Lehrstuhl für Software-Technologie
Fachbereich Informatik
Universität Dortmund
44221 Dortmund

Diplomarbeit

XAM

—

Eine abstrakte Laufzeitumgebung für ξML

Jens Schröder

29. August 2002

Gutachter:
Prof. Dr. Ernst-Erich Doberkat
Dipl.-Inf. Klaus Alfert

Inhaltsverzeichnis

Abbildungsverzeichnis

UML-Diagrammverzeichnis

Grammatikfragmentverzeichnis

XML-Dokumenteverzeichnis

1 Einleitung

Die vorliegende Diplomarbeit bewegt sich im Kontext der Erstellung von Multimediapräsentationen für die universitäre Lehre. Aufgabe der Diplomarbeit ist die Konzeption der e**X**tensible **A**bstract **M**achine (XAM), einer abstrakten Laufzeitumgebung für solche Präsentationen. Sie dient als Schnittstelle zwischen der darstellungsneutralen Spezifikation einer Multimediapräsentation und konkreten Laufzeitumgebungen, mit denen die so beschriebenen Präsentationen angezeigt werden können. Die XAM ist eingebettet in ein System namens ξML, das zur Erstellung von Multimediapräsentationen, die universitäre Lerninhalte vermitteln sollen, dient.

Auf diese kurze Charakterisierung der XAM wird im weiteren Verlauf des Kapitel ausführlicher eingegangen. Als Motivation schildere ich zunächst den Ursprung der Arbeit, ehe ich auf wesentliche Aspekte bei der Konzeption einer abstrakten Laufzeitumgebung eingehe und daran die mit dieser Arbeit zu lösende Problematik und den gewählten prinzipiellen Lösungsansatz skizziere. Abschließend wird die zur Konzeption der XAM gewählte Vorgehensweise und die sich daraus ergebende Struktur der Arbeit vorgestellt.

1.1 Ursprung der Arbeit

Ausgangspunkt der Überlegungen zum ξML-System und damit auch zu dieser Arbeit ist das Altenberger Dom-Projekt [ALFERT et al. 1999], in dem eine Multimediapräsentation erstellt wurde, die am Beispiel des Bauwerks Altenberger Dom baugeschichtliche Informationen der Epoche der Gotik für die universitäre Lehre multimedial aufbereitet (siehe Abschnitt 3.1).

In dem Projekt wurde ein generativer Ansatz gewählt, um die Erstellung der Inhalte und der Gestaltung der Multimediapräsentation von den Aspekten ihrer technischen Realisierung zu trennen. Mit Hilfe der für dieses Projekt entworfenen XML-basierten [BRAY et al. 2000] Sprache ADML werden die Inhalte der Präsentation, also die Medien und die Verlinkung zwischen ihnen, spezifiziert, und von einem Übersetzungswerkzeug, dem so genannten Converter, in verschiedene darstellbare Formate überführt. Die gestalterischen Aspekte wurden in der Projektvorbereitungsphase festgelegt und starr im Converter implementiert.

Diese Tatsache erwies sich für eine Anwendung des Converters auf andere, ähnlich geartete Multimediapräsentationen als hinderlich. Die feste Implementierung der gestalterischen Aspekte im Converter und ein Mangel an Modularität macht eine Wiederverwendung von im Altenberger Dom-Projekt entwickelten Werkzeugen unmöglich. Das ξML-Projekt hat sich die Lösung dieser Probleme als Ziel gesetzt, in dem es den im Altenberger Dom-Projekt bewährten Generatoransatz aufgreift und weiterentwickelt. ξML wendet dazu mit einem XML-basierten, komponentenorientierten Ansatz bekannte und bewährte Methoden der Software-Technologie auf die Erstellung von Multimediapräsentationen an.

Das ξML-System gliedert sich im Wesentlichen in zwei Bereiche. Der erste Bereich betrifft die Erstellung einer darstellungsneutralen Spezifikation der Multimediapräsentationen, die im Gegensatz zum Altenberger Dom-Projekt auch ihre gestalterischen Aspekte mit einschließt. In [SCHÜTTE 2002] wird für diesen Bereich ein Konzept entwickelt, um eine solche Spezifikation komponentenorientiert vornehmen zu können. Im zweiten Bereich wird diese Spezifikation in darstellbare Formen überführt. Zu diesem Zweck soll mit dieser Arbeit die XAM als abstrakte, also von einer konkreten Zielplattform losgelöste, Laufzeitumgebung des ξML-Systems konzipiert werden. Ein detaillierter Überblick über das ξML-System wird in Kapitel 3 gegeben.

Als generellen kritischen Punkt des generativen Ansatzes des Altenberger Dom-Projekts kann die Spezifikation der Multimediapräsentation angesehen werden, die die Schnittstelle zwischen den inhaltlichen und den technischen Aspekten der Multimediapräsentation darstellt. Es muss ein Weg gefunden werden, diese Spezifikation auf eine dem Autor der Inhalte der Präsentation, der üblicherweise keine informatische Ausbildung besitzt, angemessen Weise vorzunehmen. In [HEIDUCK 1999] wird diese Problematik als *Kluft in der Multimediaentwicklung* bezeichnet und zur Lösung eine Spezifikationssprache entwickelt, die auf der Verwendung von Natursprachlichkeit und Unschärfe beruht. Diese Problematik der geeigneten Erstellung einer Spezifikation der Multimediapräsentation ist allerdings im ersten Bereich des ξML-Systems anzusiedeln und wird daher im Rahmen dieser Arbeit nicht weiter verfolgt.

1.2 Konzeption einer abstrakten Laufzeitumgebung

Die Aufgabe dieser Diplomarbeit ist die Konzeption einer abstrakten Laufzeitumgebung namens XAM für das ξML-System. Mit einer solchen Laufzeitumgebung wird aus einer darstellungsneutralen Beschreibung einer Multimediapräsentation eine anzeigbare Form erzeugt. Die Konzeption der XAM umfasst einmal den Entwurf eines solchen Werkzeugs, zum anderen aber auch die Findung geeigneter Abstraktionen von Leistungsmerkmalen konkreter Laufzeitumgebungen für Multimediapräsentationen, um das Eingabeformat der XAM zu definieren, mit dem darstellungsneutral Multimediapräsentationen beschreiben werden können.

Durch die Verwendung einer solchen Laufzeitumgebung bei der Erstellung von Multimediapräsentationen hat man die Möglichkeit, das Erstellen der in der Multimediapräsentation dargestellten Informationen von einer technischen Realisierung der Darstellung zu trennen. Diese Trennung bringt mehrere Vorteile mit sich: Zum einen können die beiden Aspekte, das Erstellen des Inhalts und die technischen Realisierung der Darstellung der Präsentation, getrennt voneinander von jeweiligen Experten bearbeitet werden. Zum anderen können durch eine darstellungsneutrale Spezifikation der Inhalte leicht neue Zielplattformen unterstützt werden, ohne dass bereits formulierte Inhalte angepasst werden müssten. In [CARSON 1993] wird im Rahmen eines Referenzmodells für graphische Systeme mit der Modellierung eines Virtual Environment aus der gleichen Motivation ein ähnlicher Ansatz wie der der abstrakten Laufzeitumgebung für Multimediapräsentationen verwendet.

Neben dieser generellen Aufgabe, aus einer darstellungsneutral beschriebenen Multimediapräsentation eine anzeigbare Form zu erzeugen, müssen bei der Konzeption der XAM einige besondere Probleme gelöst werden, die sich aus den Eigenschaften bzw. Anforderungen des umschließenden ξML-System ergeben. Da das ξML-System explizit für die Realisierung der Erstel-

lung unterschiedlicher Multimediapräsentationen gedacht ist, muss die XAM erweiterbar konzipiert werden. Die Erweiterbarkeit betrifft daher zum einen die Unterstützung der Darstellung neuer multimedialer Elemente. Weil aber die Darstellung auf unterschiedlichen Zielplattformen möglich sein soll, muss die XAM zum anderen auch eine Unterstützung für die Verwendung unterschiedlicher Zielplattformen anbieten. Auf Grund des komponentenorientierten Ansatzes des ξML-Systems soll der Entwurf der XAM ebenfalls komponentenorientiert erfolgen, da das gewählte Modularisierungskonzept ein wesentlicher Bestandteil der Realisierung der Erweiterbarkeit im ξML-System darstellt.

Als genereller Ansatz zur Lösung der gerade geschilderten Aufgaben wird die XAM als ein hochgradig konfigurierbarer Generator, der sowohl in der Ein- als auch in der Ausgabe erweiterbar ist, entworfen. Dazu werden neben der Verwendung und Weiterentwicklung geeigneter Konzepte aus dem Compilerbau Methoden aus der Software-Technologie wie beispielsweise Komponenten- und Objektorientierung oder die Anwendung von Architektur- und Entwurfsmustern eingesetzt.

Da es sich bei der XAM um eine Laufzeitumgebung handelt, orientiert sich das Abstraktionsniveau der von ihr angebotenen Ausdrucksmöglichkeiten zur Beschreibung von Multimediapräsentationen an den zur Darstellung verwendeten Zielmaschinen und ist damit niedriger, als bei der Spezifikation der Präsentation, die durch den Autor der Inhalte vorgenommen wird. Die von der XAM zur Verfügung gestellten Ausdrucksmöglichkeiten bilden die plattformunabhängige Basis, auf der sich die Spezifikation abstützt.

Die Spezifizierung und die Überführung der unterschiedlichen Abstraktionsniveaus aus der Spezifikation in die Eingabe der XAM wird in [SCHÜTTE 2002] behandelt, da sie Bestandteil des ersten Bereichs des ξML-Systems ist.

1.3 Vorgehen und Struktur der Arbeit

Meine Arbeit gliedert sich grob in drei wesentliche Teile. Ich beginne mit einem einleitenden Teil (Kapitel 1 bis 4), der die Grundlagen meiner Arbeit vorstellt. Der zweite Teil (Kapitel 5 bis 10) behandelt den Entwurf der XAM. Den abschließenden Teil (Kapitel 11 und 12) bilden die Bewertung meines Entwurfs und ein Ausblick. Auf die einzelnen Kapitel werde ich bei der Erläuterung meiner Vorgehensweise detaillierter eingegangen.

Meine Vorgehensweise ergibt sich aus den einzelnen Aufgaben, die generell bei der Entwicklung von Software gelöst werden müssen. Im Einzelnen lassen sich folgende, nach der Reihenfolge ihrer Bearbeitung geordnete, Aufgaben identifizieren:

- Analyse des Problembereichs
- Aufstellen der Anforderungen an die zu erstellende Software auf Grundlage der Problembereichsanalyse
- Entwurf der zu erstellenden Software
- Überprüfen des Entwurfs an den aufgestellten Anforderungen

Die bei der Software-Entwicklung üblichen Aufgaben der Realisierung und des Testens der Software entfallen im Rahmen dieser Arbeit, da es sich um eine rein konzeptionelle Arbeit

handelt, wie in Abschnitt 4.3 noch detaillierter erörtert wird. Im Folgenden skizziere ich kurz meine Vorgehensweise beim Lösen dieser Aufgaben.

Die Analyse des Problembereichs beginnt zunächst durch die Einordnung der Arbeit in den Kontext der Literatur (Kapitel 2) und des ξML-Projektes (Kapitel 3). Sie schließt mit der Konkretisierung der Aufgabenstellung, der Definition der mit dieser Arbeit zu erreichenden Ziele (Kapitel 4) und der Vorstellung einer beispielhaften Anwendung aus dem Problembereich, der giMLi-Präsentation (Abschnitt 5.1).

Der nächste Schritt ist nun, aus der Analyse des Problembereichs einen Anforderungskatalog für die XAM abzuleiten, der mit dem Entwurf erfüllt werden muss. Dies geschieht in Abschnitt 5.2. Der Anforderungskatalog dient zum einen als Grundlage des Entwurfs der XAM und zum anderen als Kriterium, an dem die beim Entwurf der XAM gewählten Konzepte überprüft werden.

Entlang der einzelnen Bestandteile der XAM wird in den Kapiteln 6 bis 10 der Gesamtentwurf vorgestellt. Auf Grund der aufgestellten Anforderungen entwickle ich Konzepte, mit denen die einzelnen Elemente der XAM realisiert werden können und schildere deren Zusammenspiel abschließend in einem durchgängigen Beispiel.

Der letzte Punkt beim Vorgehen dieser Arbeit bildet Kapitel 11, in dem der Entwurf an Hand der an ihn gestellten Anforderungen und der gesteckten Ziele bewertet wird. Im Rahmen der Bewertung findet eine Positionierung der XAM gegenüber anderen Lösungen, die sich in der Literatur finden, statt. Die Arbeit endet mit einem Ausblick in Kapitel 12.

2 Der multimediale Kontext

In diesem Kapitel wird der multimediale Kontext, in dem sich diese Arbeit bewegt, an Hand entsprechender Fachliteratur diskutiert. Zunächst erläutere ich, was im Rahmen dieser Arbeit unter dem Begriff Multimedia verstanden wird. Dazu wird der Gebrauch einer Reihe von Begrifflichkeiten festgelegt, die in diesem Zusammenhang relevant sind. Danach stelle ich mit der Beschreibung und Darstellung multimedialer Inhalte unterschiedliche Aspekte von Multimedia vor. Das Kapitel schließt mit Betrachtungen zur Erstellung von Multimediapräsentationen.

2.1 Vokabular

Auf dem Gebiet von Multimedia gibt es eine Vielzahl von Begriffen, die in der Literatur mit unterschiedlichen Bedeutungen versehen sind. Um eine klare sprachliche Grundlage zu haben, werde ich in diesem Abschnitt die Bedeutung einiger Begriffe, die sie im Rahmen meiner Arbeit haben, festlegen.

Zunächst wird der Begriff des Mediums betracht, ehe auf Begriffe im Rahmen von Multimedia eingegangen wird.

2.1.1 Der Medienbegriff dieser Arbeit

Der Medienbegriff hat im Kontext von Multimedia eine zentrale Stellung. Er kann jedoch auf unterschiedlichen Abstraktionsniveaus verwendet werden. Um eine solide Grundlage für meine Arbeit zu schaffen, die ein klare Argumentation für Entwurfsentscheidungen ermöglicht, führe ich im Folgenden eine feingranularere Struktur des Medienbegriffs ein und lege eine einheitliche Sprechweise fest.

Mit Medien werden Einheiten von Informationen vermittelt (siehe Seite 288 in [MAYBURY 1993]). Die Art, wie die Informationen präsentiert werden, kennzeichnet solche Einheiten von Informationen: beispielsweise akustisch, optisch, diskret, kontinuierlich oder interaktiv. Diese Eigenschaften können auch kombiniert auftreten und sind nicht vollständig. Beispiele für solche Einheiten sind etwa Bilder und Texte (optisch und diskret), Videos (akustisch, optisch, kontinuierlich), Musikstücke (akustisch, interaktiv oder nichtinteraktiv) oder Virtual Realitiy-Filme (optisch, interaktiv). Ich fasse sie unter dem Begriff *Medientypen* zusammen:

Definition 1 (Medientypen) *Einheiten von Information, die sich in der Eigenschaft, wie sie die Informationen präsentieren, unterscheiden, werden* ***Medientypen*** *genannt. Sie sind unabhängig von Formaten, in denen die Information codiert werden.*

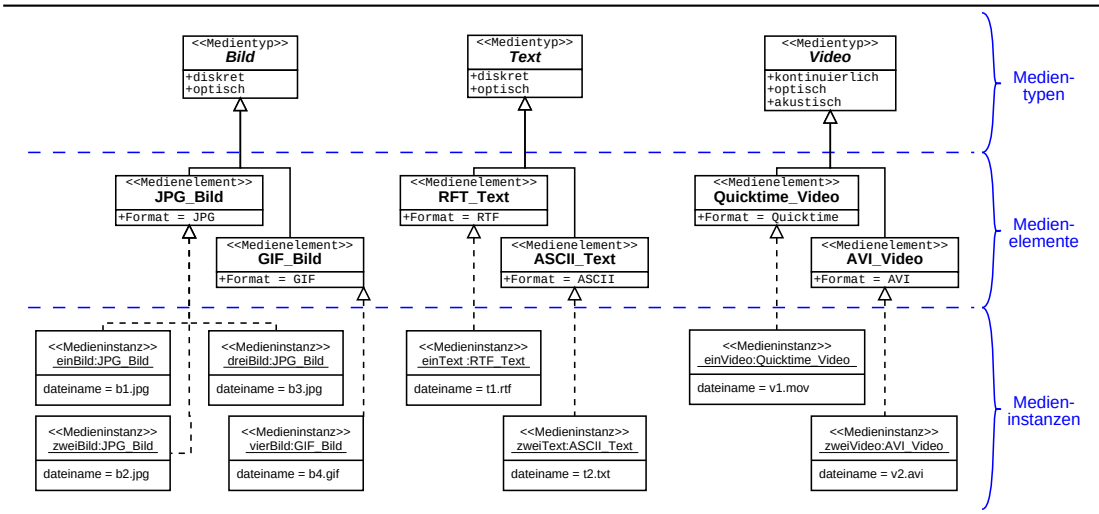
Für die einzelnen Medientypen existieren unterschiedliche Formate, in denen sie codiert werden. Für den Medientyp Bild existieren beispielsweise Formate wie JPEG oder GIF. Eine Spezialisierung von Medientypen um solche Codierungsinformationen nenne ich *Medienelemente*:

Definition 2 (Medienelemente) *Spezialisierungen von* Medientypen*, die diese um die Eigenschaft anreichern, in welchem Format sie die zu präsentierende Information codieren, werden* ***Medienelemente*** *genannt.*

Um eine Information anzeigen zu können, muss neben der Art, wie die Information präsentiert wird, und dem Format, in dem sie codiert wird, auch die eigentliche Information an sich festgelegt werden. Dazu müssen Instanzen der Medienelemente gebildet werden, die die konkret zu präsentierenden Informationen enthalten. Eine Instanz des Medienelements `JPG_Bild` kann etwa die Datei `einBild.jpg` sein. Solche konkreten Ausprägungen werden *Medieninstanzen* genannt.

Definition 3 (Medieninstanzen) *Die konkrete Ausprägung eines* Medienelements *wird* ***Medieninstanz*** *genannt. Sie beinhaltet die zu präsentierende Information.*

Zur Veranschaulichung der Strukturierung des Medienbegriffs in die unterschiedlichen Abstraktionsniveaus, die durch die obigen Definitionen beschrieben werden, stelle ich im konzeptionellen UML-Diagramm 2.1 diese Struktur exemplarisch am Hand einiger Medientypen dar. Die Unterteilung in Medientypen ergibt eine waldartige Struktur. Auf die Definition eines gemeinsamen Oberelements, das die Waldstruktur zu einer Baumstruktur zusammenführt, habe ich verzichtet, da die waldartige Struktur meines Erachtens die Wahrnehmung von Medien in der Realität geeignet modelliert. Außerdem wird ein solches einheitliches Oberelement im Rahmen dieser Arbeit nicht benötigt.



UML-Diagramm 2.1: Die Taxonomie von Medien

Mit den Definitionen werden drei Ebenen bei der Strukturierung des Medienbegriffs eingeführt. Sie werden in UML-Diagramm 2.1 auf der vorherigen Seite als horizontale, blaugestrichelte Linien angedeutet. Auf der obersten, vom Abstraktionsniveau her gesehen höchsten Stufe sind die Medientypen angesiedelt. Generelle Eigenschaften, die für einen Medientyp charakterisieren, wie er Informationen dargestellt, bilden in dem Diagramm die Attribute der Medientypen: diskret (`Bild`) oder kontinuierlich (`Video`), optisch (`Bild`, `Video`) oder akustisch (`Video`). Die Medientypen sind als abstrakte Klassen modelliert, da sie losgelöst von Formaten sind, in denen sie codiert werden. Diese Modellierung der Medientypen unterstreicht, dass sie sich auf einem Abstraktionsniveau befinden, auf dem keine konkret darstellbaren Medien beschreiben werden.

Dazu führe ich in der nächsten Ebene innerhalb meiner Struktur Medienelemente ein, die Medientypen um die Eigenschaft der Formats spezialisieren. Das Format gibt an, wie ein Medientyp codiert wird. Beispiele für eine solche Spezialisierung etwa des Medientyps `Bild` sind die Medienelemente `JPG_Bild` und `GIF_Bild`.

Um ein Medienelement, beispielsweise in einer Präsentation, zu verwenden und es dazu mit der zu vermittelnden Information zu versehen, werden Medieninstanzen, also konkrete Ausprägungen der Medienelemente, benötigt. Ein Medienelement kann aus offensichtlichen Gründen mehrfach instantiiert werden, wie in dem Diagramm zu sehen ist. Sie beinhalten die konkret darzustellenden Informationen und werden in dem Format des jeweiligen Medienelements codiert. Da sie üblicherweise in Form von Dateien vorliegen, auf die über ihren Namen zugegriffen wird, besitzen sie die Eigenschaft eines Dateinamens. Sie bilden die dritte Ebene in dem Diagramm. Beispiele für solche Medieninstanzen sind die Objekte `EinBild` für das Medienelement `JPG_Bild` oder `Zweivideo` für das Medienelement `AVI_Video`. Sie modellieren die Dateien entsprechenden Namens, mit denen in der realen Welt Instanzen von Medienelementen angegeben werden.

Diese Strukturierung wird zum einen für eine Definition multimedialer Begrifflichkeiten im folgenden Abschnitt und zum anderen im Rahmen des Entwurfs benötigt. An anderen Stellen der Arbeit, an der eine solche differenzierte Betrachtungsweise des Medienbegriffs nicht nötig ist oder es sich aus dem Kontext eindeutig ergibt, auf welchem Abstraktionsniveau er verwendet wird, benutze ich aus Gründen der besseren Lesbarkeit den Begriff Medium. Wenn ich allgemein über Medien rede, mich im Rahmen des Entwurfs aber auf der Ebene der Umsetzung von multimedialen Elementen, aus denen sich eine Multimediapräsentation zusammensetzt, befinde, wird dies durch eine kursive Schriftart hervorgehoben und der Begriff wie folgt geschrieben: *Medium*.

2.1.2 Der Multimediabegriff dieser Arbeit

Nach der Formulierung eines gültigen Medienbegriffs für diese Arbeit, betrachte ich nun den Bereich von *Multi*media. Ziel ist es, den Begriff der Multimediapräsentation im Rahmen dieser Arbeit zu formulieren und relevante Begrifflichkeiten im Kontext von Multimediapräsentationen vorzustellen.

Eine wesentliche Eigenschaft von Multimedia ist, dass zwischen einzelnen Medieninstanzen Verbindungen definiert werden können [ROISIN 1998]. Diese Verbindungen modellieren inhaltliche Zusammenhänge und können traversiert werden. Ich nenne sie *Verknüpfungen*:

Definition 4 (Verknüpfungen) *Traversierbare inhaltliche Verbindungen zwischen* Medieninstanzen *werden* ***Verknüpfungen*** *genannt.*

Eine einfache Ausprägung einer solchen Verknüpfung ist etwa ein HTML-Link.

Medieninstanzen und Verknüpfungen können komponiert werden. Die Verknüpfungen sind nicht auf Medieninstanzen der selben Komposition beschränkt, sondern können sich auch auf Medieninstanzen anderer Kompositionen beziehen. Eine solche Komposition wird *Multimediadokument* genannt. Im Rahmen der folgenden Definition eines Multimediadokuments wird der Begriff Multimedia so charakterisiert, dass er sich klar von der einfachen Verwendung mehrerer Medieninstanzen unterscheidet:

Definition 5 (Multimediadokument) *Die Komposition einer Menge von* Medieninstanzen, *von denen mindestens eine eines anderen* Medientyps *als Text sein muss, und* Verknüpfungen *wird* ***Multimediadokument*** *genannt. Die Verknüpfungen können zwischen Medieninstanzen des selben Multimediadokuments und zwischen Medieninstanzen anderer Multimediadokumente definiert werden.*

Diese Komposition, die in einem Multimediadokument gruppiert wird, muss offensichtlich auch notiert werden können. Dazu wird ein Datenmodell, das einer solchen Notation zu Grunde liegt, benötigt:

Definition 6 (Multimediadokumentmodell) *Das Datenmodell, das die Komposition von* Medieninstanzen *und* Verknüpfungen *beschreibt, wird* ***Multimediadokumentmodell*** *genannt.*

Dokumente für Internetseiten sind beispielsweise im Multimediadokumentmodell HTML [W3C 1999] verfasst. Es existieren aber auch noch eine Vielzahl anderer Modelle, von denen ich einige in Abschnitt 2.2.2 vorstelle.

Nach diesen Vorarbeiten kann nun der Begriff einer *Multimediapräsentation* betrachtet werden. Mit Multimediadokumenten bzw. mit den in ihnen verwendeten Medieninstanzen und Verknüpfungen sollen jeweils Informationen für die Wahrnehmung durch den Menschen aufbereitet werden [NEWCOMB et al. 1991]. Sie lassen sich nach inhaltlichen Aspekten zu so genannten *Multimediapräsentationen* zusammenfassen:

Definition 7 (Multimediapräsentation) *Eine Sammlung von* Multimediadokumenten, *die der Vermittlung von einem oder mehreren übergeordneten Sachverhalten dienen, wird* ***Multimediapräsentation*** *genannt.*

Ein Beispiel für eine Multimediapräsentation ist die in Abschnitt 3.1 detaillierter vorgestellte Altenberger Dom-Präsentation [ALFERT et al. 1999], deren übergeordneter Sachverhalt die Vorstellung von Aspekten der Gotik am Beispiel des Bauwerks Altenberger Dom ist.

Bei den hier vorgestellten Definitionen habe ich mich von den im Rahmen von HyTime in [NEWCOMB et al. 1991] vorgestellten Begrifflichkeiten und dem im Rahmen von PREMO, einem Standard für Multimediapräsentationssysteme, in [HERMAN et al. 1996] vorgestellten Multimedia- bzw. Hypermedia-Paradigma leiten lassen. Da bei meiner Arbeit der Fokus auf

dem multimedialen Aspekt, also der Präsentation von Medieninstanzen unterschiedlicher Medientypen, liegt und nicht so sehr auf der Betonung der Verknüpfung von diesen Medieninstanzen mit einem Link-Konzept, unterscheide ich im Gegensatz zu den beiden oben genannten Arbeiten nicht zwischen den Begriffen Multimedia und Hypermedia. Durch die oben vorgestellten Definitionen fällt im Kontext dieser Arbeit unter den Begriff Multimediadokument auch ein Hypermediadokument.

Der Begriff der Multimediapräsentation, wie er hier vorgestellt wird, spricht explizit an, woraus Multimediapräsentationen bestehen. Mittels Multimediadokumente wird festgelegt, welche Medieninstanzen verwendet und wie sie strukturiert und zueinander in Beziehung gesetzt werden. Er beschreibt die persistenten Aspekte einer Multimediapräsentationen. Wie [ALFERT 1999] feststellt, ist dieser *Dokumentaspekt* jedoch nur eine Sichtweise auf eine Multimediapräsentation. Bei der Darstellung einer Präsentation hingegen ist die Sichtweise mehr auf das dynamische Verhalten der Präsentation gelegt, die in meiner Definition implizit enthalten ist: beispielsweise müssen die Mediendaten durch geeignete Software angezeigt, Benutzerinteraktion zum Verfolgen von Verknüpfungen ermöglicht und die Reaktion der Präsentation auf Benutzerinteraktion realisiert werden. Diese Sichtweise betrifft den *Programmaspekt* einer Präsentation. Die Verschmelzung dieser beiden Aspekte kennzeichnet eine Besonderheit von Multimediapräsentationen und wird *Dokument-Programm-Dualismus* genannt [ALFERT 1999].

2.2 Multimediadokumentmodelle

In diesem Abschnitt gehe ich auf den Dokumentaspekt von Multimediapräsentationen genauer ein. Mit ihm wird die persistente Festlegung der Inhalte einer Multimediapräsentationen, die in Multimediadokumenten formuliert wird, adressiert. Ich erörtere im Folgenden einige für diese Arbeit relevante Eigenschaften von Multimediadokumenten und stelle Beispiele für die den Multimediadokumenten als Datenmodelle zu Grunde liegenden Multimediadokumentmodelle vor.

2.2.1 Eigenschaften von Multimediadokumenten

Multimediadokumente sind die anzeigbaren Beschreibungen von Multimediapräsentationen. In Multimediadokumenten wird, wie in Definition 5 auf der vorherigen Seite beschrieben, die Komposition von Medieninstanzen (als konkrete Einheiten von Information) und Verknüpfungen formuliert. Diese Definition ist konform mit der in [ROISIN 1998], Abschnitt 2.1, gegebenen Definition eines Multimediadokuments:

> [A document is] *a set of basic information entities semanticly linked together in order to constitute a message.*

Multimediadokumente können also als eine Sammlung von Entitäten angesehen werden. Die Komposition dieser Entitäten kann nach [ROISIN 1998] in unabhängige Dimensionen strukturiert werden:

Logische Dimension In der logischen Dimension werden Elemente des Dokuments nach inhaltlichen Aspekten komponiert. Inhaltliche Strukturen, die in Form eines azyklischen

Graphen modelliert werden können, werden durch die Komposition in der logischen Dimension gebildet. Ein Beispiel dafür ist die inhaltliche Strukturierung der Altenberger Dom-Präsentation in Inhaltsverzeichnis, Themen, Unterthemen und Glossar [ALFERT et al. 1999].

Navigationsdimension Durch die Komposition in der Navigationsdimension werden die Möglichkeiten des Benutzers mit der Präsentation zu Interagieren beschrieben. Nach [ROISIN 1998] besteht die Interaktion im Wesentlichen aus der Möglichkeit des Benutzers durch geeignete Aktionen die Verfolgung von Links auszulösen, und so durch die Präsentation navigieren zu können.

[BOLL et al. 1999] nennt diese Art von Interaktion *Navigationsinteraktion* und führt mit der *Design-Interaktion* eine weitere Art der Interaktion ein. Mit ihr erhält der Benutzer die Möglichkeit, beim Betrachten der Präsentation ihre visuelle und akustische Gestaltung zu verändern. Beispiele für diese Art der Interaktion wären die Möglichkeiten zum interaktiven Ändern der Auflösung von in der Präsentation verwendeten Bildern oder der Qualität von in der Präsentation abgespielten Musikstücken.

Räumliche Dimension Bei der Komposition der Elemente in der räumlichen Dimension werden zum einen die visuellen Elemente auf dem Ausgabemedium positioniert. Zum anderen werden typographische Eigenschaften der Präsentation in dieser Dimension des Multimediadokuments festgelegt. [ROISIN 1998] unterscheidet diese typographischen Eigenschaften in *Stile*, die das Layout des Inhalts, etwa durch Schriftarten und Farben, definieren und in die *physikalische Struktur*, die Eigenschaften wie Seitengröße oder Abstände festlegt.

[BOLL et al. 1999] stellt für den Aspekt der Positionierung der visuellen Elemente auf dem Ausgabemedium, das üblicherweise als zweidimensionale Präsentationsfläche modelliert wird, drei unterschiedliche Ansätze vor. Bei der *absoluten Positionierung* wird mit Hilfe eines Koordinatenpaares das visuelle Element fest auf der Präsentationsfläche plaziert. Bei dem Ansatz, der *Richtungsrelationen* verwendet, wird die Positionierung flexibler angegeben durch Ausdrücke wie nord oder nord-west. Als dritter Ansatz wird die Positionierung mit *topologischen Relationen* vorgestellt, der auf mengentheoretischen Überlegungen beruht. Zwischen zwei zusammenhängenden Regionenobjekten können acht topologische Relationen wie beispielsweise disjunkt oder beinhaltet angegeben werden ([EGENHOFER und FRANZOSA 1991] zitiert in [BOLL et al. 1999])

Zeitliche Dimension Bei der Komposition der Elemente in der zeitlichen Dimension werden die Elemente des Multimediadokuments zeitlich angeordnet und somit der zeitliche Ablauf der mit dem Dokument beschriebenen Multimediapräsentation festgelegt. In [BOLL et al. 1999] werden drei zeitliche Modelle vorgestellt, mit denen diese Anordnung vorgenommen werden kann. In dem *punkt-basierten Modell* wird die zeitliche Ausdehnung eines Elements durch seinen Anfangs- und seinen Endpunkt beschrieben. Entsprechend dieser Zeitpunkte können die Medien angeordnet werden. Das *intervall-basierte Modell* benutzt Intervalle und Beziehungen zwischen den Intervallen, um eine zeitliche Anordnung festzulegen. Bei dem *ereignis-basierten Modell* wird der Ablauf einer Präsentation

durch Ereignisse bestimmt. Ereignisse, wie etwa das Erreichen des Endes eines Videos, werden bei der Präsentation von den Elementen des Multimediadokuments ausgelöst und sind mit Aktionen verbunden, die bei Eintritt des Ereignisses ausgeführt werden. Neben der Verwendung dieser drei Modelle kann der Ablauf einer Präsentation auch über *Skript-Programme* gesteuert werden, die, je nach Mächtigkeit der verwendeten Skript-Sprache, eine sehr flexible Möglichkeit zur Definition von zeitlichen Abhängigkeiten zwischen den Medien darstellen [BOLL et al. 1999].

Die gerade vorgestellten vier Dimensionen, in denen die Elemente eines Multimediadokuments komponiert werden können, finden sich auch in den Multimediapräsentationen wieder, da Multimediapräsentationen als Sammlung von Multimediadokumenten angesehen werden können (siehe Definition 7 auf Seite 8). Im Folgenden spreche ich daher von den *vier Dimensionen von Multimediapräsentationen*. Eine vollständige Beschreibung einer Multimediapräsentation muss generell mit seinen Elementen diese vier Dimensionen adressieren.

Im Folgenden Abschnitt stelle ich kurz ausgewählte Multimediadokumentmodelle vor und erläutere ihren zum Teil unterschiedlichen Fokus auf die Adressierung der einzelnen Dimensionen.

2.2.2 Beispiele für Multimediadokumentmodelle

Ein weit verbreitetes Multimediadokumentmodell ist die Hypertext Markup Language (HTML) [W3C 1999], die auf dem SGML-Standard [ISO-SGML 1986] basiert. Ihre Syntax erlaubt es, Texte mittels SGML-Elemente um strukturelle Informationen anzureichern. Dadurch können zum einen logische Strukturen definiert werden, um eine hierarchische Untergliederung der Inhalte in Paragraphen und unterschiedliche Überschriftebenen vorzunehmen. Zum anderen werden dadurch Verknüpfungen in Form von Links formuliert, die durch Interaktionsmöglichkeiten während der Präsentationsphase verfolgt werden können. Durch die oben genannte Auszeichnung der Texte mit dem so genannten Markup ist es auch möglich, komplexere inhaltliche Elemente wie Tabellen anzugeben oder externe Objekte, wie Instanzen unterschiedlicher Medienelemente, Java-Applikationen oder Skript-Programme, einzubinden. Diese externen Objekte werden während der Präsentationszeit von einer Darstellungssoftware wie dem Netscape Navigator oder dem Mircosoft Internet Explorer interpretiert und angezeigt. Die ausgezeichneten Elemente können über einen Attributierungsmechanismus entweder direkt oder indirekt über so genannte Stylesheets mit Layoutinformationen versehen werden. Komplexere zeitliche Aspekte können durch die oben genannten Skript-Objekte realisiert werden. Dieses Multimediadokumentmodell hat sich bei der Beschreibung der Inhalte im Internet durchgesetzt. Die Inhalte des Internets werden in Form einzelner Seiten mit HTML-Dokumenten beschrieben. Ein Nachteil ist, dass die Darstellung der Layoutaspekte und der externen Objekte stark von der Implementierung der jeweiligen Darstellungssoftware abhängig und nicht einheitlich festgelegt ist.

Um ein standardisiertes Dokumentenmodell bereitzustellen, mit dem wohldefiniert synchronisierte Multimediapräsentationen im Web beschrieben werden können, wurde die Synchronized Multimedia Integration Language (SMIL) [W3C 2001] entwickelt. Sie basiert auf XML [BRAY et al. 2000], einem Standard, mit dem Markup-Sprachen definiert werden können und der die Nachfolge von SGML angetreten hat. Bei XML-basierten Sprachen werden in einer so genannten

Document Type Definition (DTD) gültige Elemente und ihre Attribute einer Sprache festgelegt. Mit Hilfe von so definierten Schedule-Elementen werden bei SMIL zeitliche Abhängigkeiten zwischen Medieninstanzen der Multimediapräsentation angegeben. Durch Switch-Elemente hat ein Autor die Möglichkeit, Alternativen für den Ablauf der Präsentation oder für die Qualität der verwendeten Medieninstanzen anzugeben. SMIL erlaubt sowohl eine absolute als auch eine relative Positionierung der Medieninstanzen. Der Fokus dieses Multimediadokumentenmodells liegt auf der Adressierung der zeitlichen Dimension von Multimediapräsentationen.

Mit dem Multimediadokumentmodell HyTime [NEWCOMB et al. 1991] wurde ein Standard entwickelt, der sehr allgemein die Beschreibung von Strukturen von Multimediadokumenten ermöglicht. Er basiert auf SGML und stellt wohldefinierte Primitive bereit, die es ermöglichen, Medieninstanzen, unabhängig von den Formaten, in denen sie codiert sind, miteinander zu verknüpfen und in Raum und Zeit anzuordnen. Diese Primitive werden Architectural Forms genannt und besitzen eine vordefinierte Semantik und Attribute. Sie sind entsprechend ihrer Aufgaben in so genannte *Module* organisiert. Durch eine Referenz auf solche Primitive können Medieninstanzen die Semantik und Attribute der Primitive erben und so an der in den Modulen bereitgestellten Funktionalität teilhaben. Es werden Module bereitgestellt, die die unterschiedlichen Dimensionen von Multimediapräsentationen adressieren. Neben dem Base-Modul, das Basisfunktionalität von HyTime definiert, existieren Module, die Aspekte der Verknüpfung und Navigation (Location-Address-Modul und Hyperlink-Modul), der zeitlichen und räumlichen Anordnung der Elemente (Finite-Coordinate-Space-Modul) und der Darstellung der Elemente (Event-Projection-Modul und Object-Modification-Modul) adressieren. Die Anordnung der Elemente einer Multimediapräsentation werden bei HyTime durch einen abstrakten Koordinatenraum modelliert, in dem sowohl zeitliche, als auch räumliche Aspekte formuliert werden können. Ähnlich wie bei dem Dexter-Modell, das ich im folgenden Abschnitt vorgestelle, wird bei HyTime auch ein abstrakter Adressierungsmechanismus eingeführt, der durch das Konzept der locations auch indirekte Adressierung ermöglicht.

Bei DoDL [DOBERKAT 1996] können die Struktur, die Verknüpfung und die Inhalte einer Multimediapräsentation getrennt voneinander spezifiziert werden. Dabei wird, im Gegensatz zu den vorherigen Multimediadokumentmodellen, keine Markup-Sprache, sondern Konzepte aus der Objektorientierung wie etwa *Klassen* oder *Spezialisierungen* verwendet. Auch hier wird ein abstrakter Mechanismus zur Adressierung der inhaltlichen Elemente mittels so genannter Positionen verwendet.

Konzepte der hier kurz vorgestellten Multimediadokumentmodelle, die Einfluss auf meinen Entwurf haben, werde ich in den entsprechenden Entwurfskapiteln detaillierter vorstellen.

2.3 Multimediapräsentationssysteme

In diesem Abschnitt stelle ich einige ausgewählte Multimediapräsentationssysteme kurz vor. Sie ermöglichen die Darstellung von Multimediapräsentationen und adressieren somit ihren Programmaspekt. Der Umfang der Systeme reicht von reinen Abspielprogrammen für bestimmte Formate von Multimediapräsentationen bis hin zu Autorensystemen, die sowohl die Erstellung als auch die Darstellung der Präsentationen ermöglichen. Die Betrachtung der hier erwähnten Systeme dient als Grundlage, um den Rahmen, in dem sich die XAM generell bewegt, abzuste-

cken.

Auf dem kommerziellen Gebiet von Multimediapräsentationen findet meist eine Trennung von Erstellung und Darstellung der Präsentationen statt. Als persistente Austauschformate werden meist standardisierte Multimediadokumentmodelle verwendet. Daher lassen sich auch hier die reinen Abspielprogramme finden. Beispiele für solche Programme etwa im Rahmen des Internets sind Webbrowser wie der Netscape Navigator oder der Mircosoft Internet Explorer, die in dem Multimediadokumentmodell HTML [W3C 1999] beschriebene Multimediapräsentationen darstellen können. Für das Multimediadokumentmodell SMIL [W3C 2001], das ebenfalls im Umfeld des Internets mit starkem Fokus auf die Modellierung von zeitlichen Beziehungen innerhalb der Präsentation entwickelt wurde, existiert beispielsweise ein Abspielprogramm namens SOJA [HELIO 1999], das als Erweiterung der oben genannten Webbrowser realisiert ist.

Auf dem Gebiet der Forschung lassen sich meist Systeme, die sowohl die Erstellung, als auch die Darstellung von Multimediapräsentationen ermöglichen, finden. Sie werden üblicherweise zur Evaluierung von Forschungsergebnissen im Kontext von Multimedia entwickelt. Auf den Erstellungsaspekt solcher Systeme gehe ich gesondert im nächsten Abschnitt ein, da er die wesentliche Aufgabe des diese Diplomarbeit umschließenden ξML-Projekts ist.

Mit dem Dexter-Modell [HALASZ und SCHWARTZ 1994] wird ein Referenzmodell für Multimediasysteme eingeführt, um bestehende Lösungen miteinander vergleichen zu können. Es untergliedert ein Multimediasystem in drei Ebenen und führt eine komponentenbasierte Beschreibung der Elemente von Multimediapräsentationen ein. Die Ebene des so genannten storage layer betrifft die persistente Beschreibung eines Multimediasystems. In ihr werden die Elemente der Präsentation, hier Komponenten genannt, und die Verbindungen zwischen diesen Komponenten definiert. Das Modell beschreibt losgelöst von technischen Realisierungen Mechanismen, die zur Beschreibung der unterschiedlichen Aspekte von Multimediapräsentationen, wie sie etwa mit den vier Dimensionen von Multimediapräsentationen in Abschnitt 2.2.1 vorgestellt werden, notwendig sind. Diese Ebene bildet das Hauptaugenmerk des Referenzmodells. Während auf dieser Ebene die Strukturen zwischen den einzelnen Komponenten beschrieben werden, wird auf Ebene des *within component layer* die Struktur innerhalb der Komponenten betrachtet. Abgesehen von der Formulierung einer Anforderung nach der Möglichkeit, Bereiche innerhalb von Komponenten adressieren zu können und so eine Schnittstelle zu der Ebene des storage layers anzubieten, wird diese Ebene in dem Dexter-Modell nicht weiter berücksichtigt. Aspekte, die die Darstellung einer Multimediapräsentation betreffen, sind auf der Ebene des so genannten runtime layer angeordnet. Auf dieser Ebene werden, wieder unabhängig von konkreten technischen Lösungen, einige Mechanismen vorgestellt, die zur Darstellung ein Multimediapräsentation benötigt werden. Das Hauptaugenmerk des Modells ist auf der Definition und Darstellung der Verknüpfungstruktur von Multimediapräsentationen gelegt, zeitliche Abhängigkeiten werden nicht berücksichtigt.

Diese zeitlichen Abhängigkeiten werden in dem Amsterdam Hypermedia Model (AHM) [HARDMAN et al. 1994] adressiert. Das AHM ist eine Erweiterung des Dexter-Modells zum einen um die zeitlichen Aspekte, aber auch um eine detailliertere Modellierung von Verknüpfungen mit Hilfe eines Kontextbegriffs, und um Präsentationsattribute, die auf einem hohen Abstraktionsniveau Eigenschaften der Darstellung festlegen. Einige Überlegungen und Konzepte dieser Referenzmodelle sind in den Entwurf der XAM eingeflossen. An den entsprechenden Stellen werde ich darauf hinweisen.

Diese beiden Modelle wurden in ein Multimediasystem namens CMIFed [VAN ROSSUM et al. 1993] umgesetzt, das auf der Grundlage des CMIF-Multimediadokumentenmodells [BULTERMAN et al. 1991] Multimediapräsentationen erstellt und darstellt. Eine Weiterentwicklung des Systems namens GRiNS [HARDMAN et al. 1998] setzt auf dem Multimediadokumentmodell SMIL auf und hat mittlerweile den Sprung zu einem kommerziellen Werkzeug vollzogen [ORATRIX DEVELOPMENT BV 2001].

Es existieren auch einige XML-basierte Ansätze [BRAY et al. 2000], wie etwa das Madeus-System ([JOURDAN et al. 1998] und [VILLARD et al. 2001]), das unter Verwendung mehrerer Transformationsschritte eine Multimediapräsentationen erstellt und darstellt, oder WCML [GAEDKE et al. 1999], das einen komponentenorientierten Ansatz verfolgt. Diese Systeme stelle ich hier nicht weiter vor, da es den Umfang dieser übersteigen würde. Auf relevante Aspekte werden ich im Rahmen der Bewertung (Kapitel 11) eingehen, in der ich meine Arbeit gegenüber anderen Lösungen positioniere.

Weiterhin sei mit PREMO [HERMAN et al. 1996] ein Standard erwähnt, der auf die Darstellung von Multimediapräsentationen ausgerichtet ist. Sein Fokus liegt dabei aber auf der Darstellung von komplexen Medientypen wie etwa virtuelle Welten, die aber aus in Kapitel 4 vorgestellten Gründen mit der hier betrachteten Arbeit gerade nicht berücksichtigt werden.

2.4 Erstellen von Multimediapräsentationen

Aufgabe des dieser Arbeit zu Grunde liegenden ξML-Systems, das ich in Kapitel 3 genauer vorstelle, ist, die Erstellung von Multimediapräsentationen im Rahmen der universitären Lehre zu unterstützen. In diesem Abschnitt werden daher einige relevante Aspekte, unter denen die Erstellung von Multimediapräsentationen in der Literatur betrachtet wird, diskutiert.

2.4.1 Aufgaben bei der Erstellung von Multimediapräsentationen

Generell lassen sich bei der Erstellung von Multimediapräsentationen gemäß des Dokument-Programm-Dualismus zwei wesentliche Aufgaben identifizieren: Zum einem die geeignete Auswahl und Erstellung der Inhalte, und zum anderen die technische Realisierung der Präsentation der Inhalte. Beide Aufgaben sind für sich genommen komplex und erfordern unterschiedliche Fähigkeiten. Daher ist eine Trennung dieser beiden Aufgaben sinnvoll, wie auch [HEIDUCK 1999], Abschnitt 1.5, feststellt.

Für die Aufgabe, die die Erstellung der Inhalte von Multimediapräsentationen betrifft, führe ich die Rolle des *Multimediaautors* ein, für die Realisierung der Darstellung der Inhalte die Rolle des *Multimediatechnikers*. In Projekten, die Multimediapräsentationen im Rahmen der universitären Lehre erstellen, wie sie in dieser Arbeit betrachtet werden, nehmen diese Rollen oft unterschiedliche Personen ein [HEIDUCK 1999], wie etwa bei dem Altenberger Dom-Projekt die Experten der Anwendungsdomäne, die Baugeschichtler, die Rolle der Multimediaautoren und die Informatiker die Rolle der Multimediatechniker eingenommen haben.

Durch die unterschiedliche Ausbildung der beiden Personengruppen traten dort in der Phase der Anforderungsanalyse einige Kommunikationsprobleme auf, zu deren Lösung in [HEIDUCK

1999] eine natursprachliche Spezifikationssprache zur prototypischen Beschreibung von Multimediapräsentationen entwickelt wird.

Das Erkennen dieser Aufgabenteilung und der damit verbundenen Probleme ist bewusst bei der Konzeption des ξML-Systems berücksichtigt worden und schlägt sich explizit in seiner Struktur nieder.

2.4.2 Paradigmen zur Erstellung von Multimediapräsentationen

Der Erstellung von Multimediapräsentationen können unterschiedliche Paradigmen zu Grunde liegen. Nach [HARDMAN et al. 1993] existieren im Wesentlichen drei unterschiedliche Ansätze.

Im *skript-basierten Ansatz* werden zeitliche und räumliche Informationen mit interpretierten, meist ungetypten Sprachen explizit programmiert. Sie erlauben einem, je nach Mächtigkeit der gewählten Sprache, auf eine sehr flexible Art, die Zusammenhänge anzugeben. Die mit ihr definierten zeitlichen und räumlichen Zusammenhänge werden jedoch, gerade bei größeren Systemen, schnell unübersichtlich und sorgen für eine schlechte Wartbarkeit des Systems. Ihre Vorteile kommen mehr bei der prototypischen Erstellung von Multimediapräsentationen zum Tragen.

Für die Realisierung komplexer zeitlicher und räumlicher Zusammenhänge bei Multimediapräsentationen, die im Multimediadokumentmodell HTML verfasst sind, wird beispielsweise häufig die Skriptsprache JavaScript [FLANAGAN 2001] verwendet. Werkzeuge zur Erstellung solcher Präsentationen, wie etwa der Macromedia Dreamweaver [WILLIAMSON und EPSTEIN 2001], bieten hierfür Unterstützung an.

Bei dem *zeitlinien-basierten Ansatz* (timeline based approach) werden die Elemente einer Multimediapräsentation entlang einer Linie, die das Fortschreiten von Zeit modelliert, angeordnet. Die Elemente sind zusätzlich mit räumlichen Informationen versehen. Der Ansatz ist intuitiv, erschwert jedoch auch die Wartung und Wiederverwendung bei größeren, mit diesem Ansatz erstellten Systemen, da beispielweise Anfangs- und Endpunkte von Elementen explizit modelliert werden und es so schwierig ist, einzelne Abschnitte einer Präsentation durch neue, die eine andere zeitliche Ausdehnung haben, zu ersetzen.

Der Macromedia Director [GROSS et al. 2000], ein Erstellungswerkzeug für Multimediapräsentationen, folgt diesem Paradigma. Gemäß der Filmmetapher, die dem Werkzeug zu Grunde liegt, manifestiert sich der zeitlinien-basierte Ansatz in dem dort benutzen Konzept des Drehbuches, in dem die Elemente der Präsentation als so genannte Darsteller angeordnet werden, um auf einer so genannten Bühne aufzutreten, sprich dargestellt zu werden.

Als drittes wird der *strukturierte Ansatz* zur Erstellung von Multimediapräsentationen vorgestellt. Grundannahme bei ihm ist, dass zum einen in Multimediapräsentationen Strukturen implizit vorhanden sind und zum anderen Autoren generell Strukturen benutzen, um ihre Arbeit zu gliedern. Die Idee des strukturierten Paradigmas ist, die den Multimediapräsentationen innewohnenden Strukturen, wie etwa die vier Dimensionen von Multimediapräsentationen ([ROISIN 1998] vorgestellt in Abschnitt 2.2.1), bei der Erstellung der Präsentation explizit zu machen und Werkzeuge, wie etwa Editoren und Beschreibungssprachen, zur Manipulation und Beschreibung dieser Strukturen bereitzustellen. Die Erstellung der Präsentationen erfolgt nach dem strukturierten Paradigma, in dem die Elemente der Multimediapräsentationen gemäß der Strukturen angeordnet werden. Idealerweise sollten dazu Werkzeuge existieren, die etwa einzelne Sichten

auf diese Strukturen anbieten, um sie editieren zu können.

Es existieren einige Editoren, die diesem Ansatz folgen und in unterschiedlichen Sichten die Bearbeitung der verschiedenen Strukturen ermöglichen. In [VAN ROSSUM et al. 1993] wird im Rahmen des CMIFed-Projekts ein Editor vorgestellt, der auf Grundlage des Multimediadokumentmodells CMIF [BULTERMAN et al. 1991] eine strukturierte Erstellung unterstützt. Desweiteren existiert ein Editor namens Thot [OPÉRA 1997], der zur Validierung von Konzepten strukturierter Multimediadokumentmodelle entwickelt wurde. Seine Technologie ist die Grundlage des Amaya-Editors [QUINT und VATTON 1997], der vom W3C als Test-Browser bzw. Test-Autorenwerkzeug zur Demonstration neuer Entwicklung von Web-Protokollen und Datenformaten benutzt wird. Im ξML-System wird dieser Ansatz auch verfolgt, wie in Abschnitt 3.3 vorgestellt wird.

Verschiedene Methoden, die zum Design von multimedialen Anwendungen existieren, wie etwa HDM [GARZOTTO et al. 1993] oder OOHDM [SCHWABE und ROSSI 1995], folgen auch diesem Ansatz und modellieren die Entwicklung multimedialer Anwendungen in unterschiedlichen Schritten, die jeweils einzelnen Sichten auf die Strukturen einer Multimediapräsentation entsprechen. Die Betrachtung von Methoden zur Erstellung multimedialer Anwendungen ist aber nicht im Fokus dieser Arbeit und wird daher, um den Umfang dieser Arbeit nicht zu überschreiten, hier nicht weiter vorgenommen.

2.4.3 Realisierungsansätze

Nach [JOURDAN et al. 1998], Kapitel 2, und [ROISIN 1998], Abschnitt 4.2, existieren bei den Autorensystemen zur Erstellung von Multimediadokumenten und somit zur Erstellung von Multimediapräsentationen zwei unterschiedliche Klassen von Systemen, die sich in der Art, wie nah die Beschreibung der Präsentation im Dokument an der Darstellung der Präsentation ist, unterscheiden: die operationalen und die regelbasierten (constraint based) Systeme.

Bei den *operationalen Systemen* werden die zeitliche und räumliche Anordnung der Elemente der Multimediapräsentation direkt spezifiziert. Es wird explizit mit Hilfe einer Skriptsprache oder einer operationalen Struktur, wie etwa einem Petrinetz oder einem Baum, angegeben, wie die Spezifikation ausgeführt werden muss. Bei der Darstellung der Präsentation wird die operationale Semantik, die durch die Struktur vorgegeben ist, direkt implementiert.

Bei den *regelbasierten Systemen* wird angegeben, welche Anforderungen (constraints) an die Darstellung der Präsentation existieren, ohne vorzugeben, wie diese Anforderungen realisiert werden sollen. Es wird somit eine deklarative Spezifikation der zeitlichen und räumlichen Anordnung der Elemente der Multimediapräsentationen vorgenommen. In einer Formatierungsphase wird diese deklarative Spezifikation in eine operationale Struktur überführt: Mit Hilfe eines so genannten Regelauflösers (constraint solver) wird die Menge der Deklarationen automatisiert in eine ausführbare Form gebracht. Es ist auch möglich, während der Ausführungszeit der Multimediapräsentation die Deklarationen zu ändern und so eine Anpassung der Präsentation an die Bedürfnisse der Betrachters vorzunehmen. Systeme, die die Erstellung von einer solche Art von anpassbaren Multimediapräsentationssystem ermöglichen, nennt man Intelligent Multimedia Presentation Systems (IMMPS). In [BORDEGONI et al. 1997] wird ein Referenzmodell hierfür vorgestellt.

Zunächst wurden in der Forschung Systeme nach dem operationalen Ansatz entwickelt, etwa

in dem oben genannten CMIFed-Projekt. Im Vordergrund standen Fragestellungen im Rahmen der strukturierten Erstellung von Multimediapräsentationen. Auch kommerziell entwickelte Autorenwerkzeuge wie etwa der Macromedia Director fallen unter diesen operationalen Ansatz, ebenso wie alle verbreiteten Standards von Multimediadokumentmodellen wie HTML oder SMIL dem operationalen Ansatz folgen. In der Forschung ist man mittlerweile einen Schritt weiter gegangen und widmet sich intensiver der Erstellung von anpassbaren Multimediapräsentationssystem und somit den regelbasierten Systemen, wie etwa bei dem Madeus-Projekt ([VILLARD et al. 2001] und [JOURDAN et al. 1998]), das auf Grundlage eines XML-basierten Multimediadokumentenmodells die Möglichkeit zur Erstellung von anpassbaren Multimediapräsentationen gibt.

Der regelbasierte Ansatz bietet offensichtlich flexiblere und ausdrucksstärkere Möglichkeiten, eine Multimediapräsentation zu beschreiben als der operationale Ansatz. Die Erstellung von solchen Autorensystemen, die auf diesen regelbasierten Formalismus beruhen, ist jedoch erheblich komplexer als bei operationalen Systemen. Es existieren noch theoretische Probleme, die mit dem regelbasierten Ansatz verbunden sind [JOURDAN et al. 1998]. Daher sind sie zur Zeit auch nur Gegenstand der Forschung und noch nicht in den kommerziellen Sektor vorgedrungen.

Mit dem im Rahmen dieser Arbeit betrachteten ξML-System wird der operationale Ansatz verfolgt. Die zu erstellenden Präsentationen benötigen zunächst keine Mechanismen zur Anpassung und können so problemlos operational erstellt werden. Außerdem müssen die begrenzten personellen Mitteln bei der Konzeption des ξML-Systems berücksichtigt werden.

3 Einführung in ξML

ξML bildet das Rahmenprojekt, in dem die XAM angesiedelt ist. Das Projekt wurde am Lehrstuhl für Software-Technologie der Universität Dortmund von Klaus Alfert, Ute Hensel, Jens Schröder und Sebastian Schütte entwickelt und in einem nichtveröffentlichen Bericht vorgestellt [ALFERT et al. 2001]. Wesentliche Aspekte, die für ein Verständnis des Entwurfs der XAM nötig sind, stelle ich in diesem Kapitel kurz vor.

Zunächst schildere ich die wesentlichen Aspekte des Altenberger Dom-Projekts als Ursprung der Überlegungen zum ξML-System und gebe einen kurzen Überblick über das ξML-System. Danach wird detaillierter auf zwei wesentliche Konzepte von ξML eingegangen und zum Abschluss die Anwendung des ξML-Systems erläutert.

3.1 Der Altenberger Dom

Der Ursprung der Überlegungen zu ξML liegen in einem anderen Projekt, dessen Ziel ebenfalls die Erstellung einer Multimediapräsentationen im Rahmen der universitären Lehre ist, dem Altenberger Dom-Projekt [ALFERT et al. 1999]. In der dort erstellten Präsentation werden am Beispiel des Bauwerks Altenberger Dom Aspekte der Gotik multimedial aufbereitet. Das Projekt wurde in Zusammenarbeit der Lehrstühle für Baugeschichte und Software-Technologie der Universität Dortmund durchgeführt.

In dem Projekt wird durch einen generativen Ansatz eine klare Trennung der inhaltlichen von den technischen Aspekten bei der Erstellung von Multimediapräsentationen vorgenommen, um so zum einen die unterschiedlichen Fähigkeiten und Ausbildung der beiden Teilnehmergruppen geeignet nutzen und zum anderen unterschiedliche Formate zur Darstellung verwenden zu können, ohne dass die Inhalte für die einzelnen Formate jeweils neu beschrieben werden müssen. Mit Hilfe eines XML-Dialekts namens ADML werden die Inhalte, die in der Multimediapräsentation verwendet werden sollen, von den Multimediaautoren, also den Baugeschichtlern spezifiziert. Aus dieser Spezifikation wird mit einem von den Multimediatechnikern, den Informatikern, entwickelten Werkzeug, dem so genannten Converter, eine darstellbare Multimediapräsentation erzeugt, wahlweise als HTML-Präsentation [W3C 1999] oder als Macromedia Director-Präsentation [GROSS et al. 2000]. Aspekte zur Gestaltung der Darstellung wurden im Vorfeld von beiden Projektteilnehmergruppen festgelegt und sind fest in das Werkzeug implementiert.

Dieser Ansatz hat sich im Rahmen des Altenberger Dom-Projektes bewährt. Bei der Übertragung der gefunden Lösungen auf ein anderes, ähnlich geartetes Projekt zeigt sich aber, dass weder die Spezifikationssprache noch das Übersetzungswerkzeug geeignet wiederverwendet werden können. Die feste Implementierung der gestalterischen Aspekte der Präsentation im Genera-

tor und die fehlende Modularisierung sowohl der Sprache und als auch des Generators bewirken einen sehr hohen Aufwand bei der Anpassung an neue Projektbedürfnisse, wie etwa die Unterstützung eines geänderten Layouts oder die Verwendung neuer inhaltlicher Konzepte. Diese Schwächen sollen beim ξML-System behoben werden.

3.2 Übersicht über das ξML-System

ξML bietet Multimediaautoren ein System zur Erstellung von Multimediapräsentationen im Rahmen der universitären Lehre. Es folgt prinzipiell dem im Altenberger Dom-Projekt angewendeten generativen Ansatz zur Erstellung von Multimediapräsentationen: zunächst wird eine Multimediapräsentation unabhängig von einer technischen Realisierung der Darstellung in einer so genannten Projektsprache beschrieben, danach werden aus dieser Spezifikation darstellbare Formen der Präsentation generiert. So können, wie beim Altenberger Dom-Projekt, einmal spezifizierte Inhalte für unterschiedliche Darstellungsformate verwendet und die einzelnen Aufgaben, also die Auswahl der Inhalte und die Realisierung der Darstellung, von jeweiligen Experten getrennt bearbeitet werden. Um Aspekte wie Wiederverwendung und Anpassbarkeit zu realisieren werden bekannte und bewährte Konzepte aus der Software-Technologie auf die Erstellung von Multimediapräsentationen angewendet. Für die Beschreibung und Generierung von Multimediapräsentationen wird ein komponentenorientierter Ansatz verfolgt.

Für die Beschreibung von Multimediapräsentationen in ξML, so genannter *ξML-Präsentationen*, benötigt der Multimediaautor Ausdrucksmöglichkeiten, mit denen er seine Vorstellung der Multimediapräsentationen auf *darstellungsneutrale Weise*, also unabhängig von Aspekten der technischen Realisierung der Darstellung der Präsentation, formulieren kann. Diese Ausdrucksmöglichkeiten sind inhaltliche Konzepte, wie beispielsweise Texte oder Inhaltsverzeichnisse und gestalterische Abstraktionen, mit denen das Erscheinungsbild der Präsentation spezifiziert werden kann. Die Ausdrucksmöglichkeiten werden bei ξML durch Komponenten, so genannte *ξML-Komponenten*, beschrieben. Sie bilden den Baukasten, aus dem ein Autor einer Präsentation auswählen kann, welche Elemente er für die Beschreibung seiner Präsentation benötigt. Die ausgewählten ξML-Komponenten werden geeignet komponiert und konfiguriert, so dass als Ergebnis dieser Vorbereitungsphase eine projektspezifische Beschreibungssprache, die so genannte *Projektsprache* entsteht. Mit dieser auf seine Bedürfnisse zugeschnittenen Projektsprache kann ein Autor die zu erstellende Multimediapräsentation beschreiben. Mit den Aspekten der Definition, Komposition und Konfiguration von ξML-Komponenten beschäftigt sich Sebastian Schütte in seiner Diplomarbeit [SCHÜTTE 2002]. Auf Grundlage einer solchen Beschreibung erzeugt die Laufzeitumgebung des ξML-Systems, die XAM, darstellbare Formen der Präsentation.

Die ξML-Komponenten geben somit den Rahmen vor, in dem sich Beschreibungssprachen erstellen lassen. Das Datenmodell in ξML wird durch die Komposition von ξML-Komponenten für jedes Projekt festgelegt und drückt sich in der jeweiligen Projektsprache aus. Es stützt sich dabei auf die von der XAM bereitgestellten Funktionalität zur Darstellung der ξML-Präsentationen, die unabhängig von einer konkreten technischen Realisierung ist. Mit der XAM wird somit der Bereich der Beschreibung einer ξML-Präsentation in einer Projektsprache von der technischen Realisierung der Darstellung der ξML-Präsentation entkoppelt.

3.3 Spezifikation von ξML-Präsentationen

Dem Prinzip des Separation of Concerns [GHEZZI et al. 1991] und dem strukturierten Paradigma (Abschnitt 2.4.2) folgend, wird in ξML die Spezifikation einer Präsentation in unterschiedliche Aspekte unterteilt. Die Unterteilung folgt dabei im Wesentlichen einer Strukturierung gemäß den in Abschnitt 2.2.1 vorgestellten Dimensionen von Multimediapräsentationen [ROISIN 1998]. Die Spezifikation von ξML-Präsentationen besteht nach [ALFERT et al. 2001] aus drei wesentlichen Strukturen:

Logische Struktur Mit der logischen Struktur wird der Inhalt einer Präsentation festgelegt und inhaltliche Beziehungen definiert.

Präsentationsstruktur Mit der Präsentationstruktur wird die generelle Erscheinung einer ξML-Präsentation angegeben. Neben Aspekten zur Anordnung der Medien auf dem Bildschirm und die Möglichkeiten der Benutzerinteraktion werden außerdem Verknüpfungen modelliert, die entweder aus den in der logischen Struktur definierten inhaltlichen Beziehungen direkt erzeugt oder, wie etwa bei Inhaltsverzeichnissen, berechnet werden.

Visualisierung Der Visualisierungsaspekt von ξML-Präsentationen umfasst die Festlegung von typografischen Informationen der Präsentation. Detaillierte Layoutinformationen wie etwa Farbgestaltung, Schriftarten oder Schriftgrößen werden spezifiziert.

Durch diese explizite Untergliederung ermöglicht es das ξML-System, dass einzelnen Bereiche von jeweiligen Experten spezifiziert werden können. Idealerweise sollte ein Experte der Anwendungsdomäne die inhaltlichen Aspekte in der logischen Struktur festlegen, ein (Medien-) Didaktiker die Medien mit der Präsentationsstruktur geeignet anordnen und ein Designer mit der Visualisierung für ein ansprechendes Layout sorgen.

Diese drei getrennt von einander spezifizierten Aspekte finden sich jeweils in den einzelnen, mit den ξML-Komponenten modellierten, inhaltlichen Konzepten wieder. Daher besteht eine ξML-Komponente aus Bausteinen, die diese einzelnen Aspekte adressieren.

Die Beschreibung der ξML-Präsentation, die jeweils aus den in den Bausteinen der ξML-Komponenten spezifizierten drei Aspekte besteht, wird mit Hilfe eines Werkzeugs zu der darstellungsneutralen Eingabe der XAM verdichtet. Dazu kann eine Auswertung der spezifizierten Informationen nötig sein, wie etwa die Erzeugung von Links aus berechenbaren Verknüpfungen. Dieser Aspekt der Überführung der Beschreibung von ξML-Präsentationen mit Hilfe von ξML-Komponenten in das Eingabeformat der XAM wird in [SCHÜTTE 2002] näher diskutiert.

Innerhalb dieser drei Strukturen werden alle zur Darstellung einer ξML-Präsentation benötigen Informationen spezifiziert, so dass die XAM die Darstellung der ξML-Präsentation realisieren kann. Durch den Zwischenschritt der Überführung der drei Teilspezifikationen einer ξML-Präsentation in die Eingabe der XAM hat man die Möglichkeit, die Spezifikation einer ξML-Präsentation auf einen höheren und dem Autor angemessenen Abstraktionsniveau, als es die XAM bietet, vorzunehmen.

3.4 Wiederverwendung und Anpassbarkeit

Zentrale Anforderungen an das ξML-System sind die Unterstützung von Wiederverwendung und Anpassbarkeit, dessen Fehlen als gravierende Schwäche beim Altenberger Dom-Projekt ausgemacht wurde.

Die Anpassbarkeit an die Bedürfnisse einzelner mit dem ξML-System zu realisierende Projekte wird durch die Verwendung von ξML-Komponenten gewährleistet. Wie bereits geschildert, können mit ihnen die zur Beschreibung der ξML-Präsentationen benötigten Projektsprachen erzeugt werden. Es ist aber zusätzlich noch nötig, dass die beiden im ξML-System verwendeten Werkzeuge, der Generator zum Erzeugen der XAM-Eingabe und die XAM selbst, an die Bedürfnisse des Projekts anpassbar sind. Dazu werden die Werkzeuge komponentenorientiert entworfen, so dass sie mit Hilfe geeigneter Software-Komponenten für die Unterstützung der in den ξML-Komponenten definierten inhaltlichen Konzepten konfiguriert werden können. Wie die Anpassbarkeit für die XAM ermöglicht wird, wird im Rahmen dieser Arbeit erläutert.

Um eine Wiederverwendung zu ermöglichen, ist es prinzipiell nötig, dass ähnlich geartete Probleme gelöst werden sollen. Die allgemeine Erstellung von Multimediapräsentationen ist da, zumindest mit den beschränkten Ressourcen, die zur Konzeption des ξML-Systems zur Verfügung stehen, ein zu breites Gebiet, als dass geeignete Mechanismen gefunden werden können, um sinnvoll Wiederverwendung anbieten zu können. Das Dilemma wird aber gelöst, indem mit dem ξML-System die Erstellung einer bestimmt Art von Multimediapräsentationen, so genannter *textzentrierter Multimediapräsentationen* , im Rahmen der universitären Lehre unterstützt wird. Eine textzentrierte Multimediapräsentation entspricht den Bedürfnissen zur multimedialen Vermittlung von universitären Lehr-Inhalten aus dem Gebiet der Geisteswissenschaften, wie sie etwa im Altenberger Dom-Projekt [ALFERT et al. 2001] identifiziert wurden. Sie wird im Rahmen dieser Arbeit wie folgt charakterisiert:

Definition 8 (Textzentrierte Multimediapräsentationen) *Eine Multimediapräsentation, die ihre (langlebigen) Inhalte vorwiegend unter Verwendung des Medientyps Text präsentiert, wird im Folgenden* ***textzentrierte Multimediapräsentation*** *genannt. Von ihr werden weitere Medientypen wie Bilder oder Videos zur Verdeutlichung und Veranschaulichung der mit den Texten zu vermittelnden Inhalte verwendet. Mittels zu definierender Beziehungen zwischen den Medieninstanzen kann durch eine solche textzentrierte Multimediapräsentation navigiert werden. Den textzentrierten Präsentationen liegt ein einfaches, ereignisorientiertes Zeitmodell zu Grunde, das neben der gleichzeitigen Anzeige von Medien, die durch Benutzerinteraktion geändert werden können, keine weiteren zeitlichen Beziehungen wie Synchronisationsabhängigkeiten vorsieht.*

Die ξML-Präsentationen sind Beispiele solcher textzentrierten Multimediapräsentationen.

Wiederverwendung im Rahmen von ξML betrifft sowohl die inhaltlichen Konzepte, als auch die gefundenen technischen Lösungen zum Erzeugen der ξML-Präsentationen. Durch eine Modularisierung in ξML mit Hilfe von Komponenten, ξML-Komponenten für die inhaltlichen Konzepte und allgemeine Software-Komponenten für die technischen Lösungen, werden geeignete Einheiten zur Wiederverwendung definiert.

3.5 Erstellung von ξML-Präsentationen

Um eine ξML-Präsentation zu erstellen, müssen zwei Phasen durchlaufen werden: Die Projektvorbereitungsphase und die Projektdurchführungsphase.

In der Projektvorbereitungsphase werden von den beteiligten Personen, den Multimediaautoren und den Multimediatechnikern, Anpassungen des ξML-Systems an die Bedürfnisse des Projekts vorgenommen. Die Multimediaautoren, im Rahmen des ξML-Projekts im Folgenden *ξML-Autoren* genannt, bestimmen die benötigten inhaltlichen Konzepte. Die Multimediatechniker wählen danach geeignete, bereits vorhandene ξML-Komponenten aus und passen sie an oder entwickeln neue ξML-Komponenten, falls die benötigten inhaltlichen Konzepte bisher noch nicht verwendet wurden. Durch die Komposition dieser ξML-Komponenten entsteht die Projektsprache, mit denen die ξML-Autoren die zu erstellende Präsentation spezifizieren. Mit einer Anpassung der Werkzeuge durch die Multimediatechniker an die verwendeten inhaltlichen Konzepte endet die Projektvorbereitungsphase.

Die Projektdurchführungsphase, schematisch in Abbildung 3.1 dargestellt, läuft in mehreren Schritten ab. In einem ersten Schritt spezifizieren die ξML-Autoren mit der aus den Bausteinen der ξML-Komponenten bestehenden Projektsprache die drei in Abschnitt 3.3 vorgestellten unterschiedlichen Aspekte der zu erstellenden ξML-Präsentation. In einem zweiten Schritt wird aus den spezifizierten Informationen das Eingabedokument der XAM generiert. Dabei werden die spezifizierten Informationen in die von der XAM angebotenen Ausdrucksmöglichkeiten überführt. Im letzten Schritt erzeugt die XAM aus dem Eingabedokument eine darstellbare Form der ξML-Präsentation. Ob die XAM das Eingabedokument interpretiert und direkt visualisiert, oder ob die XAM Ausgaben erzeugt, die von konkreten Laufzeitumgebung dargestellt werden, wird im Rahmen dieser Arbeit geklärt.

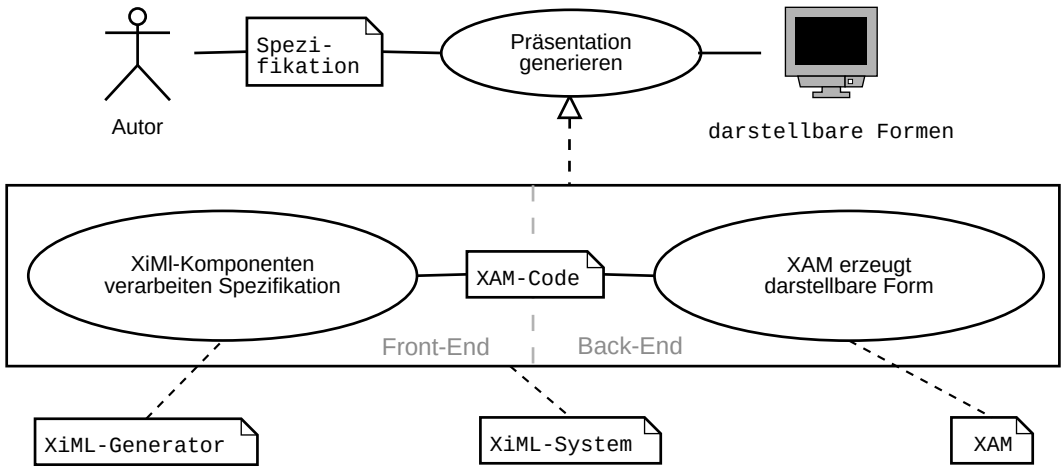


Abbildung 3.1: Überblick über das ξML-System

Das hier vorgestellt ξML-System folgt zwar dem generellen Generatoransatz des Altenberger Dom-Projekts, unterscheidet sich aber an zwei zentralen Punkten wesentlich von ihm. Zum einen

wird konsequent ein komponentenorientierter Ansatz verfolgt, sowohl bei der Erstellung der Spezifikation der Präsentationen, als auch bei der Konzeption der benötigten Werkzeuge. Zum anderen wird der Generatoransatz in ein zweistufiges Modell verfeinert (siehe Abbildung 3.1 auf der vorherigen Seite). In Anlehnung an die Sprechweisen aus dem Compilerbau wird im so genannten Front-End-Bereich des ξML-Systems die Spezifikation einer ξML-Präsentation in das darstellungsneutrale Eingabeformat der XAM überführt, im Back-End-Bereich verarbeitet die XAM diese Eingabe zu darstellbaren Formen der Präsentation. Da mit dieser Arbeit die XAM konzipiert wird, liegt der Fokus hier klar auf dem Back-End-Bereich des ξML-Systems.

4 Thema der Arbeit

Nachdem ich in den vorherigen Kapiteln meine Arbeit motiviert und in den allgemeinen Kontext der Erstellung von Multimediapräsentationen und den speziellen Rahmen des ξML-Projekts gestellt habe, konkretisiere ich in diesem Kapitel die Thematik meiner Arbeit und beende damit den einleitenden Teil meiner Diplomarbeit. Zunächst präzisiere ich die Aufgabenstellung meiner Diplomarbeit und skizzieren den von mir gewählten Lösungsansatz. Das Kapitel schließt mit der Definition der Ziele meiner Arbeit.

4.1 Aufgabenstellung

Wie der Titel meiner Diplomarbeit „XAM – Eine abstrakte Laufzeitumgebung für ξML“ besagt, besteht meine Aufgabe in der Konzeption der e**X**tensible **A**bstract **M**achine (XAM), einer abstrakten Laufzeitumgebung für Multimediapräsentationen im Rahmen des ξML-Projektes. Zunächst diskutiere ich den Begriff der Laufzeitumgebung, ehe ich einzelne Aufgaben, die bei der Konzeption der XAM im Rahmen der Diplomarbeit zu lösen sind, vorstelle.

Der Begriff der Laufzeitumgebung ist weit gefasst. Er wird in verschiedenen Kontexten, wie etwa bei der Ausführung von Programmen auf Betriebssystemen, zur Implementierung von Programmiersprachen oder bei der Bereitstellung allgemeiner Funktionalität für spezielle Anwendungen verwendet. Eine Laufzeitumgebung wird somit auf sehr unterschiedlichen Anwendungsgebieten und Abstraktionsniveaus eingesetzt.

Auf Betriebssystemebene erlaubt es eine Laufzeitumgebung, Programme auszuführen, die für das jeweilige Betriebssystem übersetzt wurden [SILBERSCHATZ und GALVIN 1994]. Dazu wird der Befehlssatz der Betriebssystemschnittstelle auf Steueranweisungen für die Hardware der Maschine, auf der das Betriebssystem läuft, abgebildet und Mechanismen bereitgestellt, die das Ausführen der Programme erlauben. Exemplarisch sei hier die Verwaltung der zur Ausführung von Programmen benötigten Ressourcen, wie etwa das Allokieren und Kontrollieren der Prozessorzeit, des Speichers für Prozesse, des Dateispeicherplatzes oder der Ein- und Ausgabegeräte genannt.

Aufgabe der Laufzeitumgebung einer Programmiersprache hingegen ist die Implementierung der Sprache. Je nach Sprache muss die Laufzeitumgebung dazu eine Reihe von abstrakteren Mechanismen bereitstellen, als sie von Betriebssystem angeboten werden. Objektorientierte Sprachen wie C++ oder Java etwa erlauben die Definition von neuen, nicht-primitiven Datentypen. Instanzen werden zur Laufzeit einer Typüberprüfung unterzogen, um ihre korrekte Benutzung zu gewährleisten. Bei der Sprache Java wird zusätzlich eine höherwertige Speicherverwaltung angeboten, die das automatische Freigeben von nicht mehr benötigtem, allokierten Speicher übernimmt, dem so genannten Garbage Collection. Die Laufzeitumgebung der jeweiligen Sprache

muss diese Funktionen implementieren. ţingegen ist eine maschinennahe Sprache, sie benutzt im Wesentlichen nur Objekte wie Zeichen, Zahlen und Adressen mit entsprechenden arithmetischen und logischen Operationen, die auf realen Maschinen auch vorhanden sind [KERNIGHAN und RITCHIE 1988]. Die Laufzeitumgebung von ţildet im Wesentlichen die Sprachkonstrukte direkt auf den Maschinenbefehlssatz ab, ohne weitere Funktionalität zu definieren.

Die Laufzeitumgebung der Programmiersprache Java, die Java Runtime Environment (JRE) stellt eine Mischform zwischen einer Betriebssystemlaufzeitumgebung und einer Laufzeitumgebung für eine Programmiersprache dar. Zum einen implementiert sie, wie oben beschrieben, die Sprache. Zum anderen definiert sie einen abstrakten Prozessor, auf dem die Java-Programme ausgeführt werden, die so genannte Java Virtual Machine (JVM), auch abstrakte Maschine genannt [LINDHOLM und YELLIN 1997]. Sie benutzt dazu einen universellen, von konkreten Prozessoren unabhängigen Befehlssatz, der sich in dem Bytecodeformat (`class`-Dateien) manifestiert, in den Java-Quellprogramme überführt werden. Um die JVM ausführen zu können, stellt die JRE für konkrete Maschinen Implementierungen bereit, die den universellen Befehlssatz und die abstrakten Mechanismen auf Grundlage der Schnittstelle der jeweiligen Maschine realisieren, auf dem die JVM benutzt werden soll.

Die Mozilla Runtime Environment (MRE, [NANGA et al. 2002]) ist die Laufzeitumgebung des Mozilla-Webbrowsers. Sie ist ein Beispiel einer Laufzeitumgebung für eine konkrete Anwendung. Die MRE stellt eine minimale Menge von Dateien zur Verfügung, die zur Ausführung von Anwendungen benötigt werden, die die eingebettete Renderingmaschine des Mozilla benutzen. Sie stellt mittels Bibliotheken allgemeine Funktionalität bereit, die zur Ausführung von Anwendungen mit der eingebetteten Renderingmaschine benötigt wird, die aber, im Gegensatz zu den vorherigen Laufzeitumgebungen, nicht die Verwaltung von Systemresourcen oder die Überführung eines Eingabebefehlssatzes in einen anderen Befehlssatz betrifft.

Der Begriff Laufzeitumgebung wird also für Aufgaben verwendet, die auf sehr unterschiedlichem Niveaus angesiedelt sind. Meiner Meinung nach ist es die allgemeine Aufgabe einer Laufzeitumgebung, Infrastruktur bereit zu stellen, die es erlaubt, Programme auszuführen.

Im Rahmen der vorliegenden Arbeit soll eine Laufzeitumgebung für Multimediapräsentationen entwickelt werden. Bei den Programmen, die ausgeführt werden sollen, handelt es sich daher um Multimediapräsentationen. Das Ausführen einer Multimediapräsentation mit Hilfe einer Laufzeitumgebung führt dazu, dass die Multimediapräsentation dargestellt wird. Das Darstellen einer Multimediapräsentation bedeutet, dass sie auf einem Ausgabemedium wie dem Bildschirm angezeigt wird, der Betrachter über Eingabegeräte wie Maus oder Tastatur die Möglichkeit erhält, mit der Präsentation zu interagieren und damit durch die Multimediapräsentation navigieren zu können. Bekannte Beispiele für Laufzeitumgebungen von Multimediapräsentationen sind Webbrowser wie der Netscape Navigator oder der Mircosoft Internet Explorer, die eine Multimediapräsentation, die in Form von HTML-Dateien und diversen Medien wie Bilddateien vorliegt, darstellen können.

Bei der XAM handelt es sich um eine Laufzeitumgebung für Multimediapräsentationen, die im Rahmen des in Kapitel 3 vorgestellten ξML-Systems erstellt werden, so genannte ξML-Präsentationen. Dadurch unterscheidet sich die XAM von den oben genannten Laufzeitumgebung für Multimediapräsentationen, den Webbrowsern, erheblich. Es müssen eine Reihe von Aspekten berücksichtigt werden, die bei den Webbrowsern nicht vorhanden sind, und eine Reihe von Aspekten, die bei den Webbrowsern betrachtet werden, sind für die XAM irrelevant. Auf die be-

sonderen Aspekte, die sich für die XAM auf Grund des ξML-Kontextes ergeben, werde ich im Folgenden eingehen.

Im Rahmen des ξML-Systems soll die Beschreibung einer Multimediapräsentation getrennt werden von der technischen Realisierung der Darstellung. Wie in Abschnitt 3.2 beschrieben, wird es so dem Autor mit dem ξML-System ermöglicht, sich bei der Erstellung einer Multimediapräsentation auf die Zusammenstellung der inhaltlichen Aspekte der Präsentation konzentrieren und von den technischen Details der Darstellung abstrahieren zu können. So sollen beispielsweise die Festlegung der logische Struktur, die Auswahl der verwendeten Medien und die Beschreibung des Layouts einer Präsentation unabhängig von einer konkreten technischen Realisierung der Darstellung sein. Aufgabe der XAM ist es, eine solche, von technischen Details weitesgehend befreite Beschreibung, die in dieser Arbeit zu definieren ist, auszuführen, d.h. darstellen zu können. Die Eingabe der XAM wird, ähnlich wie bei der JVM, aus einem universellen Satz von Instruktionen, die an keine konkrete Plattform gebunden sind, bestehen. So wie die JVM daher auch abstrakte Maschine genannt wird, spreche ich bei der XAM von einer abstrakten Laufzeitumgebung. Mit Hilfe der Abstraktion von konkreten technischen Realisierungen durch die XAM wird es dem Multimediaautor ermöglicht, sich auf das für ihn Wesentliche, nämlich die Erstellung der inhaltlichen Aspekte der ξML-Präsentation, zu konzentrieren. Allgemein gesprochen, erlaubt diese Abstraktion es also, die wichtigsten Aspekte eines Problems von den Details zu trennen und die Probleme der Reihe nach zu lösen, um so mit der Komplexität des Problems umzugehen. Diese Interpretation des Abstraktionsbegriffs ist an die in [D'SOUZA und WILLS 1999], Kapitel 1.14.1, im Rahmen von komponentenorientierter Softwareentwicklung mit UML vorgestellte Bedeutung von Abstraktion angelehnt.

Die prinzipielle Aufgabe der XAM ist es, eine ξML-Präsentation ausführen zu können. Multimediaformate wie HTML [W3C 1999] oder SMIL [W3C 2001] nutzen speziell für das Format entwickelte Abspielprogramme, so genannte Player. Multimediapräsentationen, die in Form von HTML-Dokumenten beschrieben sind, werden beispielsweise mit Hilfe von oben genannten Webbrowsern dargestellt. Analog wäre es eine Möglichkeit für die Konzeption der XAM, sie auch als einen solchen Player zu entwickeln. Es wäre denkbar, einen solchen interpretativen Ansatz, der die ξML-Präsentationen direkt innerhalb der XAM ausführt, etwa mit Hilfe des Java Media Frameworks (JMF, [Sun 1999]) umzusetzen. Im Rahmen des ξML-Systems soll die XAM nicht als neuer Player konzipiert werden, sondern auf etablierten Formaten und Technologien aufsetzen. Analog zum Altenberger Dom-Projekt, bei dem zur Darstellung auf die Kombinationen von HTML und Webbrowsern bzw. von LINGO [EPSTEIN 1998] und Macromedia Director-Projektoren [GROSS et al. 2000] zurückgegriffen wurde, ist es also Aufgabe der XAM, sich bei der Erzeugung der Darstellung von ξML-Präsentationen auf bereits existierenden Technologien abzustützen.

Mit dem ξML-System sollen Autoren bei der Erstellung von Multimediapräsentationen unterstützt werden. Für einzelne Projekte werden jeweils an die Projektbedürfnisse angepasste Sprachen formuliert, mit der die ξML-Präsentationen beschrieben werden. Hierbei ist es explizit vorgesehen, das auch multimediale Elemente, die bis zu dem Zeitpunkt noch nicht von dem ξML-Systems unterstützt wurden, neu definiert und somit zur Beschreibung der Multimediapräsentation benutzt werden können. Das hat zur Folge, das die Menge der von ξML unterstützten multimedialen Elemente bei der Entwicklung des ξML-Systems nicht abschließend festgelegt wird, sondern bei Benutzung des ξML-Systems erweitert werden kann. Für die XAM bedeutet

dies, dass die Menge der Instruktionen der Eingabe bei der Entwicklung der XAM nicht feststeht. Die XAM muss also so konzipiert werden, dass sie mit einem erweiterbaren Eingabeformat umgehen kann. Eine Aufgabe bei der Konzeption der XAM ist es daher, ein solches erweiterbares Eingabeformat zu definieren und bei der XAM eine offene Schnittstelle für die Verarbeitung dieses Formats bereit zu stellen.

Bei der Darstellung einer ξML-Präsentation soll sich die XAM, wie oben erläutert, auf bekannte und verbreitete Technologien abstützen und auf deren Plattformen distributiert werden. Der Anwendungsbereich Multimedia ist weit gefasst und entwickelt sich schnell weiter. Es existieren bereits viele Formate und Technologien zur Darstellung von Multimediapräsentationen und neue kommen hinzu. Eine Anforderung an das ξML-System ist es, sich bei der Darstellung einer ξML-Präsentation nicht nur auf ein Format festzulegen, sondern sich prinzipiell auf mehrere Formaten abzustützen. Die Menge der unterstützten Formate bzw. Technologien soll nicht bei der Entwicklung der XAM abschließend festgelegt werden, sondern sich auch später noch, ohne zu große Eingriffe in das ξML-System vornehmen zu müssen, erweitern lassen. Die XAM muss also so konzipiert werden, dass sie zum einen die Möglichkeit bereitstellt, von einer ξML-Präsentation mehrere darstellbare Formen zu erzeugen, die sich auf den unterschiedlichen Technologien abstützen und auf deren Plattformen verbreitet werden können. Zum anderen muss sie es unterstützen, dass die Menge der Technologien, auf denen sie die Darstellung abstützt und verbreitet, erweiterbar ist. Neben der Bereitstellung und Unterstützung eines erweiterbaren Eingabformats ist es also auch Aufgabe der XAM, eine erweiterbare Menge von Formaten der Darstellung erzeugen zu können, mit denen die ξML-Präsentationen verbreitet werden kann.

Bei der Konzeption der XAM müssen mehrere wesentliche Aspekte berücksichtigt werden, die ich im Folgenden kurz zusammenfassend darstelle:

- Die XAM muss eine erweiterbare Schnittstelle für die Eingabe bereitstellen. Dazu gehört Definition des Eingabeformats der XAM, mit dem ξML-Präsentationen unabhängig von einer technischen Realisierung der Darstellung beschrieben werden.
- Die XAM muss so konzipiert werden, dass sie aus der Eingabe mehrere darstellbare Formen der ξML-Präsentation erzeugen kann. Die Darstellung soll sich auf verbreitete Technologien abstützen, deren Menge jederzeit erweitert werden können soll.
- Zusätzlich muss XAM die nötige Infrastruktur bereitstellten, die es erlaubt, die abstrakte Beschreibung einer Multimediapräsentation aus der Eingabe in unterschiedliche darstellbare Formen zu überführen.

Aufgabe ist es, eine abstrakte Laufzeitumgebung für ξML-Präsentationen zu entwerfen, die ein zu definierendes erweiterbares Eingabeformat verarbeiten kann, um daraus unterschiedliche darstellbare Formen zu erzeugen, die in der Menge der dabei benutzten Formate erweiterbar sind. Zur Erzeugung der darstellbaren Formen der ξML-Präsentationen muss die XAM ein erweiterbares Eingabeformat in die unterschiedlichen Formate der Ausgabe überführen.

An dieser Stelle sei schon einmal ausdrücklich darauf hingewiesen, dass es sich bei dieser Arbeit um eine konzeptionelle Arbeit handelt. Um den zeitlichen Rahmen einer Diplomarbeit nicht zu sprengen, soll mit dieser Arbeit der Entwurf der XAM vorgenommen werden, aber nicht die Umsetzung des Entwurfs. Eine Implementierung der XAM ist nicht Teil der Diplomarbeit. Eine genauere Definition der Ziele dieser Arbeit findet sich weiter unten in Abschnitt 4.3.

4.2 Lösungsansätze

Um die oben beschriebenen Aufgaben lösen zu können, benutze ich eine Reihe von Ansätzen, die ich im Folgenden grob skizziere. Eine detaillierte Beschreibung der Konzepte der XAM wird in den entsprechenden Entwurfskapiteln weiter unten vorgenommen.

4.2.1 Generatoransatz

Wie in Abschnitt 4.1 beschrieben, ist es eine zentrale Rahmenbedingung beim Entwurf der XAM, dass kein neuer Player entwickelt werden soll. Die Darstellung der mit ξML beschriebenen Multimediapräsentationen soll auf Basis von etablierten Formaten und Technologien erfolgen. Um dies umzusetzen, bietet sich ein Generatoransatz an: Die von technischen Details weitestgehend freie Beschreibung einer Multimediapräsentation wird von der XAM dargestellt, in dem sie die Beschreibung in andere Multimediaformate überführt, die dann von bereits für diese Formate existierenden Playern angezeigt werden.

Mit der Konzeption der XAM als Generator erfülle ich zum einen die oben genannte Rahmenbedingung, zum anderen muss die XAM dadurch eine Reihe von Aufgaben nicht lösen, die in einem Player realisiert werden müssten (siehe Kap. 4 in [VAN ROSSUM et al. 1993]). Exemplarisch seien das Instantiieren der Elemente der Multimediapräsentation und das Verwalten dieser Instanzen, die technische Realisierungen der konkreten Anzeige der Multimediapräsentation auf dem Bildschirm oder der Interaktionsmöglichkeiten des Benutzers mit der Multimediapräsentation über Eingabegeräte wie Maus und Tastatur genannt. Diese Arbeit wurde bei der Implementierung der jeweiligen Player bereits erfolgreich geleistet und muss nicht noch einmal wiederholt werden.

Durch den Generatoransatz der XAM bindet man allerdings die multimediale Ausdrucksstärke der ξML-Präsentationen an die Ausdrucksstärke der Technologien, mit denen die Darstellung realisiert wird. Dieser Umstand ist nicht zu vermeiden, da zum einen auf Grund der Rahmenbedingung vorgegeben ist, sich auf etablierte Technologien abzustützen. Zum anderen bieten Formate und Technologien wie beispielsweise HTML in Verbindung mit einem Webbrowser und LINGO mit Macromedia Director-Projektoren für textzentrierte Multimediapräsentationen, für die wie in Abschnitt 3.4 beschrieben, das ξML-System gedacht ist, einen großen Satz an Ausdrucksmöglichkeiten an. Außerdem soll die XAM so konzipiert werden, dass auch nach der Entwicklungszeit der XAM neue Formate zur Darstellung der ξML-Präsentationen eingebunden werden können, und somit dadurch die Ausdrucksstärke der ξML-Präsentationen erweitert werden kann.

Die Eingabeschnittstelle der XAM wird, um sich in das in Abschnitt 3.5 auf Seite 22 beschriebene ξML-System einzufügen, dokumentenbasiert sein und nicht in Form einer Programmierschnittstelle (API) realisiert werden. Die XAM ist also ein Generator, der Elemente zur Beschreibung der ξML-Präsentationen aus den Eingabedokumenten in die entsprechenden Ausdrucksmöglichkeiten der Formate der Ausgabedokumente übersetzt. Sie fügt sich somit in den generellen Generatoransatz des ξML-Systems ein.

4.2.2 Erweiterbarer Generator

Ein Compiler [AHO et al. 1988a], beispielsweise ein Generator für Programmiersprachen, liest ein wohldefiniertes Eingabeformat ein und erzeugt daraus eine wohldefinierte Ausgabe. Die XAM hingegen muss erweiterbar konzipiert werden. Die Erweiterbarkeit betrifft sowohl die Ein- als auch die Ausgabe. Die XAM muss ein Eingabeformat verarbeiteten können, das auch nach Entwicklungszeit der XAM um neuen Elemente zur Beschreibung bisher nicht vorhandener multimedialer Funktionalität erweitert werden kann. Die XAM soll, wie in Abschnitt 4.1 beschrieben, mehrere Formate zur Darstellung der ξML-Präsentationen unterstützen und es auch nach ihrer Entwicklungszeit ermöglichen, neue Formate zur Darstellung der Präsentationen einzubinden. Die XAM muss also in der Menge der Ausgabeformate erweiterbar sein. Daher wird die XAM als ein in zwei Dimensionen, nämlich der Ein- und Ausgabe, erweiterbarer Generator entwickelt werden.

Um dies zu erreichen, bediene ich mich eines im Compilerbau üblichen Konzepts, nämlich der Entkopplung der Eingabe von der Ausgabe durch eine Aufteilung in ein Front-End und ein Back-End. Das Front-End ist nur von der Quellsprache, aber nicht von der Zielplattform des Übersetzungsvorgangs abhängig, das Back-End entsprechend ist nur von der Zielmaschine, aber nicht mehr von der Quellsprache abhängig (siehe Seite 24 in [AHO et al. 1988a]). Beim Compilerbau ist es zur Unterstützung neuer Zielmaschinen üblich, nur das Back-End des Compilers aus zu tauschen und alle anderen Elemente des Compilers beizubehalten. Mit diesem Ansatz lässt sich die XAM zum einen in der Ausgabe erweiterbaren Generator machen. Anstatt das Back-End auszutauschen, wird die XAM über mehrere von ihnen gleichzeitig verfügen, die als Werkzeuge zur Erzeugung der verschiedenen Ausgabeformate benutzt werden. Dadurch können mit der XAM unterschiedliche Formate zur Darstellung einer ξML-Präsentation genutzt werden. Um neue Darstellungsformate für Präsentationen zu unterstützen, wird die XAM wohldefinierte Schnittstellen bereitstellen, mit denen neue Werkzeuge zur Unterstützung der Erzeugung der Ausgabeformate eingebunden werden können.

Was die XAM, neben der Tatsache, dass sie mehrere Ausgabeformate gleichzeitig unterstützt, weiterhin von Compilern unterscheidet, ist ihr erweiterbares Eingabeformat. Um es zu ermöglichen, das Eingabeformat auch nach Entwicklungszeit der XAM um neue Elemente zu erweitern, wird die XAM mit einem Initialisierungsmechanismus konzipiert. Bei jedem Start werden der XAM über eine wohldefinierte Initialisierungsschnittstelle die im Eingabedokument benutzten Elemente zur Beschreibung der ξML-Präsentationen bekanntgemacht. Die Initialisierung betrifft zum einen das Front-End der XAM, dem das Format der Eingabedokumente bekanntgemacht werden muss, um es Einlesen zu können. Aber auch das Back-End mit den Werkzeugen zum Erzeugen der Ausgabeformate muss konfiguriert werden. Die Abbildung der Elemente zur Beschreibung der ξML-Präsentationen aus der Eingabe in die entsprechenden Ausdrucksmöglichkeiten der Ausgabeformate muss definiert werden. Die Realisierung des Einlesens der Eingabe und des Abbildens der Eingabe in die entsprechenden Ausgabeformate wird, entsprechend dem komponentenorientierten Ansatz des ξML-Systems, modular erfolgen, um Wiederverwendung und Anpassbarkeit zu ermöglichen (siehe Abschnitt 3.4). Die Initialisierung besteht aus einem so genannten PlugIn-Mechanismus, der für das Einlesen der Beschreibungselemente der ξML-Präsentationen und das Erzeugen der Ausgabeformate entsprechende Komponenten bei der XAM registriert.

4.2.3 Abstraktes Modell

Um das Ziel zu erreichen, die Eingabe weitestgehend von der Ausgabe zu trennen, ist es im Compilerbau-Kontext üblich, zwischen dem Front-End und dem Back-End eines Compilers eine Struktur zu definieren, die so genannte Zwischendarstellung (siehe Kapitel 8 in [AHO et al. 1988b]). Die XAM benötigt auch eine solche Zwischendarstellung. Sie unterscheidet sich hier aber wiederum von einem gewöhnlichen Compiler: Sie benötigt eine Zwischendarstellung nicht nur, um die Eingabe von der Erzeugung der Ausgabe zu entkoppeln, sondern vielmehr definiert sie dadurch eine Struktur, die den Rahmen für die Erweiterbarkeit der XAM festlegt. Mit der Definition einer solchen Zwischendarstellung bekommt die XAM einen Kern, an dem sich die erweiterbare Eingabe und die unterschiedlichen Formate der Ausgabe verankern lassen. Ohne einen solchen Rahmen erscheint es mir schwer möglich, einen Generator zu entwickeln, der nicht in seinen Strukturen bei jeder Erweiterung neu konzipiert werden müsste.

Wie in [AHO et al. 1988a], Seite 17, beschrieben, kann die Zwischendarstellung als Programm für eine abstrakte Maschine aufgefasst werden. Bei der XAM werde ich ein abstraktes Modell, das der Zwischendarstellung zu Grunde liegt und vorgibt, welche Art von Multimediapräsentationen mit Hilfe der XAM dargestellt werden können, entwerfen. Hierbei mache ich mir zu Nutze, dass mit dem ξML-System textzentrierte Multimediapräsentationen erstellt werden sollen. Das Modell besteht aus *Basisprimitiven*, die benötigt werden, um für textzentrierte Multimediapräsentationen die in Abschnitt 2.1 vorgestellten Dimensionen, die allgemein bei Multimediapräsentationen identifiziert werden können, zu adressieren. Diese Basisprimitive sind auf einem sehr hohen Abstraktionsniveau angesiedelt und stellen Ausdrucksmöglichkeiten bereit, um etwa allgemein Informationseinheiten auf einem Bildschirm anzuordnen oder Relationen zwischen diesen Informationseinheiten anzugeben. Sie werden wie folgt definiert:

Definition 9 (Basisprimitive) *Das abstrakte Modell besteht aus so genannten* ***Basisprimitiven****, oder kurz* ***Primitiven****, die prinzipielle Ausdrucksmöglichkeiten für die Modellierung von Elementen multimedialer Präsentationen bereitstellen. Das Abstraktionsniveau der Basisprimitive ist so hoch, dass sie zur konkreten Modellierung einer Multimediapräsentation erst spezialisiert werden müssen.*

Für die Basisprimitive, die im Rahmen dieser Arbeit definiert werden, verwende ich eine gesonderte Schriftart: *EinBasisprimitiv*.

Um eine ξML-Präsentation beschreiben zu können, müssen von diesen Basisprimitiven Spezialisierungen wie etwa ein Text oder ein Bild als eine konkrete Ausprägungen einer Informationseinheit oder ein Link als Spezialisierung einer Beziehung zwischen diesen konkreten Ausprägungen abgeleitet werden. Diese Spezialisierungen manifestieren sich in den Elementen der Eingabesprache und der Zwischendarstellung. Sie sind somit weiterhin abstrakt in dem Sinne, dass sie an keine technische Realisierung ihrer Darstellung gebunden sind. Daher nenne ich diese Spezialisierungen der Basisprimitive *abstrakte Elemente der ξML-Präsentationen* oder kurz *abstrakte Elemente*. Sie können als Abstraktionen für Leistungsmerkmale konkreter Zielplattformen aufgefasst und wie folgt definiert werden:

Definition 10 (Abstrakte Elemente) *Spezialisierungen der* Basisprimitive *des abstrakten Modells werden* ***abstrakte Elemente*** *genannt und bieten Ausdrucksmöglichkeiten an, mit denen*

konkrete Elemente von Multimediapräsentationen modelliert werden können. Sie sind unabhängig von einer technischen Realisierung ihrer Darstellung.

Für die konkrete Darstellung dieser abstrakten Elemente müssen Implementierungen bereitgestellt werden, die innerhalb des Back-Ends der XAM mit Hilfe der Codeerzeugungswerkzeuge jeweils die Ausgabe in den gewünschten Formaten ausführen.

Durch die Definition neuer abstrakter Elemente als Spezialisierungen der Basisprimitive ist eine Erweiterung der Eingabesprache um neue, von ihr zu verwendende Sprachelemente und somit eine Erweiterung der mit der Eingabesprache beschreibbaren multimediale Funktionalität möglich. Damit die abstrakten Elemente auch dargestellt werden können, müssen für die gewünschten Zielformate jeweils Implementierungen von ihnen entwickelt werden, mit denen die Ausgabe erweitert werden kann. Mit den in einem jeweiligen Eingabedokument benutzten abstrakten Elementen und deren Implementierungen wird die XAM beim Start initialisiert, ehe mit ihr die darstellbaren Formen aus der im Eingabedokument beschriebenen ξML-Präsentation erzeugt werden können. Die Erweiterungen der XAM, die durch die Entwicklung von abstrakten Elementen und ihren Implementierungen möglich sind, hängen also, direkt oder indirekt, von dem abstrakten Modell ab, das ich beim Entwurf der XAM festlege. Mit dem abstrakten Modell und seinen Basisprimitiven gebe ich somit den Rahmen vor, in denen sich die Erweiterungen der XAM bewegen können.

4.2.4 Zusammenfassung

Die vorgestellten Lösungsansätze sind in Abbildung 4.1 auf der nächsten Seite schematisch dargestellt. Wie beschrieben, wird die XAM als ein erweiterbarer Generator entwickelt. Die XAM untergliedert sich in ein Front-End, der auf dem abstrakten Modell beruhenden Zwischendarstellung und ein Back-End. Zusätzlich existiert ein Initialisierungsmechanismus, der die übrigen drei Teile der XAM konfiguriert. Die einzelne Elemente der drei Teile sind in UML-Notation als Pakete und darin enthaltene Komponenten dargestellt.

Der sich beim Ausführen der XAM ergebene Datenfluss wird durch die grauen Pfeile angedeutet. Man unterscheidet beim Ausführen der XAM zwei Phasen: zunächst die Initialisierungsphase, bei der die einzelnen Bestandteile der XAM für die darzustellende ξML-Präsentation konfiguriert werden und die Erzeugungsphase, während der aus der Beschreibung der ξML-Präsentation darstellbare Formen generiert werden. Die Mechanismen dieser beiden Phasen sind orthogonal zu einander, sie betreffen verschiedene Aufgaben bei der Darstellung von ξML-Präsentationen, was durch die unterschiedliche Ausrichtung der grauen Pfeile angedeutet ist.

Der Kontrollfluss der Initialisierungsphase ist durch die drei vertikalen grauen Pfeile dargestellt. Die zur Beschreibung der darzustellenden ξML-Präsentation benötigten abstrakten Elemente werden beim Einlesewerkzeug im Front-End und bei der Zwischendarstellung, ihre Implementierungen bei den jeweiligen Werkzeugen zur Erzeugung der Ausgabe im Back-End registriert. Auf diese Weise wird die XAM für die Darstellung einer ξML-Präsentation konfiguriert.

Der Kontrollfluss der Erzeugungsphase ist durch den horizontalen grauen Pfeil angedeutet. Eine als Eingabedokument vorliegende Beschreibung der darzustellenden ξML-Präsentation wird im Front-End durch einen Parser eingelesen und in die Zwischendarstellung überführt. Die Zwischendarstellung wird dann im Back-End der XAM durch die einzelnen Codeerzeugungswerk-

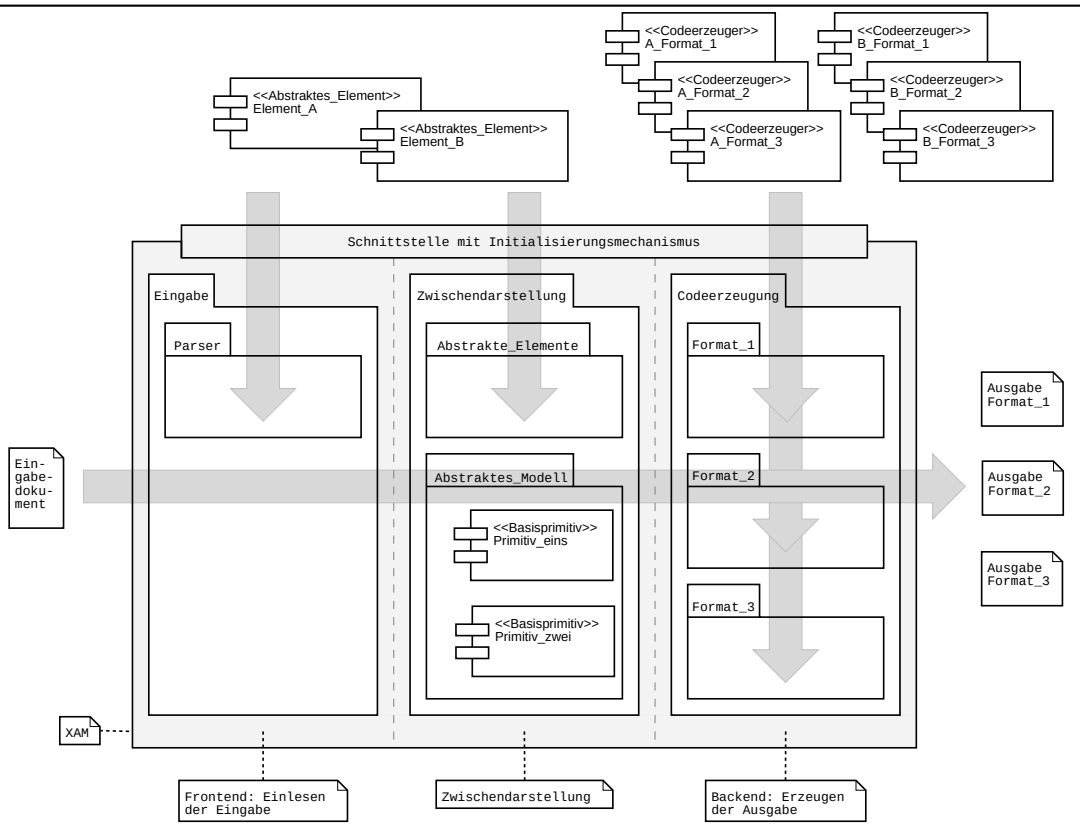


Abbildung 4.1: Der Aufbau der XAM

zeuge in die jeweiligen Darstellungsformate der Ausgabe überführt.

4.3 Ziele der Arbeit

Aus der in Abschnitt 4.1 vorgestellten Aufgabenstellung und den von mir in Abschnitt 4.2 vorgestellten Ansätzen zur Lösung der Aufgabe ergeben sich eine Reihe von Zielen, die ich mit dieser Arbeit erreichen will. Generelles Ziel dieser Arbeit ist die Erstellung eines Entwurfs einer erweiterbaren, abstrakten Laufzeitumgebung für ξML-Präsentationen. Eine Implementierung des Entwurfs im Rahmen dieser Arbeit ist aus Zeitgründen leider nicht realisierbar. Sie würde den Umfang einer Diplomarbeit sprengen. Der Entwurf der XAM wird aber so vorgestellt werden, dass sich dem software-technisch geschulten Blick des Lesers eine Implementierung der XAM erschließen sollte. Dazu bediene ich mich eines durchgängigen Beispiels, an dem ich den Entwurf der XAM verdeutliche. Mit Hilfe dieser Beispielpräsentation namens giMLi, die ich in Abschnitt 5.1 vorstelle, wird exemplarisch gezeigt, wie mit der XAM eine Multimediapräsentation dargestellt werden kann. Welche Ziele ich mir beim Entwurf der XAM im Einzelnen stecke, verdeutliche ich im Folgenden, in dem ich grob die einzelnen Schritte umreiße, die nötig sind,

um, wie in Abbildung 4.1 auf der vorherigen Seite dargestellt, mit der XAM die giMLi-Präsentation aus einer abstrakten Beschreibung in eine darstellbare Form überführen zu können.

Ein wesentliches Ziel dieser Arbeit ist die Konzeption des in Abschnitt 4.2.3 vorgestellten abstrakten Modells. Die Definition eines solchen abstrakten Modells ist der zentrale Ansatz meiner Arbeit, ohne den es meines Erachtens schwer möglich ist, die XAM als einen in zwei Dimensionen erweiterbaren Generator zu entwickeln. Zur Konzeption des abstrakten Modells gehört zum einen das Herausarbeiten und Definieren der Basisprimitive, die das Kernmodell einer jeden mit der XAM dargestellten ξML-Präsentation bilden. Um eine konkrete ξML-Präsentation mit der XAM darzustellen, sind Spezialisierungen der Basisprimitive, die abstrakten Elemente, nötig, die über einen Initialisierungsmechanismus der XAM zur Verfügung gestellt werden. Im Rahmen der Arbeit sollen eine Reihe von abstrakten Elementen definiert werden, die es erlauben, eine konkrete Präsentation, nämlich die Beispielpräsentation giMLi, darzustellen. Darüberhinaus wird der Initialisierungsmechanismus zum Registrieren der abstrakten Elemente bei der XAM entworfen.

Um mit der XAM eine ξML-Präsentation darzustellen, ist es zunächst nötig, die Präsentation mit den Ausdrucksmöglichkeiten der XAM beschreiben und die Beschreibung einlesen zu können. Ich werde die prinzipiellen Mechanismen, die zum Einlesen einer solchen Beschreibung nötig sind, vorstellen und am Beispiel der giMLi-Präsentation veranschaulichen. Dazu wird eine Eingabesprache, die e**X**tensible **a**bstract **M**achine **L**anguage (XaML), definiert, mit der die giMLi-Präsentation beschrieben werden kann. Sie bietet dazu die durch die abstrakten Elemente vorgegebenen Ausdrucksmöglichkeiten an. Diese Eingabe muss eingelesen und in die Zwischendarstellung überführt werden. Hierzu ist ein konfigurierbarer Parser nötig, der das jeweilige Format der Beschreibung der Multimediapräsentation als Eingabe verarbeiten und die Zwischendarstellung erzeugen kann. In dieser Arbeit zeige ich, wie die giMLi-Präsentation in Form eines XaML-Eingabedokuments eingelesen und daraus mit den zu definierenden abstrakten Elementen die Zwischendarstellung erzeugt wird.

Als letzter Schritt zur Darstellung der giMLi-Präsentation ist es nötig, die Zwischendarstellung in Formate zu überführen, die anzeigbar sind. Die zum Überführen der Zwischendarstellung in solche anzeigbaren Formate benötigten Mechanismen werde ich vorstellen. Aus Zeitgründen werde ich an Hand der giMLi-Präsentation die Überführung in nur ein Zielformat verdeutlichen. Zur Auswahl standen hier die beiden schon im Altenberger Dom-Projekt verwendeten Formate HTML und LINGO. Da für die Erzeugung einer Ausgabe in LINGO ein nichtdokumentiertes Framework benutzt wird, habe ich mich für HTML als standardisierte Format der Zielplattform zur Darstellung der giMLi-Präsentation entschieden. Die Konzeption der Erzeugung der Ausgabe erfolgt so, dass ersichtlich wird, wie weitere Formate unterstützt werden können, ohne dass die XAM vollständig neu entworfen werden muss. An den entsprechenden Stelle werde ich darauf hinweisen und entsprechende Lösungsmöglichkeiten skizzieren.

Als Ergebnis dieser Ziele soll ein Entwurf stehen, der die Funktionsweise der XAM aufzeigt und als Grundlage zur Implementierung der XAM dienen kann. Dem Leser sollte es nach Studium der Arbeit klar sein, wie neue multimediale Elemente zur Beschreibung von ξML-Präsentationen definiert werden und was zu tun ist, um ihre Unterstützung durch die XAM zu realisieren. Der Entwurf wird mit Hilfe der giMLi-Präsentation veranschaulicht und durch die damit verbunden fiktive Ausführung der Darstellung der giMLi-Präsentation mit der XAM gleichzeitig validiert. Eine auf dieser Grundlage beruhende Bewertung rundet die Konzeption der XAM ab.

5 Einführung in den Entwurf

Nach dem einleitenden Teil meiner Diplomarbeit, beginnt nun der Teil, in dem ich den Entwurf der XAM vorstelle. In diesem Kapitel präsentiere ich grundlegende Gedanken zu meinem Entwurf. Zunächst beschreibe ich die Beispielpräsentation giMLi. Aus der Beispielpräsentation und aus der Beschreibung meiner Aufgabenstellung und Zielen aus Kapitel 4, leite ich dann detaillierte Anforderungen an die XAM und ihre in Abschnitt 4.2.4 vorgestellten Bestandteile ab. Das Kapitel schließt mit der einer Charakterisierung des für diese Arbeit verwendeten Komponentenbegriffs ab.

5.1 Eine Beispielpräsentation: giMLi

Wie in Abschnitt 3.4 vorgestellt, soll mit dem ξML-System eine bestimmte Art von Multimediapräsentationen erstellt werden, so genannte textzentrierte Multimediapräsentationen. Ein Beispiel einer solchen Präsentation ist giMLi [1], eine vereinfachte Variante einer Präsentation aus dem Altenberger Dom-Projekt. Sie beinhaltet wesentliche Elemente, die eine textzentrierte Multimediapräsentation ausmachen, wie beispielsweise das Anzeigen von Medien wie Texte und Bilder, die Definition von Verknüpfen der Medien und die Möglichkeit zum Verfolgen dieser Verknüpfungen. Bei der Vorstellung der giMLi-Präsentation wird über die Phänomene der Anzeige einer konkreten Präsentation gesprochen. Daher ist eine Unterscheidung eines Mediums in die in Abschnitt 2.1.1 eingeführten differenzierten Medienbegriffe nicht nötig und unterbleibt aus Gründen der Lesbarkeit an den entsprechenden Stellen. Die Elemente der giMLi-Präsentation stelle ich im Einzelnen in Abschnitt 5.1.1 auf der nächsten Seite vor.

Mit dem Einführen der giMLi-Präsentation verfolge ich zwei Ziele: Zum einen benutze ich sie in dieser Arbeit als möglichst durchgängiges Beispiel, um meinen Entwurf und die mit ihm verwendeten Konzepte exemplarisch zu veranschaulichen. Zum anderen dient sie mir als Referenzpräsentation, da es ein in Abschnitt 4.3 definiertes Ziel der Diplomarbeit ist, zu zeigen, wie mit den Ausdrucksmöglichkeiten der XAM die giMLi-Präsentation beschrieben und aus der Beschreibung eine Darstellung im HTML-Format erzeugt werden kann. Daher leite ich aus der giMLi-Präsentation einerseits Anforderungen an die multimediale Ausdruckstärke meiner zu entwickelnden abstrakten Elemente ab, die ich in Abschnitt 4.2.3 eingeführt habe. Andererseits validiere ich mit ihr den Entwurf der XAM, in dem ich zeige, wie die Beschreibung der giMLi-Präsentation in das Ausgabeformat HTML überführt werden kann.

[1] Gimli ist der tapfere Zwerg in dem Buch “Herr der Ringe” [TOLKIEN 1983].

5.1.1 Elemente von giMLi

Wie in Abschnitt 2.1 auf Seite 5 beschrieben, besitzt eine Multimediapräsentation einen Dokument- und einen Programmaspekt. Dieser Dualismus [ALFERT 1999] wohnt daher auch einer giMLi-Präsentation inne. Ich beschreibe zunächst die statischen Aspekte der Präsentation, die den Dokumentaspekt adressieren, ehe ich die dynamischen, den Programmcharakter der giMLi-Präsentation betreffenden Elemente vorstelle.

Die dem Dokumentaspekt der Präsentation innewohnende logische Struktur von giMLi besitzt, da die giMLi ein bewusst einfach gehaltenes Beispiel einer Präsentation sein soll, nur eine Hierarchiestufe, die so genannten Themen. Ein Thema gruppiert Medien nach den von ihnen vermittelten inhaltlichen Informationen. Es besteht aus einem Text und mehreren Bildern, die mit Bildunterschriften versehen sind und zur Veranschaulichung der im Text gegeben Informationen dienen.

In einem Text können Verknüpfungen, die auf andere Medien verweisen, definiert werden. Es existieren zwei Arten von Verknüpfungen: interne Verknüpfungen, die auf ein Bild innerhalb eines Mediums verweisen, und externe Verknüpfungen, die zu einem anderen Thema führen. Zusätzlich gibt es ein Inhaltsverzeichnis der giMLi-Präsentation, das aus einer Liste von externen Verknüpfungen zu allen anderen Themen besteht.

Die Darstellung der giMLi-Präsentation erfolgt seitenorientiert. Von der Präsentation ist immer nur ein Thema im Darstellungsbereich der Präsentation, der so genannten Bildschirmseite, sichtbar. Die Darstellung eines Themas der giMLi-Präsentation auf einer solchen Bildschirmseite ist in Abbildung 5.1 auf der nächsten Seite zu sehen. Ein Thema wird als aufgeschlagenes Buch präsentiert, d.h. die einzelnen Elemente eines Themas sind auf zwei Buchseiten innerhalb der Bildschirmseite angeordnet. Im oberen Bereich der rechten Buchseite befindet sich das Inhaltsverzeichnis, das mit jedem Thema angezeigt wird, darunter ist der zu dem Thema gehörende Text dargestellt. Auf der linken Buchseite wird ein Bild mit seiner zugehörigen Bildunterschrift präsentiert. Wird ein Thema auf der Bildschirmseite neu dargestellt, ist auf der linken Buchseite das Startbild und auf der rechten Seite der Text von der ersten Zeile an abwärts zu sehen. Verknüpfungen werden durch Hervorhebung der Wörter, die als Startpunkt der Verknüpfung dienen, dargestellt: internen Verknüpfungen werden durch eine rote Schriftfarbe, die externen durch eine blaue Schriftfarbe markiert.

Der Programmaspekt einer Multimediapräsentation betrifft den dynamischen Aspekt der Darstellung der giMLi-Präsentation. Hierbei handelt es sich um die Interaktionsmöglichkeiten des Betrachters mit der Präsentation. Bei der giMLi-Präsentation ist dies zum einen die Navigation durch die Präsentation. Mit Hilfe der Maus als Eingabegerät kann der Betrachter die definierten Verknüpfungen auslösen. Das Auslösen einer Verknüpfung bewirkt, ähnlich wie bei einem HTML-Link, die Anzeige des Ziels der Verknüpfung. Je nach Typ der Verknüpfung hat dies unterschiedliche Auswirkungen: Das Verfolgen einer internen Verknüpfung tauscht das Bild auf der linken Buchseite der Präsentation mit seiner Bildunterschrift aus. Der Rest der Bildschirmseite mit seinen dargestellten Elementen eines Themas bleibt davon unberührt. Nur das vor dem Auslösen der Verknüpfung angezeigte Bild wird durch das als Ziel der Verknüpfung definierte Bild mitsamt Bildunterschrift ersetzt. Wenn hingegen eine externe Verknüpfung verfolgt wird, wird das gesamte Thema, also die komplette Bildschirmseite, ausgetauscht. Alle auf dem Bildschirm dargestellten Elemente eines Themas werden durch die Elemente des neuen, als Ziel der

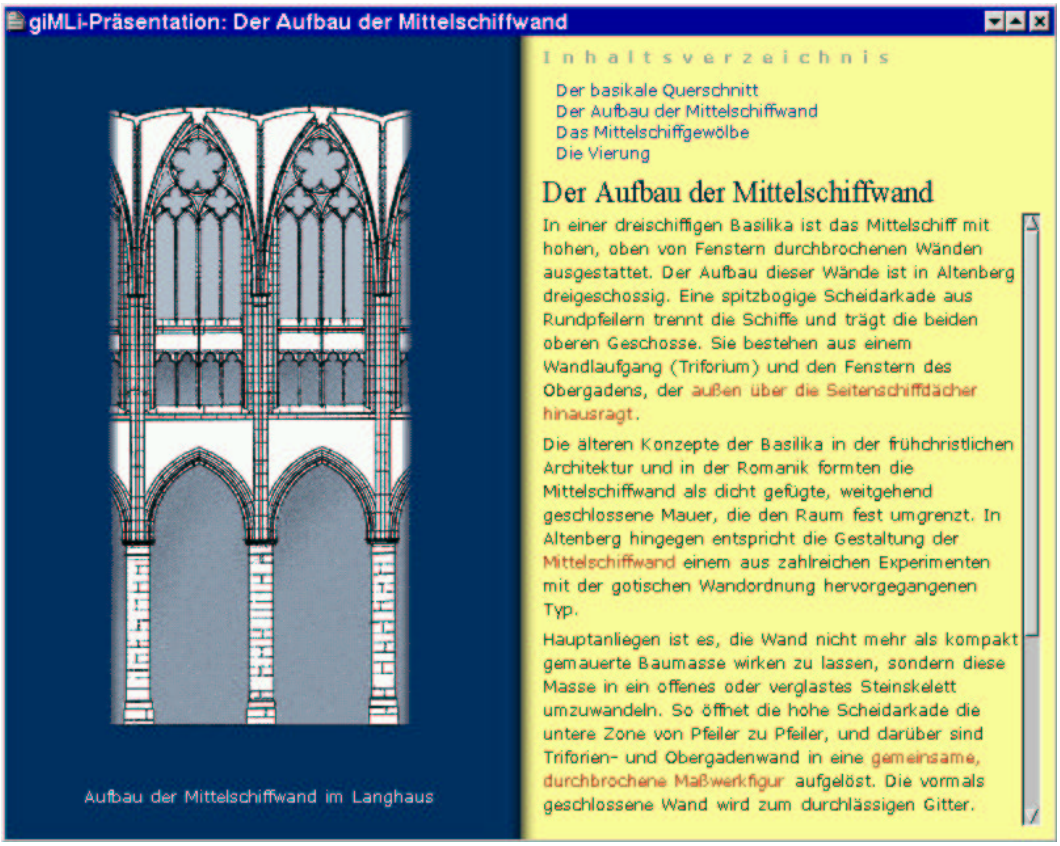


Abbildung 5.1: Die giMLi-Präsentation

Verknüpfung angegebenen Themas ersetzt. Als weitere Interaktionsmöglichkeit können zum anderen nicht zu sehende Textpassagen, falls ein Text länger ist als der für ihn vorgesehene Platz, mit Hilfe von Scrollleisten in den sichtbaren Bereich verschoben werden.

Eine giMLi-Präsentation besitzt einen Zustandsbegriff, der nur innerhalb eines Themas definiert ist. Beim Verlassen eines Themas durch einen externen Link gehen die Informationen, welches Bild gerade angezeigt wird und welcher Bereich des Textes gerade sichtbar ist, also der aktuelle Zustand der Darstellung des Themas, verloren. Bei der Anzeige eines neuen Themas, entweder initial beim Start der Präsentation oder durch Verfolgen einer externen Verknüpfung, wird immer der Grundzustand eines Themas angenommen: das Startbild und der Text von der ersten Zeile abwärts werden dargestellt. Ein globaler, also präsentationsweiter Zustandsbegriff ist nicht vorgesehen, außerhalb der Themen ist die Präsentation gedächtnislos.

Mit den hier vorgestellten Elementen stellt die giMLi-Präsentation ein repräsentatives Beispiel für ξML-Präsentationen dar. Sie orientiert sich stark an der Altenberger Dom-Präsentation und beinhaltet nur Elemente nicht, die keine neuen konzeptionellen Anforderungen stellen. Alle wesentlichen Eigenschaften einer ξML-Präsentation finden sich wieder. Auffällig ist, dass in der giMLi-Präsentation nur diskrete Medien verwendet werden. Dies bedeutet aber für die Aussa-

gekraft der giMLi-Präsentation als Referenzpräsentation keinerlei Einschränkung, da bei ξML-Präsentationen nur ein medieninterner Zustandsbegriff definiert wird und im Rahmen des einfachen, ereignisorientierten zeitlichem Modells, das giMLi als textzentrierter Multimediapräsentationen zu Grunde liegt, keine komplexen zeitlichen Abhängigkeiten zwischen Medien gefordert werden, wie in Abschnitt 5.2.2 auf der nächsten Seite zu sehen ist. Für die XAM besteht daher kein Unterschied in der Handhabung von Medien, gleichgültig ob sie einem diskreten oder kontinuierlichen Medientyp angehören.

5.2 Anforderungen an die XAM

Die Anforderungen an die XAM gliedern sich in zwei Kategorien. Die erste Kategorie fasst die konzeptionellen Anforderungen an die XAM zusammen (Abschnitt 5.2.1 bis 5.2.5). Die zweite Kategorie betrifft Anforderungen, die sich durch das Ziel, die Beispielpräsentation giMLi darzustellen, ergeben (Abschnitt 5.2.6 auf Seite 46).

Die konzeptionellen Anforderungen besitzen eine unterschiedliche Granularität. Zunächst schildere ich generelle Anforderungen, die die prinzipielle Fähigkeit der XAM zur Darstellung von Multimediapräsentationen im Rahmen des ξML-Systems betreffen (Abschnitt 5.2.1). Sie lassen sich in speziellere konzeptionelle Anforderungen verfeinern, die ich gemäß der in Abschnitt 4.2 und in Abbildung 4.1 auf Seite 32 eingeführten einzelnen Teile der XAM vorstelle: das abstrakte Modell (Abschnitt 5.2.2), das Einlesen der Eingabe (Abschnitt 5.2.3), die Codegenerierung (Abschnitt 5.2.4) und die Schnittstelle (Abschnitt 5.2.5).

Mit den Anforderungen der zweiten Kategorie verlasse ich die konzeptionelle Ebene und betrachte Anforderungen, die zur Realisierung der Beispielpräsentation giMLi aufgestellt werden. Diese konkretisieren am Beispiel der giMLi-Präsentation die konzeptionellen Anforderungen an die multimediale Ausdrucksstärke der XAM, die im Rahmen des abstrakten Modells aufgestellt werden.

Zum besseren Verständnis der Verwendung der Medienbegriffe im Rahmen des Abschnitts über die Anforderungen weise ich darauf hin, dass sich die Phänomene, deren Beobachtung mich zu der Formulierung der Anforderungen führen, auf konkrete Objekte, also Instanzen von Medien, beziehen. Die Anforderungen selbst, die an die XAM bzw. an einzelne Teile der XAM gestellt werden, sind gelöst von konkreten Instanzen und betreffen die Modellierungsebene. Entsprechend verwende ich den Begriff Medieninstanz bei der Beobachtung von Phänomenen und den Begriff Medienelement bei der Formulierung der Anforderungen.

5.2.1 Allgemeine Anforderungen

Die hier aufgeführten Anforderungen sind aus den in Kapitel 4 vorgestellten Gedanken zur Aufgabenstellung, den Lösungsansätzen und der Definition der Ziele dieser Arbeit abgeleitet. Aus Gründen des Lesbarkeit verweise ich im Folgenden nicht mehr explizit auf diese einzelnen Abschnitte.

Die XAM ist, wie der Titel der Arbeit besagt, die Laufzeitumgebung für das ξML-System. Sie soll es ermöglichen, eine von technischen Details der Realisierung der Darstellung losgelöste Beschreibung einer Multimediapräsentation im Rahmen des ξML-Systems darzustellen. Die

Darstellung soll auf der Grundlage von bereits etablierten Multimediaformaten erfolgen und die für diese Formate vorhandenen Player sollen zur endgültigen Ausführung der Multimediapräsentation genutzt werden. Die XAM soll nicht als ein neuer Player entwickelt werden. Dies lässt sich als folgende Anforderung formulieren:

Anforderung 1 (Darstellung von ξML-Präsentationen) *Generell sollen mit der XAM darstellungsneutral beschriebene ξML-Präsentationen dargestellt werden können. Zur Darstellung sollen etablierte Multimediaformate und deren Player genutzt werden.*

Zentrale Anforderung an die XAM ist es, dass sie auch mit neu definierten Elementen beschriebene ξML-Präsentationen darstellen kann, ohne sie dazu jedesmal neu entwerfen zu müssen. Wie in der Aufgabenstellung gefordert, soll sie es außerdem ermöglichen, mit vertretbarem Aufwand neue Zielformate zur Darstellung der ξML-Präsentationen nutzen zu können. Somit ergibt sich folgende Anforderung:

Anforderung 2 (Erweiterbarkeit) *Die XAM soll auch nach ihrer Entwicklungszeit mit vertretbarem Aufwand sowohl in der von ihr bereitgestellten multimedialen Ausdrucksstärke, als auch in den von ihr unterstützten Formaten zur Darstellung erweiterbar sein.*

Die XAM ist Bestandteil des ξML-Systems, das ein komponentenorientiertes Modularisierungskonzept zur Realisierung von Wiederverwendung und Anpassbarkeit verfolgt (siehe Abschnitte 3.2 und 3.4). Dies betrifft die Art, wie die Beschreibung einer ξML-Präsentation von Multimediaautoren erstellt wird und damit verbunden, wie diese Beschreibung dargestellt werden kann. Daher soll die XAM auch mit einem komponentenorientierten Ansatz zur Modularisierung entworfen werden. Diese Überlegungen führen zu folgender Anforderung:

Anforderung 3 (Komponentenorientiertheit) *Der Entwurf der XAM soll komponentenorientiert erfolgen. Einzelne Bestandteile der XAM sollen durch einen komponentenorientierten Ansatz modularisiert werden.*

Eine Charakterisierung eines im Rahmen dieser Arbeit gültigen Komponentenbegriffs erfolgt in Abschnitt 5.3.

Die Anforderungen dieses Abschnitts stecken den generellen konzeptionellen Rahmen der XAM ab. Ich verfeinere sie in den folgenden Abschnitten 5.2.2 bis 5.2.5 an Hand der einzelnen Bestandteile der XAM.

5.2.2 Das abstrakte Modell

Das abstrakte Modell definiert, wie in Abschnitt 4.2.3 vorgestellt, den Rahmen, in dem die Darstellung von Multimediapräsentationen durch die XAM generell unterstützt wird. Es setzt sich aus Basisprimitiven zusammen, die allgemeine Ausdrucksmöglichkeiten für die in Abschnitt 2.2.1 vorgestellten vier Dimensionen von Multimediapräsentationen bereitstellen. Aus den Basisprimitiven werden die abstrakten Elemente und ihre Implementierungen abgeleitet, mit denen ξML-Präsentationen beschrieben und dargestellt werden können.

Um Anforderung 1 auf der vorherigen Seite gerecht zu werden, muss das mit den Basisprimitiven definierte Modell eine solche Ausdrucksstärke anbieten, dass mit ihm prinzipiell die

Darstellung von ξML-Präsentationen ermöglicht wird. Die Beschreibungskraft des abstrakten Modells muss generell so stark sein, dass alle Aspekte der ξML-Präsentationen in den vier Dimensionen von Multimediadokumenten modelliert werden können. Für den im Rahmen dieser Arbeit betrachteten konkreten Fall einer ξML-Präsentation, der giMLi-Präsentation, bedeutet dies speziell, dass aus den Basisprimitiven abstrakte Elemente und deren Implementierungen abgeleitet werden können müssen, mit denen es möglich ist, eben diese giMLi-Präsentation darzustellen. Es muss also folgendes gefordert werden:

Anforderung 4 (Mächtigkeit der Basisprimitive) *Das abstrakte Modell muss mit den Basisprimitiven eine solche Mächtigkeit bereitstellen, dass sie die für die Bedürfnisse zur Darstellung von ξML-Präsentationen benötigten Aspekte in den vier Dimensionen von Multimediapräsentationen adressieren können. Aus den Basisprimitiven müssen durch Spezialisierungsmechanismen solche abstrakten Elemente ableitbar sein, um ξML-Präsentationen mit ihnen ausdrücken und darstellen zu können.*

Zur Findung der detaillierten Anforderungen an die Basisprimitive nutze ich die Charakterisierung von allgemeinen textzentrierten Multimediapräsentationen (siehe Definition 8 auf Seite 21) und die Eigenschaften des konkreten Beispiels einer ξML-Präsentation, der giMLi-Präsentation, die in Abschnitt 5.1 vorgestellt wird. Im Folgenden werde ich auf diese Anforderungen an die Basisprimitive eingehen.

Eine wesentliche Eigenschaft von Multimediapräsentationen ist nach Definition 7 auf Seite 8, dass sie Informationen mit Hilfe von *Medieninstanzen*, die unterschiedlichen *Medientypen* angehören, vermitteln. Nach Definition 3 auf Seite 6 sind Medieninstanzen die konkreten Ausprägungen eines Medientyps bzw. eines Medienelements des entsprechenden Medientyps. Dieses wird auch von [GÖTZE 1995] als wesentliche Charakteristik einer Multimediapräsentation gesehen. Um aus dieser Eigenschaft von Multimediapräsentationen Anforderungen abzuleiten, benötige ich einige Begriffsdefinitionen, die ich im Folgenden einführe.

In Abschnitt 2.1.1 wird ein Medientyp allgemein als eine Informationseinheit vorgestellt, die sich in der Art, wie er Informationen präsentiert, von anderen Medientypen unterscheidet (siehe Definition 1 auf Seite 5). Im Rahmen der hier zu formulierenden Anforderungen an das abstrakte Modell unterscheide ich dabei nicht zwischen einzelnen Medientypen wie etwa Bildern, Texten oder Videos, da die XAM die Darstellung der Medieninstanzen von Medienelementen der entsprechenden Medientypen mit Hilfe von externer Funktionalität vornimmt, so dass die innere Struktur der Medieninstanzen vor der XAM verborgen bleibt. Die XAM besitzt daher nur durch eine geeignete Attributierung die Möglichkeit, auf diese Medieninstanzen Einfluss zu nehmen, falls ein solcher Parametrisierungsmechanismus von dem entsprechenden Medienelement angeboten und von einem entsprechenden Player, den die XAM zur Darstellung der Instanz dieses Medienelements nutzt, unterstützt wird. Die Kapselung der internen Struktur vor der XAM soll durch einen solchen Mechanismus nicht verletzt werden. Ein Beispiel für eine solche Parametrisierung ist etwa Möglichkeit bei dem Medienelement `QTVR_Video`, über eine Schnittstelle auf die innerhalb der Instanz eines `QTVR_Video`-Medienelements definierten Hotspots zugreifen zu können. Diese gerade beschriebene Eigenschaft einer Informationseinheit nenne ich *Atomarität*:

Definition 11 (Atomarität) *Die Eigenschaft der **Atomarität** kennzeichnet eine Einheit von Information aus als die kleinste von der XAM wahrzunehmende Einheit von Information aus. Eine solche atomare Einheit von Information besitzt für die XAM keinerlei innere Struktur.*

Die atomare Eigenschaft kann einen Medientyp kennzeichnen und dadurch die von ihm spezialisierten Medienelemente und deren konkreten Ausprägungen, die Medieninstanzen. Es wird daher für Medientypen wie folgt definiert:

Definition 12 (Atomare Medientypen) *Ein Medientyp, der eine kleinste von der XAM zur Darstellung von Informationen wahrgenommene Einheit bildet, besitzt die Eigenschaft der* Atomarität *und wird **atomarer Medientyp** genannt.*

Medientypen können mehrere Eigenschaften besitzen (siehe Definition 1 auf Seite 5). Die atomare Eigenschaft zeichnet einen Medientyp neben anderen Eigenschaften wie die diskrete oder kontinuierliche Eigenschaft zusätzlich aus. Ein Medienelement, das die Spezialisierung eines als atomar gekennzeichneten Medientyps ist, nenne ich entsprechend *atomares Medienelement*:

Definition 13 (Atomares Medienelement) *Eine Spezialisierung eines atomaren Medientyps um die Eigenschaft der Codierung wird **atomares Medienelement** genannt. Es kann, falls es einen geeigneten Mechanismus anbietet, der die Eigenschaft des atomaren Medientyps nicht verletzen darf (die Kapselung der internen Struktur vor der XAM), von der XAM parametrisiert werden.*

Die Vermittlung einer Information erfolgt nicht immer nur durch die Benutzung einer einzelnen Medieninstanz. Bei der Erstellung von Multimediapräsentationen ist es üblich, mit Hilfe der Komposition von Medieninstanzen unterschiedlicher Medienelemente Informationen zu modellieren (siehe Seite 13 in [ROISIN 1998]). Eine solche aus anderen Medieninstanzen zusammengesetzte Medieninstanz besitzt eine innere Struktur, die der XAM bekannt ist und die sich aus den anderen Medieninstanzen zusammensetzt. Diese Zusammensetzung soll auch geschachtelt angewendet werden können. Dabei wird die innere Struktur nur so weit offengelegt, dass die Eigenschaft der Kapselung der inneren Struktur von atomaren Medienelementen nicht verletzt wird. Die so zu charakterisierende Eigenschaft einer Informationseinheit nenne ich *Komposition*:

Definition 14 (Komposition) *Die Eigenschaft der **Komposition** kennzeichnet eine Einheit von Information als eine Zusammensetzung aus anderen Einheiten von Information aus. Die an der Komposition teilnehmenden Informationseinheiten können atomare oder bereits komponierte Einheiten sein. Die innerer Struktur einer komponierten Informationseinheit wird dabei von der XAM, unter Einhaltung der Kapselung der inneren Struktur von an der Komposition teilnehmenden atomaren Einheiten von Informationen, wahrgenommen.*

Wie bei der Eigenschaft der Atomarität bezieht sich die Eigenschaft der Komposition auf alle Ebenen des Medienbegriffs. Für einen Medientyp der die Eigenschaft der Kompostion besitzt wird daher wie folgt definiert:

Definition 15 (Komponierter Medientyp) *Ein Medientyp heißt **zusammengesetzter** oder **komponierter Medientyp**, wenn er sich aus anderen Medientypen zusammensetzt, die sowohl atomar oder bereits komponiert sein können. Die von der XAM wahrgenommene innere Struktur des komponierten Medientyps besteht aus den an der Komposition teilnehmenden Medientypen. Nimmt ein atomarer Medientyp an der Komposition Teil, bleibt seine innere Struktur vor der XAM verborgen.*

Medienelemente, die komponierte Medientypen um Codierungsinformationen spezialisieren, werden *komponierte Medienelemente* genannt:

Definition 16 (Komponiertes Medienelement) *Eine Spezialisierung eines komponierten Medientyps um Eigenschaften der Codierung wird **zusammengesetztes** oder **komponiertes Medienelement** genannt.*

Nach diesen vorbereitenden Definitionen beginne ich mit der Formulierung von weiteren Anforderungen, die sich aus der oben genannten Eigenschaft der Verwendung von Medieninstanzen unterschiedlicher Medientypen innerhalb von Multimediapräsentationen ergeben. Diese Eigenschaft von allgemeinen Multimediapräsentationen trifft natürlich auch auf die textzentrierten ξML-Präsentationen zu. Daher ist es, wie in [ROISIN 1998], Seite 13, allgemein für die Erstellung von Multimediapräsentationen gefordert, eine Anforderung an das abstrakte Modell, dass es die Darstellung von Informationen mit Hilfe von unterschiedlichen Medienelementen modellieren kann. Dazu soll es, aus den im Rahmen ihrer Definitionen gegebenen Gründen, Instanzen atomarer und komponierter Medienelemente verwenden. Um atomare Medienelemente verwenden zu können wird daher gefordert:

Anforderung 5 (Atomare Medienelemente) *Das abstrakte Modell muss die Möglichkeit zur Definition von unterschiedlichen atomaren Medienelementen, mit deren Instanzen in der Präsentation Informationen dargestellt werden, bieten.*

Wie sich aus der Definition der Atomarität ergibt, kennt die XAM nicht die innere Struktur eines atomaren Medienelements. Da auch komponierte Medienelemente von der XAM verwendet werden sollen, muss das abstrakte Modell neben der Forderung nach atomaren Medienelementen auch die Modellierung einer Komposition von Medienelementen ermöglichen. Dabei sollen die aus den Medienelementen, die an der Komposition beteiligt sind, bestehende innere Struktur der XAM bekannt sein. Somit lässt sich folgende Anforderung formulieren:

Anforderung 6 (Komponierte Medienelemente) *Das abstrakte Modell muss es ermöglichen, Medienelemente zu zusammengesetzten Medienelementen komponieren zu können, die dann als eine Einheit mit einer der XAM bekannten inneren Struktur angesehen werden. Die innere Struktur eines atomaren Medienelements, das an der Komposition teilnimmt, bleibt per Definition verborgen.*

Um die Medienelemente, atomare oder zusammengesetzte, darzustellen, müssen ihre Instanzen auf dem Bildschirm angezeigt werden. Dazu sollten die Medieninstanzen geeignet auf dem Bildschirm angeordnet werden können. Um dies zu erreichen, muss das abstrakte Modell die Möglichkeit bereitstellen, Medienelemente auf dem Ausgabemedium positionieren zu können:

Anforderung 7 (Positionierung) *Im Rahmen des abstrakten Modells müssen Medienelemente auf dem Ausgabemedium positioniert werden können.*

Wie in Abschnitt 2.2.1 im Rahmen der räumlichen Dimension vorgestellt, gehört zum optischen Erscheinungsbild einer Präsentation neben der räumlichen Anordnung der Medienelemente auch die typografischen Eigenschaften der Präsentation. Diese Eigenschaften bestehen aus der *physikalische Struktur* und den *Stilen* und bilden das Layout. Ein solches Layout betrifft daher zum einen die Gesamterscheinung der Präsentation und zum anderen das Erscheinungsbild der Medieninstanzen, die über oben geschilderten Mechanismus parametrisiert werden können. Das Layout einer Präsentation wird über Informationen, wie etwa die zu verwendenden Farben, Schriftarten oder Abstände festgelegt. Diese Informationen müssen für eine Präsentation angegeben werden können. Das abstrakte Modell muss es daher neben der Positionierung auch ermöglichen, Layoutinformationen für eine Präsentation festlegen zu können:

Anforderung 8 (Layout) *Mit Hilfe des abstrakten Modells müssen Layoutinformationen für das optische Erscheinungsbild der Darstellung einer ξML-Präsentation definiert werden können.*

Neben der Präsentation von Informationen mit Hilfe von Medieninstanzen und der Möglichkeit zur Komposition der Medieninstanzen ist nach [ROISIN 1998], Seite 13, die Fähigkeit zur Navigation durch die Präsentation eine weitere zentrale Eigenschaft von Multimediapräsentationen. Um es bei der Darstellung von ξML-Präsentationen zu ermöglichen, dass der Benutzer durch die Präsentation navigieren kann, sind einige Anforderungen an das abstrakte Modell nötig. Zunächst müssen die Pfade, entlang derer navigiert werden soll, definiert werden. Dazu werden die Medieninstanzen miteinander in Beziehung gesetzt. Da mit einer Medieninstanz meist mehr als nur eine Information vermittelt wird, ist es für eine detaillierte Modellierung der Beziehungen zwischen den in den Medieninstanzen dargestellten Informationen nötig, auf eine feingranularere Struktur, als sie mit der Einteilung in atomare und zusammengesetzte Medienelemente definiert wird, zurückzugreifen. Nicht nur zwischen Medienelementen als ganzes, sondern auch, falls vorhanden, zwischen Einheiten der internen Strukturen von Medienelementen müssen solche Beziehungen formuliert werden können. Hierbei ist zu beachten, dass die in Anforderung 5 auf der vorherigen Seite geforderte Kapselung der internen Struktur von atomaren Medienelementen vor der XAM aufrecht gehalten wird. Es muss hierfür ein geeigneter Mechanismus gefunden werden, um innerhalb von atomaren Medienelementen Bereiche ansprechen zu können, ohne dass die innere Struktur der Medien offengelegt wird. Das abstrakt Modell muss daher folgende Anforderung erfüllen:

Anforderung 9 (Beziehungen) *Mit dem abstrakten Modell muss es ermöglicht werden, zwischen Medienelementen, bzw. zwischen einzelnen Informationseinheiten von Medienelementen, Beziehungen definieren zu können. Die Kapselung der inneren Struktur von atomaren Medienelementen vor der XAM soll dabei aufrecht erhalten bleiben.*

Für die Definition von Beziehungen ist es nötig, dass man die Informationen, zwischen denen eine solche Beziehung besteht, benennen kann. Hierfür müssen zum einen atomare und komponierte Medienelemente als Ganzen adressiert werden können. Da aber nach Anforderung 9 auch für Informationen innerhalb von Medienelementen Beziehungen definiert werden sollen, muss

das abstrakte Modell eine Adressierung auch innerhalb von Medienelementen ermöglichen. Es wird gefordert:

Anforderung 10 (Adressierung) *Das abstrakte Modell muss die Adressierung von Informationen unterstützen. Mit dem Adressierungsschema müssen Medienelemente als Ganzes, sowohl atomare als auch komponierte, und Informationseinheiten innerhalb von Medienelementen ausgezeichnet werden können.*

Bis zu dieser Stelle habe ich Anforderungen formuliert, die die statischen Aspekte des abstrakten Modells betreffen. Da das abstrakte Modell aber die Grundlage der von der XAM unterstützten Multimediapräsentationen ist, müssen mit ihm neben den statischen auch die dynamischen Aspekte, die einer Multimediapräsentation innewohnen, adressiert werden. Mit den folgenden Anforderungen gehe ich auf diese Aspekte ein. Sie beziehen sich daher nicht auf die statische Ebene der Basisprimitive, sondern auf die dynamische Ebene einer konkreten Ausprägung des abstrakten Modells.

Da mit den Beziehungen eine Navigation durch die Multimediapräsentation ermöglicht werden soll, muss der Benutzer die Möglichkeit haben, einen Pfad innerhalb der Präsentation auszuwählen, dem er folgen möchte. Dazu muss es dem Benutzer ermöglicht werden, seine Wahl der Präsentation kund zu tun. Er muss mit der Präsentation interagieren können:

Anforderung 11 (Interaktion) *Eine Realisierung abstrakte Modell muss er erlauben, dass der Benutzers über Eingabegeräte mit der Präsentation interagieren kann.*

Um durch eine Präsentation navigieren zu können, müssen nicht nur mögliche Navigationspfade durch die Beziehungen angegeben und durch Interaktion ausgewählt werden können, sondern das Auswählen eines solchen Pfades muss die Verfolgung einer Beziehung nach sich ziehen. Der Benutzer muss durch Auswählen eines Pfades zu der in Beziehung gesetzten Information gelangen können:

Anforderung 12 (Verfolgung von Beziehungen) *Das Verfolgen von Beziehungen durch Interaktion mit der Präsentation muss für den Benutzer mit einer Instanziierung des abstrakten Modells ermöglicht werden, um so durch die Präsentation navigieren zu können.*

Ein weiterer, von den bisherigen Anforderungen noch nicht berührter Aspekt, ist die zeitliche Dimension von Multimediapräsentationen. Wie in Abschnitt 3.4 vorgestellt, liegt den ξML-Präsentationen ein einfaches, ereignisorientiertes zeitliches Modell zu Grunde. Im Rahmen dieser Arbeit wird aus Umfangsgründen darauf verzichtet, differenzierte zeitliche Kompositionsmöglichkeiten von Medienelementen, wie etwa Synchronisationsbeziehungen oder die Definition von beliebigen Ordnungen bei zeitlichen Relationen, anzubieten. Solche Aspekte können nur medienintern, also mit einem Erstellungswerkzeug der jeweiligen Instanz der Medienelemente, definiert werden, und obliegen daher den Ausdrucksmöglichkeiten des Formats der verwendeten Medienelemente und nicht der XAM. Als einzige zeitliche Relation, deren Darstellung die XAM unterstützt, wird im Rahmen der Arbeit gemäß den Eigenschaften von textzentrierten Multimediapräsentationen (siehe Definition 8 auf Seite 21) die Gleichzeitigkeit der Anzeige von Medieninstanzen definiert. Die Ansicht von parallel anzuzeigenden Medieninstanzen muss der

Benutzer mit Ereignissen, die er durch Interaktion mit der Präsentation auslöst, wechseln können. Zusammenfassend wird von dem abstrakten Modell gefordert:

Anforderung 13 (Zeitliche Beziehungen) *Mit Hilfe des abstrakten Modell müssen zwischen Medienelementen zeitliche Relationen definiert werden können, die in einer konkreten Ausprägung des abstarkten Modells die gleichzeitige Anzeige der an den Relationen teilnehmenden Medieninstanzen bewirken. Der Wechsel zwischen unterschiedlichen, als parallel anzuzeigend ausgezeichneten Medieninstanzen muss durch Benutzerinteraktion möglich sein.*

Im Rahmen dieser Arbeit besitzen ξML-Präsentationen, und damit auch die giMLi-Präsentation, wie in Abschnitt 5.1.1 auf Seite 35 beschrieben, keinen präsentationsweiten Zustandsbegriff. Die in den Zielplattformen angebotenen Möglichkeiten zur Realisierung eines präsentationsweiten Zustandsbegriffs sind zu unterschiedlich, als dass sie auf abstrakte Weise beschrieben werden können. Lediglich medienweit kann ein Zustandsbegriff definiert werden. Für atomare Medienelemente hängt er von den im jeweiligen Format des Medienelements gegeben Möglichkeiten ab und kann, da die Erstellung der Medienelementen außerhalb der XAM vorgenommen wird, nicht von dem abstrakten Modell unterstützt werden. Für komponierte Medienelemente hängt der Zustandsbegriff von den Fähigkeiten der konkret verwendeten Medien und der gewählten Zielplattform ab. Auch hier ist daher eine Unterstützung auf Niveau des abstrakten Modells nicht möglich und muss durch abstrakte Elemente, die die konkrete multimediale Funktionalität definieren, bzw. implizit in deren Implementierungen, für die jeweilige Zielplattform erfolgen. Eine Anforderung an einem Zustandsbegriff wird daher nicht in den Katalog mit aufgenommen.

5.2.3 Das Einlesen der Eingabe

Die Eingabe der XAM besteht aus der Beschreibung einer ξML-Präsentation. Sie wird in Form eines Dokuments vorliegen, das eingelesen und in die Zwischendarstellung der XAM überführt werden muss. Hierbei müssen, wie in [RUTLEDGE et al. 1998] geschildert, die in der Eingabesprache verwendeten Konzepte auf die Konzepte der Zwischendarstellung abgebildet werden. Für das Einlesen wird daher gefordert:

Anforderung 14 (Einlesen) *Aus einem Eingabedokument, mit dem eine ξML-Präsentation beschrieben wird, muss beim Einlesen die entsprechende Zwischendarstellung erzeugt werden können.*

Das Einlesen der Eingabe erfolgt mit einem Parser. Da sich die in einer ξML-Präsentation, und somit in dem Eingabedokument, verwendeten multimedialen Elemente von Präsentation zu Präsentation unterscheiden können, muss der Parser an die Ausdrucksstärke der jeweiligen Präsentation anpassbarer sein. Dieser Aspekt fällt unter die in Anforderung 2 auf Seite 38 geforderte Erweiterbarkeit der XAM und wird daher nicht mehr explizit in den Anforderungskatalog mit aufgenommen.

Ein weiterer Punkt in diesem Zusammenhang ist, dass für die Sprache, mit der die Eingabe der XAM formuliert wird, Aspekte wie gute Lesbarkeit oder leichte Erstellbarkeit durch einen Menschen keine Rolle spielen. Diese Aspekte sind unerheblich, da im Rahmen des ξML-Systems die Eingabe durch einen vorgelagerten Generator automatisiert aus einer auf die Bedürfnisse des

ξML-Autors angepassten Beschreibung der jeweiligen ξML-Präsentation erzeugt wird. Für die im Rahmen dieser Arbeit zu erstellende Eingabesprache XaML, mit der eine giMLi-Präsentation in den Ausdrucksmöglichkeiten der XAM beschrieben werden kann, sind diese Aspekte daher auch irrelevant.

5.2.4 Die Codegenerierung

Um eine ξML-Präsentation mit der XAM darstellen zu können, muss die von technischen Details weitesgehend abstrahierte Beschreibung einer Präsentation in Formate der Zielplattform überführt werden. Die Codegenerierung ist dabei für den Schritt der Überführung der Zwischendarstellung in die Zielsprache zuständig. Sie besteht aus Werkzeugen, die für die einzelnen Zielplattformen Code generieren. Aufgabe dabei ist es, die Präsentation zu rendern, in dem die Konzepte der Eingabesprache in die der Zielsprache abgebildet werden [RUTLEDGE et al. 1998]:

Anforderung 15 (Rendern) *Die Codeerzeugungswerkzeuge der XAM müssen eine ξML-Präsentation rendern können. Die Konzepte der Eingabsprache müssen von den Codeerzeugungswerkzeugen in die entsprechenden Ausdrucksmöglichkeiten der jeweiligen Zielplattform abgebildet werden können.*

Die Aspekte der Erweiterbarkeit der Werkzeuge und der Auswahl der konkret zu überführenden Konzepte werden nicht an dieser Stelle behandelt. Sie treffen nicht nur auf die Codeerzeugung zu und werden in einem allgemeineren Kontext betrachtet. Entsprechende Anforderungen werden in Abschnitt 5.2.1 auf Seite 37 und Abschnitt 5.2.6 auf der nächsten Seite gestellt.

Die Darstellung der ξML-Präsentation mit der XAM erfolgt durch Erzeugung von Code für die gewünschte Zielplattformen. Zum Betrachten der fertigen Präsentation wird der erzeugte Code durch einen entsprechenden Player für das jeweilige Format ausgeführt. Hierfür ist die XAM nicht mehr erforderlich. Die Codeerzeugung erfolgt nur im Rahmen der Erstellung der Präsentation. Ein häufiges Ausführen der XAM findet also nicht statt. Daher sind Effizienzbetrachtungen an dieser Stelle nicht wesentlich und werden aus Zeitgründen im Rahmen dieser Arbeit auch nicht vorgenommen.

5.2.5 Die Schnittstelle

Wie in Abschnitt 4.2 vorgestellt, dient die Schnittstelle dazu, die XAM bei jedem Start mit der von der darzustellenden ξML-Präsentation benötigten multimedialen Funktionalität zu initialisieren. Dazu muss sie einen Mechanismus bereitstellen, mit dem die entsprechenden abstrakten Elemente und deren Implementierungen bei den entsprechenden Teilen der XAM registriert werden können: bei dem Parser, der das Eingabedokument verarbeitet, bei der Zwischendarstellung und bei den jeweiligen Generierungswerkzeugen. Für die Schnittstelle wird daher gefordert:

Anforderung 16 (Initialisierung) *Die Schnittstelle muss es ermöglichen, dass die XAM mit der multimedialen Funktionalität initialisiert werden kann, die zur Darstellung einer ξML-Präsentation nötig ist.*

Die Schnittstelle manifestiert sich unter anderem in den abstrakten Elementen und deren Implementierungen, mit denen die XAM initialisiert wird. Sie sollen entsprechend des Gesamtansatzes des ξML-Systems komponentenorientiert entworfen werden. Dieses ist bereits in Anforderung 3 auf Seite 38 festgehalten worden.

Die abstrakten Elemente und ihre Implementierungen definieren die multimediale Funktionalität, die der XAM zum Darstellen von ξML-Präsentationen zur Verfügung steht. Welche Funktionalität sie konkret definieren sollen, wird in den Anforderungen in Abschnitt 5.2.6 vorgestellt. Damit sie benutzbar sein können, muss offengelegt werden, welche multimediale Funktonalität sie bereitstellen. Es muss also ihre Bedeutung definiert werden. Im Rahmen dieser Arbeit wird dies auf informale Weise bei der Beschreibung des Entwurfs der einzelnen abstrakten Elemente und ihrer Implementierungen geschehen. Eine formale Spezifikation übersteigt den Rahmen dieser Diplomarbeit und wird daher auch nicht gefordert:

Anforderung 17 (Bedeutung der multimedialen Funktionalität) *Für die multimediale Funktionalität, die mit den abstrakten Elementen beschrieben wird, muss ihre Bedeutung festgelegt werden. Im Rahmen dieser Arbeit geschieht dies informal.*

Mit diesem Abschnitt endet die Definition der konzeptionellen Anforderungen an XAM. Im nächsten Abschnitt wird die multimediale Funktionalität betrachtet, die zur Darstellung einer ξML-Präsentation benötigt wird.

5.2.6 Die Darstellung der giMLi-Präsentation

Die multimediale Ausdrucksstärke bzw. die multimediale Funktionalität ermöglicht der XAM, eine ξML-Präsentation darstellen zu können. Sie betrifft also die Fähigkeit, die multimedialen Elemente einer ξML-Präsentation mit den Ausdrucksmöglichkeiten der XAM beschreiben und diese Beschreibung in darstellbare Formen überführen zu können. Sie wird in den abstrakten Elementen und ihren Implementierungen definiert.

Die Anforderungen an die multimediale Funktionalität hängen daher von den multimedialen Elementen ab, die in einer giMLi-Präsentation verwendet werden. Die multimedialen Elemente sind aber je nach Präsentation unterschiedlich und an den Vorlieben des Autors, der die Präsentation erstellt, orientiert, was eine Verallgemeinerung der Anforderungen an multimediale Funktionalität schwierig macht, wie auch in [ROISIN 1998], Abschnitt 4.1, festgestellt wird. Da mit dieser Arbeit, wie in Abschnitt 4.3 als Ziel formuliert, die Beispielpräsentation giMLi betrachtet werden soll, löse ich diese Schwierigkeit, in dem ich im Rahmen dieser Arbeit die multimediale Funktionalität der XAM, die zur Darstellung von ξML-Präsentationen benötigt wird, gemäß der Ausdrucksstärke, die zur Darstellung der giMLi-Präsentation benötigt wird, definiere. Dies bedeutet keine Einschränkung der XAM, da später weitere multimediale Funktionalität, wie in Anforderung 2 auf Seite 38 beschrieben, außerhalb dieser Arbeit definiert werden kann. Generell lässt sich somit für die multimediale Funktionalität fordern:

Anforderung 18 (Multimediale Funktionalität) *Im Rahmen dieser Arbeit soll für die XAM eine solche multimediale Funktionalität zur Darstellen von ξML-Präsentationen entwickelt werden, dass mit ihr die giMLi-Präsentation dargestellt werden kann.*

Die multimediale Funktionalität, die die XAM zum Darstellen von ξML-Präsentationen besitzt, wird, wie bereits gesagt, in den abstrakten Elementen und ihren Implementierungen definiert. Die Anforderungen, die sich im einzelnen an die multimediale Funktionalität ergeben, müssen daher durch geeignete abstrakte Elemente und ihre Implementierungen realisiert werden. Im Folgenden wird deshalb die Abschnitt 2.1.2 eingeführte in kursive Schreibweise des Begriffs *Medium* verwendet, um zu verdeutlichen, dass ich mich bei seiner Verwendung auf der Ebene der Realisierung von geforderten inhaltlichen Konzepten zur Darstellung von Multimediapräsentationen bewege. Wenn ich mich auf die Realisierung der inhaltlichen Konzepte beziehe, verwende ich auch als Kurzschreibweise ebenfalls die kursive Schriftart. Beziehe ich mich beispielsweise bei einem Bild oder einem Text auf die Realisierung des inhaltlichen Konzepts Bild oder Text, so benutze ich die Schreibweise *Bild* und *Text* anstelle der Schreibweise das *Medium* Bild oder das *Medium* Text.

Die nun folgenden einzelnen Anforderungen leiten sich aus den in Abschnitt 5.1.1 auf Seite 35 vorgestellten Elementen ab, aus denen die giMLi-Präsentation besteht.

Mit der giMLi-Präsentation werden Medieninstanzen von Medienelementen zweier atomarer Medientypen verwendet, um dem Benutzer der Präsentation Informationen zu vermitteln: Texte und Bilder. Die nächsten beiden Anforderungen sind somit eine Spezialisierung der Anforderung an das abstrakte Modell nach atomaren Medienelementen (Anforderung 5 auf Seite 41). Multimediale Elemente des Medientyps Text sind in der giMLi-Präsentation neben dem *Medium* Text auch das *Medium* Bildunterschrift und das *Medium* Inhaltsverzeichnis:

Anforderung 19 (Texte) *Mit der giMLi-Präsentation müssen textgestützt Informationen dem Betrachter vermittelt werden können. Dazu sollen neben der Verwendung des Mediums Text auch die Medien Bildunterschrift und Inhaltsverzeichnis verwendet werden können. Die drei Medien sollen atomar sein.*

Neben diesen textgestützten Elementen werden, wie oben erwähnt, in der giMLi-Präsentation auch Bilder verwendet. Sie verdeutlichen einzelne Aspekte der in dem *Text* dargestellten Informationen:

Anforderung 20 (Bilder) *In der giMLi-Präsentation müssen Bilder als Medien definiert werden können. Das Bild soll atomar sein.*

In der giMLi-Präsentation werden neben diesen atomaren Medientypen auch komponierte Medientypen verwendet. Die folgenden zwei Anforderungen spezialisieren daher Anforderung 6 auf Seite 41. Zum einen werden in der giMLi-Präsentation die Bilder mit Bilderunterschriften annotiert. Es wird daher gefordert:

Anforderung 21 (Bilder mit Bildunterschriften) *Das Medium Bild muss mit dem Medium Bildunterschrift zu einem Medium Bild mit Bildunterschrift komponiert werden können.*

Zum anderen werden in der giMLi-Präsentation die Medieninstanzen nach übergeordneten inhaltlichen Aspekten zu einem *Medium* Thema gruppiert. Ein *Thema* fasst Elemente, deren Informationen zu einen größeren inhaltlichen Aspekt gehören, zusammen und stellt sie gemeinsam dar:

Anforderung 22 (Themen) *Die in der giMLi-Präsentation verwendeten multimedialen Elemente müssen nach übergeordneten inhaltlichen Aspekten zu dem Medium Thema gruppiert und eine Auswahl von ihnen gleichzeitig dargestellt werden können.*

Für die Darstellung eines *Themas* wird eine Buchmetapher benutzt. Die Medieninstanzen eines *Themas* werden jeweils auf einer Doppelseite eines Buches angeordnet dargestellt. Zusätzlich zu der einheitlichen Anordnung soll die Darstellung eines *Themas* durch die Wahl von gleichen Farben, Schriften und Schriftgrößen mit einem einheitliches Layout erfolgen. Als Spezialisierung der Anforderungen 7 auf Seite 42 und 8 auf Seite 42 an das abstrakte Modell, wird für die multimediale Funktionalität daher gefordert:

Anforderung 23 (Anordnung und Layout) *Zu Themen gruppierte Medienelemente müssen einheitlich dargestellt werden können. Dass heißt, sie müssen auf dem Bildschirm gleich angeordnet und mit einem gleichen Layout versehen werden können. Ihre konkrete Anordnung und die Wahl des Layouts sollen, wie in Abbildung 5.1 auf Seite 36 dargestellt, erfolgen können.*

In der giMLi-Präsentation kann der Benutzer mit Hilfe von inhaltlichen Beziehungen, so genannte *Verknüpfungen*, navigieren. Sie definieren mit einem Start- und einem Zielpunkt eine gerichtete 1-1-Relation zwischen zwei (Teilen von) Medieninstanzen. Die nächsten drei Anforderungen, die Verknüpfungen betreffen, sind somit eine Spezialisierung der in Anforderung 9 auf Seite 42 vorgestellten Beziehung. Es ergibt sich folgende Anforderung:

Anforderung 24 (Verknüpfungen) *Zwischen zwei Medien oder Teilen von ihnen müssen inhaltliche Beziehungen als gerichtete 1-1-Relationen, so genannte Verknüpfungen, definiert werden können.*

In der giMLi-Präsentation werden durch die Auswahl der von ihnen in Beziehung gesetzten Medieninstanzen zwei Arten von Verknüpfungen unterschieden: interne und externe Verknüpfungen. Die erste Art von Verknüpfung definiert Beziehungen zwischen Informationen innerhalb eines *Themas*:

Anforderung 25 (Interne Verknüpfungen) *Zwischen den einzelnen Informationen, die im Text dargestellt werden und zwischen den Bildern mit Bildunterschrift müssen interne Verknüpfungen definiert werden können.*

Die zweite Art von Verknüpfungen beschreibt die inhaltlichen Verbindungen von *Themen* untereinander:

Anforderung 26 (Externe Verknüpfungen) *Es müssen zwischen einzelnen Informationen, die in einem Inhaltsverzeichnis dargestellt werden, und den Themen externe Verknüpfungen definiert werden können.*

Die Startpunkte von Verknüpfungen werden bei der giMLi-Präsentation in den textgestützten Medieninstanzen definiert. Einzelne Informationen dieser *Medien* müssen dazu als Startpunkt einer Verknüpfung ausgezeichnet werden können. Als Spezialisierung der allgemeinen Adressierung (Anforderung 10 auf Seite 43) wird daher gefordert:

Anforderung 27 (Textanker) *Es müssen in dem Text Buchstaben oder ganze Worte als Startpunkt einer Verknüpfung ausgezeichnet werden können. Diese Startpunkte heißen Textanker.*

Als Ziel einer Verknüpfung werden in giMLi Medieninstanzen als Ganzes bzw. Instanzen von Mediengruppierungen (die *Themen*) definiert. Als weitere Spezialisierung von Anforderung 10 auf Seite 43 wird daher gefordert:

Anforderung 28 (Verknüpfungsziele) *In der giMLi-Präsentation müssen gesamte Medien und Gruppen von Medien als Zielpunkt einer Verknüpfung ausgezeichnet werden können.*

Die definierten Verknüpfungen müssen von dem Benutzer zur Navigation ausgewählt werden können. In der giMLi-Präsentation erfolgt die Auswahl einer Verknüpfung mit Hilfe einer Mausklicks. Als Spezialisierung von Anforderung 11 auf Seite 43 wird für die Interaktionsmöglichkeiten daher gefordert:

Anforderung 29 (Interaktion) *Die Interaktion des Benutzers mit der giMLi-Präsentation soll mit dem Eingabegerät Maus erfolgen. Verknüpfungen müssen durch einen Mausklick ausgewählt werden können.*

Das zeitliche Modell, das der giMLi-Präsentation als Beispiel einer ξML-Präsentation zu Grunde liegt, ist, neben der parallelen Darstellung von Medieninstanzen innerhalb von *Themen*, ein ereignisgesteuertes Modell [Boll et al. 1999], das keine expliziten zeitlichen Abhängigkeiten definiert, wie es für textzentrierte Multimediapräsentationen vorgestellt wurde (Definition 8 auf Seite 21). Durch das Verfolgen der *Verknüpfungen* wird die Darstellung der giMLi-Präsentation gesteuert. Das Verfolgen hat, entsprechend der beiden Arten von *Verknüpfungen*, unterschiedliche Auswirkungen. Aus den in Abschnitt 5.1.1 vorgestellten Verhaltensweisen der giMLi-Präsentation bei der Verfolgung von *internen* und *externen Verknüpfungen* lassen sich die folgenden beiden Anforderungen an das dynamische Verhalten der giMLi-Präsentation ableiten. Die nächsten beiden Anforderungen sind somit Spezialisierungen von Anforderung 12 auf Seite 43. Für *interne Verknüpfungen* gilt:

Anforderung 30 (Dynamisches Verhalten interner Verknüpfungen) *Mit dem Verfolgen einer internen Verknüpfung soll der Tausch des aktuell angezeigten Bildes mit dem als Ziel der internen Verknüpfung angegeben Bildes ausgelöst werden.*

Und für *externe Verknüpfungen* muss folgende Anforderung erfüllt werden:

Anforderung 31 (Dynamisches Verhalten externer Verknüpfungen) *Mit dem Verfolgen einer externen Verknüpfung sollen alle zu einem Thema gruppierten Medieninstanzen durch die Medieninstanzen des als Ziel der externen Verknüpfung definierten Themas ausgetauscht werden. Das als Startbild ausgezeichnete Bild mit Bildunterschrift soll dargestellt werden und der Fließtext von der ersten Zeile an abwärts zu sehen sein.*

Mit den Anforderungen an die multimediale Funktionalität endet die Definition der Anforderungen an die XAM. Sie müssen mit dem im Folgenden vorgestellten Entwurf erfüllt werden.

5.3 Komponentenbegriff der XAM

Im Rahmen des ξML-Systems soll der Entwurf der XAM komponentenorientiert erfolgen, wie in Anforderung 3 auf Seite 38 formuliert wird. Da der Begriff der Komponente in der softwaretechnologischen Welt weit verbreitet und unterschiedlich verwendet wird, erläutere ich in diesem Abschnitt einen im Rahmen der XAM gültigen Komponentenbegriff, ehe ich in den nachfolgenden Kapiteln den konkreten Entwurf der XAM vorstelle. Dazu führe zunächst den Begriff der *XAM-Komponenten* ein. XAM-Komponenten werden auf der Ebene der im Lösungsansatz in Abschnitt 4.2.3 vorgestellten *abstrakten Elemente und ihren Implementierungen* benutzt, mit denen die XAM für die Darstellung einer konkreten ξML-Präsentation angepasst bzw. konfiguriert wird. Eine genauere Charakterisierung von XAM-Komponenten erarbeite ich im Folgenden.

Allgemein ist die Komponentenorientierung bei der Software-Entwicklung ein Ansatz, bei dem Funktionalität in jeder Granularität, von der kompletten Anwendung bis hin zu kleinen Teilen einer Anwendung, und auf jedem Abstraktionsniveau, von der Modellierung der Anwendungsdomäne, des Entwurfs der Architektur, der Schnittstellenspezifikation bis hin zur Entwicklung des ausführbaren Codes, durch das Zusammenfügen und Anpassen von Komponenten gebildet werden kann (siehe Kapitel 10.1.1 in [D'SOUZA und WILLS 1999]). Sie bietet somit einen modularen Ansatz zur Erstellung von Software. Eine genauere Betrachtung der Beziehung zwischen Modularisierung und Komponentenorientierung wird in Abschnitt 6.2.4 vorgenommen. Eine Komponente bildet also allgemein eine Einheit, die Funktionalität definiert und die eingesetzt wird, um größer Einheiten von Funktionalität zu bilden. Da es sich bei dieser Arbeit um eine konzeptionelle Arbeit handelt, wird die Bildung von Funktionseinheiten durch XAM-Komponenten auf der Ebene des Entwurfes betrachtet.

Um einen Komponentenbegriff zu formulieren, stellt sich die Frage, welche Funktionseinheiten geeignet mit einer Komponente definiert werden. Als Richtlinie zur Beantwortung dieser Frage für die XAM-Komponenten dient mir die Charakterisierung einer Komponente von Jed Harris, der in [ORFALI et al. 1996], Seite 34, und [GRIFFEL 1998], Seite 31, wie folgt zitiert wird:

> *Eine Komponente ist ein Stück Software, das klein genug ist, um es in einem Stück erzeugen und pflegen zu können, das groß genug ist, um eine sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen sowie mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten.*

Für die XAM-Komponenten scheint es mir daher angemessen, dass sie Funktionseinheiten zusammenfassen, die die XAM zur Darstellung von einzelnen multimedialen Elementen benötigt, die in der Beschreibung einer ξML-Präsentation verwendet werden. Diese Unterteilung gruppiert somit einzelne abstrakte Elemente und ihre Implementierungen gemäß der Abstraktionen von Leistungsmerkmalen von Multimediapräsentationen, für die sie für die XAM Funktionalität bereitstellen. Diese Unterteilung ist somit orthogonal zu der in dem Lösungsansatz vorgestellte Unterteilung in abstrakte Elemente und ihre Implementierungen. Die im Lösungsansatz vorgestellte Unterteilung ist sinnvoll, um die dort vorgestellten Mechanismen, die die Erweiterbarkeit der XAM ermöglichen, geeignet darzustellen. Mit der Einteilung der für die Darstellung der Abstraktionen, also der multimedialen Elemente der Beschreibung von ξML-Präsentationen, benö-

tigten Funktionalität in XAM-Komponenten wird dagegen eine Unterteilung für einen geeigneten Entwurf unternommen, der den Fokus auf die spätere Benutzung der XAM legt. In Projekten wird die Erweiterung der XAM auch entlang der einzelnen benötigten multimedialen Elemente erfolgen. Es erscheint mir daher sinnvoll, die zur Darstellung dieser Abstraktionen benötigten einzelnen Aspekte, die Funktionalität für das Einlesen, für die Repräsentation in der Zwischendarstellung und für das Erzeugen der Ausgaben, in einer XAM-Komponente zusammenzufassen.

Als nächstes stellt sich im Zusammenhang der Findung eines Komponentenbegriffes die Frage, woraus sich eine Komponente genauer zusammensetzt, um die mit ihr gekapselte Funktionalität bereitszustellen. In [D'SOUZA und WILLS 1999], Seite 386, wird eine Komponente recht allgemein wie folgt charakterisiert:

> [A component is] *a coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other components to build something larger.*

Eine Komponente besteht also aus Software-Artefakten, die unabhängig entwickelt und vertrieben werden können, um zusammen mit anderen Komponenten eine größere Funktionseinheit zu bilden. Bei den XAM-Komponenten bestehen diese Softwareartefakte zum einen aus Schnittstellen, um, wie bei Komponenten üblich, die Dienste einer Komponente anbieten und fremde Dienste in Anspruch nehmen zu können, und aus Implementierungen, die diese Dienste realisieren. Bei den XAM-Komponenten liefern diese Dienste die Funktionalität, die von der zur Darstellung von ξML-Präsentationen notwendigen Konfiguration der XAM bzw. der Konfiguration der entsprechenden Teile der XAM benötigt wird. Für ein mit der XAM darzustellendes multimediales Element liefert eine XAM-Komponente die benötigten Informationen beispielsweise in Form einer Menge von geeigneten Schnittstellen, abstrakten Datentypen und Implementierungen aus, um das Front-End für das Einlesen dieses Elements, die Zwischendarstellung zum Repräsentieren des Elements und das Back-End zum Erzeugen einer Ausgabe für dieses Element konfigurieren zu können. Wie diese Informationen genau bereitgestellt werden, wird in den einzelnen Entwurfkapiteln erörtert. Eine Dokumentation der von einer XAM-Komponente bereitgestellten Funktionalität ist damit gleichbedeutend mit der Festlegung der Bedeutung der Abstraktionen von Leistungsmerkmalen von Multimediapräsentationen. Sie erfolgt im Rahmen dieser Arbeit informell und wird ebenfalls in den entsprechenden Entwurfskapiteln vorgenommen.

Bei beiden Zitaten wird die Unabhängigkeit der Entwicklung und Auslieferung einer Komponente betont. Für die XAM-Komponenten scheint dies nicht gewährleistet zu sein, da die abstrakten Elemente und ihre Implementierungen als Spezialisierungen der Basisprimitive des abstrakten Modells vorgestellt werden und somit eine klare Abhängigkeit zu ihnen haben. Die mit den Basisprimitiven definierte Funktionalität stellt aber Systemfunktionalität bereit und muss daher nicht zusätzlich mit den XAM-Komponenten distributiert werden. Eine Komponente hat immer eine Abhängigkeit zu dem System, für das die Komponente entwickelt wird. Nach dem Komponentenmodell von Java entwickelte Komponenten, die JavaBeans, können auch nur mit der Laufzeitumgebung von Java verwendet werden. Die Basisprimitive der XAM nehmen in dem Vergleich etwa die Rolle der Standardbibliotheken von Java ein, auf denen aufbauend die JavaBeans entwickelt werden. So wie JavaBeans nur im Zusammenhang mit der Laufzeitumgebung von Java verwendet werden können, können XAM-Komponenten nur im Rahmen der

XAM verwendet werden. Dies schränkt ihre Unabhängigkeit und Wiederverwendung im Rahmen einzelner mit dem ξML-System zu realisierende Projekte aber nicht ein.

Durch die Realisierung der Funktionalität für die Anpassbarkeit der XAM als XAM-Komponenten werden die in der Software-Technik und speziell beim objektorientierten Entwurf bewährten Richtlinien und Konzepte der *Programmierung gegen Schnittstellen und nicht gegen Implementierungen* [GAMMA et al. 1996], des *Information Hiding* (siehe Kapitel 4.2.2 in [GHEZZI et al. 1991]) und der *Annahme zur Änderungsnotwendigkeit* (siehe Kapitel 3.5 in [GHEZZI et al. 1991]), die dem Gesamtentwurf der XAM zu Grunde liegen, berücksichtigt.

Wenn in den folgenden Kapiteln der Begriff XAM-Komponente verwendet wird, liegt ihm die hier gegebene Charakterisierung zu Grunde. Aus Gründen des Lesbarkeit wird dies im Folgenden nicht mehr explizit angeführt.

6 Systemarchitektur

In diesem Kapitel stelle ich die Systemarchitektur der XAM vor. Zunächst fasse ich kurz die für die Systemarchitektur relevanten Anforderungen zusammen, bevor ich auf die Konzepte eingehe, mit denen die Anforderungen erfüllt werden sollen. Das Kapitel schließt mit einer Beschreibung der konkreten Architektur: an Hand der einzelnen Elemente der Systemarchitektur erläutere ich die statischen Strukturen der XAM, ehe ich durch eine exemplarische Anwendung der XAM das Zusammenspiel der einzelnen Elemente veranschauliche und somit die dynamischen Strukturen der Architektur darstelle.

6.1 Anforderungen an die Systemarchitektur

In dieser Arbeit verwende ich den in [FOEGEN und BATTENFELD 2001] vorgestellten Architekturbegriff, bei dem unter der Architektur die Struktur einer Anwendung verstanden wird, wobei die Struktur sowohl die statischen als auch die dynamischen Aspekte des Systems betreffen. Unter Verwendung der allgemeinen Definition des Architekturbegriffs aus der Encyclopædia Britannica [ENCYCLOPAEDIA BRITANNICA 1978] werden dort drei wesentliche Ziele einer Software-Architektur vorgestellt: die Architektur soll sicherstellen, dass das spätere System die Anforderungen erfüllt (utilitas), robust gegenüber Änderungen ist (firmitas) und eine gewisse Schönheit besitzt (venustas), sprich klar und verständlich strukturiert ist.

Diese Ziele sind die Richtlinien, die ich mir beim Entwurf der Systemarchitektur der XAM gesetzt habe. Die Architektur der XAM muss es ermöglichen, ein System zu entwickeln, das die in Abschnitt 5.2 vorgestellten Anforderungen an die XAM erfüllt, zukünftige Änderungen einkalkuliert und durch einen modularen bzw. komponentenorientierten Ansatz die von ihm geforderte Funktionalität übersichtlich strukturiert und für die Leser der Arbeit verständlich bereitstellt.

Für den Entwurf der Architektur sind die drei generellen konzeptionellen Anforderungen aus Abschnitt 5.2.1 wesentlich: die Architektur muss es ermöglichen, ξML -Präsentationen darzustellen (siehe Anforderung 1 auf Seite 38), die XAM muss nach der Entwicklungszeit mit vertretbarem Aufwand erweiterbar sein (sihe Anforderung 2) und sie soll mittels eines komponentenorientierten Ansatzes realisiert werden (siehe Anforderung 3). Welche Konzepte ich zur Erfüllung dieser Anforderungen einsetze und zu welcher Architektur das im einzelnen führt, werde ich in den folgenden Abschnitten des Kapitels erläutern.

6.2 Verwendete Konzepte

Um die oben genannten Anforderungen an die XAM zu erfüllen, bediene ich mich beim Entwurf der XAM mehrerer Konzepte, die sich unmittelbar in der Architektur der XAM niederschlagen.

Um den in Abschnitt 4.2.1 als Lösungsansatz vorgestellten Generatoransatz zu realisieren, verwende ich naheliegenderweise Konzepte aus dem Compilerbau ([AHO et al. 1988a] bzw. [AHO et al. 1988b] und [WAITE und GOOS 1985]). Welche das sind und wie sie sich auf die Architektur auswirken, stelle ich in Abschnitt 6.2.1 vor.

Als ein Entwurfsmuster der XAM-Architektur, das mir die in Abschnitt 4.2.2 vorgestellte Erweiterbarkeit des Generatoransatzes ermöglicht, adaptiere ich Elemente des Architekturmusters Microkernel [BUSCHMANN et al. 1996], welches generell für die Entwicklung erweiterbarer Architekturen gedacht ist. Die relevanten Aspekte des Musters und wie ich sie für die XAM anwende, erläutere ich in Abschnitt 6.2.2 auf der nächsten Seite.

Elemente des Architekturmusters Reflection [BUSCHMANN et al. 1996] finden sich in den Überlegungen zu dem abstrakten Modell wieder, das in Abschnitt 4.2.3 als weiterer Lösungsansatz vorgestellten wird. Das Muster ist ebenfalls in dem Kontext erweiterbarer Software-Systeme angesiedelt. Welche Elemente sich wiederfinden lassen und wie sie im Rahmen des Entwurfs der XAM eingesetzt werden, erläutere ich in Abschnitt 6.2.3.

Als generelles Modularisierungskonzept wird in dieser Arbeit, entsprechend dem ξML -System, die Komponentenorientierung verwendet. In Abschnitt 6.2.4 stelle ich kurz wesentliche Aspekte vor.

Die hier vorgestellten Konzepte sollen es mir ermöglichen, die in Abschnitt 6.1 auf der vorherigen Seite vorgestellten Anforderungen an die Systemarchitektur zu erfüllen. Mit den Ausführungen zu den Konzepten positioniere ich die in Abschnitt 4.2 auf Seite 28 vorgestellten generellen Lösungsansätze des Entwurfs der XAM in der Literatur und erläutere, wie sie sich auf die Architektur der XAM auswirken.

6.2.1 Compilerbau

Wie in Anforderung 1 auf Seite 38 verlangt, soll die XAM bei der Darstellung von ξML-Präsentationen auf etablierte Multimediaformate und deren Player zurückgreifen und nicht selbst als Player entwickelt werden. Daher wird beim Entwurf der XAM der in Abschnitt 4.2.1 als Lösungsweg vorgestellte Generatoransatz verfolgt. Die darstellungsneutrale Beschreibung einer ξML-Präsentation wird in Form eines Eingabedokuments der XAM bereitgestellt und von ihr in entsprechende Multimediaformate überführt, die von geeigneten Playern angezeigt werden können. Diese Aufgabe des Überführens eines Eingabeformats in ein Ausgabeformat hat die XAM generell mit einem Compiler gemeinsam. Für diese Aufgabe mache ich mir daher einige beim Compilerbau erprobte Konzepte zunutze, die ich im folgenden Abschnitt vorstelle. An dieser Stelle sei darauf hingewiesen, das sich die XAM jedoch unter anderem durch den Aspekt der Erweiterbarkeit von einem gewöhnlichen Compiler unterscheidet, wie in Abschnitt 4.2.2 auf Seite 29 geschildert wird. Daher nenne ich die XAM einen Generator, um sie auch sprachlich von gewöhnlichen Compilern abzuheben.

Im Compilerbau ist es üblich, nach einem Analyse-Synthese-Modell (siehe Kapitel 1 in [AHO et al. 1988a]) vorzugehen. Danach setzt sich der Übersetzungsprozess aus zwei Teilen zusammen, der Analyse und der Synthese, wie auch in [WAITE und GOOS 1985], Seite 3, festgestellt wird. Bei der Analyse wird das Quellprogramm eingelesen und eine Zwischendarstellung der Eingabe erzeugt. Im Synthese-Teil wird aus der Zwischendarstellung das gewünschte Zielprogramm konstruiert. Die Teile eines Compilers, die sich mit der Analyse beschäftigen, werden

unter dem Begriff *Front-End* zusammengefasst, die für die Synthese zuständige Teile nennt man *Back-End* (siehe Seite 24 in [AHO et al. 1988a]). Charakteristisch für die Unterteilung eines Compilers in Front-End, Zwischendarstellung und Back-End ist, dass das Front-End meist nur von der Quellsprache abhängt und von der Zielmaschine unabhängig sein soll, und entsprechend die Teile des Back-Ends sich nur auf die Zielmaschine beziehen und von der Zwischendarstellung und nicht von der Quellsprache abhängen sollen. Durch die maschinenunabhängige Zwischendarstellung wird die Eingabe von der zu erzeugenden Ausgabe entkoppelt. Beim Compilerbau macht man sich dieses Konzept zu nutze, um mit einem möglichst geringen und auf das Back-End begrenzten Änderungsaufwand einen Compiler für die Unterstützung neuer Zielmaschinen anzupassen. Unüblich ist hingegen, mit der Entkoppelung eine leichte Anpassung des Compilers im Front-End für die Unterstützung neuer Eingabeformate, die für die gleiche Zielmaschine übersetzt werden sollen, zu erreichen (siehe Seite 24f in [AHO et al. 1988a]).

Dieses Konzept der Entkoppelung der Eingabe von der Ausgabe wird auch bei der XAM verwendet, da eine solche Unterteilung die Umsetzung der Forderung nach der Erweiterbarkeit sowohl in der Eingabe als auch in der Ausgabe (siehe Anforderung 2) unterstützt. Bei der XAM lassen sich daher, wie bereits in Abschnitt 4.2.4 skizziert, die oben vorgestellten drei Teile wiederfinden. In dem Front-End-Teil der XAM ist ein Parser angesiedelt, der die in einem Eingabedokument gespeicherte darstellungsneutrale Beschreibung einer ξML-Präsentation einliest und in die Zwischendarstellung überführt. In dem Back-End der XAM befinden sich die Werkzeuge, die aus der Zwischendarstellung die Ausgaben in den jeweils gewünschten Multimediaformaten erzeugen. Der generelle Aufbau des Generators XAM ist dem eines Compilers also durchaus ähnlich.

In wie fern weitere Konzepte des Compilerbaus beim Entwurf der einzelnen Teile der XAM angewendet werden, diskutiere ich in den entsprechenden Entwurfskapiteln. Dort werden auch Fragestellungen, ob sich einzelne, aus dem Compilerbau bekannte Phasen des Übersetzungsprozesses bei der XAM wiederfinden lassen oder an welchen Stellen sich die XAM, neben der Erweiterbarkeit, von einem Compiler unterscheidet, erörtert.

6.2.2 Architekturmuster Microkernel

Neben dem Generatoransatz, der zu der in Abschnitt 6.2.1 vorgestellten Strukturierung der Architektur führt, ist ein wesentliches Aspekt der XAM, wie in Anforderung 2 auf Seite 38 formuliert, die Erweiterbarkeit. Die XAM muss, um die einzelnen ξML-Präsentationen darstellen zu können, anpassbar sein. In dem Kontext der anpassbaren Systeme ist das Architekturmuster Microkernel [BUSCHMANN et al. 1996] angesiedelt, so dass ich es für den Entwurf der XAM betrachte. Das Muster ist für Software-Systeme gedacht, die sich an ändernde Systemanforderungen anpassen müssen. Es findet häufig Anwendung bei der Entwicklung von Betriebssystemen. Ich stelle die für die XAM relevanten Aspekte des Musters im Folgenden kurz vor, ehe ich sie auf die XAM übertrage.

Bei dem Microkernel-Muster wird die von dem zu erstellenden System angebotene Funktionalität klar untergliedert. Ein minimaler Funktionalitätskern wird von erweiterter Funktionalität und spezifischen Anforderungen an das System getrennt. Der Kern stellt rudimentäre Dienste bereit, so genannte *Mechanismen*, die als Basis für die Erstellung von erweiterter Funktionalität genutzt werden, und koordiniert die Zusammenarbeit dieser Erweiterungen. Das Muster führt

noch eine feingranularere Struktur von internen und externen Diensten, Adaptern und Klienten ein und legt Verantwortlichkeiten und Kommunikationsmöglichkeiten für diese einzelnen Komponenten fest. Diese Struktur wird im Muster eingesetzt, um unter anderem möglichst ressourcenschonend, also etwa die Prozessor- und Arbeitsspeicherauslastung niedrig haltend, das System konzipieren zu können. Da die XAM sich nicht im Betriebssystemkontext bewegt und eine so vielschichtige Struktur für den Entwurf der XAM nicht nötig ist, sind diese Elemente im Rahmen der vorliegenden Arbeit nicht relevant und werden hier nicht weiter vorgestellt.

Die Grundidee des Musters, dass ein System mit einem minimalen Funktionalitätskern entworfen wird, der Basisfunktionalität bereitstellt und bei dem sich erweiterte Funktionalität registriert, wende ich auch bei dem Entwurf der XAM an. Neben einem Mechanismus, der das Registrieren der erweiterten Funktionalität ermöglicht, besteht die Basisfunktionalität, die ich für die XAM definiere, aus der Infrastruktur, die nötig ist, um prinzipiell einen Übersetzungsprozess durchführen zu können. Es werden die nötigen Strukturen in dem Front-End, der Zwischendarstellung und dem Back-End bereitgestellt, um prinzipiell eine Eingabe einlesen, eine Zwischendarstellung aufbauen und daraus Ausgaben erzeugen zu können. Aus welchen Elementen die jeweiligen Formate der Strukturen dieser drei Teile bestehen, wird als erweiterte Funktionalität definiert, die sich bei dem Funktionalitätskern registriert, um von der XAM beim Übersetzungsvorgang genutzt zu werden. Durch das Anmelden der erweiterten Funktionalität wird die XAM so konfiguriert, dass sie als Eingabe die Beschreibung einer konkreten ξML-Präsentation verarbeiten und die Ausgaben in den gewünschten Formaten erzeugen kann. Diese erweiterte Funktionalität wird bei der XAM mit den abstrakten Elementen und ihren Implementierungen, also in Form der XAM-Komponenten, definiert. Sie stellen, wie in Abschnitt 4.2 vorgestellt, die Funktionalität bereit, die zum Darstellen der in den Beschreibungen der ξML-Präsentationen verwendeten multimedialen Ausdrücke benötigt wird.

Konkret bedeutet dies, dass die XAM beim Start nur minimale Basisfunktionalität besitzt. Diese Basisfunktionalität der XAM besteht, wie oben geschildert, unter anderem aus einem Initialisierungsmechanismus, der es ermöglicht, die erweiterte Funktionalität, sprich die XAM-Komponenten, bei den zum Erzeugen von darstellbaren Formen aus der darstellungsneutralen Beschreibung einer ξML-Präsentationen benötigten Teilen der XAM, dem Front-End, der Zwischendarstellung und dem Back-End, zu registrieren, um sie so für die Verarbeitung der konkret vorliegenden Eingabe zu konfigurieren. Somit kommt als vierter wesentlicher Teil der Architektur der XAM, neben dem Front-End, der Zwischendarstellung und dem Back-End, eine Schnittstelle mit Initialisierungsmechanismus hinzu.

Ein Vorteil der oben vorgestellten Anwendung der Microkernel-Musters bei der XAM ist, dass alle zum Darstellen einer konkreten Beschreibung einer ξML-Präsentation benötigte (erweiterte) Funktionalität gleichförmig behandelt wird. Durch diesen Mechanismus wird nicht unterschieden, wann die erweiterte Funktionalität, also die XAM-Komponenten, definiert werden. Es macht keinen Unterschied, ob die XAM-Komponenten im Rahmen dieser Diplomarbeit entwickelt werden, oder ob sie erst bei einer späteren Nutzung des ξML-System hinzukommen, um für konkrete Projekte benötigte multimediale Ausdrucksstärke bereitzustellen. Als weiterer Vorteil erlaubt mir dieser Ansatz, mich im Rahmen meiner Diplomarbeit bei der Definition der XAM-Komponenten auf die Darstellung einer möglichst einfach gehaltenen ξML-Präsentation, der giMLi-Präsentation, zu beschränken, ohne dass die generellen Fähigkeiten der XAM darunter leiden.

6.2.3 Architekturmuster Reflection

Ein Aspekt der Erweiterbarkeit der XAM ist, dass sie für die Darstellung der ξML-Präsentationen an die von ihnen verwendeten multimedialen Elemente anpassbar sein soll, ohne dass die XAM dafür in ihrer Struktur verändert und neu übersetzt werden muss. In diesem Zusammenhang der dynamischen Erweiterbarkeit existiert ein weiteres Architekturmuster aus dem Kontext der anpassbaren Systeme, das Reflection-Muster [BUSCHMANN et al. 1996]. Es ist für die Entwicklung von Systemen gedacht, die ihre eigene Adaptierbarkeit von vornherein unterstützen sollen. Welche Aspekte des Musters für den Entwurf der XAM interessant sind und wie sie angewendet werden, stelle ich im Folgenden kurz vor.

Das Reflection-Muster stellt einen Mechanismus bereit, mit dem die Struktur und das Verhalten von Software-Systemen dynamisch verändert werden kann. Ein wesentliches Ziel, das mit diesem Muster verfolgt wird, ist die Anpassung der Software möglichst übersichtlich zu gestalten und die bei der Anpassung auftretende Komplexität zu reduzieren. Dazu führt es zwei Ebenen ein: Es wird eine *Metaebene* definiert, die ausgewählte Systemeigenschaften zur Verfügung stellt und dadurch eine Software ihrer selbst bewusst macht. In einer zweiten Ebenen, der so genannten *Basisebene* wird die Anwendungslogik des Systems definiert. In der Metaebene werden Objekte, so genannte *Metaobjekte*, definiert, die Systemaspekte, die Gegenstand von Änderungen sein können, kapseln. Implementierungen der Anwendungslogik, also Elemente der Basisebene, benutzen diese Metaobjekte, um von eventuell ändernden Systemaspekten unabhängig zu bleiben. Zugriffe auf solche Funktionalität erfolgt nur über die Metaobjekte. Änderungen der Systemeigenschaften bei den Metaobjekten sind durch ein eigens definiertes Zugriffsprotokoll möglich. Auf diesen Aspekt gehe ich aber nicht näher ein, da er aus Gründen, die ich in nächsten Abschnitt vorstelle, für den Entwurf der XAM nicht relevant ist.

Die Idee, durch eine Kapselung von relevanten Aspekten der Systemfunktionalität und somit durch einen zweischichtigen Zugriff auf diese Funktionalitäten ein System leichter anpassbar zu gestalten, wende ich auch bei der XAM an. Bei einer Übertragung des Musters auf die XAM entspricht das abstrakte Modell, das die generelle Systemfunktionalität bereitstellt, der Metaebene. Die Basisprimitive des abstrakten Modells sind also, im Sprachgebrauch des Musters, die Metaobjekte. Die Basisebene wird von den XAM-Komponenten, also den abstrakten Elementen und ihren Implementierungen gebildet, die die Anwendungslogik definieren, also im Rahmen der XAM die Funktionalität bereitstellen, die nötig ist, um eine konkrete ξML-Präsentation darstellen zu können.

Ein wesentlicher Unterschied zu der ursprünglichen Anwendung des Musters ist jedoch, dass bei dem Muster die Metaobjekte, im Rahmen des Musters durch ein eigenes Protokoll, explizit verändert werden können. Selbst Veränderungen grundlegender Systemmechanismen können auf diese Weise vor der eigentlichen Anwendungslogik verborgen werden. Dies ist bei der XAM nicht nötig. Der durch die Basisprimitive definierte Rahmen ist gerade so ausgelegt, dass die für die Darstellung der textzentrierten ξML-Präsentationen benötigten generellen Mechanismen bereitgestellt werden. Änderungen dieser grundlegenden Strukturen sind, zumindest im Rahmen dieser Arbeit, durch die explizite Festlegung auf die Unterstützung von textzentrierten Multimediapräsentationen nicht vorgesehen. Änderungen finden somit nicht auf der Metaebene statt. Erweiterungen sind durch das Entwickeln neuer XAM-Komponenten auf der Basisebene angesiedelt, um die XAM an die jeweils darzustellende Präsentation anpassen zu können.

Die Kapselung von Systemfunktionalitäten in einer Metaebene, die noch unabhängig von der für die Darstellung konkreter ξML-Präsentationen benötigten Anwendungslogik ist, erlaubt es mir, die Implementierung von rudimentärer Systemfunktionalität vor Anwendungslogik zu verbergen und gezielt Aspekte anbieten zu können, die von den XAM-Komponenten erweitert werden sollen. Dadurch wird die Komplexität der Anpassung der XAM an die jeweils darzustellenden ξML-Präsentationen deutlich reduziert.

6.2.4 Modularisierung und Komponentenorientierung

Wie in Abschnitt 6.1 auf Seite 53 dargestellt, soll die Architektur eines Software-Systems übersichtlich strukturiert sein. Um dies bei der Architektur der XAM zu erreichen, bediene ich mich, analog zu dem Gesamtansatz des ξML-Systems (siehe Abschnitt 3.2), des software-technologischen Prinzips der *Modularisierung*, das, wie in [GHEZZI et al. 1991], Kapitel 3.3, erläutert, komplexe Systeme in einfachere Teile, so genannte *Module*, unterteilt. Mit der Modularisierung sollen drei Ziele erreicht werden: Zum einen soll sie es ermöglichen, ein komplexes System dekomponieren zu können. Dies entspricht einem Top-Down-Ansatz, bei dem man große Probleme auf kleinere Teilprobleme zurückführt. Aus der entgegengesetzten Betrachtungsweise, also einem Bottom-Up-Ansatz folgend, soll außerdem mit der Modularisierung ermöglicht werden, ein Gesamtsystem aus bestehenden Modulen komponieren zu können. Und als drittes und generelles Ziel der Modularisierung soll mit ihm das Verständnis des Gesamtsystems durch ein schrittweises Verständnis der einzelnen Teilsysteme ermöglicht werden. Um diese drei Ziel erreichen zu können, wird in [GHEZZI et al. 1991], Seite 51, gefordert, das Module einen großen Zusammenhalt im Innern (*hohe Kohäsion*), aber geringe Abhängigkeiten untereinander (*lose Koppelung*) besitzen. Dazu wird das Prinzip der Trennung von Schnittstelle und Implementierung verwendet. Schnittstellen definieren, hier im Kontext von Modulen, welche Funktionalität ein Modul anbietet und welche Informationen von einem Modul benötigt werden, um diese Funktionalität anbieten zu können. Sie definieren sozusagen Verträge zwischen den Modulen, die von ihnen eingehalten werden müssen, um ein korrekt arbeitendes System zu ermöglichen. Dieser Gedanke wird bei auch dem Entwurfprinzip des Design by Contract [MEYER 1992] sehr klar herausgestellt. Der Zugriff auf die angebotene Funktionalität erfolgt nur über die Schnittstellen, nie über die Implementierung der Funktionalität selbst. Schnittstellen beschreiben somit eine Sicht auf das System, die die Abhängigkeiten zwischen einzelnen Modulen veranschaulicht. Mit den Implementierungen wird dagegen eine Sicht auf die Interna eines Moduls eingenommen. Da sich die Abhängigkeiten zwischen den einzelnen Teilen, wie oben geschildert, auf die Verwendung der wohldefinierten Schnittstellen reduzieren und diese losgelöst von Implementierungen sind, wird eine lose Koppelung zwischen den Modulen erreicht.

Das Konzept der Komponentenorientierung wird in [GHEZZI et al. 1991], Kapitel 4.2, als eine Technik zur Modularisierung vorgestellt. Es ist noch losgelöst von der Komponentenorientierung, wie sie im Rahmen der Objekttechnologie eingesetzt wird und sehr allgemein gehalten. Die prinzipiellen Überlegungen, die dort angestellt werden und die ich hier betrachte, um die Stellung von Komponenten in einer Architektur einzuordnen, sind aber meines Erachtens mit den Konzepten der Komponentenorientierung im objektorientierten Kontext zu vereinbaren. Eine Komponente und ein Modul unterscheiden sich dabei in dem Abstraktionsniveau der Sichten, die sie auf ein System anbieten. Ein Modul setzt sich, gemäß des Bottom-Up-Ansatzes,

aus mehreren Komponenten zusammen. Dieses Zusammensetzen eines Module aus einzelnen Komponenten kann auf konzeptioneller Ebene als die Implementierung eines Moduls angesehen werden (siehe Kapitel 4.2.2 in [GHEZZI et al. 1991]). Mit Modulen wird somit eine abstraktere Sichtweise auf ein System als mit Komponenten eingenommen. Komponenten wiederum bieten innerhalb von Modulen eine abstrakte Sicht auf die bereitzustellende Funktionalität. Auch auf dem Abstraktionsniveau der Komponenten gilt wieder die Richtlinie der losen Koppelung zwischen Komponenten und des starken Kohäsion innerhalb einer Komponente und damit verbunden das Prinzip der Trennung von Schnittstelle und Implementierung. Diese Charakterisierung kennzeichnet den Umgang mit Modulen und Komponenten in meiner Arbeit. Für die Implementierung eines komponentenorientierten Ansatzes existieren verschiedene Komponentenmodelle, wie etwa Corba [BEN-NATAN 1995], JavaBeans [ENGLANDER 1997] Enterprise Java-Beans [MONSON-HAEFEL 2001] oder COM [BOX 1998], die unter Ausrichtung auf unterschiedliche Aspekte wie Verteilung oder Sprachparadigmen, die zur Realisierung verwendetet werden, Standards definieren, nach denen Komponenten entwickelt und ausgeliefert werden. Eine breitere Erörterung der Komponentenorientierung im Rahmen der Literatur wäre sicher angemessen. Da die Betrachtung der Anwendung von Komponentenorientierung speziell im Rahmen der Erstellung von Multimediapräsentationen nicht das zentrale Thema dieser Arbeit ist, unterbleibt sie aber aus Umfangsgründen. Für eine detaillierte Diskussion dieser Aspekte im Kontext des ξML-Systems verweise ich auf die Diplomarbeit von Sebastian Schütte [SCHÜTTE 2002], in der ein Kompositionskonzept für Komponenten im ξML-System entwickelt wird. Einen für meine Arbeit geltenden Komponentenbegriff, den der so genannten XAM-Komponente, habe ich in Abschnitt 5.3 auf Seite 50 vorgestellt.

Die Untergliederung der Gesamtarchitektur der XAM in die vier Teile Front-End, Zwischendarstellung, Back-End und Schnittstelle mit Initialisierungsmechanismus, entspricht dem oben vorgestellten Modularisierungsprinzip. Die Komplexität des Gesamtsystems der XAM wird in diese vier Teile, aus den in den Abschnitten 6.2.1 und 6.2.2 vorgestellten Gründen, dekomponiert. Eine solche Herangehensweise folgt dem oben vorgestellten Top-Down-Ansatz bei der Modularisierung. Auf eine explizite Formulierung dieser Teile in Form von Modulen mit Schnittstellen usw. wird verzichtet, da das Konzept der Modularisierung nur zur groben Strukturierung der Architektur dient, eine Beschreibung aber auf der Ebene der einzelnen Elemente dieser Teile vorgenommen wird.

Mit der Komponentenorientierung wird der Fokus hingegen auf die Bottom-up-Sicht bei der Modularisierung gelegt. Einzelne Teilfunktionalitäten werden in Komponenten definiert, die zusammengenommen Aufgaben eines Moduls bereitstellen. Explizit angewendet wird die Komponentenorientierung auf die Teile der XAM, die die Anpassbarkeit an konkret darzustellende ξML-Präsentationen betreffen: das sind nach dem Lösungsansatz aus Abschnitt 4.2.3 auf Seite 30 die sich auf das abstrakte Modell abstützenden abstrakten Elemente und ihre Implementierungen. Nach welchen Richtlinien hierbei diese Teilfunktionalitäten abgegrenzt und Komponenten gebildet werden, habe ich in Abschnitt 5.3 auf Seite 50 mit der Einführung des Begriffs der XAM-Komponente vorgestellt.

Wie sich die Anwendung der in Abschnitt 6.2 vorgestellten Konzepte auf die Architektur der XAM konkret niederschlagen, erörtere ich in den folgenden beiden Abschnitten.

6.3 Elemente der Systemarchitektur

Wie in den vorherigen Abschnitten dieses Kapitels beschrieben und in Abbildung 6.1 zu sehen ist, besteht die statische Struktur der Systemarchitektur der XAM aus vier wesentlichen Einheiten: die drei zum eigentlichen Generator gehörenden Bereiche Front-End, Zwischendarstellung und Back-End und die Schnittstelle mit dem Initialisierungsmechanismus, über den die drei übrigen Teile der XAM konfiguriert werden. Ihre Aufgaben im Kontext der Gesamtarchitektur und Möglichkeiten zur Realisierung dieser Aufgaben stelle ich in den folgenden Abschnitten vor.

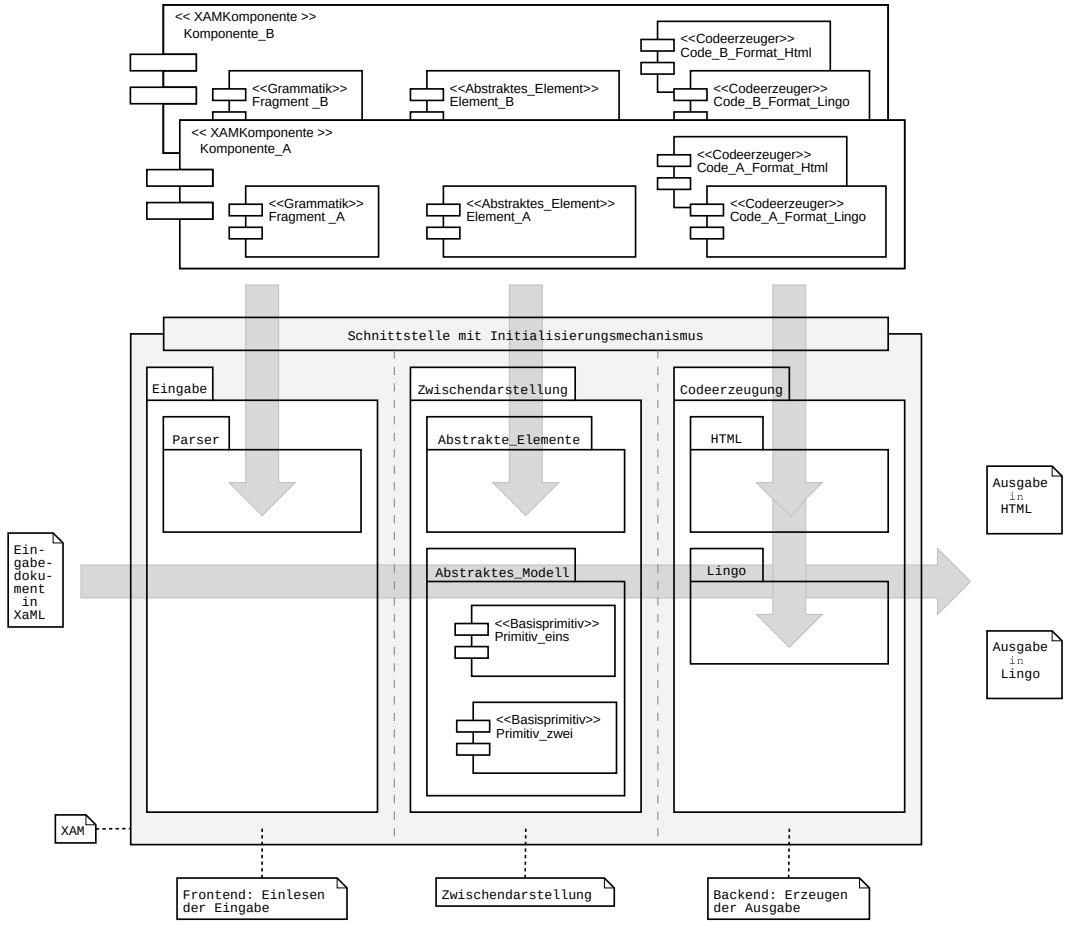


Abbildung 6.1: Die Architektur der XAM

6.3.1 Das Front-End

Wie bei einem Generator üblich, ist es Aufgabe des Front-Ends der XAM, die Eingabe zu verarbeiten. Die Eingabe ist die Beschreibung einer ξML-Präsentation mit den Ausdrucksmöglichkeiten der XAM. Sie liegt als Dokument vor, das in der im Rahmen dieser Arbeit zu formulierenden

Sprache e**X**tensible **a**bstract **M**achine **L**anguage (XaML) notiert ist (siehe Abschnitt 4.3). Die Eingabe wird von einem Parser eingelesen, der von ihr eine Zwischendarstellung im Hauptspeicher erzeugt (siehe Anforderung 14 auf Seite 44).

Eine Besonderheit im Vergleich zu üblichen Generatoren ist es, das die Eingabe und somit auch der Parser an die jeweils darzustellende ξML-Präsentation zur Laufzeit angepasst werden soll (siehe Anforderung 2 auf Seite 38 und Abschnitt 5.2.3). Die Eingabesprache XaML setzt sich daher für die einzelnen ξML-Präsentationen aus unterschiedlichen Elementen zusammen, mit denen die in den Präsentationen jeweils verwendete multimediale Funktionalität beschrieben werden kann. Für die unterschiedlichen ξML-Präsentationen liegen XaML somit unterschiedliche Grammatiken zu Grunde. Entsprechend muss der Parser mit der jeweiligen Grammatik konfiguriert werden können und Implementierungen bereitgestellt bekommen, mit denen er die Zwischendarstellung aufbauen kann.

Mit XML [RAY 2001] existiert eine Technologie, die sich zur Realisierung der Eingabesprache und des Parsers im Front-End anbietet. Sie besteht zum einen aus dem XML-Standard [BRAY et al. 2000], der eine einheitliche Notationsform für eine Familie von Auszeichnungssprachen (Markup-Sprachen) definiert. Zum Anderen existieren eine Reihe von Werkzeugen, die die Verarbeitung von in XML verfassten Dokumenten unterstützen: die Programmierschnittstellen DOM (Document Object Model, [APPARAO et al. 1998]) und SAX (Simple API for XML, [MEGGINSON 1998]) und Parser wie der Xerces [APACHE XML PROJECT 2000], der unter Verwendung dieser Programmierschnittstellen entwickelt wurde. Der Parser kann alle Dokumente, die der XML-Notation gehorchen (so genannte wohlgeformte Dokumente) verarbeiten und unter Verwendung einer Document Type Definition (DTD), mit der die Elemente und Struktur einer XML-Sprache festgelegt werden können, validieren, d.h. auf ihre grammatikalische Korrektheit überprüfen.

Daher bietet es sich an, die Eingabesprache XaML XML-basiert zu notieren und so die Vorteile der XML-Technologie zu nutzen. Mit dem DTD-Mechanismus können für die Beschreibung der einzelnen ξML-Präsentationen jeweils eigene Grammatiken definiert werden, um so einerseits XaML an die ξML-Präsentationen anpassen zu können und andererseits den XML-Parser für das Einlesen des spezifischen Eingabedokuments zu konfigurieren. Die DOM- und SAX-Programmierschnittstellen ermöglichen es, den Xerces-Parser an spezielle Bedürfnisse der XAM anzupassen. Da ein XML-Parser eine XML-Datei auch ohne die Existenz einer DTD verarbeiten kann, ist die Definition einer DTD für XaML nicht zwingend erforderlich. Die Verwendung einer DTD bringt aber einige Vorteile mit sich: Sie erlaubt es zum einen, die Grammatik einer Sprache standardisiert zu definieren, was die Beschreibung von XaML im Rahmen dieser Arbeit eindeutig präzisiert. Zum Anderen kann der Parser ein Dokument mittels einer DTD validieren, wodurch er, wie später detaillierter erläutert wird, für die Codeerzeugung benötigte Überprüfungen von Referenzen durchführen kann. Desweiteren wird eine Fehlersuche bei einer Realisierung der XAM durch einen validierenden Parser erheblich vereinfacht. Daher werde ich bei der Konzeption der XAM die Definition einer DTD für XaML vornehmen.

In dieser Arbeit wird die giMLi-Präsentation als beispielhafte ξML-Präsentation verwendet. Entsprechend werden für XaML die Sprachelemente herausgearbeitet, die zur Beschreibung der giMLi-Präsentation mit den Ausdrucksmöglichkeiten der XAM benötigt werden. Welche das im Einzelnen sind und wie XaML generell aufgebaut ist, stelle ich in Kapitel 9 vor.

6.3.2 Die Zwischendarstellung

Die Zwischendarstellung definiert die Struktur zwischen der Ein- und der Ausgabe eines Generators und entkoppelt sie voneinander. Wie in Abschnitt 4.2.3 vorgestellt, hat die Zwischendarstellung bei der XAM eine weitere Aufgabe, nämlich den Rahmen vorzugeben, in dem die Erweiterung der multimedialen Ausdrucksstärke der XAM möglich ist. Diese beiden Aufgaben führen zu einer zweischichtigen Struktur der Zwischendarstellung (siehe Abschnitt 6.2.3): das abstrakte Modell als fester Kern und die abstrakten Elemente als erweiterbare Spezialisierung dieses Modells, mit denen die Repräsentation einer als Eingabe vorliegenden ξML-Präsentation aufgebaut wird. Diese beiden Elemente sind in Abbildung 6.1 auf Seite 60 als UML-Pakete dargestellt.

Das abstrakte Modell bildet mit seinen Basisprimitiven ein Metamodell der mit der XAM darstellbaren Multimediapräsentationen, also ein Metamodell der ξML-Präsentationen. Die Basisprimitive modellieren Konzepte, die zur Beschreibung einer ξML-Präsentationen verwendet werden können und stellen eine Kapsel der Systemfunktionalität da, die die XAM zur Unterstützung dieser Konzepte bereitstellt. Die abstrakten Elemente spezialisieren diese Basisprimitive, um so eine konkrete ξML-Präsentation beschreiben zu können. Mit den Basisprimitiven wird somit die Klasse der ξML-Präsentationen modelliert. Mit den abstrakten Elementen hingegen kann die Repräsentation einer Instanz der Klasse der ξML-Präsentationen, beispielsweise die giMLi-Präsentation, in der Zwischendarstellung beschrieben werden.

Die Basisprimitive und damit das abstrakte Modell der XAM werden zur Entwicklungszeit festgelegt. Die abstrakten Elemente hingegen werden jeweils für die unterschiedlichen ξML-Präsentationen entwickelt und sind Teil des Erweiterbarkeitsbegriffs der XAM. Zur Darstellung einer ξML-Präsentation muss die XAM mit den entsprechenden abstrakten Elementen konfiguriert werden. Die Möglichkeiten, in denen die Ausdrucksstärke der XAM erweitert werden kann, ist durch das abstrakte Modell vorgegeben. Wird in einer Multimediapräsentation Funktionalität verwendet, die sich nicht durch Spezialisierung eines Basisprimitives als abstraktes Element modellieren lässt, so kann diese Funktionalität nicht von der XAM unterstützt, sprich dargestellt werden, und die Multimediapräsentation, die diese Funktionalität verwendet, gehört nicht zu der Klasse der ξML-Präsentationen, für deren Darstellung die XAM konzipiert wird.

Da das gesamte ξML-System objektorientiert entworfen wird, bietet sich an, die Basisprimitive und abstrakten Elemente mit dem Konzept der Klasse zu modellieren, Beziehungen zwischen ihnen mit den in der Objektorientierung üblichen Konzepten wie etwa Assoziationen oder Vererbung. Eine Umsetzung einer solchen Modellierung ist mit einer objektorientierten Programmiersprache möglich. Als konkrete Sprache bietet sich Java [GOSLING et al. 2000] an, da sie neben den oben genannten Konzepten auch eine klare Trennung von Schnittstellen und Implementierungen bereitstellt, was eine saubere Umsetzung des Entwurfs unterstützt. Außerdem bietet ihre Infrastruktur einige Dienste an, die zur Realisierung der XAM gut geeignet sind. An den entsprechenden Stellen, wie etwa beim Initialisierungsmechanismus, werde ich näher darauf eingehen. Für die Realisierung der Basisprimitive sind abstrakte Java-Klassen, für die abstrakten Elemente konkrete Java-Klassen sinnvoll. Mit abstrakten Java-Klassen können die Schnittstellen der Basisprimitive definiert und ihre Systemfunktionalität implementiert werden, ohne dass ihre direkte Verwendung, sprich Instantiierung, möglich ist. Um sie verwenden zu können, müssen die abstrakten Java-Klassen, also die Basisprimitive, von konkreten Java-Klassen, also den abstrakten

Elementen, spezialisiert werden.

Aus welchen Basisprimitiven sich das abstrakte Modell zusammensetzt, stelle ich in Kapitel 7 vor. Die abstrakten Elemente, die zur Beschreibung der giMLi-Präsentation benötigt werden, erörtere ich in Kapitel 9.

6.3.3 Das Back-End

Wie in Abschnitt 6.2.1 beschrieben, findet im Back-End die Synthese des Übersetzungsprozesses statt: Aus der Zwischendarstellung wird das gewünschte Zielprogramm erzeugt. Bei der XAM bedeutet dies, dass die Zwischendarstellung in das Multimediaformat der gewünschten Zielplattform überführt wird.

Entgegen üblichen Generatoren wird die XAM für die Unterstützung mehrerer Zielplattformen konzipiert (siehe Anforderung 1 auf Seite 38). Dies hat Auswirkungen auf die Konzeption des Back-Ends. Allgemein gesprochen ist es Aufgabe der Codegenerierung, für die einzelnen Zielplattformen jeweils über die Datenstruktur der Zwischendarstellung zu iterieren und an den geeigneten Stellen den Code für die entsprechende Zielplattform zu erzeugen.

Auf den ersten Blick erscheint es wünschenswert, die von allen Zielplattformen benötigte Funktionalität des Iterierens über die Datenstruktur der Zwischendarstellung einmal generisch zu realisieren und nur die Erzeugung des Ausgabecode für jede Zielplattform seperat umzusetzen. Dazu müsste eine allen Zielplattformen gemeinsame Durchlaufstrategie entwickelt werden, was aber nicht möglich ist, da die Zielplattformen zu unterschiedliche Konzepte zur Darstellung von Multimediapräsentationen verwenden. Zum einen müssten Annahmen über alle zukünftig zu verwendende Zielplattformen gemacht werden, was zur Entwicklungszeit der XAM nicht möglich ist, zum anderen hat sich beim Altenberger Dom-Projekt am konkreten Fall der Zielformate HTML und LINGO keine einheitliche Durchlaufstrategie finden lassen. Selbst für die unterschiedlichen Varianten des Formats HTML ist es schwierig, eine gemeinsame Durchlaufstrategie zu definieren: Für die Codegenerierung bedeutet es etwa einen wesentlichen Unterschied, ob die HTML-Variante die Verwendung von Frames erlaubt oder nicht. Daher wird bei der XAM für jede Zielplattform ein eigenes Werkzeug benötigt, das die Zwischendarstellung als Eingabe und den Code der Zielplattform als Ausgabe hat und eine eigene Durchlaufstrategie definiert.

Wie die Elemente des Front-Ends und auch der Zwischendarstellung, müssen die Codeerzeugungswerkzeuge konfigurierbar sein, um an die von der jeweils darzustellenden ξML-Präsentation benötigten Ausdruckstärke angepasst werden zu können (siehe Anforderung 2 auf Seite 38). Da mit dem abstrakten Modell der generelle Rahmen der ξML-Präsentationen fest definiert ist, sollte die Durchlaufstrategie durch die Zwischendarstellung für eine Zielplattform weitesgehend unabhängig von den konkret verwendeten multimedialen Elementen sein, aus denen sich die darzustellende ξML-Präsentation zusammensetzt. Der Kontrollfluss bei der Codeerzeugung ist im Wesentlichen durch die Durchlaufstrategie festgelegt: Bei Erreichen der einzelnen Elemente der Zwischendarstellung wird jeweils die Erzeugung des entsprechenden Codefragments angestoßen. Daher bietet es sich an, die Werkzeuge jeweils als Frameworks zu konzipieren, die die einzelnen Durchlaufstrategien fest implementieren, und bei denen sich jeweils Codeerzeugungskomponenten registrieren, die für die einzelnen multimedialen Elemente die benötigte Abbildung aus der Zwischendarstellung in die Konzepte der Zielplattform vornehmen (siehe Anforderung 15 auf Seite 45), also den entsprechenden Code erzeugen.

Eine Realisierung der Codeerzeugungswerkzeuge kann mit Programmiersprache Java vorgenommen werden. Durch die klare Trennung von Schnittstellen und Implementierung in Java ist es möglich, die Durchlaufstrategie unter Verwendung von Schnittstellen zur Codegenerierung mittels Java-Interfaces zu realisieren, die dann jeweils von den Codeerzeugern für die einzelnen Elemente der Zwischendarstellung in Form von Java-Klassen implementiert werden.

In Abbildung 6.1 auf Seite 60 sind die einzelnen Werkzeuge als UML-Pakete dargestellt. Exemplarisch sind dort nur die Zielplattformen HTML und LINGO aufgeführt. Im Rahmen dieser Arbeit wird aus Umfangsgründen die Erzeugung einer Ausgabe in HTML betrachtet. Sie wird in Kapitel 10 näher vorgestellt.

6.3.4 Die Schnittstelle mit Initialisierungsmechanismus

Da die multimediale Ausdrucksstärke der XAM erweiterbar sein soll (siehe Anforderung 2 auf Seite 38), werden, wie oben beschrieben, die drei Einheiten der XAM, das Front-End, die Zwischendarstellung und das Back-End, konfigurierbar konzipiert. Beim Start wird die XAM so initialisiert (siehe Anforderung 16 auf Seite 45), das die einzelnen Einheiten für die Elemente der darzustellenden ξML-Präsentation konfiguriert sind.

Um die Konfiguration der XAM zu ermöglichen, ist zum einen die Definition einer Schnittstelle nötig, die die Möglichkeiten zur Erweiterung der XAM um die Unterstützung neuer multimedialer Ausdrücke anbietet. Zum anderen wird ein Initialisierungsmechanismus benötigt, der die XAM mit den über die Schnittstelle definierten Erweiterungen konfiguriert.

Wie oben beschrieben, wird zur Konfiguration der XAM jeweils für ein darzustellendes multimediales Element einer ξML-Präsentation Funktionalität für jeden der drei Teile der XAM benötigt: Für das Front-End ein Grammatikfragment, mit dem das multimediale Element in der Eingabesprache modelliert wird, ein abstraktes Element als Spezialisierung eines Basisprimitivs, das ein multimediales Element in der Zwischendarstellung der XAM repräsentiert, und für jede Zielplattform ein Codeerzeuger, der den entsprechenden Code in der Zielplattform für das multimediale Element der ξML-Präsentation erzeugt. Wie in Abschnitt 5.3 beschrieben und in Abbildung 6.1 auf Seite 60 dargestellt, wird diese Funktionalität für die Darstellung eines multimedialen Elements unter dem Begriff der XAM-Komponente zusammengefasst. Die XAM-Komponenten bilden somit die Erweiterungschnittstelle der XAM.

Durch den Initialisierungsmechanismus werden mit den einzelnen Bestandteilen einer XAM-Komponente jeweils die entsprechenden Teile der XAM für die Darstellung einer ξML-Präsentation konfiguriert. Dazu sind drei Elemente nötig: Zum einen muss es einen Dienst geben, der aus den einzelnen Grammatikfragmenten der XAM-Komponenten eine Grammatik der Eingabesprache XaML erzeugt, auf deren Grundlage das Parsen des Eingabedokuments erfolgt. Des Weiteren wird ein Dienst benötigt, der das Auffinden und Initialisieren der abstrakten Elemente der XAM-Komponenten mit den Daten des Eingabedokuments ermöglicht, um die Zwischendarstellung des Eingabedokuments zu erzeugen. Und als letztes muss ein Dienst existieren, der die Codeerzeuger der XAM-Komponenten bei den Generierungswerkzeugen der entsprechenden Zielplattformen initialisiert. Als Konzept zur Umsetzung dieser Dienste bietet sich das Entwurfsmuster Fabrikmethode (siehe Seite 131 in [GAMMA et al. 1996]) an, das eine Klassenschnittstelle zum Erzeugen von Objekten definiert, aber Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist. Auf die XAM angewendet, kann die Definition von Klassenschnitt-

stellen zum Erzeugen von Instanzen der abstrakten Elemente bzw. der Codeerzeuger verwendet werden, um so Stellen, an denen eine Initialisierung vorgenommen werden muss, zu bündeln und die dazu benötigte Funktionalität des Auffindens und Initialisieren der Elemente der XAM-Komponenten zu kapseln.

Eine Realisierung der XAM-Komponenten mit der Programmiersprache Java ist mit ihrem Komponentenmodell JavaBeans [ENGLANDER 1997] möglich. In einem JavaBean werden die zu einer Komponente gehörenden Klassen und Schnittstellen zusammengefasst und und in einer jar-Datei distributiert. Der Inhalt einer JavaBean-Komponente wird üblicherweise in einer so genannten Manifestdatei angegeben. Für die XAM-Komponenten habe ich eine XML-Notation zur Beschreibung ihres Inhalts in der Manifestdatei gewählt, derer Struktur ich durch eine entsprechende DTD beschreibe. An Hand eines Ausschnitts der DTD (siehe Grammatikfragment 6.1) lässt sich so der Lieferumfang einer XAM-Komponente präzise beschreiben: sie besteht aus einem DTD-Fragment für die Eingabesprache (`xamlfrag`), einer Java-Datei für die Zwischendarstellung (`abstractelement`) und aus den Java-Klassen der Codeerzeuger (`codegeneration`). Da in einer jar-Datei auch mehr als eine XAM-Komponente ausgeliefert werden kann, wird dem Element `xamcomponent` ein Name zugewiesen. Die vollständige DTD ist in Anhang A zu finden. Dort verwendete Elemente, die dem Leser unbekannt vorkommen, werden in späteren Kapiteln vorgestellt.

```
1 <!ELEMENT xamcomponent (xamlfrag, abstractelement, codegeneration) >
2   <!ATTLIST xamcomponent
3       name CDATA #REQUIRED >
```

Grammatikfragment 6.1: Bestandteile einer XAM-Komponente

Zur Realisierung der Initialisierungsmechanismen innerhalb der Fabriken bietet die Java-Umgebung mit den Klassenlader (Classloader), der das dynamische Nachladen von Klassen zur Laufzeit ermöglicht, und dem durch die Reflektions-API bereitgestellten Introspektionsmechanismus, der das Instantiieren und Analysieren von Klassen zur Laufzeit erlaubt, mächtige Mechanismen an, die das Konfigurieren der entsprechenden Teile der XAM mit den XAM-Komponenten ermöglichen.

Im Rahmen dieser Arbeit entwickele ich exemplarisch die zur Darstellung der giMLi-Präsentation benötigten XAM-Komponenten. Welche das im einzelnen sind, wird in Kapitel 9 vorgestellt.

6.4 Zusammenspiel der Elemente

Mit dem Zusammenspiel der im vorherigen Abschnitt vorgestellten Elemente bei der Darstellung einer ξML-Präsentation beschreibe ich die dynamischen Aspekte der Systemarchitektur. Wie in Abschnitt 4.2.4 kurz skizziert, läuft die Darstellung einer ξML-Präsentation in zwei Phasen ab: erst die Initialisierungsphase, dann die Ausführungsphase.

In der Initialisierungsphase wird die XAM durch entsprechende XAM-Komponenten mit der zur Darstellung der ξML-Präsentation benötigten Funktionalität initialisiert, in dem die einzelnen Teile der XAM mit den entsprechenden Elementen der XAM-Komponenten konfiguriert werden.

Dies ist in Abbildung 6.1 auf Seite 60 durch die drei senkrechten grauen Pfeile angedeutet: Aus den Grammatikfragmenten der XAM-Komponenten wird die Grammatik der Eingabsprache zusammengefügt und der Parser initialisiert. Bei dem Dienst, der dem Parser die abstrakten Elemente zur Verfügung stellt, aus denen er die Zwischendarstellung aufbaut, wird das abstrakte Element jeder XAM-Komponenten angemeldet. Die Codeerzeuger für die unterschiedlichen Zielplattformen der einzelnen XAM-Komponenten werden den Frameworks zur Codeerzeugung bekannt gemacht.

In einer zweiten Phase findet die eigentliche Darstellung der ξML-Präsentation statt: die mit den XAM-Komponenten initialisierte XAM überführt die in dem Eingabedokument beschriebene ξML-Präsentation in den Code der gewünschten Zielplattformen. Der Kontrollfluss hierbei ist in Abbildung 6.1 auf Seite 60 als waagerechter grauer Pfeil visualisiert. Zunächst liest der Parser das Eingabedokument ein und erzeugt daraus mit den abstrakten Elementen die Zwischendarstellung. Zur Codeerzeugung wird für die jeweiligen Zielplattformen mit einer in den entsprechenden Frameworks festgelegten Durchlaufstrategie über die Zwischendarstellung iteriert. Dabei wird mit Hilfe des Codeerzeugers für jedes durch ein abstrakte Element modellierte multimediale Element der ξML-Präsentation das Codefragment für die Zielplattform generiert.

Nach der allgemeinen Betrachtung der Abläufe innerhalb der XAM während der beiden Phasen, beschreibe ich nun, wie sie sich mit der oben zur Umsetzung der einzelnen Elemente der XAM vorgestellten XML-Technologie und der Programmiersprache Java realisieren lassen.

Die Initialisierung der XAM erfolgt mit den als JavaBeans im Dateisystem vorliegenden XAM-Komponenten. Es kann eine Komponente mit Hilfe der Ein-/Ausgabe-Programmierschnittstelle der Java-Standardbibliothek lokalisiert und unter Verwendung der in der Manifestdatei eines JavaBeans abgelegten Informationen auf ihre Elemente zugegriffen werden. Die Java-Klassen der abstrakten Elemente und der Codeerzeuger werden so bei den entsprechenden Fabriken bekannt gemacht. Aus den einzelnen DTD-Fragmenten der XAM-Komponenten wird die DTD der Eingabesprache zusammengebaut und mittels der DOM- bzw. SAX -Programmierschnittstellen dem Xerces-Parser zur Verfügung gestellt.

Der eigentliche Generierungsprozess beginnt mit dem Einlesen der XaML-Eingabedatei. Der Xerces-Parser übersetzt sie in eine lineare Aufrufreihenfolge von Methoden: Bei Erreichen und bei Verlassen eines Elements des Eingabedokuments während des Einlesevorgangs ruft der Parser definierte Methoden auf. Mittels der DOM- bzw. SAX-Programmierschnittstellen kann diese Abfolge von Methodenaufrufen verwendet werden, um jeweils durch Instantiieren und Zusammenfügen der entsprechenden Java-Klassen der abstrakten Elemente die Zwischendarstellung aufzubauen. Wie die Klassen zur Laufzeit geladen und instantiiert werden können, habe ich bereits oben gezeigt. Die Initialisierung der Instanzen der abstrakten Elemente mit den Informationen der darzustellenden ξML-Präsentation, die in den Datenfeldern der Elementen der Eingabesprache abgelegt sind, erfolgt mit Hilfe des von der Reflektions-Programmierschnittstelle der Java-Standardbibliothek angebotenen Introspektionsmechanismus. Er ermöglicht die Analyse der Struktur einer Java-Klasse zur Laufzeit, so dass die von den Instanzen der Java-Klassen der abstrakten Elemente benötigten Daten ermittelt und mit den Informationen der entsprechenden Elemente der Eingabesprache gefüllt werden können. Um den Code für die Zielplattform zu erzeugen, kann das Besucher-Muster (Visitor-Pattern, [GAMMA et al. 1996]) verwendet werden: Mit einer in einem Java-Framework implementierten Durchlaufstrategie wird über die Elemente der Zwischendarstellung iteriert und für jedes Element die Codeerzeugung über die einheutliche

Besucher-Schnittstelle an den entsprechenden Codeerzeuger der XAM-Komponente delegiert.

Wie aus den Ausführungen in diesem Abschnitt ersichtlich ist, unterscheidet sich der Ablauf der Generierung bei der XAM von dem eines gewöhnlichen Compilers. Die Initialisierungsphase und die in ihr stattfindenden Vorgänge sind XAM-spezifisch und bei anderen Compilern, die nicht wie die XAM erweiterbar gestaltet sind, nicht zu finden. Zum anderen folgt die XAM zwar generell dem Analyse-Synthese-Modell, einzelne sonst übliche Phasen sind bei der XAM jedoch nicht zu finden. So entfällt die übliche semantische Analyse der Eingabe, da diese von einem vorgelagerten Generator, für den im Rahmen dieser Arbeit angenommen wird, das er eine korrekte Ausgabe erstellt, erzeugt wird. Bei der Erzeugung der Zwischendarstellung innerhalb der XAM entfällt eine komplexe Abbildung der Konzepte der Eingabe in Konzepte der Zwischendarstellung. Die XAM-Eingabe stellt im Wesentlichen eine serialisierte Form der Zwischendarstellung der XAM dar. Dies liegt in der Architektur des ξML-Systems begründet. Üblicherweise wird bei Compilern die Zwischendarstellung als Programm für eine abstrakte Maschine aufgefasst (siehe Kap. 1 in [AHO et al. 1988a]). Dieses gedachte Konzept der abstrakten Maschine wird in der ξML-Architektur durch die XAM explizit gemacht (siehe Abbildung 3.1 auf Seite 22). Die XAM-Eingabe ist daher im Rahmen des ξML-System die persistente Form der Zwischendarstellung einer ξML-Präsentation, aus der Code für die Zielplattformen erzeugt werden soll. Daher ist die Eingabe schon das Programm für die abstrakte Maschine, so dass eine komplexere Abbildung der XAM-Eingabe in die XAM-Zwischendarstellung nicht nötig ist. Das Eingabedokument wird nur in eine Repräsentation im Hauptspeicher überführt, um die Informationen der Eingabe zugänglich zu machen. Die Phase der Codeoptimierung, durch den effizienterer Maschinencode erzeugt werden soll, entfällt bei der XAM ebenfalls, da sie für die zu erzeugenden Multimediaformate unwesentlich ist. Abgesehen von Stilfragen beim Formulieren von ξML-Präsentationen in den Formaten der Zielplattformen sind übliche Optimierungsziele, wie das Erzeugen von Code, der zur Laufzeit möglichst schnell ausgeführt werden kann, oder dessen Ausführung möglichst wenig Arbeitspeicher benötigt, bei der XAM nicht von Bedeutung.

6.5 Zusammenfassung

In diesem Kapitel habe ich die Systemarchitektur der XAM mit ihren statischen und dynamischen Aspekten vorgestellt. Dazu wurden zunächst die Architekturmuster, die der Konzeption zu Grunde liegen, erläutert. Die sich daraus ergebene Systemarchitektur wurde an Hand der einzelnen Elemente und ihres Zusammenspiels vorgestellt. Um die Umsetzbarkeit dieses Entwurfes zu zeigen, ist für die in der Systemarchitektur verwendeten Konzepte und Mechanismen ein Realisierung mit der XML-Technologie und der Programmiersprache Java skizziert worden.

In den folgenden Kapiteln gehe ich jetzt näher auf die einzelnen Teile der XAM ein. Hierbei lege ich den Schwerpunkt auf die Modellierung des abstrakten Modells (siehe Kapitel 7) und der für die giMLi-Präsentation benötigten XAM-Komponenten (siehe Kapitel 9). Um meine vorgenommene Modellierung der XAM mit ihren Abstraktionen für Elemente von ξML-Präsentationen zu überprüfen, skizziere ich in Kapitel 10, wie die giMLi-Präsentation als Beispiel einer ξML-Präsentation von ihrer Repräsentation in der Eingabesprache XaML in das Zielformat HTML überführt werden kann. Ein detaillierter Entwurf der Infrastruktur der XAM obliegt aus Umfangsgründen einer nicht im Rahmen dieser Arbeit stattfindenden Implementierung der XAM.

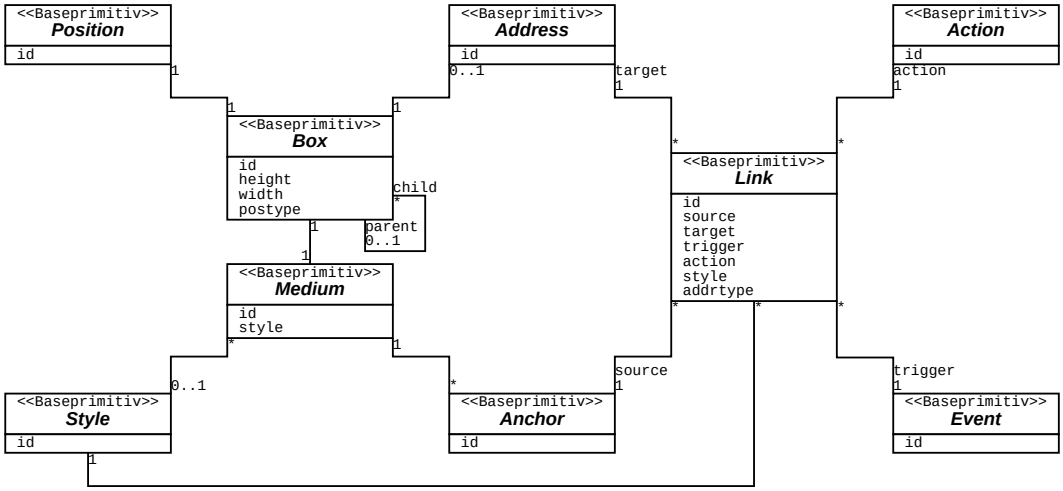
7 Abstraktes Modell

In diesem Kapitel stelle ich das abstrakte Modell der XAM vor. Es definiert das Metamodell, das den ξML-Präsentationen, deren Darstellung von der XAM unterstützt wird, zu Grunde liegt. Die so genannten *Basisprimitive* definieren die Konzepte, die zur Modellierung der ξML-Präsentationen verwendet werden dürfen. Die Anforderungen, die das abstrakte Modell mit seinen Basisprimitiven erfüllen muss, habe ich in Abschnitt 5.2.2 hergeleitet und vorgestellt. Die sich daraus im Einzelnen ergebenden Basisprimitive stelle ich in den folgenden Abschnitten des Kapitels vor. Die Überprüfung, ob mit den gewählten Basisprimitiven meine aufgestellten Anforderungen erfüllt werden, findet daran anschliessend zentral in Abschnitt 7.10 statt.

Als generelles Prinzip, das ich dem Entwurf des abstrakten Modells zu Grunde lege, werden Mechanismen, die ich im Rahmen meiner Anforderungsanalyse als notwendig zur Beschreibung des Metamodells der ξML-Präsentationen identifiziert habe, weitestgehend als eigene Basisprimitive modelliert, um sie konzeptionell eigenständig zu definieren. Davon erwarte ich zum einen eine Entkoppellung der einzelnen Mechanismen, die es mir erlaubt, möglichst keine Annahmen über die Umsetzung dieser Mechanismen in den entsprechenden Zielplattformen vorzunehmen. Es ist leichter, klar getrennt modelliert Konzepte bei der Codeerzeugung zu mischen, falls dies für eine Umsetzung in der Zielplattform nötig ist, als einmal vermischt modellierte Konzepte bei der Codegenerierung wieder zu trennen. Zum anderen verhindert eine getrennte Modellierung einzelner erweiterbarer Konzepte, eine unübersichtliche Hierarchie mit einer Vielzahl von unterschiedlichen Spezialisierungen. Aus dem selben Grund wurde bei dem Entwurf von PREMO großer Wert auf die Definition eigener Schnittstellen für einzelne Aspekte von dort spezifizierten Multimediaobjekttypen gelegt (siehe Abschnitt 2.3 in [HERMAN und DUKE 1997]).

Um diesem Entwurfsprinzip zu treu zu bleiben, spezialisiert im Rahmen der Definition der XAM-Komponenten in Kapitel 9 ein abstraktes Element genau ein Basisprimitiv. Bei der Spezialisierung mehrerer Basisprimitive durch ein abstraktes Element würden die oben genannten Vorteile außer Kraft gesetzt.

Das abstrakte Modell beschreibe ich an Hand seiner Schnittstellen, die es zur Erweiterung der multimedialen Funktionalität der XAM durch die XAM-Komponenten definiert. Diese Erweiterungsschnittstellen haben zwei Repräsentationen: Zum einen eine Repräsentation, die von den abstrakten Elementen der Zwischendarstellung verwendet wird. Zum anderen eine Repräsentation, die zur Definition der Eingabesprache XaML bzw. seiner Grammatik dient. Für die einzelnen Basisprimitive stelle ich jeweils die beiden Repräsentationen ihrer Schnittstellen vor. Da die Schnittstelle zu Codegenerierung gemäß des Besucher-Musters für alle Basisprimitive einheitlich ist und aus einer geeigneten Methode zum Codeerzeugen besteht (siehe Abschnitt 6.4), wird nicht hier sondern im Rahmen der detaillierten Beschreibung der Codegenerierung in Kapitel 10 näher auf sie eingegangen.



UML-Diagramm 7.1: Das abstrakte Modell

Zur Beschreibung der Repräsentation der Schnittstelle des abstrakte Modells für die abstrakten Elemente der Zwischendarstellung wird ein UML-Klassenmodell verwendet (siehe UML-Diagramm 7.1). Die Basisprimitive sind als abstrakte Klassen dargestellt, da es mit ihnen nicht möglich ist, eine konkrete ξML-Präsentation zu modellieren. Beziehungen zwischen einzelnen Basisprimitiven sind als Assoziationen dargestellt, Eigenschaften als Attribute. Sie werden im Klassendiagramm ungetypt und ohne Sichtbarkeitsbereich nur mit ihrem Namen angegeben, da diese Aspekte auf dem hier verwendeten Abstraktionsniveau irrelevant sind. Auf eine Referenzierung des UML-Diagramms in den einzelnen Abschnitten wird zu Gunsten einer besseren Lesbarkeit verzichtet.

Die Repräsentation der Schnittstelle des abstrakten Modells für die Eingabesprache XaML stelle ich in Form von DTD-Fragmenten vor. In ihnen wird mit Hilfe so genannter Entity-Definitionen die Eigenschaften der Basisprimitive als Attribute modelliert. Assoziationen drücke ich mit Hilfe von zwei unterschiedlichen Mechanismen in einer DTD aus: Zum einen als Eigenschaft einer symbolische Referenz auf das entsprechende Basisprimitiv, die kanonisch als Attribut modelliert wird. Zum anderen über Enthaltenseinsbeziehungen von XML-Elementen, die sich aber nur auf der Ebene der abstrakten Elemente in der DTD definieren lassen. Eine so modellierte Assoziation kann daher nicht in der Entity-Definition eines Basisprimitivs ausgedrückt werden, sondern findet sich in dem DTD-Fragment der entsprechenden abstrakten Elemente wieder (siehe Kapitel 9). Diese unterschiedliche Modellierung der Assoziationen liegt in der Abbildung der generellen Struktur der Schnittstelle des abstrakten Modells in die Repräsentation für die Eingabesprache begründet. Auf Grund von Eigenschaften der gewählten XML-Technologie ist das Abbilden nicht so offensichtlich, wie es bei der Repräsentation für die abstrakten Elemente in den UML-Klassendiagramm und ihrer Umsetzung mit Java ist. Daher werden die Aspekte der technischen Umsetzung dieser Abbildung gesondert in Abschnitt 7.11 vorgestellt.

Allen Basisprimitiven ist gemein, dass sie die Identität von Instanzen ihrer Spezialisierun-

gen über die Eigenschaft einer eindeutigen Kennung explizit vorschreiben. Diese Eigenschaft wird in beiden Repräsentationen durch ein Attribut `id` modelliert. Damit besitzt später jede Instanz einer Spezialisierung eines Basisprimitivs unabhängig ihrer unterschiedlichen physischen Darstellungen im Eingabedokument und im Hauptspeicher eine eindeutige Kennung, mit der ihre Objektidentität definiert ist. Dieser gleichförmige Mechanismus ermöglicht eine einheitliche Umsetzung der symbolische Referenzierung als Modell für Assoziationen in beiden Darstellungsformen. Diese Tatsache wird für unterschiedliche Aufgaben der XAM verwendet. An den entsprechenden Stellen werde ich näher darauf eingehen.

Nach diesen Vorbemerkungen wird nun das abstrakte Modell an Hand seiner Basisprimitive vorgestellt. Die strukturellen Zusammenhänge des Modells erläutere ich im Rahmen der Beschreibung der drei zentralen Basisprimitive *Box*, *Medium* und *Link*.

7.1 Das Basisprimitiv *Box*

Das Basisprimitiv *Box* modelliert einen rechteckigen Bereich auf dem Ausgabegerät, in dem die in einer ξML-Präsentation darzustellenden Informationen, die Medienelemente, angezeigt werden. Jedes Medienelement, das dargestellt werden soll, kann nur innerhalb des Bereichs einer *Box* angezeigt werden. Diese Bereiche können geschachtelt werden. Dabei muss der eingeschlossene Bereich vollständig im umschließenden enthalten sein. Ein Bereich ist auch dann vollständig ein einem anderen enthalten, wenn sich ihre Ränder überdecken. Der eingeschlossene Bereich wird immer vor dem umschließenden Bereich liegend auf dem Ausgabegerät dargestellt. Bereiche hingegen, die sich nicht beinhalten, sondern deren *Boxen* die gleiche Eltern-*Box* besitzen, dürfen sich nicht überlappen. Ihre Ränder dürfen entweder voreinander stoßen oder sich gar nicht berühren. Ansonsten ist ihre Lage innerhalb der umschließenden Eltern-*Box* frei wählbar. Für die Modellierung der ξML-Präsentationen bedeutet dies jedoch keine Einschränkung, da ihr Layout auch ohne Überlappungen beschreibbar ist. Um generell Überlappungen zuzulassen, müsste eine dritte Dimension in Form von beliebig anzuordnenden Ebenen explizit modelliert werden. Da in Abschnitt 6.4 angenommen wird, dass die Eingabe vom vorgelagerten Generator semantisch korrekt erzeugt wird, kann die Einhaltung der Nichtüberlappung vorausgesetzt werden. Eine Überprüfung diese Bedingung innerhalb der XAM findet daher nicht statt.

Ein *Box* kann auf dem Ausgabemedium angeordnet werden. Dadurch hat man mit der *Box* die Möglichkeit, die ihr zugeordneten Medienelemente auf dem Ausgabegeräte zu positionieren. Wie Abschnitt 2.2.1 vorgestellt, kann eine Positionierung auf sehr unterschiedlichen Arten vorgenommen werden kann, wie etwa eine absolute oder eine auf Relationen beruhende Positionierung. Daher wird der Positionsbegriff als eigenes Basisprimitiv *Position* (siehe Abschnitt 7.5) modelliert. Zur Positionierung ist einer *Box* immer eine *Position* zugeordnet.

Eine *Box* fungiert als ein Container für ein anzeigbares Element einer ξML-Präsentation. Diese Medienelemente werden im abstrakten Modell mit dem Basisprimitiv *Medium* modelliert (siehe Abschnitt 7.2). Unter einem anzeigbaren Element wird bei der XAM auch der Hintergrund einer *Box* verstanden und somit auch als ein *Medium* modelliert. Einer *Box* ist daher immer genau ein *Medium* zugeordnet.

Da *Boxen* geschachtelt werden können, ermöglicht eine *Box* die Gruppierung von in den *Boxen* dargestellten Medienelementen. Durch die Schachtelung ist es somit möglich, die Komposi-

tion einzelner Medienelemente zu einer Einheit zu modellieren. Durch diesen Gruppierungsmechanismus können auch zeitliche Beziehungen zwischen Medienelementen angegeben werden. Dazu ist es nötig, die oberste *Box* einer solcher Gruppierung durch eine Spezialisierung des Basisprimitivs *Medium* auszuzeichnen und so ein komponiertes Medienelement zu formulieren, das sich aus allen in der Gruppierung unterhalb dieser *Box* in weiteren *Box*-Primitiven befindlichen Medienelementen zusammensetzt. Für ein solches komponiertes Medienelement ist es dann möglich, die gleichzeitige Anzeige aller an der Komposition teilnehmenden Medienelemente zu fordern. Diese Forderung muss bei der Codegenerierung für eine Spezialisierung eines solchen *Medium*-Primitivs umgesetzt werden. Ein Beispiel einer solchen Spezialisierung eines *Medium*-Primitivs wird später in Abschnitt 9.7 mit der XAM-Komponente *Scene* gegeben, mit der der seitenorientierte Aufbau der giMLi-Präsentation modelliert wird.

Eine Adressierung der Medienelemente findet über die *Box* statt. Um unterschiedliche Adressierungsschemen unterstützen zu können, wird eine Adresse als eigenes Basisprimitiv *Address* modelliert (siehe Abschnitt 7.7), das einer *Box* zugeordnet werden kann, wenn sie adressierbar sein soll. Da einer *Box* genau ein *Medium* zugeordnet ist, kann eine Medienelement über eine *Box* adressiert werden. Ein Vorteil der Modellierung der Adressierung eines Medienelements über ein *Box*-Primitiv ist, dass dadurch die Adressierung sowohl der atomaren als auch der komponierten Medienelemente gleichförmig behandelt wird.

Besondere Fähigkeiten, die das Verhalten einer *Box* betreffen und die unabhängig von den in den *Boxen* dargestellten Medienelementen sind, wie etwa die Möglichkeit, einen Inhalt zu Scrollen, werden in Spezialisierungen einer *Box* angegeben. Layoutinformationen hingegen, die das Aussehen einer *Box* verändern und sich ebenfalls nicht auf die in einer *Box* dargestellten Medienelemente beziehen, werden als Eigenschaften einer *Box* modelliert. Ein Beispiel hierfür wäre etwa das Vorhandensein eines Randes um die *Box*. Um das abstrakte Modell möglichst übersichtlich zu gestellten, verzichte ich im Rahmen dieser Arbeit auf die Definition solcher Layouteigenschaften einer *Box*. Sie müssten als entsprechende Attribute einer *Box* modelliert werden. Drei Eigenschaften einer Box sind jedoch zwingend notwendig: Zum einen besitzt sie die Eigenschaften Höhe und Breite, um ihre die Ausdehnung definieren zu können. Zum anderen ist es nötig, angeben zu können, auf welche Art eine *Box* positioniert wird, um die entsprechenden Angaben in einer *Position* verarbeiten zu können. Die Ausdehnung wird in Pixel angegeben, da als Ausgabemedium der Bildschirm vorgesehen ist. Die maximale Ausdehnung eine *Box* ist von der Größe und der gewählten Auflösung des Bildschirms abhängig und wird daher nicht fest vorgegeben. Es obliegt dem Autor einer ξML-Präsentation, sinnvolle Angaben zur Breite und Höhe vorzunehmen.

Das führt zu einer Schnittstelle, in der die oben genannten Eigenschaften als die Attribute `height`, `width` und `postype` modelliert sind. Die Assoziationen zu einem *Medium* und zu einer *Position* werden in der Repräsentation für die Eingabesprache über Enthaltenseinsbeziehungen modelliert und tauchen daher nicht im DTD-Fragment auf (Grammatikfragment 7.1 auf der nächsten Seite). Die Assoziation zwischen einem *Adress*-Primitiv und einer *Box* wird in Richtung der *Box* navigierbar modelliert und hat daher in dieser Schnittstelle keine Entsprechung.

```
<!ENTITY % box-attr "
  id       ID    #REQUIRED
  width    CDATA #REQUIRED
  height   CDATA #REQUIRED
  postype CDATA #REQUIRED">
```

Grammatikfragment 7.1: DTD-Fragment des *Box*-Primitivs

7.2 Das Basisprimitiv *Medium*

Das Basisprimitiv *Medium* modelliert Informationseinheiten, die in einer ξML-Präsentation dargestellt werden. Es abstrahiert von den unterschiedlichen Medientypen, in denen Informationen formuliert werden können. Eine mit einem *Medium* modellierte Information wird als Einheit angesehen, deren innere Struktur der XAM nicht bekannt ist.

Einem *Medium* ist immer ein durch eine *Box* modellierter Bereich auf dem Ausgabegerät zugeordnet, in dem die mit dem *Medium* definierte Information dargestellt wird. Diese Modellierung führt zu einer 1-1-Beziehung zwischen den Basisprimitiven *Box* und *Medium*, die durch ein Zusammenfassen der beiden Primitive entfallen würde. Da dann aber die unterschiedlichen Aspekte wie das Anordnen und Gruppieren von Medienelementen und die Definition einer Information in einem Basisprimitiv zusammengefasst würden, scheidet dieses aus den im Rahmen des oben vorgestellen Entwurfsprinzips genannten Gründen aus.

Einem *Medium* kann ein Stil, modelliert durch das Basisprimitiv *Style* (siehe Abschnitt 7.4), zugeordnet werden. Mit einem *Style* ist es möglich, Parameter anzugeben, mit denen die Darstellung einer mit einem *Medium* modellierten Information parametrisiert werden kann, falls in der Zielplattform für den entsprechenden Medientyp ein solcher Mechanismus angeboten wird (siehe Seite 39).

Um Beziehungen zu einem Teil einer Information innerhalb eines *Mediums* definieren zu können, ohne dabei die Kapselung der internen Struktur der Information vor der XAM zu verletzen, können einem *Medium* Anker, modelliert durch das Basisprimitiv *Anchor* (siehe Abschnitt 7.6), zugeordnet werden. Mit einem *Anchor* wird eine Schnittstelle zu einer Position innerhalb eines *Mediums* definiert.

Spezialisierungen eines *Medium*-Primitivs können abstrakte Elemente wie ein *Text* oder ein *Bild*, aber auch komplexere Medientypen sein. Neben der Modellierung von inhaltlichen Informationeinheiten, die in einer ξML-Präsentation dargestellt werden sollen, kann eine Spezialisierung eines *Medium*-Primitivs auch den Hintergrund einer *Box* angeben oder Stellen in der mit der Schachtelung der *Box*-Primitive modellierten baumartigen Hierarchie auszeichnen. Eine solche Kennzeichnung kann etwa mit der Spezialisierung eines *Medium*-Primitivs zu einem abstrakten Element *Scene* modelliert werden, das in der Baumrepräsentation des Eingabedokuments den Beginn der Beschreibung eines Themas auszeichnet. Alle Medienelemente, die sich in der Baumhierarchie in *Boxen* unterhalb einer so ausgezeichneten *Box* befinden, sollen gleichzeitig auf dem Ausgabegerät angezeigt werden. Auf diese Art besteht die Möglichkeit, einen seitenorientierten Aufbau, wie er der giMLi-Präsentation durch die Verwendung von unterschiedlichen Themen zu Grunde liegt, zu modellieren.

Für ein *Medium* wird bewusst auf die Definition einer Eigenschaft der Ausdehnung verzichtet. Die Fläche, die zum Anzeigen eines Medienelements auf dem Ausgabemedium zur Verfügung steht, ist durch das *Box*-Primitiv fest vorgegeben. Stimmt die Größe eines Medienelements nicht mit der Ausdehnung der entsprechenden *Box* überein, obliegt es der *Box*, geeignet darauf zu reagieren. Zur Festlegung eines bestimmten Verhaltens in einem solchen Fall, etwa eine scrollbare Darstellung des Medienelements, kann dies in einer Spezialisierungen einer *Box* definiert werden. Wenn kein bestimmtes Verhalten modelliert wird, kommt das Standardverhalten der jeweiligen Zielplattformen zum Einsatz, was beispielweise ein Clippen oder Skalieren des Medienelements sein kann. Ebenso wird darauf verzichtet, das Format eines *Mediums* als Eigenschaft zu modellieren. Der Medientyp eines darzustellenden Medienelements wird durch entsprechende Spezialisierungen des Basisprimitivs *Medium* modelliert. Die für einen Medientyp erlaubten Formate sind durch die jeweiligen Zielplattformen vorgegeben. Falls mehrere Formate für einen Medientyp von einer Zielplattform unterstützt werden, wirkt sich dies bei den derzeit üblichen Plattformen nicht auf ihre Mechanismen zur Verwendung des Medienelements aus. Sollte eine Unterscheidung jedoch doch nötig werden, kann dieses durch eine Spezialisierung des *Medium*-Primitivs etwa in ein getyptes Bild modelliert werden. Im Rahmen des in dieser Arbeit betrachten Beispiels der Darstellung der giMLi-Präsentation in der HTML-Zielplattform ist das jedoch nicht der Fall.

Da die Assoziation zu einem *Style*-Primitiv in beiden Repräsentationen über eine symbolische Referenz umgesetzt wird, wird diese als Eigenschaft eines *Medium* modelliert. Die Schnittstelle des *Medium*-Primitivs besitzt somit neben des kanonischen Attributs `id` ein Attribut `style`, in dem die entsprechende Referenz abgelegt werden kann. Die Assoziation zu einer *Box* wird in der Repräsentation für die Eingabesprache als Enthaltenseinsbeziehung modelliert und findet sich daher nicht im DTD-Fragment des *Medium*-Primitivs wieder (Grammatikfragment 7.2).

```
1 <!ENTITY % medium-attr "
2   id    ID    #REQUIRED
3   style IDREF #IMPLIED">
```

Grammatikfragment 7.2: DTD-Fragment des *Medium*-Primitivs

7.3 Das Basisprimitiv *Link*

In ξML-Präsentation dargestellte Informationen können durch das Basisprimitiv *Link* miteinander in Beziehung gesetzt werden. Diese Beziehungen sind navigierbar. Die Verfolgung einer Beziehung wird durch ein Ereignis ausgelöst und hat die Ausführung einer Aktion zur Folge. Im Abstrakten Modell werden sie als so genannter *unärer, unidirektionaler Link* modelliert, d.h. eine Beziehung besitzt einen ausgezeichneten Startpunkt und einen ausgezeichneten Zielpunkt. Eine Alternative wäre zum einen die Modellierung als so genannter *n-ärer Link*, bei dem eine Beziehung mehr als einen Start- und mehr als einen Zielpunkt besitzt. N-äre Links sind zwar mittlerweile in den Linkspezifikation des XML-Standards aufgenommen worden (XLink, [DeRose et al. 2001]), ihre Umsetzung ist aber noch nicht geklärt. Von den handelsüblichen Sys-

temen werden sie derzeit nicht umgesetzt. Daher habe ich mich gegen eine Modellierung von n-ären Links mit dem abstrakten Modell entschieden. Für eine Unterstützung von n-ären Links müsste das abstrakte Modell an dieser Stelle angepasst werden. Durch die Modellierung als ein eigenes Basisprimitiv werden Links von den anderen Konzepten entkoppelt, so dass sich eine solche Anpassung an ein anderes Linkkonzepts im Wesentlichen nur in dem Basisprimitiv *Link* niederschlagen und andere Primitive unberührt lassen würde. Eine weitere Alternative wäre, eine Beziehung auch als bidirektionaler Link zu modellieren, bei dem eine Navigation in beiden Richtungen erlaubt ist. Da aber eine Modellierung einer Beziehung als bidirektionaler Link in eine Modellierung mit zwei entsprechenden unidirektionalen Links, bei denen die Start- und Zielpunkte jeweils vertauscht sind, abgebildet werden kann, habe ich mich für die Modellierung von unidirektionalen Links entschieden.

Als Startpunkt einer Beziehung werden Teile einer Informationseinheit ausgezeichnet. Im abstrakten Modell sind solche Positionen innerhalb von Medienelementen als *Anchor*-Primitiv modelliert (siehe Abschnitt 7.6 auf Seite 77). Das Ziel einer Beziehung ist ein gesamtes Medienelement. Eine Adressierung von Medienelementen, unabhängig davon, ob sie atomar oder zusammengesetzt sind, wird im abstrakten Modell mit dem Basisprimitiv *Address* ermöglicht (siehe Abschnitt 7.7). Da mit dem abstrakten Modell ein unärer Link modelliert wird, der genau einen Start- und einen Zielpunkt hat, sind einem *Link*-Primitiv entsprechend genau ein *Anchor*- und ein *Address*-Primitiv zugeordnet.

Gemäß des oben vorgestellten Entwurfsprinzips, wird ein Ereignis, das die Verfolgung einer Beziehung auslöst, und die Aktion, die als Ergebnis der Verfolgung eines Links ausgeführt wird, in eigenen Basisprimitiven, als *Event*- (siehe Abschnitt 7.9) und als *Action*-Primitiv (siehe Abschnitt 7.8), modelliert. Die Alternative, diese beiden Aspekte über eine Typisierung eines Links zu modellieren, scheidet daher aus bekannten Gründen aus. Da das Auslösen der Verfolgung ein und derselben Beziehung durch unterschiedliche Ereignisse ebensowenig Sinn macht, wie die Modellierung einer Beziehung, deren Verfolgung durch kein Ereignis ausgelöst werden kann, schreibt es das abstrakte Modell vor, das einem *Link*-Primitiv genau ein *Event*-Primitiv zugeordnet werden muss. Ebenso ist einem *Link*-Primitiv genau ein *Action*-Primitiv zugeordnet, da bei der Verfolgung einer Beziehung nur eine Aktion ausgeführt werden kann. Diese eine Aktion kann aber, im Rahmen der von den Zielplattformen vorgegebenen Möglichkeiten, beliebig komplex sein.

Eine Beziehung wird durch ein Element, das den Startpunkt repräsentiert, visualisiert. Ein *Link* kann mit Hilfe des Basisprimitivs *Style* das Layout seines Erscheinungsbilds, etwa die Farbe des Textes, der den Startpunkt eines *Links* modelliert, verändern. Einem *Link* kann daher ein Basisprimitiv *Style* zugeordnet werden.

Die Assoziationen zu den einzelnen Basisprimitiven *Anchor*, *Address*, *Event*, *Action* und *Style* werden als symbolische Referenzen modelliert und bilden somit Eigenschaften eines *Link*-Primitivs. Ebenso besitzt es die Eigenschaft eines Adressierungstyps, mit dem angegeben wird, welches Adressierungsschema für das Ziel einer Beziehung verwendet wird.

Die Schnittstelle des Basisprimitivs *Link* besitzt daher die entsprechenden Attribute `start`, `target`, `trigger`, `action` und `style`, die die Referenzen modellieren, und zur Definition des Adressierungsschema das Attribut `addrtype`. Die Repräsentation dieser Schnittstelle für die Eingabesprache ist in Grammatikfragment 7.3 auf der nächsten Seite zu sehen.

```
<!ENTITY % link-attr "
  id       ID       #REQUIRED
  source   IDREF    #REQUIRED
  target   IDREF    #REQUIRED
  trigger  NMTOKEN  #REQUIRED
  action   IDREF    #REQUIRED
  style    IDREF    #IMPLIED
  addrtype CDATA    #REQUIRED">
```

Grammatikfragment 7.3: DTD-Fragment des *Link*-Primitivs

7.4 Das Basisprimitiv *Style*

Die Möglichkeit, für ein mit einem *Medium* modellierten Medienelement bzw. für den Bereich eines Medienelements, der als Startpunkt einer mit einem *Link* modellierte Beziehung ausgezeichnet ist, Layoutinformationen angeben zu können, wird mit einem Stil, modelliert als Basisprimitiv *Style*, gegeben. Die Unterscheidung zwischen einem gesamten Medienelement und dem Bereich eines Medienelements, der als Startpunkt eines *Links* ausgezeichnet ist, nehme ich in der folgenden Diskussion des Basisprimitivs aus Lesbarkeitsgründen nicht mehr vor. Im Rahmen der Beschreibung eines Stils fasse ich unter den Begriff eines Medienelements beide Fälle. Ein *Style* kann einem oder mehreren *Medium*- bzw. *Link*-Primitiven zugeordnet werden. Durch die explizite Modellierung eines Stils wird die Darstellung von der eigentlichen Information getrennt, wie es etwa mit den Cascading Stylesheets (CSS, [Lie und Bos 1996]) für HTML auch nachträglich eingeführt wurde. In einem *Style* werden Parameter definiert, mit der die Darstellung von Medienelementen beeinflusst werden kann. Welche Parameter das sind, hängt von der Umsetzung der Darstellung in der jeweiligen Zielplattform ab (siehe Seite 39).

Die Modellierung der Parameter und ihre Wertbelegungen wird durch Schlüssel-Werte-Paare vorgenommen, die bei dem jeweiligen *Medium* abgelegt werden. Sie liegen somit außerhalb des eigentlichen Typsystems des abstrakten Modells und sind daher nicht in UML-Diagramm 7.1 auf Seite 69 aufgeführt. Die Schnittstelle eines solche Schlüssel-Wert-Paar-Elements ist denkbar einfach und besteht aus einem Schlüssel- und einem Wert-Attribut. Bei der Codegenerierung müssen für eine Spezialisierung des *Style*-Primitivs die einzelnen Schlüssel-Wert-Paare in die Parameter abgebildet werden, mit denen die Darstellung der Medienelemente in der jeweiligen Zielplattform beeinflusst werden kann.

Durch die Verwendung dieses Schlüssel-Wert-Paar-Mechanismus wird bewusst auf die Modellierung eines festen Satzes an Parametern verzichtet, um flexibel auf neue Medientypen und die Unterstützung neuer Zielplattformen vorbereitet zu sein. Unabhängig von konkreten Parametern kann so immer derselbe Mechanismus verwendet werden. Beispiele solcher Parameter sind für den Medientyp Text etwa Formatierungsangaben wie Schriftgröße, Schriftfarbe oder Schriftart. Für weitere Beispiele solcher Parameter sei auf den Standard Extensible Stylesheet Language Formating Objects (XSL-FO, [Adler et al. 2001]) verwiesen, in dem unter anderem im Rahmen der Festlegung einer Semantik für (plattformunabhängige) Formatierungseigenschaften eine Vielzahl solcher Parameter fest definiert werden.

Die Assoziationen zu einem *Box*- bzw. *Link*-Primitiv werden so modelliert, dass sie nicht von einem *Style* wegführend navigierbar sind. Daher sind sie nicht in dieser Schnittstelle, sondern bei den beiden anderen Primitiven modelliert. Somit besitzt die Schnittstelle eines *Style*-Primitivs neben der kanonischen `id` keine weitere Attribute (Grammatikfragment 7.4).

```
1 <!ENTITY % style-attr "
2   id ID #REQUIRED">
```

Grammatikfragment 7.4: DTD-Fragment des *Style*-Primitivs

7.5 Das Basisprimitiv *Position*

Das Basisprimitiv *Position* definiert einen Positionsbegriff, mit dem *Boxen* und somit die darin dargestellten Medienelemente auf dem Ausgabemedium angeordnet werden können (Anforderung 7 auf Seite 42). Mit einer *Position* werden Koordinaten in einem abstrakten Koordinatensystem modelliert.

Bei konkreten Koordinatensystemen, die in den Zielplattformen für die Positionierung zur Verfügung stehen, werden zur Modellierung der entsprechenden Koordinaten abstrakte Elemente von dem Basisprimitiv *Position* spezialisiert. Eine Spezialisierung für ein absolutes zweidimensionales Koordinatensystem wäre etwa ein abstraktes Element, das die Eigenschaft einer `x`- und `y`-Koordinate besitzt. Entsprechend besitzt eine spezialisierte *Position* für ein relatives zweidimensionales Koordinatensystem nur die Eigenschaft einer Koordinate, die etwa die Werte `{oberhalb|links|unterhalb|rechts}` annehmen kann. Entsprechendes gilt für ein zweidimensionales topologisches Koordinatensystem, bei dem die Werte der Koordinate nur anders benannt sind, etwa mit `{nord|ost|süd|west}`.

Das Basisprimitiv *Position* selbst besitzt keine Eigenschaft. Koordinaten werden erst in den Spezialisierungen einer *Position* angegeben. Die Alternative, diese Koordinaten auf Ebene des abstrakten Modells mit Hilfe des oben vorgestellten Schlüssel-Werte-Paar-Mechanismus generisch zu modellieren ist nicht sinnvoll, da für jede Spezialisierung einer *Position* fest vorgegeben wird, wie die Koordinaten des entsprechenden Koordinatensystems aussehen. Bei Spezialisierungen des Basisprimitivs *Style* hingegen sind die Eigenschaften nicht fest vorgegeben und müssen somit generisch modelliert werden.

Wie in Grammatikfragment 7.5 zu sehen ist, besitzt die Schnittstelle abgesehen von dem `id`-Attribut kein weiteren Attribute. Die Assoziation zu einem *Box*-Primitiv ist nur von dem *Box*-Primitiv aus navigierbar. Sie wird daher in der dortigen Schnittstelle modelliert.

```
1 <!ENTITY % position-attr "
2   id ID #REQUIRED">
```

Grammatikfragment 7.5: DTD-Fragment des *Position*-Primitivs

7.6 Das Basisprimitiv *Anchor*

Mit dem Basisprimitiv *Anchor* wird eine abstrakte Position innerhalb eines atomaren Medienelements, ein so genannter Anker, modelliert, der als Startpunkt einer Beziehung dient. Die Kapselung der internen Struktur eines atomaren Medienelements vor der XAM wird durch die Definition eines Ankers nicht verletzt. Ein Anker verbirgt die Realisierung des Auffindens eines solchen Teiles einer Informationseinheit vor einer Beziehung. Mit einem *Anchor* wird somit eine feingranularere Schnittstelle für den Zugriff auf Informationseinheiten angeboten, als sie mit einem *Medium* bereitgestellt wird.

Durch die externe Definition einer Referenz auf einen Teil einer Information innerhalb eines Medienelements wird eine Beziehung entkoppelt von dem Typ und dem Format eines Medienelements. Es wird ein für alle Medienelemente einheitlicher Mechanismus bereitgestellt, den Startpunkt einer Beziehung angegeben zu können. Da eine Beziehung nur einen Startpunkt besitzt, wird einem *Link* genau ein *Anchor* zugeordnet.

Der hier vorgestellte Mechanismus findet sich auch bei anderen Systemen wieder. Das Referenzmodell Dexter ermöglicht mit einem auch Anchor genannten Element den Zugriff auf Informationen innerhalb des Within-Component-Layer, also auf Strukturen innerhalb von Medienelementen, ohne die inneren Strukturen der Medienelement im Gesamtsystem zu spezifizieren. Von der Sprache HyTime wird eine so genannte Document Location verwendet, um auf Bereiche innerhalb von Medienelementen zuzugreifen. DoDL stellt mit seinem abstrakten Positionsbegriff ebenfalls einen Mechanismus bereit, Einheiten innerhalb von Medien außerhalb der Medien zu adressieren, ohne dessen innere Struktur offenzulegen. Ein entgegengesetzter Ansatz wird mit A-Links in HTML verfolgt. Dort wird der Startpunkt des Links als Markierung von Wörtern direkt in einem Text vorgenommen. Ein gleichförmiger Mechanismus zur Definition von Startpunkten einer Beziehung ist dort nicht möglich. Die Codegenerierung der XAM für die Zielplattform HTML muss daher eine Abbildung des Anker-Mechanismus der XAM in das A-Link-Konzept von HTML anbieten.

Spezialisierungen eines Links realisieren für die unterschiedlichen Medientypen, wie etwa für Bilder oder Texte, den Zugriff auf deren internen Strukturen. Sie definieren alle Informationen, die zum Auflösen der mit einem Anker definierten symbolischen Referenz auf die Interna eines Medienelements benötigt werden. Um einen Bereich innerhalb eines Textes als Anker zu markieren, werden etwa Informationen wie Start- und ein Endbuchstabe benötigt. Für Bilder müssten beispielsweise Koordinaten, die eine entsprechende Fläche beschreiben, angegeben werden.

Wie gerade gesehen, sind alle weiteren Angaben, die für das Definieren einer Position innerhalb eines Medienelements benötigt werden, von dem Medientyp des Elements abhängig und werden somit auf die Ebene der abstrakten Elemente definiert. Das Basisprimitiv *Anchor* besitzt somit keine weiteren Eigenschaften. Da auch die Assoziation zwischen einem *Anchor* und einem *Link* nur aus Richtung des *Link*-Primitivs navigierbar ist, wird sie nicht in der Schnittstelle des Basisprimitivs *Anchor* modelliert. Die Schnittstelle besteht daher nur aus dem `id`-Attribut (Grammatikfragment 7.6 auf der nächsten Seite).

```
<!ENTITY % anchor-attr "
  id ID #REQUIRED">
```

Grammatikfragment 7.6: DTD-Fragment des *Anchor*-Primitivs

7.7 Das Basisprimitiv *Address*

Das Basisprimitiv *Address* modelliert im abstrakten Modell eine Adresse, über die eine *Box* von einem *Link* als Zielpunkt einer Beziehung referenziert wird. Die explizite Modellierung einer Adresse als eigenes Basisprimitiv ermöglicht es, unterschiedliche Adressierungsschemen zu verwenden, ohne eine *Box* modifizieren zu müssen. Diese Indirektion entkoppelt die Umsetzung der Adressierung einer *Box* von der *Box* an sich.

Für unterschiedliche Adressierungsschemen müssen entsprechende Spezialisierungen einer *Address* definiert werden, die die zur Auflösung einer Adresse benötigten Informationen enthält. Für eine Auflösung einer Adresse muss eine Funktion bereitgestellt werden, die die Abbildung einer mit einem *Address*-Primitiv modellierten Adresse in die Adresse des referenzierten Elements realisiert. Bei der XAM ist das die Id eines *Box*-Primitivs. Bei einem einfachen Adressierungsschema können Adressen aus global eindeutigen Id's bestehen. Bei einer Adressierung in baumartigen Strukturen kann eine Adresse als Pfad in diesem Baum modelliert werden. Eine solche Adressierung im Rahmen von Hypermediadokumenten wird etwa in [FRONK 2001], Kap. 4, verwendet. In solchen Baumstrukturen ist auch ein relatives Adressieren möglich. Der Mechanismus dieser indirekten Adressierung wird auch in HyTime verwendet. Dort werden unterschiedliche Ausprägungen von so genannten Location Address Architectural Forms definiert, um eine Adressierung mit unterschiedlichen Adressierungsschemen zu unterstützen.

Wie oben gesehen, wird die Assoziation zu dem zu adressierenden Element, hier also zu dem Basisprimitivs *Box*, in konkreten Ausprägungen modelliert, also bei der Schnittstelle der entsprechenden abstrakten Elemente. Neben einer eindeutigen Kennung besitzt ein Basisprimitiv *Address* daher keine weiteren Eigenschaften, so dass die Schnittstelle eines *Address*-Primitivs nur aus dem Attribut `id` besteht (Grammatikfragment 7.7).

```
<!ENTITY % address-attr "
  id ID #REQUIRED">
```

Grammatikfragment 7.7: DTD-Fragment des *Address*-Primitivs

7.8 Das Basisprimitiv *Action*

Eine Aktion, modelliert durch das Basisprimitiv *Action*, adressiert einen Aspekt der Interaktionsmöglichkeiten, die das abstrakte Modell anbietet. Mit einer Aktion wird die Handlung modelliert, die bei der Verfolgung einer Beziehung ausgeführt wird. Da eine Aktion, wie etwa das Austauschen eines referenzierten Medienelements, für mehrere Beziehungen gleich sein kann,

können einer *Action* mehrere *Links* zugeordnet werden. Die so modellierten Beziehungen zeigen dann bei ihrer Verfolgung das gleiche Verhalten.

Neben dem oben erwähnten Austauschen eines Medienelements, können auch komplexere Handlungen ausgeführt werden. Dafür können weitere Informationen, wie etwa Parameterwerte für Funktionen, die eventuell zum Ausführen der Aktion aufgerufen werden, benötigt werden. Diese werden in den Spezialisierungen eines *Action*-Primitivs als Eigenschaften modelliert.

Auf Ebene des abstrakten Modells besitzt eine *Action* jedoch neben dem als `id`-Attribut modellierten eindeutigen Bezeichner keine weiteren Eigenschaften (Grammatikfragment 7.8). Die Assoziation zwischen einem *Action*- und einem *Link*-Primitiv ist nur vom *Link* aus navigierbar und wird daher in dessen Schnittstelle modelliert.

```
1 <!ENTITY % action-attr "
2   id ID #REQUIRED">
```

Grammatikfragment 7.8: DTD-Fragment des *Action*-Primitivs

7.9 Das Basisprimitiv *Event*

Ein weiterer Aspekt der mit dem abstrakten Modell modellierten Interaktionsmöglichkeiten wird durch das Basisprimitiv *Event* adressiert. Ein *Event* modelliert ein Ereignis, das die Verfolgung einer Beziehung auslöst. Da eine Verfolgung von unterschiedlichen Beziehungen durch das gleiche Ereignis ausgelöst werden kann, können einem *Event* mehrere *Links* zugeordnet werden.

In Spezialisierungen des Basisprimitivs *Event* werden die konkreten Ereignisse modelliert, die üblicherweise durch Eingabegeräte wie Maus oder Tastatur erzeugt werden. Ein Beispiel eines solchen Ereignisses wäre etwa ein Klick mit der rechten Maustaste auf einen als Startpunkt einer Beziehung ausgezeichneten sensitiven Bereich. Informationen, wie etwa auf welchem Element ein Ereignis ausgelöst wurde, sind im Rahmen der XAM irrelevant und werden daher nicht modelliert. Die XAM bewegt sich, in Fall des Ereignisses, auf der Ebene der Beschreibung von Verhalten. Eine Ausführung dieses Verhaltens wird von dem Player der jeweiligen Zielplattform umgesetzt.

Die Schnittstelle besitzt daher nur das `id`-Attribut und keine weiteren als Attribute modellierten Eigenschaften (Grammatikfragment 7.9). Die Assoziation zwischen einem *Link* und einem *Event* ist nur aus Richtung des *Link*-Attributs navigierbar und wird daher in der Schnittstelle des *Link* modelliert.

```
1 <!ENTITY % event-attr "
2   id ID #REQUIRED">
```

Grammatikfragment 7.9: DTD-Fragment des *Event*-Primitivs

7.10 Ausdrucksstärke des Modells

Im Folgenden belege ich an Hand der einzelnen Anforderungen von Abschnitt 5.2.2, dass mit den in diesem Kapitel eingeführten Basisprimitiven die geforderte Mächtigkeit des Modells erfüllt wird (Anforderung4).

Mit einem *Medium* wird die Möglichkeit zur Modellierung von atomaren Medienelementen bereitgestellt (Anforderung 5). Komponierte Medienelemente können mit Hilfe der Schachtelung von *Medien* in *Boxen* ausgedrückt werden (Anforderung 6). Mit Hilfe einer *Position* können *Boxen* und somit die dort zugeordneten *Medien* auf dem Ausgabegerät angeordnet werden (Anforderung 7). Mit einem *Style* bzw. über Attribute einer *Box* besteht die Möglichkeit, Layoutinformationen für das Erscheinungsbild einer ξML-Präsentation anzugeben (Anforderung 8).

Neben den im vorherigen Absatz vorgestellten Fähigkeiten, Informationen darstellen zu können, muss das abstrakte Modell auch die Möglichkeit anbieten, durch eine ξML-Präsentation navigieren zu können. Mit einem *Link* kann eine Beziehung zwischen einzelnen Informationseinheiten modelliert werden (Anforderung 9). Unter der Verwendung der Basisprimitive *Anchor* und *Address* können so Medienelemente als ganzes oder einzelne Teile eines Medienelements in Beziehung gesetzt werden (Anforderung 10). Die Möglichkeit, dass ein Benutzer mit der Präsentation interagiert, um einen *Link* auszuwählen, wird durch das Basisprimitiv *Event* angeboten (Anforderung 11). Mit dem Primitiv *Action* hingegen wird modelliert, wie sich das Auswählen eines *Links*, also die Verfolgung einer Beziehung, auswirkt (Anforderung 12).

Zur Definition der geforderten zeitlichen Beziehungen können *Boxen* verwendetet werden (Anforderung 13). Eine *Box* kann durch ein *Medium* ausgezeichnet werden, so dass alle in dieser *Box* enthaltenen Medienelemente als gleichzeitig anzuzeigend markiert sind. Mit den im vorherigen Absatz vorgestellten Mechanismen zur Navigation und Benutzerinteraktion kann der Austausch von gleichzeitig anzuzeigenden Medienelementen ausgedrückt werden.

In diesem Abschnitt wurde gezeigt, dass das abstrakte Modell mit den von mir gewählten Basisprimitiven die in Anforderung 4 verlangte Ausdruckstärke besitzt. Wie auf Grundlage dieses Modells XAM-Komponenten zur Beschreibung einer ξML-Präsentation entwickelt werden können, wird am Beispiel der giMLi-Präsentation in Kapitel 9 detaillierter vorgestellt.

7.11 Repräsentation in XML

Die Schnittstelle des abstrakten Modells besitzt eine Repräsentation für die Eingabesprache. Das abstrakte Modell gibt einige Schnittstellen und Strukturen vor, die für alle von der XAM zu verarbeitenden Eingabesprachen gelten. Da die Eingabe mit der XML-Technologie realisiert wird, werden die Schnittstellen und Strukturen in DTD-Ausdrücken modelliert. In den vorherigen Abschnitten wurden bereits die DTD-Fragmente vorgestellt, die die Eigenschaften der Basisprimitive, also die Attribute ihrer Schnittstelle, beschreiben. In diesem Abschnitt werden noch weitere Fragmente eingeführt, mit denen zum einen der generelle Aufbau der Eingabesprachen und zum anderen die Assoziationen zwischen den Basisprimitiven modelliert werden. Die Summe aller DTD-Fragmente repräsentiert die von dem abstrakten Modell definierte Schnittstelle zur Eingabe. Die Fragmente bilden das Gerüst für die DTD's der jeweiligen Eingabesprachen. Wie dieses Gerüst für die einzelnen Sprachen zu einer DTD, die die Grammatik einer konkreten

Eingabesprache der XAM vollständig beschreibt, erweitert wird, stelle ich im letzten Abschnitt vor. Exemplarisch ist in Anhang B die vollständige DTD für die Eingabesprache angegeben, mit der die giMLi-Präsentation beschrieben werden kann.

7.11.1 Struktur der Eingabe

Die Eingabesprache gliedert sich in vier unterschiedliche Bereiche, deren Einteilung sich grob an den verschiedenen Aspekten einer ξML-Präsentation orientiert, die im Rahmen der vier Dimensionen von Multimediadokumenten identifiziert wurden. Ein XaML-Dokument besteht aus den Bereichen `structure`, `navigation`, `interaction` und `layout` (siehe Grammatikfragment 7.10 auf der nächsten Seite). Diese als XML-Elemente modellierten Bereiche besitzen keine eigenen Attribute, da sie nur als Container für die XML-Elemente dienen, mit denen die ξML-Präsentationen beschrieben werden. Der `structure`-Bereich beinhaltet die Instanzen der Spezialisierungen des *Box*-Primitivs und entsprechend den oben vorgestellten Enthaltenseinsbeziehungen auch die Instanzen der Spezialisierungen weiterer Basisprimitive. Mit ihm wird daher der logische Aufbau und die zugehörigen darzustellenden Inhalten der zu beschreibenden ξML-Präsentation angegeben. Im `navigation`-Bereich werden die Instanzen der Spezialisierungen der Basisprimitive *Address* und *Link* angegeben, mit denen die Navigationsmöglichkeiten einer ξML-Präsentation modelliert werden. Die Interaktionsmöglichkeiten mit der zu beschreibenden ξML-Präsentation werden durch die Spezialisierungen der Basisprimitive *Action* und *Event* modelliert. Ihre Instanzen finden sich daher im `interaction`-Bereich wieder. Zur Definition des Aussehens einer ξML-Präsentation werden im `layout`-Bereich die Instanzen der Spezialisierungen des *Style*-Primitivs angegeben. Diese Unterteilung ermöglicht eine übersichtliche Erstellung der Zwischendarstellung, wie in Abschnitt 8.3 näher vorgestellt wird.

Um die Struktur des Eingabedokuments weitesgehend unabhängig von den abstrakten Elementen definieren und Anpassungen an konkrete ξML-Präsentationen bündeln zu können, definiere ich für jedes Basisprimitiv eine Liste in Form einer `Entity`-Definition. In diese Liste werden die für eine Eingabesprache einer ξML-Präsentation verwendeten abstrakten Elemente als Spezialisierungen der Basisprimitive eingetragen. Auf diese Weise können die einzelnen Listennamen als symbolische Referenz (`%<BASISPRIMITIVNAME>-spez;`) auf die jeweilige Menge der abstrakten Elemente verwendet werden. Eine solche Referenzierung wird bei der Definition der Struktur der Eingabe (siehe Grammatikfragment 7.10 auf der nächsten Seite) und bei der Definition der DTD-Fragmente der XAM-Komponenten in Kapitel 9 (siehe Beispielhaft für das abstrakte Element *SimpleBox* in Grammatikfragment 7.12 auf Seite 83) verwendet.

Alle Spezialisierungen eines Basisprimitivs werden als Menge von Alternativen, in DTD-Syntax durch ein '`|`' getrennt, in die Listen eingetragen. Stellvertretend ist das Fragment einer solchen Listendefinition namens `%box-spez` für das Basisprimitiv *Box* mit den beiden Spezialisierungen *SimpleBox* und *ScrollBox* in Grammatikfragment 7.11 auf der nächsten Seite angegeben. Für die anderen Basisprimitive sind die Listen entsprechend aufgebaut.

7.11.2 Modellierung von Assoziationen

In der Repräsentation der Eingabe der Schnittstelle des abstrakten Modells werden Assoziationen auf zwei unterschiedliche Arten modelliert: zum einen über symbolische Referenzen, zum

```
1  <!-- Wurzelelement -->
2  <!ELEMENT xaml (structure, navigation, interaction, layout) >
3    <!ATTLIST xaml>
4
5  <!-- die Struktur der Praesentation (logisch + Inhalt) -->
6  <!ELEMENT structure (%box-spez;)+ >
7    <!ATTLIST structure >
8
9  <!-- die Links der Praesentation -->
10 <!ELEMENT navigation ((%addr-spez;)*,(%link-spez;)*) >
11   <!ATTLIST navigation >
12
13 <!-- die moeglichen Ereignisse der Praesentation -->
14 <!ELEMENT interaction ((%event-spez;)*, (%action-spez;)*) >
15   <!ATTLIST interaction >
16
17 <!-- die typografischen Informationen fuer die Praesentation -->
18 <!ELEMENT layout (%style-spez;)* >
19   <!ATTLIST layout >
```

Grammatikfragment 7.10: Die Spezifikation der Struktur der Eingabesprachen

```
1  <!ENTITY % box-spez "simplebox | scrollbox">
```

Grammatikfragment 7.11: Listendefinition der Spezialisierungen des Basisprimitivs *Box*

anderen über Enthaltenseinsbeziehungen. Assoziationen, die zwischen Basisprimitiven bestehen, deren Instanzen ihrer Spezialisierungen innerhalb des `structure`-Bereichs des Eingabedokuments angegeben werden, werden über Enthaltenseinsbeziehungen modelliert, alle anderen mittels symbolischer Referenzen.

Die Enthaltenseinsbeziehungen werden als strukturelle Vorschrift in Element-Definitionen einer DTD, also auf Ebene der abstrakten Elemente, formuliert. Die Umsetzung einer solchen Vorschrift stelle ich exemplarisch am DTD-Fragment für das bereits oben erwähnte abstrakte Element *SimpleBox* vor (Grammatikfragment 7.12 auf der nächsten Seite): Als Spezialisierung des Basisprimitivs *Box* besitzt es als Enthaltenseinsbeziehungen modellierte Assoziationen zu Spezialisierungen der Basisprimitive *Medium, Position* und *Box* (siehe Abschnitt 7.1). Unter Verwendung der oben vorgestellten Listen von Spezialisierungen von Basisprimitiven und entsprechenden Kardinalitäten wird dies in der Element-Definition angegeben. Ein validierender Parser überprüft beim Einlesen der Eingabedatei die Einhaltung dieser Regel und beendet bei Verstößen, etwa durch die Angabe falsch getypter Instanzen innerhalb einer *Simplebox*-Instanz oder falscher Kardinalitäten, wohldefiniert den Einlesevorgang. Die Verwendung einer als Enthaltenseinsbeziehung modellierten Assoziation in einem Eingabedokument kann also durch den validierenden Xerces-Parser auf ihre Korrektheit überprüft werden: ein erfolgreiches Parsen garantiert für die Codeerzeugung, dass das Ziel einer solche Assoziation existiert und vom richti-

gen Typ ist.

```
1 <!ELEMENT simplebox ((%pos−spez;), (%medium−spez;), (%box−spez;)∗) >
2   <!ATTLIST simplebox
3       %box−attr;
4       <!−− ... −−>
5   >
```

Grammatikfragment 7.12: Enthaltenseinsbeziehungen am Beispiel des abstrakten Elements *SimpleBox*

Die Modellierung von Assoziationen mittels symbolischer Referenzen wird auf der Ebene der Basisprimitive mit Hilfe von Attributen formuliert: Jedes Basisprimitiv besitzt eine global eindeutige Kennung als Attribut `id`. Eine Assoziation wird durch ein Attribut angegeben, in dem die Referenz auf eine solche Kennung abgelegt ist. Beispielsweise wird die Assoziation des Basisprimitivs *Medium* zu einem *Style*-Primitiv als symbolische Referenz modelliert (siehe Abschnitt 7.2). Daher besitzt das *Medium*-Primitiv ein Attribut `style`, in dem die Kennung des zu referenzierenden *Style*-Primitivs abgelegt wird (siehe Grammatikfragment 7.2 auf Seite 73). Bei der Verwendung einer so modellierten Assoziation in einem Eingabedokument, kann ein validierender Parser feststellen, ob das Element, das als Ziel der symbolischen Referenz angegeben wurde, eindeutig ist und existiert. Dazu werden die Kennung und die Referenz als XML-Typ `ID` bzw. `IDREF` definiert. Der Wert eines Attributs des Typs `ID` muss dokumentenweit einmalig sein. Ein als `IDREF`-typisiertes Attribut darf nur einen Wert des Typs `ID` referenzieren, der in dem Dokument auch verwendet wird. Bei Verstößen gegen eine der beiden Regeln bricht ein validierender Parser das Einlesen eines Dokumentes wohldefiniert ab. Liest der Xerces-Parser also ein Dokument fehlerfrei ein, ist für die Codegenerierung garantiert, dass die Vorwärtsreferenz aufgelöst werden kann, also die symbolische Referenz auf ein eindeutiges Ziel verweist, das auch existiert. Eine statische Typüberprüfung des Ziels einer Assoziation beim Parsen ist bei dieser Umsetzung einer Assoziation im Gegensatz zu der oben vorgestellten Umsetzung als Enthaltenseinsbeziehung nicht möglich. Sie muss in die Codegenerierung verschoben werden. Ein entsprechender Mechanismus wird in Abschnitt 8.3 vorgestellt.

7.11.3 Erweiterung für konkrete Eingabesprache

Im Gegensatz zu der Repräsentation der Schnittstelle des abstrakten Modells für die Elemente der Zwischendarstellung, bei der eine Spezialisierung der Basisprimitive durch die abstrakten Elemente über einen Vererbungsmechanismus realisiert wird, sind die zu verwendenden Mechanismen in der Repräsentation für die Eingabesprache nicht so offensichtlich.

Ein abstraktes Element spezialisiert ein Basisprimitiv und soll daher auch die für das Basisprimitiv als Attribute in der Entity-Definition festgelegen Eigenschaften besitzen. Eine solche Vererbung kann in XML realisiert werden, in dem gefordert wird, das ein jedes abstrakte Element in der Attlist-Definition seines DTD-Fragments eine Referenz auf die Entity-Definition des zu spezialisierenden Basisprimitivs besitzen muss. Wie in Grammatikfragment 7.12 am Beispiel des DTD-Fragments für das abstrakten Elements *SimpleBox* zu sehen ist, findet sich dort in Zei-

le 3 die Referenz auf die Entity-Definition seines Basisprimitivs *Box*. Bei der Definition des DTD-Fragments jedes abstrakten Elements muss diese Vorschrift berücksichtigt werden.

Um eine Grammatik für eine konkrete Eingabesprache festzulegen, also eine vollständige DTD zu definieren, die dieser Sprache zu Grunde liegt, müssen neben den oben vorgestellten DTD-Fragmenten des abstrakten Modells entsprechende Einträge für die konkret verwendeten abstrakten Elemente zusammengefügt werden. Für jedes abstrakte Element muss dazu ein Eintrag in der entsprechenden Liste des spezialisierten Basisprimitivs erzeugt und das DTD-Fragment, das die Struktur des abstrakten Elements spezifiziert, an die DTD angehängt werden.

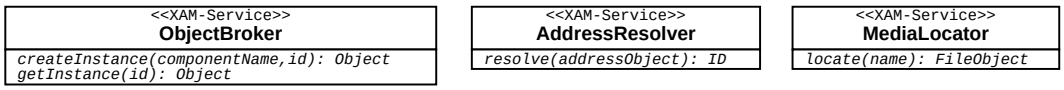
Der Mechanismus wird wiederum am Beispiel des Basisprimitivs *SimpleBox* verdeutlicht. Es wird der entsprechende Eintrag `simplebox` in der Liste der *Box*-Spezialisierungen eingefügt (siehe Grammatikfragment 7.11 auf Seite 82) und sein DTD-Fragment (siehe Grammatikfragment 7.12 auf der vorherigen Seite) an die DTD angehängt. Dieser Vorgang ist für jedes abstrakte Element zu wiederholen.

8 Dienste

In diesem Kapitel werden die Dienste der XAM beschrieben. Ein Dienst stellt Systemfunktionalität bereit, die die XAM zum Darstellen von ξML-Präsentationen anbietet. Die Dienste der XAM gliedern sich zum einen in Dienste, die allgemeine Funktionalität bereitstellen, die von anderen Diensten genutzt wird. Diese Dienste werden unter dem Begriff Infrastruktur zusammengefasst. Zum anderen existieren Dienste, die die einzelnen Phasen der XAM unterstützen: die Initialisierung, der Aufbau der Zwischendarstellung und die Codeerzeugung. Gemäß des diesem Entwurf zu Grunde liegenden Mircokernel-Musters besitzt die XAM initial nur minimale Dienste. Weitere Funktionalität zur Darstellung von ξML-Präsentationen wird in den XAM-Komponenten definiert.

8.1 Infrastruktur

Mit der Infrastruktur werden Dienste definiert, die zur Realisierung der einzelnen Phasen der XAM benötigt werden. Sie reichen vom Erzeugen von Objekten bis hin zum Auffinden von Mediendateien. Diese Dienste werden als Schnittstellen definiert, für die entsprechende Implementierungen bereitgestellt werden müssen. Darauf gehe ich bei den einzelnen Dienste näher ein. Die einzelnen Schnittstellen der Infrastrukturdienste sind als Klassen in UML-Diagramm 8.1 dargestellt.



UML-Diagramm 8.1: Schnittstellen der allgemeinen Dienste der XAM

8.1.1 Erzeugen und Auffinden von Instanzen

Die XAM muss einen Dienst zur Verfügung stellen, der den Umgang mit den in der XAM verwendeten Objekten regelt. Dieser Dienst wird mit dem so genannten `ObjectBroker` realisiert. Er stellt zwei Funktionalitäten bereit:

- Das Erzeugen einer Instanz eines Objekts mit einer festzusetzenden Kennung. Dazu wird eine Methode namens `createInstance` angeboten. Sie bekommt den Namen der XAM-Komponente, zu der das zu instantiierende Objekt gehört, und die Kennung, die der Instanz zugeordnet wird, übergeben und liefert eine Referenz auf die gewünschte Instanz

zurück. Jede erzeugte Instanz wird unter seiner Kennung in eine geeignete Datenstruktur, etwa in ein Dictionary, eingetragen.

- Das Zuordnen einer Kennung zu der entsprechenden Instanz. Dazu wird eine Methode namens `getInstance` angeboten, die eine Kennung übergeben bekommt und unter Verwendung der oben gewählten Datenstruktur eine Referenz auf das entsprechende Objekt zurückliefert.

Mit einem solchen `ObjectBroker` wird zum einen der Erzeugungsprozess von Objekten gekapselt, der aus dem Auffinden und Instantiieren der entsprechenden Dateien der XAM-Komponenten besteht. Zum anderen realisiert der `ObjectBroker` die Zuordnung von symbolischen Referenzen zu den Hauptspeicherrepräsentationen der Objekte.

Für die einzelnen Objekttypen, die im Rahmen der XAM benötigt werden, etwa die abstrakten Elemente oder die Codeerzeuger, werden nach dem Entwurfsmuster Abstrakte Fabrik (siehe Seite 107 in [GAMMA et al. 1996]) entsprechende Implementierungen der `ObjectBroker`-Schnittstelle bereitgestellt, so dass es für jeden Objekttyp eine eigene Fabrik gibt. Unter dem Namen der XAM-Komponente wird je nach Fabrik das entsprechende Objekt erzeugt. Um den Umgang mit den Referenzen der Objekte für die Codeerzeugung übersichtlicher zu gestellten, besitzen die Instanzen der unterschiedlichen Elemente einer XAM-Komponente alle die gleiche Kennung. Die unterschiedlichen Fabriken liefern unter der Id die Instanzen der entsprechenden Typen zurück.

Wie eine Realisierung der vom `ObjectBroker` bereitgestellten Funktionalität in den einzelnen Implementierungen mit der Programmiersprache Java vorgenommen werden kann, wurde bereits im Rahmen der Abschnitte 6.3 und 6.4 skizziert.

8.1.2 Auflösen von Adressen

Ein weiterer Dienst, den die XAM bereitstellen muss, ist das Auflösen von den im abstrakten Modell definierten Adressen. Dazu wird eine `AddressResolver`-Schnittstelle, die eine `resolve`-Methode besitzt, definiert. Sie bekommt eine Instanz einer Adresse übergeben und liefert die eindeutige Kennung der Instanz des adressierten Elements zurück. Da nur abstrakte Elemente mit diesem Mechanismus adressiert werden, kann damit über eine `ObjectBroker`-Fabrik für abstrakte Elemente die zur Kennung gehörende Referenz auf die Instanz des abstrakten Elements erhalten werden.

Zur Unterstützung der unterschiedlichen Adressierungschemen müssen entsprechende Implementierungen der `AddressResolver`-Schnittstelle bereitgestellt werden. Gemäß des `addrtype`-Attributs eines *Link*-Primitivs wird eine entsprechende Implementierung zum Auflösen der Adresse benutzt.

8.1.3 Zugriff auf Medien

In einer ξML-Präsentation werden Informationen dargestellt, die in Form von Medien vorliegen und der XAM zur Codegenerierung geeignet zugänglich sein müssen. Eine `MediaLocator`-Schnittstelle besitzt dazu die Methode `getMedia`, die den Zugriff auf solche Medien von der

Art und Weise entkoppelt, wie sie der XAM zur Verfügung gestellt werden. Die Methode erwartet den Namen eines Medienelements und liefert ein entsprechendes Dateiobjekt zurück.

Je nachdem wie die Medien der XAM zur Verfügung gestellt werden, ob sie beispielsweise in einer Datenbank oder an geeigneten Stellen im Dateisystem vorliegen, müssen entsprechende Implementierungen der `MediaLocator`-Schnittstelle bereitgestellt werden.

8.1.4 Bereitstellen der Diensteimplementierungen

Die oben geschilderten Dienste sind als Schnittstellen definiert, für die unterschiedliche Implementierungen bereitgestellt werden müssen. Dazu hat man zwei Möglichkeiten: Entweder während der Entwicklungszeit der XAM oder bei jedem Start der XAM neu. Die zweite Alternative ist deutlich flexibler, da sie die Unterstützung neuer Adressierungschemen oder neuer Arten, Medien der XAM zur Verfügung zu stellen, auch nach der Entwicklungszeit der XAM ermöglicht. Daher ziehe ich sie vor.

Es müssen somit bei jedem Start der XAM zunächst die Implementierungen der Infrastrukturdienste bereitgestellt werden, ehe die XAM mit den XAM-Komponenten konfiguriert werden kann. Diese Dienste können analog zu den XAM-Komponenten als so genannte *XAM-Systemkomponente* betrachtet und mittels eines JavaBean distributiert werden. Um die Lauffähigkeit der XAM zu gewährleisten, muss eine XAM-Systemkomponente mindestens eine Implementierung der `AddressResolver`-Schnittstelle, eine Implementierung der `MediaLocator`-Schnittstelle und eine Implementierung der `ObjectBroker`-Schnittstelle für jede in der XAM verwendete Objektart, wie beispielsweise eine für die abstrakten Elemente, beinhalten. Entsprechende Regeln finden sich im Ausschnitt der DTD der Manifestdatei für die Systemkomponente in Grammatikfragment 8.1 wieder. Die Gesamt-DTD mit einer vollständigen Definition des Lieferumfangs einer Systemkomponente und aller ihrer Bestandteile findet sich in Anhang A.

```
1 <!ELEMENT systemcomponent (objectbroker+, addressresolver+, medialocator+) >
2   <!ATTLIST systemcomponent>
```

Grammatikfragment 8.1: Bestandteile der Systemkomponente

Diese Modellierung folgt klar dem Gedanken des Mircokernelmusters: die Dienste, die die XAM initial zur Verfügung stellt, ist minimal. Im Wesentlichen bietet die XAM initial nur die Möglichkeit, die XAM-Systemkomponente zu registrieren. Nach dieser Registration sind weitere Dienste initialisiert, die die Konfiguration der XAM mit den XAM-Komponenten ermöglichen.

8.2 Initialisierung

Der Initialisierungsdienst arbeitet in zwei Schritten. Zunächst werden die Mechanismen des zweiten Initialisierungsschritts mit den in der XAM-Systemkomponente ausgelieferten Implementierungen initialisiert. Im zweiten Schritt wird die XAM mit den XAM-Komponenten der darzustellenden ξML-Präsentation konfiguriert. Dazu müssen zunächst die Dateien, die die XAM-

Komponenten beinhalten, an einer geeigneten Stelle abgelegt werden, so dass die im ersten Initialisierungsschritt bereitgestellten `ObjectBroker`-Implementierungen auf sie zugreifen können. Die Implementierungen können dann selbst feststellen, welche XAM-Komponenten bereitstehen. Die für diese beiden Schritte benötigte Funktionalität stellt, wie oben geschildert, Java zur Verfügung.

Desweiteren muss der Initialisierungsdienst aus den mit den XAM-Komponenten gelieferten Informationen die Grammatik der Eingabesprache erzeugen und damit den Parser konfigurieren. Dazu stellt er die DTD der Sprache nach dem in Abschnitt 7.11.3 vorgestellten Schema mit den einzelnen Fragmenten zusammen. Die DTD-Fragmente sind über die XAM-Komponenten zugänglich. Die Information, welches Basisprimitiv von einer XAM-Komponenten spezialisiert wurde, kann über den Introspektionsmechanismus aus dem mit der Komponente ausgelieferten abstrakten Element in Erfahrung gebracht werden.

Sind auf diese Art der Parser initialisiert und die XAM-Komponenten für die `ObjectBroker` verfügbar, kann die Verarbeitung des Eingabedokuments beginnen. Mit den hier vorgestellten Mechanismen wird somit die in Anforderung 16 auf Seite 45 verlangte Initialisierung der XAM mit der benötigten multimedialen Funktionalität, sprich den XAM-Komponenten, ermöglicht.

8.3 Aufbau der Zwischendarstellung

Aufgabe beim Aufbau der Zwischendarstellung ist es, das Eingabedokument in eine entsprechende Hauptspeicherrepräsentation zu überführen. Die Zwischendarstellung besteht aus zwei unterschiedlichen Strukturen: Zum einen aus einer baumartigen Struktur, die der Struktur der im `structure`-Bereich des Eingabedokuments abgelegten Elemente entspricht, und zum anderen aus einer flachen Struktur, in der die Elemente der übrigen Bereiche des Eingabedokuments nebeneinander liegen.

Der Aufbau wird durch den Parser, der das Eingabedokument einliest, gesteuert. Wie in Abschnitt 6.4 beschrieben, erzeugt der Xerces-Parser während des Einlesens beim Erreichen und Verlassen eines Elements des Eingabedokuments Methodenaufrufe, mit denen der Aufbau der Zwischendarstellung angestoßen werden kann. Über die DOM- und SAX-Programmierschnittstellen besteht die Möglichkeit, auf Informationen des zu verarbeitenden Elements zuzugreifen.

Jedes so gefundene Element des Eingabedokuments muss in seine Repräsentation der Zwischendarstellung überführt werden. Dazu werden drei Schritte durchgeführt:

- Erzeugen einer Instanz des abstrakten Elements, das das entsprechende Element des Eingabedokuments im Hauptspeicher repräsentiert. Dabei wird die Id des Elements der Eingabedatei bei der Instanz des abstrakten Elements gesetzt.
- Initialisieren der Instanz des abstrakten Elements mit den Daten des Elements der Eingabedatei. Dabei müssen die Werte der korrespondierenden Attribute gesetzt werden.
- Ablegen der Instanz in der jeweiligen Struktur im Hauptspeicher.

Die Instantiierung der entsprechenden abstrakten Elemente wird mit der oben beschriebenen `ObjectBroker`-Implementierung für abstrakte Elemente ermöglicht. Eine so erhaltene Instanz muss mit den Informationen des Elements des Eingabedokuments initialisiert werden. Da die

Informationen in den Attributen modelliert sind und diese mit den Attributen der entsprechenden abstrakten Elemente auf Grund der Konstruktion der Schnittstellen übereinstimmen, können über Namenskonventionen bzw. Introspektion ihre Inhalte gesetzt werden.

Für die Repräsentationen in der Zwischendarstellung von Elementen, die nicht im `structure`-Bereich des Eingabedokuments definiert sind, wird keine besondere Datenstruktur benötigt. Sie werden bei ihrer Instantiierung automatisch im Dictionary der `ObjectBroker`-Implementierung hinterlegt und sind mit ihrer `Id` über die `getInstance`-Methode verfügbar. Für die Instanzen in der Zwischendarstellung der Elemente, die innerhalb des `structure`-Bereichs des Eingabedokuments angegeben sind, wird eine Baumstruktur im Hauptspeicher aufgebaut, die der im Eingabedokument über die Enthaltenseinsbeziehungen modellierten Struktur entspricht. Der Aufbau einer solchen Baumstruktur kann als ein in der software-technologischen Welt gelöstes Problem angesehen und ihre detaillierte Gestaltung daher einer Implementierung überlassen werden.

In der Zwischendarstellung werden also die Assoziationen, die über Enthaltenseinsbeziehungen modelliert sind, als Java-Referenzen realisiert. Solche Assoziationen finden sich nur bei Elementen, die im `structure`-Bereich des Eingabedokuments angegeben werden. Es handelt sich dabei um die drei Arten von Assoziationen, die von den Spezialisierungen des *Box*-Primitivs ausgehen: Beliebig viele (`0...n`) zu weiteren Spezialisierungen des *Box*-Primitivs, über die die oben erwähnte baumartige Struktur gebildet wird, eine zu einer Spezialisierung des *Position*-Primitivs und eine zu einer Spezialisierung des *Medium*-Primitivs. Alle anderen durch das abstrakte Modell vorgegebenen Assoziationen werden auch in der Zwischendarstellung über symbolische Referenzen modelliert, die mit Hilfe der `ObjectBroker`-Implementierung verfolgt werden können.

Da beim Aufbau der Zwischendarstellung die in Abschnitt 7.11.2 vorgestellten Prüfmechanismen des validierenden Parsers verwendet werden, garantiert ein erfolgreiches Einlesen des Eingabedokuments und somit der erfolgreiche Aufbau der Zwischendarstellung für die Codegenerierung, dass

- die über Java-Referenzen realisierten Assoziationen ein existierendes Ziel haben, das auch vom richtigen Typ ist.
- die über symbolische Referenzen realisierten Assoziationen auf ein existierendes Ziel verweisen.

Die Überprüfung der Korrektheit des Typs des referenzierten Elements bei symbolischen Referenzen hingegen muss während der Codegenerierung vorgenommen werden. Dies geschieht, wenn über die `getInstance`-Methode der `ObjectBroker`-Implementierung auf die Instanz eines abstrakten Elements zugegriffen wird. Die zurückgegebene Referenz der Instanz muss auf den konkreten Typ des abstrakten Elements gecastet werden. Schlägt das Casten fehl, liegt ein Typfehler vor und die Codegenerierung wird wohldefiniert beendet, etwa durch eine geeignete Ausnahme-Behandlung mit Java.

Informationen der Elemente, die über den Schlüssel-Wert-Mechanismus in einem *Pair*-Element (DTD-Fragment siehe Anhang B, Zeile 146 ff) angegeben sind, wie etwa bei Spezialisierungen des *Style*-Primitivs, werden mittels Java-Referenzen an die Instanz des entsprechenden abstrakten Elements gehängt. Werden die *Pair*-Elemente geschachtelt verwendet, werden sie in derselben Struktur beim entsprechenden Element abgelegt. Bei der Codegenerierung weiß der

zum abstrakten Element gehörige Codeerzeuger, wie er die in den *Pair*-Elementen angegeben Informationen zu verarbeiten hat. Diese Informationen werden unabhängig von der Struktur, in der das abstrakte Element in der Zwischendarstellung abgelegt wird, an das entsprechende Element gehängt. Zur Erzeugung ihrer Instanzen kann die `ObjectBroker`-Implementierung für abstrakte Elemente verwendet werden. Wie schon erwähnt, liegen diese *Pair*-Elemente außerhalb des Typsystems der XAM und werden außer von den Codeerzeugern nie ausgewertet.

Mit dem hier geschilderten Dienst wird somit ein Eingabedokument eingelesen und die Zwischendarstellung aufgebaut, wie in Anforderung 14 auf Seite 44 verlangt. Die so erzeugte Zwischendarstellung kann dann zur Codeerzeugung verwendet werden.

8.4 Codeerzeugung

Nach dem Aufbau der Zwischendarstellung wird von der XAM der Code für die entsprechenden Zielplattformen erzeugt. Wie bereits diskutiert, unterscheiden sich die einzelnen Plattformen zu sehr voneinander, so dass kein allgemeiner Dienst angeboten werden kann, der eine einheitliche Durchlaufstrategie bei der Codeerzeugung für jede beliebige Zielplattform bereitstellt. Es wird daher für jede Zielplattform ein Framework definiert, das die jeweilige Durchlaufstrategie festlegt. An den Hotspots des Frameworks werden die mit den XAM-Komponenten ausgelieferten Codeerzeuger aufgerufen, um die entsprechenden Codefragmente des Zielformats zu erzeugen. Ein solches Framework wird exemplarisch für das Format HTML im Rahmen der giMLi-Präsentation in Abschnitt 10.3 näher vorgestellt. Bei der Codegenerierung werden aber weitere Dienste benötigt, die unabhängig von einer Zielplattform sind.

Zum einen müssen den Frameworks die entsprechenden Instanzen der Codeerzeuger zur Verfügung gestellt werden. Als Fabrik dient hierfür eine `ObjectBroker`-Implementierung für die Codeerzeuger. Mit ihrer `createInstance`-Methode werden die entsprechenden Codeerzeuger instantiiert.

Des Weiteren kann es vorkommen, dass für die Codeerzeugung Informationen über ein abstraktes Element benötigt werden, die nicht an Hand seines Typs ablesbar sind. Falls solche Informationen für eine Zielplattform benötigt werden, werden bei der Erstellung des Frameworks der Plattform für die abstrakten Elemente Schnittstellen definiert, die den Zugriff auf die benötigten Informationen ermöglichen. Jede XAM-Komponente muss, falls sie in der entsprechenden Zielplattform darstellbar sein will, eine Implementierung dieser Schnittstelle bereitstellen. Neben den Codeerzeugern für die jeweilige Plattform kann somit zusätzlich ein entsprechendes Metadatenobjekt als Implementierung der obigen Schnittstelle mit zum Lieferumfang einer XAM-Komponente gehören (siehe Anhang A, Zeile 45ff). Mit einem solchen Metadatenobjekt können gleichförmig Informationen bei den Instanzen der unterschiedlichen abstrakten Elemente abgelegt werden. Wird das Wissen über das Vorhandensein von Eigenschaften benötigt, liefert eine Implementierung etwa den entsprechenden booleschen Wert zurück. Bei komplexeren Informationen können die Implementierungen auch anspruchsvoller sein.

Der Metadatenobjekt-Mechanismus kann auch dazu genutzt werden, um Informationen, die sich während der Codegenerierung etwa aus der Analyse des Eingabedokuments ergeben, bei einem abstrakten Element abzulegen. Dazu müssen bei der Erstellung des Frameworks entsprechende Methoden in der Metadaten-Schnittstelle definiert werden. Eine Verwendung dieses Me-

chanismus wird im Rahmen des oben angekündigten Beispiels in Abschnitt 10.3 gegeben.

Da, wie bei einem Codeerzeuger, eine solche Implementierung der Metadaten-Schnittstelle je Zielplattform für jede XAM-Komponente definiert und mit ihr distributiert wird, wird ihre Instanz über eine Implementierung der `ObjectBroker`-Schnittstelle für die Metadatenobjekte erzeugt.

Zur Codegenerierung benötigen die Codeerzeuger neben den in den abstrakten Elementen abgelegten Daten und auch Informationen über die Umgebung bzw. den aktuellen Zustand, in dem sich Codegenerierung befindet. Sie können entweder vorab in dem Metadatenobjekt des abstrakten Elements abgelegt oder erst beim Aufruf der Codeerzeugung gesammelt werden. Diese Informationen müssen dem Codeerzeuger neben der Instanz des abstrakten Elements, für das er gerade Code generieren soll, übergeben werden.

Um dies umzusetzen hat man zwei Alternativen: Entweder definiert man eine allgemeine Codeerzeugerschnittstelle, die für alle Codeerzeuger gleich ist. Sie besteht aus einer `generateCode`-Methode, die ein abstraktes Element und ein allgemeines Parameterobjekt übergeben bekommt, bei dem ungetypt die einzelnen Parameter abgelegt sind. Oder man fordert eine Schnittstelle, die auch aus einer `generateCode`-Methode besteht, aber neben einem abstrakten Element eine Reihe weiterer getypter Attribute erwartet, die für jedes Element unterschiedlich seien können. So schön einheitliche Schnittstellen auch sind, die Variante der getypten Parameter scheint aus einer software-technologischen Sicht jedoch weniger fehleranfällig. Für Codeerzeuger wird daher gefordert, dass sie eine `generateCode`-Methode implementieren, die auf jeden Fall eine Instanz eines abstrakten Elements als Parameter erwartet. Benötigt ein Codeerzeuger weitere Parameter, werden sie der Parameterliste getypt hinzugefügt.

Mit dem Dienst der Codegenerierung werden somit die Konzepte der Eingabe in die entsprechenden Konzepte der Zielplattform überführt und Anforderung 15 auf Seite 45 erfüllt.

9 XAM-Komponenten für giMLi

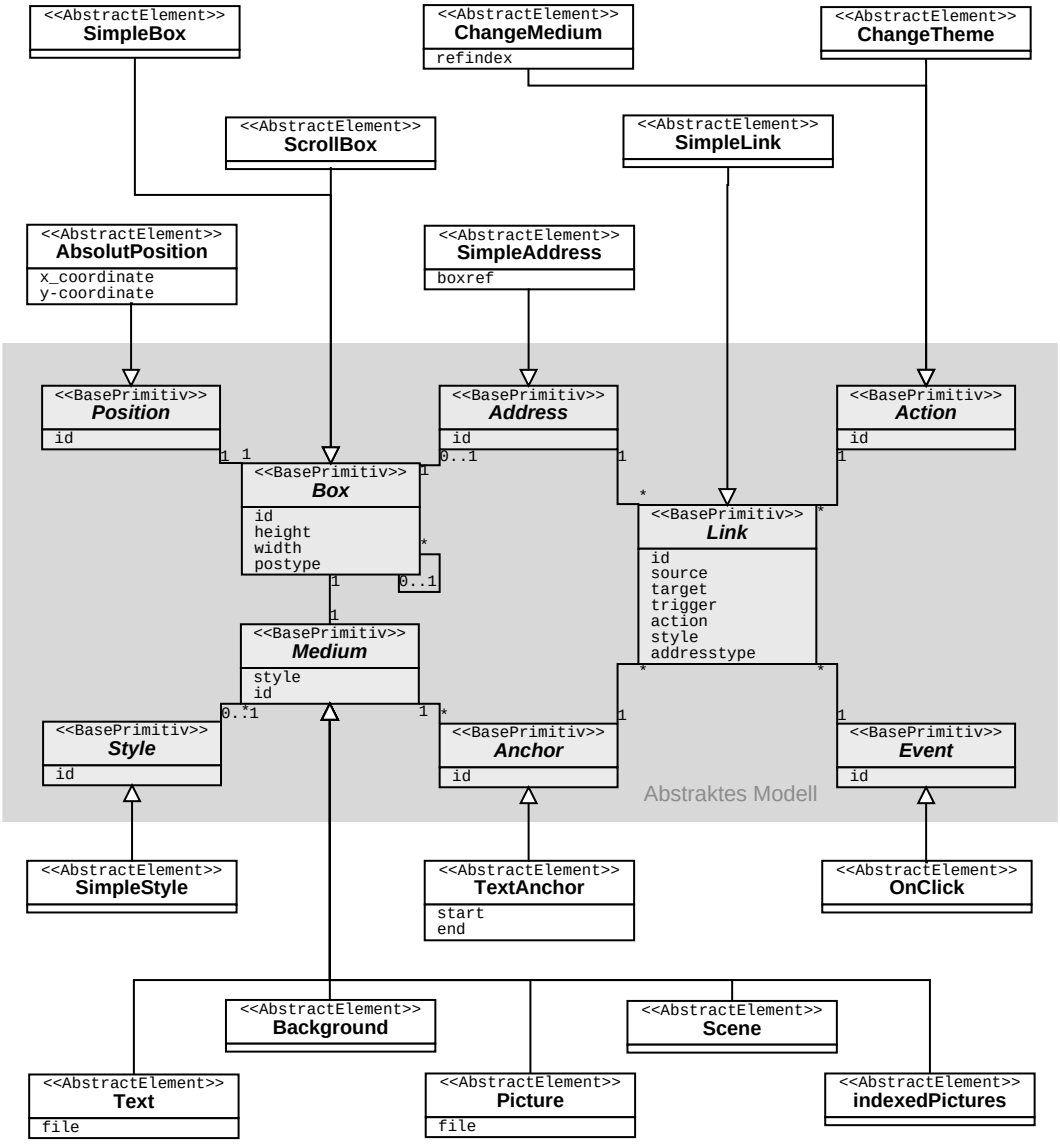
In diesem Kapitel stelle ich die XAM-Komponenten vor, die zur Darstellung der giMLi-Präsentation benötigt werden, die in dieser Arbeit als Beispiel einer ξML-Präsentation näher betrachtet wird. Die XAM-Komponenten sind aus den in Abschnitt 5.2.6 aufgestellten Anforderungen abgeleitet.

Eine XAM-Komponente besteht im wesentlichen aus drei Elementen: einem DTD-Fragment, das die Repräsentation der Komponente im Eingabedokument definiert, einem abstrakten Element als Spezialisierung eines Basisprimitivs und Softwareartefakte wie die Codeerzeuger und Metadatenobjekte, die zur Generierung des Codes für eine entsprechende Zielplattform benötigt werden. Im Rahmen dieser Arbeit wird die Codegenerierung exemplarisch für das Zielformat HTML vorgestellt. Um eine bessere Lesbarkeit zu gewährleisten, verwende ich bei der Vorstellung der einzelnen XAM-Komponenten den Komponentennamen stellvertretend für die einzelnen mit einer XAM-Komponente ausgelieferten Elemente. Welches Element einer Komponente dann jeweils gemeint ist, ergibt sich aus dem Zusammenhang.

An Hand dieser drei Aspekte werden die einzelnen XAM-Komponenten, die zur Beschreibung der giMLi-Präsentation benötigt werden, vorgestellt. Die mit den XAM-Komponenten definierten abstrakten Elemente werden zusammenhängend in UML-Diagramm 9.1 auf der nächsten Seite als Klassen dargestellt. Zur besseren Übersichtlichkeit ist dort die Assoziation zwischen dem *Style*- und dem *Link*-Primitiv ebenso weggelassen worden, wie die Benennung der von dem *Link*-Primitiv ausgehenden Assoziationen. Ihr DTD-Fragment und eine Abbildung in die entsprechende Konzepte des Zielformats HTML wird für jede Komponente einzeln vorgestellt. Um die Verwendung der XAM-Komponenten zur Beschreibung der giMLi-Präsentation exemplarisch vorzustellen, habe ich in Anhang C beispielhaft für ein Thema das Eingabedokument angegeben. Der Gebrauch der einzelnen XAM-Komponenten kann dort nachgeschlagen werden.

9.1 Die XAM-Komponente *SimpleBox*

Die *Simplebox* stellt einen rechteckigen Bereich auf dem Ausgabemedium dar, in dem Medienelemente angezeigt werden. Sie ist eine Spezialisierung des Basisprimitivs *Box*, die die mit einer *Box* modellierten Eigenschaften implementiert, aber keine neue Eigenschaften hinzufügt (siehe Grammatikfragment 9.1 auf Seite 94). Die *Simplebox* besitzt somit alle Fähigkeiten, die auch für das *Box*-Primitiv angegeben wurden, wie etwa die Möglichkeit zur Gruppierung. Angaben zu Layoutaspekten einer *Simplebox*, die unabhängig von den in ihr dargestellten Medienelementen sind, können über eine geeignete Attributierung vorgenommen werden. Wie schon bei der Vorstellung des Basisprimitivs *Box* erwähnt, verzichte ich aus Gründen der Übersichtlichkeit im Rahmen dieser Arbeit auf die Definition solcher Layoutattribute.



UML-Diagramm 9.1: Die abstrakten Elemente der XAM

```
1 <!ELEMENT simplebox ((%pos−spez;), (%medium−spez;), (%box−spez;)∗) >
2   <!ATTLIST simplebox
3       %box−attr;>
```

Grammatikfragment 9.1: DTD-Fragment der XAM-Komponente *SimpleBox*

Eine *SimpleBox*-Komponente kann durch zwei unterschiedliche Konzepte in HTML realisiert werden: Entweder als eine Tabelle oder als ein Frame innerhalb einer Frameset-Datei. Mit beiden Konzepten werden positionierbare Bereiche definiert, die weitere Inhalte aufnehmen können. Da diese Inhalte sowohl Medienelemente als auch weitere positionierbare Bereiche sein können, erlauben sie zum einen die Anordnung von Medienelementen, zum anderen deren Komposition durch die mögliche Schachtelung. Für beide Konzepte können über die Attribute ihrer HTML-Tags Informationen zur Positionierung und zum Layout angegebene werden. Welches der beiden Konzepte verwendet wird, hängt von den Elementen ab, die sie beinhalten. Diese Entscheidung muss für jede *Simplebox* im Rahmen der Codegenerierung getroffen werden. Wie dies geschieht, wird in Abschnitt 10.3 beschrieben.

9.2 Die XAM-Komponente *ScrollBox*

Eine *ScrollBox*-Komponente spezialisiert ebenfalls das *Box*-Primitiv und besitzt dieselben Eigenschaften wie eine *SimpleBox* (siehe Grammatikfragment 9.2). Als weitere Fähigkeit können aber die in ihr dargestellten Inhalte gescrollt werden. Bei der giMLi-Präsentation wird der erläuternde Text eines Themas in einer solchen *ScrollBox* angegeben, da er auch größer als der für ihn vorgesehene Bereich auf dem Ausgabegerät sein kann.

```
1 <!ELEMENT scrollbox ((%pos−spez;), (%medium−spez;), (%box−spez;)∗) >
2   <!ATTLIST scrollbox
3       %box−attr;>
```

Grammatikfragment 9.2: DTD-Fragment der XAM-Komponente *ScrollBox*

Eine *ScrollBox* wird in HTML als Frame realisiert, bei dem das Attribut `scrolling` auf `yes` gesetzt ist. Weitere Informationen, etwa zur Positionierung oder zum Layout, können ebenso wie bei der *SimpleBox* mittels entsprechender Attribute des Frame-Elements angegeben werden.

9.3 Die XAM-Komponente *Text*

Die XAM-Komponente *Text* spezialisiert das *Medium*-Primitiv und modelliert einen Text, der in einer ξML-Präsentation dargestellt werden soll. Neben den durch die Spezialisierung geerbten Eigenschaften besitzt eine *Picture*-Komponente den Namen einer Datei, in dem der anzuzeigende unformatierte Text abgelegt ist (siehe Grammatikfragment 9.3 auf der nächsten Seite).

Formatierungen, wie etwa die zu verwendende Schriftart oder Schriftgröße, können über eine der *Text*-Komponente zugeordnete Spezialisierung des *Style*-Primitivs angegeben werden.

```
1 <!ELEMENT text (textanchor)* >
2   <!ATTLIST text
3       %medium−attr;
4       file  CDATA #REQUIRED>
```

Grammatikfragment 9.3: DTD-Fragment der XAM-Komponente *Text*

Eine *Text*-Komponente wird in HTML als einfacher Text realisiert, der in die entsprechenden Codefragmente der umgebenden Spezialisierung eines *Box*-Primitivs generiert wird. Die Realisierung der Formatierungseigenschaften wird in Abschnitt 9.9 mit der XAM-Komponente *SimpleStyle* vorgestellt.

9.4 Die XAM-Komponente *Picture*

Analog zu einem Text wird für ein Bild das Basisprimitiv *Medium* zu einer XAM-Komponente *Picture* spezialisiert. Ebenso wie die *Text*-Komponente besitzt sie die Eigenschaft eines Dateinamens, unter dem die Binärdaten des darzustellenden Bildes abgelegt sind (siehe Grammatikfragment 9.4). Der Farbverlauf zwischen den aufgeschlagenen Buchseiten eines Themas der giMLi-Präsentation (siehe Abbildung 5.1 auf Seite 36) wird mit einem durch die *Picture*-Komponente modellierten Bild realisiert.

```
1 <!ELEMENT picture EMPTY >
2   <!ATTLIST picture
3       %medium−attr;
4       file  CDATA #REQUIRED>
```

Grammatikfragment 9.4: DTD-Fragment der XAM-Komponente *Picture*

Eine *Picture*-Komponente wird in HTML als <img>-Tag umgesetzt, das in dem Codefragment der umgebenen Spezialisierung des *Box*-Primitivs angegeben wird. Der Dateiname des darzustellenden Bildes ist im `src`-Attribut des <img>-Tags anzugeben.

9.5 Die XAM-Komponente *IndexedPictures*

Eine weitere Spezialisierung des Basisprimitivs *Medium* ist die *IndexedPictures*-Komponente. Sie modelliert ein Medienelement, das aus mehreren Bildern besteht, die jeweils mit einer Bildunterschrift versehen sind. Von diesen Bildern wird immer nur eins angezeigt, die anderen sind nicht sichtbar. Bei der *IndexedPictures*-Komponente kann von außen gesteuert werden, welches Bild mit Bildunterschrift angezeigt wird. Dazu wird jedem Bild mit seiner Bildunterschrift ein Index zugeordnet. Die Informationen, welche Bilder mit welcher Bildunterschrift unter welchem

Index abgelegt sind, werden mit einer geschachtelten Verwendung der *Pair*-Hilfselemente modelliert. Eine *IndexedPictures*-Komponente besitzt keine weiteren zusätzlichen Eigenschaften, außer den übernommenen vom *Medium*-Primitiv und den *Pair*-Elementen.

```
1 <!ELEMENT indexedpictures (pair)+ >
2   <!ATTLIST indexedpictures
3       %medium−attr;>
```

Grammatikfragment 9.5: DTD-Fragment der XAM-Komponente *IndexedPictures*

Ein Medienelement mit einem solche Verhalten wird benötigt, um bei der giMLi-Präsentation die Anzeige der zu einem Thema gehörenden Bilder zu realisieren. Eine alternative Modellierung wäre mit der *Picture*- und *Text*-Komponente denkbar, die jeweils ein Bild und seine Unterschrift repräsentieren. Diese Alternative scheidet wegen zweier Modellierungsvorschriften des abstrakten Modells aus:

- Da einer *Box* nur ein *Medium* zugeordnet werden darf, ist es nicht erlaubt, mehrere Bilder mit ihren Bildunterschriften, die durch einzelne *Medium*-Primitive modelliert werden, dem selben Bereich auf dem Ausgabemedium zuzuordnen.
- Da *Box*-Primitive, die das selbe Eltern-*Box*-Primitiv besitzen, sich nicht überlappen dürfen, können die *Medium*-Primitive der Bilder und der Bildunterschriften nicht jeweils einzelnen *Box*-Primitiven zugeordnet werden, die den selben Bereich auf dem Ausgabemedium modellieren.

Bei der Realisierung der *IndexedPictures*-Komponente für HTML werden jeweils die einzelnen Bilder und ihre Bildunterschriften als <img>-Tags und einfacher Text in zweizeiligen Tabellen abgelegt. Die Namen der Dateien, in denen sich die Binärdaten der Medien befinden, werden aus den entsprechenden *Pair*-Elementen entnommen. Jede Tabelle wird an dieselbe Stelle positioniert und in einem -Tag-Bereich abgelegt, dem eine eindeutige Kennung zugeordnet ist, die dem im entsprechenden *Pair*-Attribut abgelegten Index entspricht. Jedes -Tag besitzt ein Attribut `visible`, mit dem die Inhalte des markierten Bereichs wahlweise sichtbar oder unsichtbar geschaltet werden können. Während der Darstellung der giMLi-Präsentation kann mit JavaScript, der Skriptsprache für HTML, über den Index auf das zugehörige -Element zugegriffen und der Wert des `visible`-Attributs verändert werden. Eine entsprechende JavaScript-Funktion ist bei der Codeerzeugung mitzugenerieren.

9.6 Die XAM-Komponente *Background*

Mit der XAM-Komponente *Background* wird eine Spezialisierung des Basisprimitivs *Medium* angeboten, die im Gegensatz zu den vorherigen *Medium*-Spezialisierungen nicht dazu dient, Informationen einer ξML-Präsentation darzustellen. Sie bietet vielmehr die Möglichkeit, Layoutinformationen für den von einer Spezialisierung des *Box*-Primitiv beschriebenen Bereich anzugeben, falls kein Medienelement in ihm dargestellt werden soll. Diese Layoutinformationen wer-

den benötigt, wenn weitere in ihm definierte Bereiche den umgebenen nicht vollständig überdecken und so Randflächen bleiben. Für diesen Fall wird eine *Background*-Komponente verwendet, um Layoutinformationen für solche Randflächen, wie etwa die Hintergrundfarbe, zu setzen. Dazu wird der für *Medium*-Primitive übliche *Style*-Mechanismus auch für die *Background*-Komponente verwendet. Sie definiert daher neben den vom Basisprimitiv *Medium* übernommenen keine weiteren Eigenschaften (siehe Grammatikfragment 9.6). Mit einer *Background*-Komponente wird in der giMLi-Präsentation etwa angegeben, dass die Ränder um die als *IndexedPictures*-Komponente modellierten Bilder auf der linken Seite der Präsentation die selbe Hintergrundfarbe besitzt wie *IndexedPictures*-Komponente.

```
1 <!ELEMENT background EMPTY >
2   <!ATTLIST background
3       %medium−attr;>
```

Grammatikfragment 9.6: DTD-Fragment der XAM-Komponente *Background*

Die mit der Komponente in einer entsprechenden Spezialisierung des *Style*-Primitivs definierten Eigenschaften werden in HTML als Belegung geeigneter Attribute bei der umgebenen *ScrollBox* bzw. *SimpleBox* realisiert, wie etwa durch Setzen des `bgcolor`-Attributs.

9.7 Die XAM-Komponente *Scene*

Eine weitere besondere Spezialisierung des *Medium*-Primitivs, die, neben der *Background*-Komponente, kein anzeigbares Medienelement modelliert, ist die XAM-Komponente *Scene*. Sie zeichnet alle in einer Spezialisierung des Basisprimitivs *Box*, wie etwa die *SimpleBox*, enthaltenen Medienelemente als komponiertes Medienelement aus. Da dadurch angegeben wird, dass alle Medienelemente innerhalb des einer *Scene*-Komponente zugeordneten Bereichs gleichzeitig dargestellt werden sollen, kann der seitenorientierte Aufbau einer ξML-Präsentation realisiert werden. Ein Thema in der giMLi-Präsentation ist ein Beispiel für die Verwendung der *Scene*-Komponente.

```
1 <!ELEMENT scene EMPTY >
2   <!ATTLIST scene
3       %medium−attr;>
```

Grammatikfragment 9.7: DTD-Fragment der XAM-Komponente *Scene*

Bei einer Umsetzung in HTML werden alle Inhalte, die innerhalb der durch die *Scene*-Komponente ausgezeichneten Spezialisierungen einer *Box* angegeben sind, in eine eigene Datei generiert.

9.8 Die XAM-Komponente *SimpleLink*

Eine XAM-Komponente *SimpleLink* ist eine Spezialisierung des *Link*-Primitivs. Sie implementiert die mit dem Basisprimitiv *Link* modellierten Fähigkeiten und stellt somit einen einfachen, unidirektionalen Link dar. Weitere Eigenschaften werden nicht definiert (siehe Grammatikfragment 9.8). Mit einer *SimpleLink*-Komponente werden die Beziehungen zwischen Medienelementen modelliert. Bei der giMLi-Präsentation sind das zum einen Beziehungen zwischen dem Text eines Themas und den entsprechenden Bildern und zum anderen zwischen den Einträgen des Inhaltsverzeichnisses und den anderen Themen.

```
1 <!ELEMENT simplelink EMPTY >
2   <!ATTLIST simplelink
3       %link-attr;>
```

Grammatikfragment 9.8: DTD-Fragment der XAM-Komponente *SimpleLink*

Die *SimpleLink*-Komponente wird in HTML durch das <a>-Tag realisiert. Die weiteren Informationen, die zur Generierung eines Links benötigt werden, sind in den Spezialisierungen der *Anchor*-, *Address*-, *Event*- und *Action*-Primitive abgelegt und werden dort erläutert.

9.9 Die XAM-Komponente *SimpleStyle*

Mit der *SimpleStyle*-Komponente wird das Basisprimitiv *Style* implementiert. Mit ihm können Layoutinformationen für Spezialisierungen der Basisprimitive *Medium* und *Link* mittels *Pair*-Elemente angegeben werden. Weitere Eigenschaften besitzt es nicht (siehe Grammatikfragment 9.9). Das Layout der Medienelemente und Links der giMLi-Präsentation wird durch entsprechende Instanzen der *SimpleStyle*-Komponente definiert.

```
1 <!ELEMENT simplestyle (pair)+ >
2   <!ATTLIST simplestyle
3       %style-attr;>
```

Grammatikfragment 9.9: DTD-Fragment der XAM-Komponente *SimpleStyle*

HTML bietet mit dem CSS-Mechanismus die Möglichkeit, *SimpleStyle*-Komponenten umzusetzen. Die in den *Pair*-Elementen abgelegten Layoutmerkmale und ihre Wertebelegungen können als entsprechende Paare in einer eigenen CSS-Datei abgelegt werden, die über das `class`-Attribut des Elements, zu dem die Instanz der *SimpleStyle*-Komponente gehört, referenziert wird. Alternativ können sie direkt beim Element unter dem `style`-Attribut als Aufzählung hinterlegt werden.

9.10 Die XAM-Komponente *AbsolutPosition*

Die XAM-Komponente *AbsolutPosition* spezialisiert das *Position*-Primitiv. Es definiert eine Position in einen absoluten zweidimensionalen Koordinatensystem. Das Koordinatensystem bezieht sich auf den Bereich, den das Elternelement der zu positionierenden *Box* beschreibt. Falls eine Position für eine *Box* angegeben wird, die keine Eltern-*Box* besitzt, so wird mit ihr die Position der gesamten giMLi-Präsentation auf dem Ausgabemedium angegeben. Zur Angabe der Position besitzt eine *Position*-Komponente daher eine x- und eine y-Koordinate (siehe Grammatikfragment 9.10). Ebenso wie die Ausdehnung der Spezialisierungen eines *Box*-Primitivs, werden die Koordinaten in Pixel angegeben. Es sind nur Koordinatenwerte sinnvoll, die innerhalb der umschließenden *Box* liegen. Da für diese Arbeit angenommen wird, dass die Eingabe vom vorgelagerten Generator semantisch korrekt erzeugt wird (siehe Abschnitt 6.4), findet eine überprüfung der Koordinatenwerte innerhalb der XAM nicht statt. Die Anordnung aller Medienelemente wird in der giMLi-Präsentation mit Instanzen der *AbsolutPosition*-Komponente vorgenommen.

```
1 <!ELEMENT absolutposition EMPTY >
2   <!ATTLIST absolutposition
3       %position-attr;
4       x CDATA #REQUIRED
5       y CDATA #REQUIRED>
```

Grammatikfragment 9.10: DTD-Fragment der XAM-Komponente *AbsolutPosition*

Für den Normalfall, dass die zu positionierende *Box* eine Eltern-*Box* besitzt, erfolgt die Umsetzung mittels des CSS-Mechanismus. Für die absolute Positionierung innerhalb der umgebenen Spezialisierung eines *Box*-Primitivs wird das CSS-Attribut `position` auf `absolut` gesetzt und die x- und y-Koordinate werden als `left`- und `top`-Attribute eingetragen. Ist kein Elternelement vorhanden, wird die Gesamtpräsentation mit Hilfe von JavaScript-Funktionalität auf dem Ausgabegerät positioniert.

9.11 Die XAM-Komponente *TextAnchor*

Um Startpunkte eines Links in einem Text markieren zu können, spezialisiert die XAM-Komponente *TextAnchor* das Basisprimitiv *Anchor*. Über die zusätzlich definierten Attribute `start` und `end` werden in der assoziierten *Text*-Komponente das Start- und das Endwort des Ankerbereichs angegeben (siehe Grammatikfragment 9.11 auf der nächsten Seite). Die beiden Attribute geben die Nummer des Wortes an, die durch Abzählen zugeordnet wird. Alle zwischen diesen Wörtern liegenden Worte, einschließlich des Start- und Endwortes werden so als Anker ausgezeichnet. Soll nur ein Wort ausgezeichnet werden, sind die Werte für das Start- und Endwort gleich. In der giMLi-Präsentation werden so Wörter des erläuternden Textes als Startpunkte von Links zu den Bildern eines Themas ausgezeichnet. In Abbildung 5.1 auf Seite 36 sind die rot dargestellten Wörter im Text solche Startpunkte dargestellt.

```
1 <!ELEMENT textanchor EMPTY >
2   <!ATTLIST textanchor
3       %anchor−attr;
4       start  CDATA #REQUIRED
5       end    CDATA #REQUIRED>
```

Grammatikfragment 9.11: DTD-Fragment der XAM-Komponente *TextAnchor*

In HTML wird der Startbereich eines Links in einem Text ausgezeichnet, in dem die zu markierenden Wörter vom <a>-Tag des Links eingeschlossen werden. Wird also das Codefragment für einen Text generiert, müssen die zu einer *Text*-Komponente gehörenden *TextAnchor* mittels <a>-Tags der entsprechenden *SimpleLink*-Komponente in den Text eingefügt werden.

Eine solche Modellierung erscheint auf den ersten Blick umständlich: Ein Textanker wird im Rahmen der Spezifikation einer ξML-Präsentation zunächst *inline*, also innerhalb des Textes, angegeben [SCHÜTTE 2002]. Für die XAM-Eingabe muss er dann extrahiert werden, da in der XAM, wie hier beschrieben, ein Textanker *outline*, also außerhalb des Textes stehend, modelliert ist. Um ihn in das HTML-Format zu überführen, muss der outline-definierte Textanker wieder in eine Inline-Form zurücküberführt werden. Diese Modellierung ist in dem in Kapitel 7 vorgestellten Entwurfsprinzip, unterschiedliche Konzepte eigenständig zu modellieren, begründet. Sie wird unabhängig von konkreten Zielplattformen vorgenommen. Bei der Implementierung der Altenberger Dom-Präsentation für das Format LINGO etwa werden die Textanker im Gegensatz zu HTML outline definiert. Diese umständlich anmutende Konstruktion eines Textankers für HTML bedeutet für den Anwender des ξML-Systems jedoch keinen Nachteil, da alle Transformationen automatisiert erfolgen.

9.12 Die XAM-Komponente *SimpleAddress*

Die XAM-Komponente *SimpleAddress* spezialisiert das Basisprimitiv *Address*. Mit einer solchen Adresse wird das Ziel eines Links angegeben. Eine *SimpleAddress*-Komponente modelliert in einem einfachen Adressierungsschema eine Adresse als die eindeutige Kennung einer Spezialisierung eines *Box*-Primitivs. Sie wird in dem `boxref`-Attribut der *SimpleAddress*-Komponente angegeben (siehe Grammatikfragment 9.12). Eine Implementierung des `AddressResolver`-Dienstes für dieses Adressierungsschema liefert mit seiner `resolve`-Methode diese Kennung zurück. Alle Ziele eines Links werden in der giMLi-Präsentation nach diesem Adressierungsschema angegeben.

```
1 <!ELEMENT simpleaddress EMPTY >
2   <!ATTLIST simpleaddress
3       %address−attr;
4       boxref IDREF #REQUIRED>
```

Grammatikfragment 9.12: DTD-Fragment der XAM-Komponente *SimpleAddress*

In HTML wird die XAM-Komponente *SimpleAddress* nicht direkt umgesetzt. Vielmehr wird die Instanz einer Spezialisierung eines *Box*-Primitivs, die sich hinter einer Adresse verbirgt, durch den `AddressResolver`-Dienst herausgefunden und im `href`-Attribut des `<a>`-Tags eines HTML-Links eingetragen.

9.13 Die XAM-Komponente *ChangeMedium*

Um die Aktion zu modellieren, die ausgeführt wird, wenn ein Link zu einem in einer *IndexedPictures*-Komponente abgelegten Bild mit Bildunterschrift verfolgt wird, spezialisiert die XAM-Komponente *ChangeMedium* das Basisprimitiv *Action*. Wie oben beschrieben, wird auf ein Bild innerhalb der *IndexedPictures*-Komponente mit einem ihm zugeordneten Index zugegriffen. Eine *ChangeMedium*-Komponente besitzt daher ein Attribut `refindex` (siehe Grammatikfragment 9.13). Da der Index des anzuzeigenden Bildes fest an eine *ChangeMedium*-Komponente gebunden ist, wird für jedes anzuzeigende Bild mit seiner Unterschrift eine eigene Instanz der *ChangeMedium*-Komponente mit dem entsprechenden Index verwendet. Bei der giMLi-Präsentation definiert eine solche Komponente das Verhalten eines Links, der aus dem auf der rechten unteren Seite der giMLi-Präsentation befindlichen Textes eines Themas auf ein Bild mit Bildunterschrift verweist (siehe Abbildung 5.1 auf Seite 36).

```
1 <!ELEMENT changemedium EMPTY >
2   <!ATTLIST changemedium
3       %action-attr;
4       refindex  CDATA #REQUIRED>
```

Grammatikfragment 9.13: DTD-Fragment der XAM-Komponente *ChangeMedium*

Die Aktion, die eine *ChangeMedium*-Komponente ausführt, wird in HTML als Aufruf der von einer *IndexedPictures*-Komponente bereitgestellten JavaScript-Funktion realisiert. Als Parameter wird der unter dem `refindex`-Attribut angegeben Index übergeben. Ein solcher Funktionsaufruf wird unter dem `href`-Attribut des `<a>`-Tags des zugehörigen Links angegeben und definiert so dessen Ziel. Die Instanz einer *ChangeMedium*-Komponente etwa, die ein unter dem Index `02` abgelegten Bild mit seiner Bildunterschrift innerhalb einer Instanz der *IndexedPictures*-Komponente anzeigen soll, führt im Rahmen der Codeerzeugung bei dem zugehörigen Link zu folgender Belegung des `href`-Attributs: `javascript:showIndexedMedium('im2')`.

9.14 Die XAM-Komponente *ChangeTheme*

Eine weitere Aktion wird mit der *ChangeTheme*-Komponente modelliert. Als Spezialisierung des *Action*-Primitivs definiert sie eine Aktion, die den Austausch eines mit einer *Scene*-Komponente angegeben komplexen Medienelements bewirkt. Sie benötigt neben den vom *Action*-Primitiv definierten Eigenschaften keine weiteren (siehe Grammatikfragment 9.14 auf der nächsten Seite). In der giMLi-Präsentation wird die mit einer *ChangeTheme*-Komponente modellierte Aktion ausgeführt, wenn ein im Inhaltsverzeichnis definierter Link, der auf ein anderes als das

gerade dargestellte Thema verweist, verfolgt wird: Das aktuell dargestellte Thema wird durch das referenzierte Thema ersetzt.

```
1 <!ELEMENT changetheme EMPTY >
2   <!ATTLIST changetheme
3       %action-attr;>
```

Grammatikfragment 9.14: DTD-Fragment der XAM-Komponente *ChangeTheme*

In HTML wird eine solche Aktion umgesetzt, in dem beim <a>-Tag des zugehörigen Links ein Attribut `target` auf den Wert `_top` gesetzt und im `href`-Attribut die Datei angegeben wird, in dem das darzustellende Thema abgelegt ist. Das `target`-Attribut bewirkt, dass der Inhalt der als Ziel angegeben Datei in dem umschließenden Frameset anstatt des vor Auslösen des Links angezeigten Inhalts dargestellt wird. Der gesamte Inhalt einer Präsentationsseite wird auf diese Weise ausgetauscht.

9.15 Die XAM-Komponente *OnClick*

Mit der XAM-Komponente *OnClick* wird das Basisprimitiv *Event* spezialisiert. Sie modelliert einen Mausklick mit der rechten Maustaste auf den als Startpunkt des Links markierten Bereich eines Elements der giMLi-Präsentation als ein Ereignis, das zum Verfolgen eines Links führt. Bei dieser Spezialisierung des *Event*-Primitivs werden keine weiteren Eigenschaften benötigt (siehe Grammatikfragment 9.15). Dieses mit der *OnClick*-Komponente modellierte Ereignis ist der Auslöser für alle in der giMLi-Präsentation definierten Links.

```
1 <!ELEMENT onclick EMPTY >
2   <!ATTLIST onclick
3       %event-attr;>
```

Grammatikfragment 9.15: DTD-Fragment der XAM-Komponente *OnClick*

HTML-Links besitzen ein Standardereignis, das zu ihrem Verfolgen führt. Dieses Standardereignis ist gerade das mit der *OnClick*-Komponente modellierte. Zu seiner Umsetzung muss daher kein HTML-Code erzeugt werden, sondern es reicht die oben beschriebene Umsetzung des zugehörigen Links mit dem <a>-Tag aus. Andere Ereignisse können über vordefinierte Attribute wie etwa `onmouseover` oder `onkeypress` bei einem HTML-Link als Auslöser registriert werden, für die mit JavaScript angegeben werden kann, wie auf sie zu reagieren ist.

9.16 Anmerkungen

In diesem Abschnitt stelle ich abschließende Überlegungen zur Definition der XAM-Komponenten vor. Nach der Betrachtung einiger alternativer Modellierungmöglichkeiten endet das Kapitel

mit der Überprüfung der Ausdrucksstärke der oben definierten XAM-Komponenten an den in Abschnitt 5.2.6 auf Seite 46 definierten Anforderungen.

9.16.1 Entwurfsalternativen

An dieser Stelle diskutiere ich potentielle XAM-Komponenten, die als mögliche Kandidaten während des Entwurfs in Betracht gezogen, aber aus nachfolgend geschilderten Gründen wieder verworfen wurden.

Mit den oben vorgestellten XAM-Komponenten wird ein Inhaltsverzeichnis durch *SimpleBox*-Komponenten und in ihnen enthaltenden *Text*- und *TextAnchor*-Komponenten beschrieben (siehe Anhang C, Zeile 67ff). Es wäre denkbar, eine eigene XAM-Komponente *TOC* als ein komplexes atomares Medienelement wie etwa die *IndexedPictures*-Komponente zu definieren, die ein komplettes Inhaltsverzeichnis modelliert. Dies macht aber nur Sinn, wenn komplexere Funktionalität als mit dem in der giMLi-Präsentation verwendeten Inhaltsverzeichnis angeboten werden soll, wie etwa das Hervorheben des Eintrags des aktuell dargestellten Themas. Die mit der hier gewählte Modellierung bereitgestellte Ausdrucksstärke reicht zur Beschreibung des Inhaltsverzeichnisses der giMLi-Präsentation vollkommen aus. Diese Modellierung mit den einzelnen *SimpleBox*-Primitiven scheint zwar etwas umständlich, fällt aber, wie ebenfalls bei der *TextAnchor*-Komponente, nicht ins Gewicht, da die Eingabe von dem vorgelagerten Generator automatisiert erzeugt und von der XAM automatisiert ausgewertet wird. Dem Anwender entstehen also keinerlei Nachteile.

Ein Thema einer giMLi-Präsentation wird mit der *Scene*-Komponente als ein komponiertes Medienelement beschrieben. Da ein Zustandsbegriff nur für ein atomares Medienelement existiert (siehe Seite 44), ist es nicht möglich, einen Zustand für ein Thema explizit zu modellieren. Als Konsequenz kann beim erneuten Besuchen eines Themas während der Betrachtung der giMLi-Präsentation nicht garantiert werden, dass das beim vorherigen Verlassen des Themas zuletzt dargestellte Bild und die zuletzt eingenommene Position des Fließtexts wieder angezeigt wird. Um dies zu erreichen, hätte eine *Theme*-Komponente als atomares Medienelement definiert werden müssen. Abgesehen davon, dass ein solches Verhalten für ein Thema einer giMLi-Präsentation nicht gefordert wurde (siehe Seite 36), hat eine Modellierung eines Themas als atomares Medienelement einen entscheidenden Nachteil: Sie ist viel zu grob, als dass sie eine sinnvolle Einheit zur Wiederverwendung definieren würde. Neben einer *Theme*-Komponente würden keine weiteren Komponenten benötigt, die die einzelnen Elemente modellieren, aus denen sich ein Thema einer giMLi-Präsentation zusammensetzt. Die Möglichkeiten zur Wiederverwendung, die ja ein zentrales Anliegen des ξML-Systems ist, wären dadurch stark eingeschränkt. Nur das Konzept eines Themas als Ganzes und nicht einzelne Elemente eines Themas könnten so zur Darstellung weiterer ξML-Präsentationen mit der XAM wiederverwendet werden. Mit einer so gewählten Granularität der XAM-Komponenten hätte man im Vergleich zum ADML-Converter nicht viel gewonnen, da dann auch mit der XAM bzw. mit dem gesamten ξML -System nur die anzuzeigenden Inhalte einer Präsentation flexibel zu setzen wären, die Änderung der Gestaltung einer Präsentation aber einen kompletten Neuentwurf nach sich zögen. Dieses soll mit dem ξML-System aber gerade vermieden werden.

In der giMLi-Präsentation werden zwei unterschiedliche Arten von Beziehungen zwischen Informationseinheiten verwendet: Zum einen Links innerhalb eines Themas zwischen Wörtern

des Texts auf der rechten Seite und den Bildern auf der linken Seite, zum anderen Links zwischen den Einträgen des Inhaltsverzeichnisses und anderen Themen. Diese beiden unterschiedlichen Arten werden einheitlich als *SimpleLink*-Komponenten realisiert, die sich neben den unterschiedlichen Start- und Zielpunkten in der Aktion unterscheiden, die bei ihrer Verfolgung ausgeführt wird. Alternativ wären auch zwei verschiedene Link-Komponenten denkbar, die das unterschiedliche Verhalten durch die Typisierung des Links modellieren. Meinem im Rahmen des abstrakten Modells vorgestellten Entwurfsprinzip folgend, unterschiedliche Konzepte wegen dort genannter Vorteile getrennt voneinander zu definieren, habe ich mich dagegen entschieden. Mit der expliziten Definition des Basisprimitivs *Action* im abstrakten Modell wurde eine von der eigentlichen Beziehung getrennte Modellierung ihres Verhaltens implizit vorgegeben. Eine *SimpleLink*-Komponente modelliert daher das Konzept, zwei Informationseinheiten gerichtet in Beziehung zu setzten. Das unterschiedliche Verhalten bei einer Linkverfolgung, das an die entsprechenden Beziehungen geknüpft ist, wird mit eigenen Komponenten modelliert, nämlich mit den XAM-Komponenten *ChangeTheme* und *ChangeMedium*.

9.16.2 Ausdrucksstärke

Als zentrale Forderung an die mit dieser Arbeit zu entwickelnden XAM-Komponenten wurde in Anforderung 18 auf Seite 46 formuliert, dass mit ihnen die giMLi-Präsentation modelliert werden können soll. Im Folgenden gehe ich detaillierter darauf ein, wie mit den von mir oben definierten XAM-Komponenten die einzelnen Anforderungen aus Abschnitt 5.2.6 erfüllt werden.

Bei der giMLi-Präsentation werden eine Reihe von Medienelementen verwendet, die mit entsprechenden XAM-Komponenten modelliert werden können müssen. Einfache Texte (Anforderung 19) und Bilder (Anforderung 20) können mit den XAM-Komponenten *Text* und *Picture* angegeben werden. Bilder mit Bildunterschriften (Anforderung 21) werden aus in Abschnitt 9.5 vorgestellten Gründen als atomares Medienelement mit der XAM-Komponente *IndexedPictures*, ein Thema (Anforderung 22) hingegen als komponiertes Medienelement mit einer *Scene*-Komponente modelliert. Weitere Medienelemente werden für die giMLi-Präsentation nicht benötigt.

Um diese Medienelemente zu dem gewünschten Erscheinungsbild anordnen und ein einheitliches Layout definieren zu können (Anforderung 23), werden verschiedene XAM-Komponenten bereitgestellt. Damit die Medienelemente überhaupt dargestellt werden können, wird ihnen mit der XAM-Komponente *SimpleBox* ein Bereich auf dem Ausgabegerät zugeordnet. Diese Bereiche können mit Hilfe der *AbsolutPosition*-Komponente angeordnet werden. Um die einzelnen Medienelemente, falls benötigt, mit Layoutinformationen zu versehen, können mit der XAM-Komponente *SimpleStyle* entsprechende Angaben, wie etwa Schriftgröße etc., gemacht und ihnen zugeordnet werden. Es existieren in der giMLi-Präsentation aber auch Bereiche, die nicht von mit oben genannten XAM-Komponenten modellierten Medienelementen bedeckt werden. Diesen Bereichen wird mit der *Background*-Komponente ein Medienelement zugeordnet, das zwar kein inhaltliche Information präsentiert, für das aber über eine *SimpleStyle*-Komponente Layoutinformationen wie etwa die Farbe, mit dem der ihr zugeordnete Bereich darzustellen ist, angegeben werden können.

Da bei der giMLi-Präsentation, wie bei allen Multimediapräsentationen auch, der Benutzer auf vorgegebenen Pfaden navigieren können soll (Anforderung 24), definiert die XAM-Kompo-

nente *SimpleLink* eine navigierbare Beziehung zwischen zwei Informationseinheiten. Mit einer *SimpleLink*-Komponente können sowohl Beziehungen zwischen dem Fließtext und den Bildern eine Themas (Anforderung 25), als auch zwischen dem Inhaltsverzeichnis und anderen Themen (Anforderung 26) angegeben werden. Der Startpunkt solcher Beziehungen liegt jeweils in Texten. Einzelne Wörter können durch die XAM-Komponente *TextAnchor* als ein solcher Startpunkt markiert werden (Anforderung 27). Das Ziel einer solchen Verknüpfung wird mit der *SimpleAddress*-Komponente angegeben, die auf eine *SimpleBox*-Komponente und damit indirekt auf die ihr zugeordneten atomaren oder komponierten Medienelemente (Anforderung 28) verweist.

Neben dem statischen Aspekt der giMLi-Präsentation, der mit den XAM-Komponenten angegeben wird, mit denen die Medienelemente und die Beziehungen zwischen ihnen modelliert werden, muss auch noch der dynamische Aspekt der giMLi-Präsentation mit den von mir definierten XAM-Komponenten beschrieben werden können. Das dynamische Verhalten besteht aus dem Auswählen und Verfolgen der Beziehungen zwischen den Medienelementen. Mittels der XAM-Komponente *OnClick* wird ein Ereignis modelliert, das der Benutzer auslöst, um die Verfolgung einer solchen Beziehung anzustoßen (Anforderung 29). Das Verfolgen der beiden oben beschriebenen verschiedenden Arten von Beziehungen führt zu zwei unterschiedlichen Handlungen. Während die mit der *ChangeMedium*-Komponente modellierte Aktion den Austausch eines innerhalb eines Thema angezeigten Bildes und seiner Unterschrift bewirkt (Anforderung 30), wird durch die mit der *ChangeTheme*-Komponente definierte Aktion der Austausch des angezeigten Themas der giMLi-Präsentation durch ein neues veranlasst (Anforderung 31).

Wie zu sehen ist, erfüllen die oben definierten XAM-Komponenten alle in Abschnitt 5.2.6 definierten Anforderungen, und sind somit ausdrucksstark genug, die giMLi-Präsentation zu modellieren. In Anhang C wird exemplarisch für ein Thema das Eingabedokument aufgeführt, mit dem die giMLi-Präsentation in den Ausdrucksmöglichkeiten der oben vorgestellten XAM-Komponenten beschrieben wird.

Neben der Ausdrucksstärke der XAM-Komponenten ist es nötig, ihre Bedeutung festzulegen (Anforderung 17 auf Seite 46), um sie geeignet verwenden zu können. Dies geschieht im Rahmen dieser Arbeit informell und ist in den obigen Abschnitten für die einzelnen XAM-Komponenten mit den textuellen Erläuterungen, dem Klassendiagramm und den DTD-Fragmenten erfolgt.

10 Beispielanwendung für giMLi

Nachdem in den vorherigen Kapiteln aus den Anforderungen der Entwurf der XAM entwickelt wurde, stelle ich in diesem Kapitel am Beispiel der giMLi-Präsentation stellvertretend für die ξML-Präsentationen die Funktionsweise der XAM zusammenhängend vor. Sie wird an Hand der einzelnen Phasen erläutert, die die XAM bei der Darstellung, also beim Erzeugen einer darstellbaren Form der giMLi-Präsentation durchläuft: zunächst die Initialisierung, dann der Aufbau der Zwischendarstellung und abschließend die Codeerzeugung.

Das Eingabedokument, mit der die giMLi-Präsentation in der Sprache XaML beschrieben wird, ist in Anhang C zu finden. Exemplarisch wird dort die Beschreibung eines Themas detailliert angegeben. Als Zielformat wird HTML gewählt. Eine genauere Vorstellung der Zielplattform wird im Rahmen der Beschreibung der Codegenerierung in Abschnitt 10.3 gegeben.

10.1 Initialisierung

Ehe die XAM mit dem Einlesen des Eingabedokuments die Generierung einer darstellbaren Form der giMLi-Präsentation beginnen kann, muss sie mit der entsprechenden Funktionalität initialisiert werden. Diese Funktionalität wird in Form von Komponenten bereitgestellt. Sie bestehen aus der in Abschnitt 8.1.4 vorgestellten XAM-Systemkomponente und den in Kapitel 9 definierten XAM-Komponenten. Die Initialisierung erfolgt dann in zwei Schritten.

In einem ersten Schritt wird die Systemkomponente registriert, die die Implementierungen der Dienste zur Verfügung stellt, die zum Erzeugen der HTML-Ausgabe benötigt werden (siehe Abschnitt 8.2). In XML-Dokument 10.1 auf der nächsten Seite ist der Lieferumfang der Systemkomponente, die für die giMLi-Präsentation benötigt wird, in Form ihrer Manifestdatei dargestellt. Wie dort zu sehen ist, besteht die Systemkomponente aus den folgenden Implementierungen der drei in Abschnitt 8.1 vorgestellten Diensteschnittstellen (in Klammern ist jeweils der in der Manifestdatei verwendete Dateiname der Implementierung angegeben): Die `ObjectBroker`-Schnittstelle benötigt jeweils eine Implementierung zum Erzeugen und Verwalten der abstrakten Elemente (`AbElOBImpl.class`), der Codeerzeuger (`CoGeHTMLOBImpl.class`) und der Metadatenobjekte (`MeDaOBHTMLImpl.class`). Die Codeerzeuger und die Metadatenobjekte sind jeweils für die Zielplattform HTML entwickelt worden. Desweiteren muss zum einen eine Implementierung für die `AddressResolver`-Schnittstelle bereitgestellt werden (`SIDARImpl.class`), die das in Abschnitt 9.12 vorgestellte Adressierungsschema unterstützt, und eine Adresse auf die Kennung einer Instanz einer Spezialisierung eines *Box*-Primitivs abbildet. Zum anderen wird eine Implementierung der `MediaLocator`-Schnittstelle benötigt (`FiSyMLImpl.class`), die den Zugriff auf die im Rahmen der giMLi-Präsentation im Dateisystem abgelegten Mediendaten ermöglicht.

```
1  <?xml version="1.0" encoding="iso-8859-1" ?>
2  <!DOCTYPE manifest
3      PUBLIC "-//UniDo//MANIFESTDTD 1.0//EN" "manifest.dtd">
4  <manifest>
5   <systemcomponent>
6
7  <objectbroker type="AbstractElement" file="AbElOBImpl.class">
8      </objectbroker>
9  <objectbroker type="CodeGenerator" file="CoGeHTMLOBImpl.class"
10     targetplatform="html"></objectbroker>
11 <objectbroker type="MetaData" file="MeDaOBHTMLImpl.class"
12     targetplatform="html"></objectbroker>
13
14 <addressresolver type="SimpleID" file="SIDARImpl.class"></addressresolver>
15
16 <medialocator type="FileSystem" file="FiSyMLImpl.class"></medialocator>
17
18  </systemcomponent>
19 </manifest>
```

XML-Dokument 10.1: Manifestdatei der XAM-Systemkomponente für die giMLi-Präsentation

In einem zweiten Schritt wird die XAM mit den in Kapitel 9 vorgestellten XAM-Komponenten für die Darstellung der giMLi-Präsentation konfiguriert. Der Lieferumfang einer XAM-Komponente ist am Beispiel der *SimpleBox*-Komponente in XML-Dokument 10.2 auf der nächsten Seite ebenfalls in Form ihrer Manifestdatei dargestellt. Bei der folgenden Erläuterung der Bestandteile der *SimpleBox*-Komponente ist in Klammern wieder der in der Manifestdatei verwendete Dateiname der entsprechenden Elemente angegeben. Der Lieferumfang besteht aus dem DTD-Fragment (`SimpleBox_Frag.dtd`), das die Repräsentation der Komponente in der Eingabesprache angibt, einer Java-Binärdatei, die die Komponente in der Zwischendarstellung repräsentiert (`SimpleBox_AbEl.class`) und jeweils einer Java-Binärdatei für den Codeerzeuger (`SimpleBox_CoGe.class`) und das Metadatenobjekt (`SimpleBox_MeDa.class`), die zur Codegenerierung für die Zielplattform HTML benötigt werden. Wie in Abschnitt 8.2 vorgestellt, wird aus den DTD-Fragmenten der einzelnen XAM-Komponenten die DTD der Eingabesprache XaML zusammengebaut und dem Xerces-Parser zur Verfügung gestellt. Die vollständige DTD der Eingabesprache XaML zur Beschreibung der giMLi-Präsentation ist in Anhang B angeben. Die weiteren als Java-Binärdateien vorliegenden Bestandteile der XAM-Komponenten stehen der XAM über die mit den entsprechenden Implementierungen im ersten Schritt initialisierten `ObjectBroker`-Schnittstellen zur Verfügung.

Ist die XAM nach diesen beiden Schritten für die Darstellung der giMLi-Präsentation initialisiert, kann die nächste Phase ausgeführt werden.

```
1  <?xml version="1.0" encoding="iso-8859-1" ?>
2  <!DOCTYPE manifest
3      PUBLIC "-//UniDo//MANIFESTDTD 1.0//EN" "manifest.dtd">
4  <manifest>
5   <xamcomponent name="SimpleBox">
6
7   <!-- Dateien fuer die Repraesentation in der Eingabe -->
8   <xamlfrag file="SimpleBox_Frag.dtd"></xamlfrag>
9   <!-- Dateien fuer die Repraesentation in der Zwischendarstellung -->
10  <abstractelement file="SimpleBox_AbEl.class"></abstractelement>
11  <!-- Dateien fuer die Codeerzeugung-->
12  <codegeneration targetplatform="html">
13   <codegenerator file="SimpleBox_CoGe.class"></codegenerator>
14   <metadata file="SimpleBox_MeDa.class"></metadata>
15  </codegeneration>
16
17  </xamcomponent>
18 </manifest>
```

XML-Dokument 10.2: Manifestdatei der XAM-Komponente *SimpleBox*

10.2 Aufbau der Zwischendarstellung

Nach der Initialisierung der XAM erfolgt der Aufbau der Zwischendarstellung. Dazu wird die als Eingabedokument vorliegende Beschreibung der giMLi-Präsentation mit Hilfe des Xerces-Parsers validierend eingelesen und, wie in Abschnitt 8.3 beschrieben, sukzessiv die Zwischendarstellung aufgebaut.

Zur Beschreibung der giMLi-Präsentation werden die mit den DTD-Fragmenten definierten XML-Elemente, die die XAM-Komponenten im Eingabedokument repräsentieren, verwendet. Für die einzelnen Elemente des Eingabdokuments werden mit Hilfe der `ObjectBroker`-Schnittstellenimplementierung die Instanzen der abstrakten Elemente der entsprechenden XAM-Komponenten erzeugt.

Diese Instanzen werden in unterschiedlichen Strukturen der Zwischendarstellung organisiert (siehe Abschnitt 8.3). Für die im `structure`-Bereich des Eingabedokuments als XML-Elemente angegebenen XAM-Komponenten wird aus den Instanzen ihrer abstrakten Elemente eine Baumhierarchie aufgebaut, die der Enthaltenseinsbeziehungen ihrer XML-Elemente im Eingabedokument entspricht. Die Baumhierarchie wird dabei aus den Instanzen der abstrakten Elemente der *SimpleBox*- und *ScrollBox*-Komponenten gebildet, denen gemäß den im Abschnitt 8.3 auf Seite 89 beschriebenen Assoziationen jeweils ein abstraktes Element der *AbsolutPosition*-Komponente und ein abstraktes Element einer für die giMLi-Präsentation benötigten Spezialisierung des *Medium*-Primitivs, also entweder einer *Text*-, *Picture*-, *IndexedPictures*-, *Background*- oder *Scene*-Komponente, zugeordnet ist. Das Wurzelelement dieser Struktur stellt einen Sonderfall dar. Es steht außerhalb des Typsystems der XAM und bildet die mit dem XML-Element `structure` im Eingabedokument angegebene Klammer um die Instanzen der abstrakten Elemente der Spe-

zialisierungen des *Box*-Primitivs (siehe Grammatikfragment 7.10 auf Seite 82). Es wird bei der Codeerzeugung gesondert behandelt. Die Instanzen der abstrakten Elemente der in den übrigen Bereichen des Eingabedokuments angegebenen Elemente werden dagegen in einer geeigneten Datenstruktur der `ObjectBroker`-Schnittstellenimplementierung verwaltet und unter ihrer eindeutigen Kennung für die Codegenerierung zur Verfügung gestellt.

Wie im Abschnitt 8.3 auf Seite 89 beschrieben, steht das *Pair*-Element außerhalb des eigentlichen Typsystems der XAM. Seine Instanzen werden unabhängig von dem Bereich, in dem sie im Eingabedokument verwendet werden, bei den Instanzen der abstrakten Elemente abgelegt, denen sie zugeordnet sind. Im Falle der giMLi-Präsentation werden diese *Pair*-Elemente von der *SimpleStyle* und der *IndexedPictures*-Komponente verwendet.

Mit Hilfe der so erzeugten Zwischendarstellung kann in der letzten Phase der Code der giMLi-Präsentation für die Zielplattform HTML generiert werden.

10.3 Codeerzeugung

Die letzte Phase beim Erzeugen einer darstellbaren Form der giMLi-Präsentation ist die Codeerzeugung. Ihr Aufgabe ist es, die in der Zwischendarstellung zur Beschreibung der giMLi-Präsentation verwendeten Konzepte in geeignete Konzepte der Zielplattform zu überführen und den entsprechenden HTML-Code zu generieren. Wie das Abbilden der Konzepte für das Zielformat HTML aussieht, wurde bereits in Kapitel 9 für die einzelnen XAM-Komponenten der giMLi-Präsentation gezeigt. In diesem Abschnitt skizziere ich daher das Framework zur HTML-Codeerzeugung, mit dem die benötigte Infrastruktur bereitgestellt und die Durchlaufstrategie über die Zwischendarstellung definiert wird.

10.3.1 Definition der Zielplattform

Ehe ich detaillierter auf die Generierung des Codes eingehe, müssen die Eigenschaften der im Rahmen dieses Beispiels verwendeten Zielplattform präziser als es bisher nötig war beschrieben werden, da sich eine Zielplattform durch mehr als nur die Eigenschaft des verwendeten Formats auszeichnet.

So kann sich die Unterstützung von einzelnen Eigenschaften einer Sprache direkt auf die Durchlaufstrategie auswirken. Für HTML etwa hängen die Durchlaufstrategie und Codeerzeugung erheblich davon ab, ob die Verwendung von Frames angeboten wird oder nicht. Für jeden der beiden Fälle muss daher eine eigene Zielplattform definiert werden. Ein weitere Eigenschaft, die eine Zielplattform kennzeichnet, ist das Programm, mit dem der erzeugte Code dargestellt werden soll, da die Browser den HTML-Standard zum Teil unterschiedlich interpretieren und umsetzen.

Die hier betrachtete Zielplattform beschreibe ich im Folgenden an Hand der Eigenschaften, die sie kennzeichnet. Als Format wird, wie bereits erwähnt, HTML gewählt und zwar in der Version 4.01 mit der Unterstützung von Frames. Ebenfalls unterstützt werden die CSS, und zwar in der Variante, in der die einzelnen Layoutinformationen unter dem `style`-Attribut bei den entsprechenden HTML-Elementen angegeben und nicht als Klassen in einer eigenen Datei abgelegt werden. In dieser Zielplattform wird der Opera-Browser [OPERA SOFTWARE 2002] als Anzei-

geprogramm für den erzeugten HTML-Code festgelegt. Werden andere Browser verwendet, wird eine korrekte Anzeige des HTML-Codes, der Ergebnis der hier vorgestellten Codeerzeugung ist, nicht gewährleistet.

Ist von der Zielplattform HTML die Rede, ist im Rahmen dieser Arbeit eine wie oben charakterisierte Zielplattform gemeint. Die Umsetzung der XAM-Komponenten und die im Folgenden geschilderte Durchlaufstrategie zur Codeerzeugung sind auf diese Zielplattform ausgerichtet.

10.3.2 Infrastruktur

Das Framework definiert nicht nur die Durchlaufstrategie, sondern muss auch eine Infrastruktur, in der der Durchlauf eingebettet ist, bereitstellen. Im Folgenden stelle ich zwei wesentliche Aspekte dieser Infrastruktur vor, die zur Erzeugung des HTML-Codes benötigt werden.

DOM-Repräsentation

Um die Codeerzeugung zu vereinfachen, wird aus der Zwischendarstellung nicht direkt Code in Form von HTML-Dateien generiert. Da für die Repräsentation der giMLi-Präsentation in der Zwischendarstellung andere Konzepte als in der HTML-Zielplattform verwendet werden, müssen die Informationen aus der Zwischendarstellung zum Teil umstrukturiert und für HTML geeignet neu zusammengestellt werden.

Es kann sein, dass durch diese Umstrukturierung die zur Codeerzeugung in der Zielplattform benötigten Informationen nicht in der richtigen Reihenfolge in der Zwischendarstellung angegeben sind. Sie können unter Umständen erst zu einem späteren Zeitpunkt während des Durchlaufs über die Zwischendarstellung zur Verfügung stehen, wie etwa einige Wertebelegungen von Attributen der HTML-Elemente. Bei direkter Erzeugung des HTML-Codes müsste daher entweder ein unter Umständen sehr hoher Verwaltungsaufwand betrieben werden, um benötigte Informationen für die einzelnen HTML-Elemente und ihre Attribute zusammenzutragen, bevor sie in eine Datei geschrieben werden. Oder diese Informationen müssten nachträglich zu bereits generierten Elementen in die entsprechende Datei eingetragen werden, was ebenfalls sehr aufwendig ist.

Um diesen Aufwand zu vermeiden, wird, wie bei Compilern zur Lösung dieses Problems üblich, zunächst eine Baumdarstellung des Zielcodes aufgebaut und zur eigentlichen Erzeugung der Ausgabe anschließend serialisiert. Dazu greife ich, wie bereits beim Einlesen, auf das Document Object Model (DOM) und die mit der DOM-Programmierschnittstelle zur Verfügung stehenden Funktionalität zurück. Es wird daher zunächst aus der Zwischendarstellung eine Repräsentation des HTML-Codes im Hauptspeicher, ein so genannter DOM-Baum, erzeugt. Nach dem vollständigen Aufbau des DOM-Baumes kann er recht einfach unter Verwendung von mit der DOM-Programmierschnittstelle zur Verfügung gestellter Funktionalität serialisiert werden. Im Lieferumfang des Xerces-Parsers ist etwa ein Beispielprogramm namens `DOMWriter` enthalten, dass genau dies leistet.

Da die DOM-Programmierschnittstelle neben der Unterstützung von XML-Elementen auch Schnittstellen und Implementierungen für HTML-Elemente (Version 4 mit Unterstützung von CSS) zur Verfügung stellt (für Java im Paket `org.w3c.dom.html`), kann der DOM-Baum mit den einzelnen HTML-Elementen komfortabel aufgebaut und deren Attribute auch nachträglich über die angebotenen Methoden gesetzt werden. Man hat somit die Möglichkeit, schrittweise

die Informationen der Zwischendarstellung in die Struktur von HTML mit seinen Elementen und Attributen überführen zu können.

Der Zwischenschritt mit dem Aufbau des DOM-Baums erspart einem den Verwaltungsaufwand, der bei direkter Generierung der HTML-Dateien benötigt würde. Die Codeerzeuger der einzelnen XAM-Komponenten implementieren für diese Zielplattform daher Transformationsregeln, die ihre Repräsentation aus der Zwischendarstellung in den DOM-Baum überführen. Wenn ich im Folgenden von der Codeerzeugung für die einzelnen XAM-Komponenten rede, ist diese Transformation in die Repräsentation im DOM-Baum gemeint.

Bereitstellung von Zusatzinformationen

Ehe der DOM-Baum aus der Zwischendarstellung aufgebaut werden kann, muss zur Codegenerierung in der hier betrachtete HTML-Zielplattform eine Information zur Verfügung gestellt werden, die nicht direkt in den Instanzen der einzelnen abstrakten Elemente enthalten ist: Bei der Codeerzeugung für eine *SimpleBox*-Komponente muss bekannt sein, ob in ihr *ScrollBox*-Komponenten enthalten sind oder nicht, da für diese beiden Fälle die *SimpleBox*-Komponente unterschiedlich in HTML umgesetzt wird (siehe Abschnitt 9.1).

Diese Information kann nur durch eine Analyse der Zwischendarstellung erfolgen. Mit Hilfe des Metadatenobjekt-Mechanismus wird diese Information zur Laufzeit bei jeder *Box*-Spezialisierung abgelegt. Zu diesem Zweck benötigt das Metadatenobjekt für die Zielplattform HTML eine Eigenschaft `containsScrollBox` mit entsprechenden `get` - und `set` -Methoden. Um die Information zu erhalten, wird mit einem Postorder-Durchlauf durch die mit den *Box*-Spezialisierungen gebildete Baumstruktur geschaut, ob eine *SimpleBox*-Komponente eine *ScrollBox*-Komponente beinhaltet, und entsprechend die `containsScrollBox` -Eigenschaft gesetzt. Ist die Zwischendarstellung auf diese Weise um die benötigten Informationen angereichert, kann mit dem Aufbau des DOM-Baumes und somit der Codeerzeugung begonnen werden.

10.3.3 Allgemeine Durchlaufstrategie

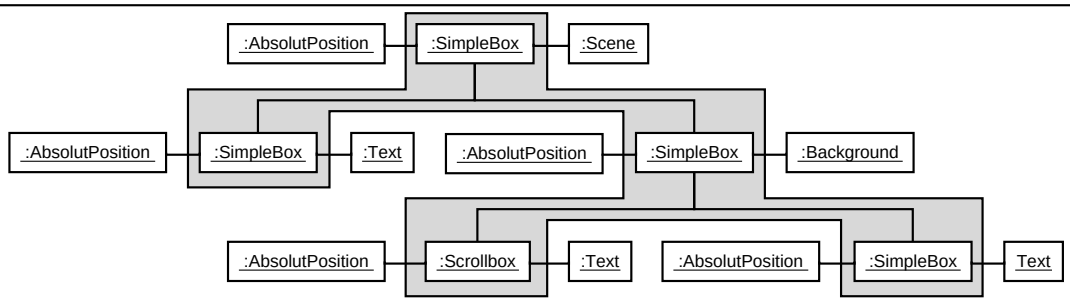
Um den DOM-Baum aufbauen zu können, muss zunächst eine Durchlaufstrategie festgelegt werden. In diesem Abschnitt beschreibe ich erst das generelle Vorgehen beim Durchlauf durch den Baum, ehe ich im folgenden Abschnitt auf einige Details der Codeerzeugung eingehe.

Für die Codeerzeugung wird über die baumartige Struktur der Zwischendarstellung iteriert, die sich, wie in Abschnitt 10.2 beschrieben, aus Instanzen

- der abstrakten Elemente der *SimpleBox*- und *ScrollBox*-Komponenten als Spezialisierungen des *Box*-Primitivs,
- des abstrakten Elements der *AbsolutPosition*-Komponente als Spezialisierung des *Position*-Primitivs und
- der abstrakten Elemente der *Text*-, *Picture*-, *IndexedPictures*-, *Background*- und *Scene*-Komponenten als Spezialisierungen des *Medium*-Primitiv

zusammensetzt. Dabei ist einer Instanz eines abstrakten Elements einer *SimpleBox*- bzw. *ScrollBox*-Komponenten jeweils genau eine Instanz des abstrakten Elements einer *AbsolutPosition*-Komponente und eine Instanz eines abstrakten Elements einer der Spezialisierungen des *Medi-*

um-Primitivs zugeordnet (siehe UML-Diagramm 10.1). Diese zugeordneten Instanzen besitzen in der Baumstruktur der Zwischendarstellung keine Kinder.



UML-Diagramm 10.1: Ausschnitt eines Objektbaums der Zwischendarstellung

Die Durchlaufstrategie lässt sich wie folgt beschreiben: Über die sich aus den Instanzen der abstrakten Elemente der *Box*-Spezialisierungen ergebene Baumstruktur, die in UML-Diagramm 10.1 grau hinterlegt ist, wird in einer Preorder-Strategie iteriert. Bei Erreichen eines Knotens in dieser grau hinterlegten Baumstruktur, also bei Erreichen einer Instanz des abstrakten Elements einer *SimpleBox*- bzw. *ScrollBox*-Komponente, werden zunächst die beiden zugeordneten Instanzen der abstrakten Elemente einer Spezialisierung des *Position*- und einer Spezialisierung des *Medium*-Primitivs besucht, die in UML-Diagramm 10.1 nicht grau hinterlegt sind. Danach wird der Preorder-Durchlauf über die Instanzen der abstrakten Elemente der *SimpleBox*- bzw. *ScrollBox*-Komponente fortgesetzt. Auf diese Weise wird über die Baumstruktur der Zwischendarstellung iteriert und jeder Knoten besucht.

Der Besuch eines Knotens stößt die Codegenerierung für diesen Knoten an, dass heißt, entsprechende DOM-Elemente für die Instanzen der abstrakten Elemente werden erzeugt und in den aufzubauenden DOM-Baum eingefügt bzw. Attribute von bereits existierenden Elementen über entsprechende Methoden gesetzt. Die genaue Definition der dabei verwendeten Schnittstellen der Codegenerierer der abstrakten Elemente bleibt einer Implementierung überlassen.

Neben der Baumstruktur besteht die Zwischendarstellung aus der Dictionary-Datenstruktur der `ObjectBroker`-Implementierung für die abstrakten Elemente. Besitzt ein Knoten in der Baumstruktur eine symbolische Referenz auf Instanzen von abstrakten Elementen, die in der Dictionary-Datenstruktur abgelegt sind, wird die Referenz im Zuge der Codeerzeugung für den referenzierenden Knoten aufgelöst. Für eine so referenzierte Instanz eines abstrakten Elements wird ebenfalls Code erzeugt, ehe nach dem obigen Schema weiter über die Baumstruktur der Instanzen der abstrakten Elemente der *Box*-Spezialisierungen iteriert wird.

Auf den ersten Blick besteht an dieser Stelle die Gefahr, das durch das Auflösen der Referenz Zyklen entstehen können, die eine Terminierung des mit der Durchlaufstrategie beschriebenen Algorithmus der Codeerzeugung verhindern. Es kann jedoch kein Zyklus auftreten, da im Rahmen der Codegenerierung diese Referenzen nur ausgewertet und in HTML-Code umgesetzt, aber nicht aktiv weiterverfolgt werden. Dieses gilt auch gerade für Links, so dass die Tatsache, dass sie prinzipiell die baumartige Struktur, die von den Inhalten der Multimediapräsentation gebildet wird, zu einer meist nicht zyklenfreien Graphstruktur erweitern, für deren Codeerzeugung

unkritisch ist.

Wie für einzelne Knoten, die mit der hier beschriebenen Durchlaufstrategie besucht werden, Code erzeugt wird, schildere ich im nächsten Abschnitt.

10.3.4 Details der Codeerzeugung

In diesem Abschnitt betrachte ich die Codeerzeugung einzelner XAM-Komponenten etwas genauer. Ich unterscheide dabei zwischen XAM-Komponenten, deren Instanzen ihrer abstrakten Elemente in der Baumstruktur der Zwischendarstellung abgelegt sind, und XAM-Komponenten, deren Instanzen ihrer abstrakten Elemente aus dieser Baumstruktur symbolisch referenziert werden.

Codeerzeugung für Elemente der Baumstruktur der Zwischendarstellung

Als erstes Element wird das obligatorische Wurzelelement der Baumstruktur besucht. Wie im Abschnitt 10.2 auf Seite 108 beschrieben, handelt es sich bei ihm um kein abstraktes Element. Im Framework der Zielplattform wird fest angegeben, wie seine Umsetzung in HTML-Code erfolgt. Denkbar ist, dass eine Startseite generiert wird, die ein neues Fenster des Browsers für die Darstellung der eigentliche Präsentation öffnet. Die Größe des Fensters ergibt sich aus der Größe der als Kinder an den Wurzelknoten hängenden *SimpleBox*-Instanzen. Entsprechend der oben beschriebenen Durchlaufstrategie wird nach Abarbeitung des Wurzelelements der DOM-Baum für die an ihm hängenden Unterbäume erzeugt.

Beim Besuch eines *SimpleBox*-Knotens wird an Hand des oben beschriebenen Metadatenobjekts unterschieden, ob in seinem Unterbaum *ScrollBox*-Knoten enthalten sind oder nicht. Sind sie enthalten, wird ein *SimpleBox*-Knoten in HTML als Frame-Element, bei dem das `scrolling`-Attribut den Wert `no` hat, umgesetzt. Für jede Ebene in der Baumhierarchie, die so umzusetzende *SimpleBox*-Knoten enthält, wird zunächst ein Frameset-Element in den DOM-Baum eingefügt, an das dann die Frame-Elemente angehängt werden. Enthält der Unterbaum eines *SimpleBox*-Knotens keinen *ScrollBox*-Knoten, wird er als HTML-Tabellendefinition umgesetzt, bei dem weitere Inhalte als Tabelleneinträge realisiert werden. Ein *ScrollBox*-Knoten hingegen wird in HTML immer als ein Frame-Element, bei dem das `scrolling`-Attribut auf `yes` gesetzt ist, realisiert.

Wurde so ein *SimpleBox*- bzw. *ScrollBox*-Knoten verarbeitet, wird zunächst der ihm zugeordnete *AbsolutPosition*-Knoten und der Knoten, der die Spezialisierung eines *Medium*-Primitivs enthält, besucht und für ihn Code erzeugt, ehe mit dem Preorder-Durchlauf über weitere *SimpleBox*- bzw. *ScrollBox*-Knoten fortgefahren wird. Für den *AbsolutPosition*-Knoten werden wie in Abschnitt 9.10 beschrieben die Koordinaten als `top`- und `left`-Einträge unter dem `style`-Attribut, das die CSS-Formatierungen für HTML-Elemente enthält, hinzugefügt. Die einzelnen Inhalte der *SimpleBox* bzw. *ScrollBox*-Knoten, die mit dem Konten der Spezialisierung eines *Medium*-Primitivs angeben sind, werden wie in ihren Abschnitten in Kapitel 9 beschrieben in HTML umgesetzt und geeignet in den DOM-Baum eingefügt. Ein Sonderrolle nimmt dabei die *Scene*-Komponente ein, da sie keine direkte Entsprechung in HTML hat und somit als allgemeiner DOM-Knoten und nicht als spezieller HTML-DOM-Knoten in den Baum eingefügt wird. Er wird bei der Serialisierung des DOM-Baums ausgewertet, um eine geeignete Aufteilung in HTML-Dateien vorzunehmen.

Codeerzeugung für symbolisch referenzierte Elemente

Die bisher geschilderte Codeerzeugung behandelt nur Elemente, die in der baumartigen Struktur der Zwischendarstellung angegeben sind und daher beim Durchlauf über sie direkt verarbeitet werden. Es existieren aber, wie bereits erwähnt, weitere abstrakte Elemente in der Zwischendarstellung, die im Dictionary der `ObjectBroker`-Schnittstellenimplementierung verwaltet werden und über symbolische Referenzen mit den Elementen der Baumstruktur assoziiert sind. Wenn ein Element der Baumstruktur eine solche Referenz besitzt, wird nach der Codeerzeugung dieses Elements die symbolische Referenz aufgelöst und Code für das referenzierte Element erzeugt (siehe auf Seite 112). Im Falle der giMLi-Präsentation können nur Instanzen der abstrakten Elemente der *SimpleStyle*- und der *TextAnchor*-Komponente aus der Baumstruktur direkt referenziert werden.

Das abstrakte Element der *SimpleStyle*-Komponente ist über eine Referenz einer Spezialisierung des *Medium*-Primitivs zugeordnet. Nach der Codeerzeugung für die Instanz einer solchen Spezialisierung wird mit Hilfe der `ObjectBroker`-Schnittstellenimplementierung diese Referenz aufgelöst und dann für die so erhaltene Instanz eines abstrakten Elements einer *SimpleStyle*-Komponente Code erzeugt. Wie in Abschnitt 9.9 beschrieben, werden dazu entsprechende CSS-Einträge bei dem `style`-Attribut des referenzierenden Elements vorgenommen.

Ein weiteres abstraktes Element, das aus der Baumstruktur referenziert wird, ist das der *TextAnchor*-Komponente. Es kann nur von der Instanz eines abstrakten Elements der *Text*-Komponente referenziert werden. Für eine Instanz eines abstrakten Elements der *Text*-Komponente, die keine Referenzen auf solche Textanker besitzt, wird Code generiert, in dem der durch die Komponente repräsentierte Text als eigener Knoten an den entsprechenden *SimpleBox*- oder *ScrollBox*-Knoten im DOM-Baum gehängt wird. Besitzt eine Instanz eines abstrakten Elements der *Text*-Komponente hingegen Referenzen auf solche Textanker, muss ihr Text um diese Textanker und die mit ihm wiederum über symbolische Referenzen verbundenen Links angereichert werden, ehe er in den DOM-Baum eingefügt werden kann: Die als Textanker markierten Worte werden vom HTML-Element des zum Textanker gehörenden Links umschlossen. Der Aufbau eines Links wird in den Abschnitten 9.8 und 9.11 bis 9.15 detaillierter beschrieben. Die dazu benötigten Informationen werden durch die Codeerzeugung der entsprechenden über symbolische Referenzen assoziierten Instanzen der abstrakten Elemente bereitgestellt. Ein so angereicherter Text besteht in der Repräsentation des DOM-Baumes dann nicht mehr aus einem Knoten, sondern aus einer Baumstruktur von mehreren solcher *Text*-Knoten, die an den geeigneten Stellen auch Knoten für das a-Element beinhaltet, das in HTML einen Link beschreibt.

10.3.5 Anmerkungen für die Implementierung

Der an Hand dieses Ablaufs beschriebene Aufbau des DOM-Baums muss dann im Detail im Framework umgesetzt werden. Ein wesentlicher Teil dabei ist die Implementierung der oben beschriebenen Durchlaufstrategie. Für eine Realisierung bietet sich die Verwendung des Besuchermusters (siehe Seite 301ff in [GAMMA et al. 1996]) an, um die Codeerzeugung für die einzelnen Knoten von der Durchlaufstrategie zu trennen. Der eigentlichen Aufbau des DOM-Baums wird dann durch die Codeerzeuger der einzelnen XAM-Komponenten realisiert. Nach dem Aufbau des DOM-Baums muss dieser noch in entsprechende HTML-Dateien serialisiert werden. Alle für eine Realisierung benötigten Informationen sind dazu im DOM-Baum enthal-

ten: Die HTML-Tags und ihre Attribute sind als entsprechende DOM-Elemente im DOM-Baum abgelegt. Die Informationen, die zur Aufteilung des erzeugten HTML-Codes in einzelne Dateien benötigt werden, sind durch die HTML-Elemente Frameset- bzw. Frame und die *Scene*-Knoten explizit angegeben. Die Umsetzung des Frameworks bleibt aber ebenso wie die Serialisierung des DOM-Baums in entsprechende HTML-Dateien einer Implementierung überlassen, um den Umfang meiner Diplomarbeit nicht zu überschreiten.

11 Zusammenfassung und Bewertung

Mit der Vorstellung der Anwendung der XAM am Beispiel der giMLi-Präsentation im letzten Kapitel endet der Entwurfsteil meiner Arbeit. Es beginnt nun der Teil, in dem abschließende Betrachtungen zur XAM vorgestellt werden. In diesem Kapitel erläutere ich zunächst kurz im Rahmen einer Zusammenfassung wesentliche Aspekte des Entwurfs der XAM und positioniere ihn gegenüber anderen Lösungen. Den Abschluss bildet eine Bewertung des Entwurfs der XAM.

11.1 Zusammenfassung

In der Zusammenfassung stelle ich zum einen wesentliche charakteristische Eigenschaften der XAM kurz vor und ordne die XAM in den Compilerbaukontext, in dem sie sich als Generator bewegt, ein. Zum anderen werfe ich einen Blick über den Tellerrand der XAM und stelle ausgewählte Arbeiten vor, die sich in einem ähnlichen Kontext wie die XAM bewegen.

11.1.1 Eigenschaften der XAM

Die XAM ist die Laufzeitumgebung des ξML-Systems, die die Darstellung von ξML-Präsentationen realisiert, in dem sie die darstellungsneutrale Beschreibung einer ξML-Präsentation in anzeigbare Formate überführt, wie im letzten Kapitel exemplarisch mit der giMLi-Präsentation für das Format HTML gezeigt wurde. Sie stellt somit einen Generator dar, der aber auf Grund der Bedürfnisse des umschließenden ξML-Systems eine Reihe von charakteristischen Eigenschaften besitzt, die ihn von anderen Generatoren unterscheiden:

- Die XAM ist ein in zwei Dimensionen erweiterbarer Generator, da sowohl sein Eingabeformat, und damit die von der XAM bereitzustellende multimediale Funktionalität, als auch die Menge der zu unterstützenden Ausgabeformate erweiterbar ist.
- Die XAM ist hochgradig konfigurierbar, um diese Erweiterungsmöglichkeiten anbieten zu können. Initial besitzt sie mit einem Dienst, der die Registration weiterer Dienste ermöglicht, nur minimale Funktionalität. Die zur Darstellung der ξML-Präsentationen benötigte Funktionalität wird der XAM ebenfalls zur Laufzeit im Rahmen der Initialisierung zur Verfügung gestellt.
- Die XAM besitzt mit dem abstrakten Modell ein Metamodell, das allen mit ihr darzustellenden ξML-Präsentationen zu Grunde liegt und vorgibt, welche multimediale Funktionalität prinzipiell mit der XAM unterstützt wird.

- Die XAM ist ein komponentenorientierter Generator. Die Funktionalität zur Darstellung einzelner multimedialer Elemente einer ξML-Präsentation wird in XAM-Komponenten definiert.

Zur Realisierung dieser gerade beschriebenen Funktionalität werden im Entwurf eine Reihe von Architektur- und Entwurfsmuster eingesetzt, auf deren Verwendung an den entsprechenden Stellen in den Entwurfskapiteln hingewiesen wurde.

Wie an den obigen Eigenschaften der XAM zu sehen ist, unterscheidet sie sich in einigen Punkten deutlich von gewöhnlichen Compilern. Stellvertretend für die in den einzelnen Entwurfskapiteln bereits vorgenommene Abgrenzung zu gewöhnlichen Compilern, seien an dieser Stelle zwei wesentliche Unterschiede hervorgehoben:

- Die XAM ist sowohl in ihrer Eingabe, als auch in ihrer Ausgabe erweiterbar. In der Literatur wird wohl der Fall der Erweiterbarkeit der Ausgabe eines Compilers, also der Unterstützung einer neuen Zielplattform durch den Austausch des Back-Ends, vorgestellt, eine Erweiterbarkeit des Eingabeformats ist hingegen nicht prominent vertreten. In [AHO et al. 1988a], Seite 24, wird dafür die Unterschiedlichkeit der einzelnen Eingabesprachen angeführt. Dieser Problematik wird mit der XAM auf zwei unterschiedlichen Ebenen begegnet: Zum einen mit der Definition des abstrakten Modells, das den Rahmen der von der XAM zu unterstützenden Ausdrucksstärke vorgibt, auf einer eher konzeptionellen Ebene. Zum anderen auf einer syntaktischen Ebene, in dem es mit der Verwendung von XML ermöglicht wird, eine Familie von Eingabesprachen definieren zu können, um so die Abläufe beim Einlesen zu vereinheitlichen.

- Die Elemente der Zwischendarstellung der XAM werden durch die Initialisierung zur Laufzeit nachgeladen, um die Erweiterung der Eingabe der XAM ohne einen Neuentwurf des Front-Ends zu ermöglichen. Dieser Ansatz einer dynamisch erweiterbaren Zwischendarstellung taucht bei gewöhnlichen Compilern nicht auf.

Die XAM folgt prinzipiell dem für Compiler üblichen Vorgehen des Analyse-Synthese-Modells, wobei sich der Ablauf aber im Detail unterscheidet, wie im Abschnitt 6.4 beschrieben wird.

11.1.2 Verwandte Arbeiten

Neben der hier vorgenommenen Einordnung der XAM in die allgemeinen Compilerbautechniken aus der Literatur ist es sinnvoll, sie auch gegenüber Systemen bzw. Technologien zu positionieren, die zur Lösung von der XAM ähnlichen Problemstellungen verwendet werden.

In [VILLARD et al. 2001] und [JOURDAN et al. 1998] wird mit Madeus ein System vorgestellt, das prinzipiell ein ähnliches Vorgehen wie ξML und die XAM verfolgt, sich aber im Detail doch deutlich unterscheidet. Im Madeus-System wird zwar auch eine Multimediapräsentation darstellungsneutral in einem XML-basierten Format beschrieben und aus dieser Beschreibung eine anzeigbare Form (etwa unter Verwendung des Formats SMIL) oder die Anzeige direkt erzeugt (mit Hilfe des Java Media Frameworks). Durch die explizite Unterstützung von zeitlichen Beziehungen innerhalb von Multimediapräsentationen und die Verwendung eines regelbasierten Ansatzes (siehe Abschnitt 2.4.3) zur Bestimmung der endgültigen Gestaltung der Präsentation geht das Madeus-System sogar einen deutlichen Schritt weiter als das ξML-System mit der

XAM. Allerdings wird dort weder ein komponentenorientierter Ansatz zur Beschreibung von Multimediapräsentationen verfolgt, noch die Möglichkeit bereitgestellt, das System und die dort verwendete Laufzeitumgebung dynamisch um die Unterstützung neuer multimedialer Elemente zu erweitern. Diese Aspekte sind im Rahmen des ξML-Systems zentral und daher für die XAM unerlässlich.

Im Kontext von DoDL, einer objektorientierten Spezifikationssprache für Hyperdokumente (siehe Abschnitt 2.2.2), wurde ein Generator namens dodl2html [PLEUMANN 2000] entwickelt, der ein mit DoDL darstellungsneutral beschriebenes Hyperdokument sowohl in das Format HTML als auch in das Format LaTeX überführen kann. Wie die XAM verwendet dodl2html etablierte Formate, um eine anzeigbare Form einer darstellungsneutral beschriebenen Multimediapräsentation zu erzeugen. Jedoch beschränken sich seine Erweiterungsmöglichkeiten auf das Back-End, da mit DoDL das Eingabeformat fest definiert ist. Durch die Verwendung objektorientierter Technologien bei der Architektur des Generators soll der Änderungsaufwand für die Unterstützung neuer Zielsysteme möglichst gering gehalten werden. Im Bereich des Back-Ends wird somit ein ähnlicher Weg wie bei der XAM gegangen. Aspekte, wie etwa die Unterstützung neuer multimedialer Funktionalität zur Laufzeit, spielen jedoch im Gegensatz zur XAM wegen des fest definierten Eingabeformats beim dodl2html-Generator keine Rolle. Hingegen sind dort auftauchende Probleme, wie etwa das automatisierte Erzeugen eines Layouts und das damit verbundene Aufteilen der zu erzeugenden Ausgabe in HTML-Dateien (siehe Kap. 5.5 in [PLEUMANN 2000]), für die XAM irrelevant, da diese Informationen explizit im XAM-Eingabedokument angegeben werden.

Es existieren noch eine Vielzahl weiterer Multimediapräsentationssysteme in der Literatur, bei denen jedoch häufig kein Schwerpunkt auf die Konzeption einer Laufzeitumgebung gelegt wurde. Es existieren daher nicht sehr viele Beispiele, die im Rahmen der Positionierung der XAM relevant wären. Die beiden oben geschilderten Systeme wurden ausgewählt, da mit ihnen eine ähnliche Problematik, wie die, die mit der XAM gelöst werden soll, behandelt wird. Abschließend stelle ich eine Technologie vor, die im Kontext von Multimediapräsentationen zur Realisierung der Erzeugung der Darstellung häufiger verwendet wird und erläutere ihre Bezüge zur XAM.

Ein zur Zeit recht populärer Ansatz zur Überführung von in XML-Dokumenten verfassten Inhalten in darstellbare Formen, der auch bei einigen Multimediapräsentationssysteme wie etwa dem oben vorgestellten Madeus-System verwendet wird, ist der Standard Extensible Stylesheet Language (XSL, [ADLER et al. 2001]). Er besteht zum einen aus den XSL Formatting Objects (XSL-FO), die die Positionierung von Elementen auf einer Ebene ermöglichen sowie Semantik für Formatierungseigenschaften unabhängig von einem konkreten Zielsystem festlegen. Mit einem so genannten Formatter werden XSL-FO-Dokumente in ein gewünschtes Zielsystem überführt. Zum anderen definiert er die deklarative Sprache XSL Transformations (XSLT, [CLARK 1999]), mit der die Regeln zur Überführung eines XML-Eingabedokuments in ein XSL-FO-Dokument beschrieben werden. Die Transformation, die auch eine Umstrukturierung der Informationen beinhalten kann, wird von einen XSLT-Prozessor, einem generischem Programm, das die mit XSLT verfassten Transformationsregeln auf das XML-Eingabedokument anwendet, durchgeführt. Der XSL-Ansatz unterscheidet sich in zwei wesentlichen Punkten vom ξML-System bzw. von der XAM. Zum einen bietet XSL keine ausreichenden Möglichkeiten zur Modularisierung an, die zur Definition von weitesgehend unabhängigen Einheiten benötigt wird,

um eine Wiederverwendung, wie sie mit dem ξML-System durch die Komponentenorientierung erreicht werden soll, zu ermöglichen. Zum anderen arbeiten die Transformationsregeln direkt auf den Inhalten des Dokuments, und greifen nicht, wie beim ξML-System im Allgemeinen und bei der XAM im Besonderen über durch die Komponenten definierte Schnittstellen auf die Inhalte des Dokuments zu, um eine Datenkapselung zu ermöglichen. Eine Umsetzung einer Laufzeitumgebung für das ξML-System allein durch die Definition geeigneter XSL-Stylesheets scheint daher unpraktikabel. Die Gefahr, das eine so realisierte Laufzeitumgebung ähnlich wie der Converter des Altenberger Dom-Projekts einen monolithischen Charakter bekommt und schnell unübersichtlich und unwartbar wird, erscheint doch beträchtlich. Aber für eine Realisierung einzelner Teilaspekte des hier vorgestellten Entwurfs der XAM könnte sich eine genauere Betrachtung dieser Technologie lohnen, etwa zur Definition der Transformationsregeln in den Codeerzeugern für das HTML-Zielformat oder bei der Findung plattformunabhängiger Layouteigenschaften, die in Spezialisierungen des *Style*-Primitivs verwendet werden.

11.2 Bewertung

Nach der Zusammenfassung wesentlicher Eigenschaften der XAM und einer Positionierung gegenüber anderen Ansätzen, erfolgt nun die Bewertung. Sie wird an Hand der beiden folgenden Kriterien vorgenommen. Das erste Kriterium ist der in Abschnitt 5.2 aufgestellte Anforderungskatalog. Die dort formulierten Anforderungen müssen mit dem hier vorgestellten Entwurf der XAM erfüllt werden. Als zweites Kriterium ist die Beurteilung einer Anwendung des Entwurfs, was allerdings eigentlich eine Implementierung der XAM voraussetzt, die nicht Bestandteil dieser Diplomarbeit ist. Im Idealfall sollte zusammen mit einer Implementierung des in [SCHÜTTE 2002] vorgestellten Generators eine rudimentär funktionierendes ξML-System vorhanden sein, um den Entwurf der XAM validieren zu können. Da dies aber nicht der Fall ist, dient als Grundlage der Validierung die in dieser Arbeit verwendete Beispielpräsentation giMLi.

11.2.1 Anforderungen

Im Anforderungskatalog werden Anforderungen aufgestellt, die eine unterschiedliche Granularität besitzen. Die in den Abschnitten 5.2.2 bis 5.2.6 formulierten Anforderungen betreffen detaillierte Funktionalität einzelner Teile der XAM. In den Kapiteln, in denen der Entwurf der Teilbereiche vorgestellt wurde, ist die Erfüllung dieser Anforderungen bereits gezeigt worden. Die drei in Abschnitt 5.2.1 aufgestellten Anforderungen sind hingegen allgemeinerer Natur und betreffen die XAM als Ganzes. Ihre Erfüllung wird daher abschließend an dieser Stelle diskutiert.

Die zentrale Aufgabe der XAM ist die Darstellung von abstrakt beschriebenen ξML-Präsentationen unter Verwendung etablierter Multimediaformate (Anforderung 1 auf Seite 38). Um zu zeigen, dass der hier vorgestellte Entwurf diese Aufgabe auch erfüllt, wurde in Kapitel 10 an Hand der giMLi-Präsentation als Beispiel einer ξML-Präsentation für die Zielplattform HTML erfolgreich gezeigt, wie eine solche Darstellung mit der XAM realisiert werden kann.

Ein wesentlicher Aspekt beim Entwurf der XAM ist ihre Erweiterbarkeit (Anforderung 2). Sie soll zum einen unterschiedliche Zielplattformen unterstützen können, was sich beim Entwurf durch die Definition einzelner Generierungsframeworks äußert. Die Frameworks kapseln

plattformabhängige Details der Codeerzeugung und greifen über wohldefinierte Schnittstellen auf Funktionalität der übrigen Teilen der XAM zu. Auf diese Weise werden die Stellen, an denen Erweiterungen zur Unterstützung neuer Zielplattformen vorgenommen werden müssen, konzentriert, so dass die übrigen Teile der XAM davon unberührt bleiben. Zum anderen soll es mit dem Entwurf ermöglicht werden, die unterstützte Ausdruckstärke der XAM zu erweitern. Dies kann durch die Definition der dazu benötigten Funktionalität in einzelnen XAM-Komponenten, die der XAM mit Hilfe des Initialisierungsmechanismus zur Laufzeit zur Verfügung gestellt werden, auf eine flexible und gleichförmige Art und Weise geschehen. Da die XAM initial als Funktionalität nur den Initialisierungsmechanismus anbietet, wird sie somit bei jedem Start mit der Funktionalität zur Unterstützung der benötigten Ausdrucksstärke erweitert.

Mit Hilfe der gerade geschilderten Mechanismen beantwortet sich die Frage nach der in Anforderung 3 verlangten Komponentenorientiertheit. Durch die explizite Definition der benötigten Funktionalität zur Darstellung einer ξML-Präsentation in Form von XAM-Komponenten und der Verwendung geeigneter Architektur- und Entwurfsmuster, die zu der in Kapitel 6.3 vorgestellten modularen Architektur der XAM führen, wird diese Anforderung erfüllt.

Abschließend kann daher festgestellt werden, dass mit dem hier vorgestellten Entwurf der XAM der in Abschnitt 5.2 aufgestellte Anforderungskatalog vollständig erfüllt wird.

11.2.2 Anwendung der XAM

Eine Anwendung der XAM kann nicht losgelöst von dem umschließenden ξML-System gesehen werden, da einige eingeschlagene Lösungswege direkt in der durch das ξML-System definierten Umgebung begründet sind. Im Folgenden wird daher bei der Bewertung der Anwendung an einigen Stellen auch explizit zum ξML-System und nicht nur zur XAM Stellung genommen.

Um die XAM anzuwenden, also um eine konkrete ξML-Präsentation wie etwa die giMLi-Präsentation darstellen zu können, ist es nötig, einige Vorarbeiten zu leisten: Es muss zum einen für die gewünschte Zielplattform ein Framework definiert werden, das die Durchlaufstrategie über die Zwischendarstellung und einige plattformabhängige Infrastruktur anbietet. Zum anderen müssen für die in der Präsentation verwendeten multimedialen Elemente entsprechende XAM-Komponenten definiert werden, die die von den einzelnen Teilen der XAM benötigte Funktionalität zur Erzeugen einer darstellbaren Form einer ξML-Präsentation bereitstellen. Dies ist für die giMLi-Präsentation und die Zielplattform HTML in dieser Arbeit geschehen (siehe Kapitel 9 und 10).

Es stellt sich die Frage, ob sich der Aufwand, der betrieben werden muss, um die XAM so konfigurierbar zu gestalten, überhaupt lohnt, oder ob nicht besser die Darstellung einer ξML-Präsentation für eine Zielplattform einmal fest implementiert werden soll. Die Frage kann endgültig nur auf Grund empirischer Untersuchungen bei einem realen Einsatz geklärt werden, was im Rahmen der Arbeit nicht möglich ist, da keine Implementierungen vorliegen. Es ist aber offensichtlich, das die Vorteile der XAM, wie auch beim ξML-System an sich, erst richtig zum Tragen kommen, wenn mehr als ein Projekt realisiert werden soll. Der Aufwand, die von der XAM und den ξML-System benötigte Infrastruktur zur Verfügung zu stellen, ist erheblich höher als etwa beim Altenberger Dom-Projekt, relativiert sich aber mit jedem Projekt, das mit dem ξML-System bzw. der XAM durchgeführt wird. Denn gerade bei der dann nötigen Anpassung bzw. Erweiterung, um neue Multimediapräsentationen bzw. ξML-Präsentationen verarbeiten zu

können, zeigen sich die Vorteile der XAM im Gegensatz zum Converter des Altenberger Dom-Projekts, die den oben beschriebenen Mehraufwand rechtfertigen, wie ich im Folgenden näher erläutere.

Mit dem abstrakten Modell und den XAM-Komponenten ist eine klare Schnittstelle zur Erweiterung der XAM um die Unterstützung der Darstellung neuer multimedialer Elemente definiert worden. Soll ein neues Element unterstützt werden, muss für dieses Element

- ein DTD-Fragment definiert werden, das die Darstellung des Elements in der Eingabe repräsentiert.
- eine Java-Binärdatei als abstraktes Element bereitgestellt werden, das die Funktionalität für die Repräsentation in der Zwischendarstellung realisiert.
- für jede zu unterstützende Zielplattform jeweils eine Java-Binärdatei als Codeerzeuger und, falls benötigt, als Metadatenobjekt bereitgestellt werden, um die Codeerzeugung für die Zielplattform zu ermöglichen.

Zur Unterstützung neuer multimedialer Elemente ist es also nicht nötig, die XAM an sich zu verändern. Ganz im Gegensatz zum Converter des Altenberger Dom-Projektes, kann eine solche Erweiterung daher zur Laufzeit erfolgen.

Diese Feststellung gilt allerdings nur unter der Voraussetzung, dass zur Erzeugung von Code für ein neues Element in der jeweiligen Zielplattform die verwendete Durchlaufstrategie nicht geändert werden muss. Tritt dies jedoch ein, wird durch die Modifikation der Durchlaufstrategie eine neue Zielplattform definiert, da die Durchlaufstrategie Teil des Umfangs einer Zielplattform ist, wie etwa auch das verwendete Zielformat (siehe Abschnitt 10.3.1). Für diesen Fall ist mit der Definition eines neuen Generierungsframeworks ein größerer Entwicklungsaufwand erforderlich, als nur die Bereitstellung einer neuen XAM-Komponente. Diese Tatsache ist aber nicht im Entwurf der XAM begründet, sondern liegt in der Natur der verwendeten Zielformate.

Durch die klare Trennung von Konzepten zur Darstellung einer Multimediapräsentation bereits auf Ebene des abstrakten Modells, wie beispielsweise die getrennte Modellierung der Anordnung von Medien durch das *Box*-Primitiv und der Darstellung von solchen Medien durch das *Medium*-Primitiv, wird aber versucht, die Anzahl des Auftretens dieses Falles möglichst gering zu halten. Bei Versuchen zur Modellierung anderer Multimediapräsentationen als die giMLi-Präsentation hat sich gezeigt, dass das von mir definierte abstrakte Modell recht universell einsetzbar zu seien scheint. Es lies sich bei ersten Überlegungen kein Element finden, das eine Veränderung der Durchlaufstrategie erfordert. Für die Modellierung eines anderen Aufbaus einer Bildschirmseite reichten die zur Verfügung stehenden Ausdrucksmöglichkeiten aus. Zur Unterstützung anderer Medien können neue Elemente wie Anker in Bildern oder neue komplexe Medienelemente problemlos in die bereits vorhandene Struktur eingefügt werden, da sich die in Abschnitt 10.3.3 vorgestellte Durchlaufstrategie zwar fest an der Struktur der Anordnung der Medien orientiert, für die Codeerzeugung der Medien selbst jedoch flexibel ist. Diese Art der Modellierung hat sich im Rahmen dieser Überlegung daher bewährt und als robust gegenüber der Unterstützung neuer multimedialer Elemente herausgestellt.

Zur Vermeidung dieser Abhängigkeit von einer fest definierten Durchlaufstrategie für eine Zielplattform müsste ein komplett anderer Weg gegangen werden, wie er etwa mit dem Eli-System [Eli 1998] eingeschlagen wurde. Das Eli-System stellt eine Umgebung bereit, mit der

aus einer deklarativen Spezifikation von Eigenschaften eines Compilers dieser automatisiert erzeugt wird. Bei dem Eli-System wird ein Compiler auf diese Weise vollständig spezifiziert, was eine Spezifikation der Durchlaufstrategie mittels attributierter Grammatiken einschließt. Eine Anwendung dieses Ansatzes auf die Problemstellung der XAM würde bedeuten, das keine Konfiguration eines Generierungsframeworks mit fest definierter Durchlaufstrategie stattfindet, sondern bei jedem Start automatisiert eine vollständig neue Erzeugung des Generators auf Grund der bereitgestellten Spezifikation vorgenommen würde. Ob dieser Ansatz praktikabel ist und wie dabei eine Komponentenorientierung angewendet werden kann, in dem etwa XAM-Komponenten Spezifikationsfragmente des Eli-Systems beinhalten, muss gesondert untersucht werden. Auf Grund der Komplexität des Eli-Systems ist dies im Rahmen dieser Arbeit nicht möglich.

Abschließend kann ich feststellen, dass es meines Erachtens mit dem hier vorgestellten Entwurf der XAM gelungen ist, die in Abschnitt 4.3 gesteckten Ziele zu erreichen. Wie im Rahmen der Arbeit dargestellt wurde, ermöglicht es der Entwurf der XAM auf der Basis des abstrakten Modells ein darstellungsneutral beschriebene ξML-Präsentation in etablierte Zielformate zu überführen. Exemplarisch habe ich diese Überführung für die giMLi-Präsentation in das Zielformat HTML verdeutlicht und allgemein gezeigt, wie eine Erweiterung der XAM sowohl um die Unterstützung neuer multimedialer Elemente als auch neuer Zielplattformen möglich ist. Auf Grundlage dieser Diplomarbeit sollte eine Implementierung der XAM möglich sein.

12 Ausblick

Zum Abschluss der Arbeit bleibt es nun noch übrig zu klären, wie es mit der XAM weitergehen kann. Neben ihrer Implementierung, die sich offensichtlich anbietet und zu einer umfassenden Validierung benötigt wird (siehe Abschnitt 11.2), werden im Folgenden noch eine Reihe weiterer Aspekte vorgestellt, die eine detailliertere Betrachtung verdienen, als es im Rahmen dieser Arbeit möglich wäre.

Eine wesentliche Aufgabe bei einer Anwendung der XAM ist, mit den XAM-Komponenten eine geeignete Modellierung der in der darzustellenden ξML-Präsentation verwendeten multimedialen Elemente auf dem Abstraktionsniveau der XAM zu finden, wie es im Rahmen der Arbeit für die giMLi-Präsentation in Kapitel 9 gezeigt wurde. Das abstrakte Modell gibt als Metamodell mit den Basisprimitiven zwar den Rahmen klar vor, in dem sich die Modellierung bewegt, überlässt sie im Detail aber dem Anwender der XAM. Einige solcher Modellierungsalternativen im Rahmen der Bestimmung der XAM-Komponenten für die giMLi-Präsentation habe ich in Abschnitt 9.16.1 vorgestellt. Im Verlauf der Arbeit hat sich gezeigt, dass solche Alternativen unterschiedlich gut von der XAM unterstützt werden. Es erscheint daher eine sinnvolle Ergänzung zu dieser Arbeit zu sein, dem Benutzer der XAM einen Leitfaden an die Hand zu geben, der ein geeignetes Vorgehen bei der Definition der XAM-Komponenten beschreibt. Für die Erstellung eines solchen Leitfadens wäre es sicher wünschenswert, auf Erfahrungen aus einer praktischen Validierung des Entwurfs der XAM zurückgreifen zu können.

Ein weiterer Bereich im Kontext der Modellierungsmöglichkeiten der XAM, der eine nähere Betrachtung verdient, sind die bei der Definition von XAM-Komponenten vorzunehmenden Spezialisierungen des abstrakten Modells. Im Rahmen der Arbeit werden nur XAM-Komponenten definiert, die *direkte* Spezialisierungen *eines* Basisprimitiv sind. Ein solches Vorgehen ist mit dem abstrakten Modell intendiert, wird aber nicht zwingend vorgeschrieben.

Prinzipiell ist es nicht verboten, eine XAM-Komponente als Spezialisierung einer anderen XAM-Komponente zu definieren. Eine solche Spezialisierungshierarchie macht in meinen Augen allerdings nur im Rahmen der Wiederverwendung von Implementierungen der einzelnen Bestandteile der XAM-Komponenten Sinn. Diese Wiederverwendung wäre daher auf Ebene der zur Realisierung verwendeten Programmiersprache angesiedelt und hätte somit auf der Modellierungsebene der XAM keine explizite Entsprechung. Würde dies gewünscht, besteht jedoch prinzipiell auch auf der dazu benötigten Spezifikationsebene mit den in Abschnitt 7.11 beschrieben Mechanismen von XML die Möglichkeit, dies zu tun: Anstatt nur die Eigenschaften und Enthaltenseinsbeziehungen des Basisprimitivs zu übernehmen, müssten auch zusätzlich die Eigenschaften der zu spezialisierenden XAM-Komponenten übernommen werden.

Ebenso wenig ist es explizit verboten, dass eine XAM-Komponenten mehr als ein Basisprimitiv erweitert. Es scheint auf den ersten Blick jedoch nicht sinnvoll, dies zu tun, da das abstrakte Modell ja gerade so definiert wurde, das Konzepte zur Darstellung von Multimediaprä-

sentationen in einzelnen Basisprimitiven getrennt modelliert werden. Durch eine solche mehrfache Spezialisierung von Basisprimitiven würden diese explizit getrennt modellierten Konzepte in einer XAM-Komponente wieder zusammengeführt. Prinzipiell ist eine Beschreibung solcher Beziehungen mit den in Abschnitt 7.11 vorgestellten Mechanismen jedoch möglich: Die Enthaltenseinsbeziehungen und Attribute aller zu spezialisierenden Basisprimitive müssten bei der Definition der XAM-Komponenten aufgenommen werden.

Weiterhin stellt sich die interessante Frage, was passiert, wenn bei einer Spezialisierung nicht alle im zu spezialisierenden Element definierten Enthaltenseinsbeziehungen und Eigenschaften (Attribute) von der spezialisierenden XAM-Komponente übernommen werden, unabhängig davon, ob ein Basisprimitiv oder eine XAM-Komponenten spezialisiert werden soll. Ein Beispiel hierfür wäre etwa die Definition eines Bereiches auf dem Ausgabemedium, der keinen weiteren enthalten darf. Ein solches Vorgehen kann als eine Einschränkung der Vorbedingung der Eingabe einer Spezialisierung aufgefasst werden, was nach dem Prinzip des Design by Contract [MEYER 1992] zu einer Verletzung der Typkonformität in der Spezialisierungshierarchie führt. Da sich diese Einschränkung bei der XAM aber auf statische und nicht auf dynamischen Aspekte wie etwa Wertebelegungen von Attributen bezieht, muss sie nicht unbedingt ein Problem darstellen.

Diese drei gerade vorgestellten Überlegungen zu den Spezialisierungsbeziehungen zwischen XAM-Komponenten und dem abstrakten Modell müssten aber in einer eigenen Arbeit weiter vertieft werden.

Als neuer Punkt wäre es auch überlegenswert, ob das abstrakte Modell um eine explizite Modellierung komplexerer zeitlicher Zusammenhänge, wie sie für die hier betrachteten ξML-Präsentationen nötig sind, erweitert werden kann. Es wäre zu klären, wie eine solche Erweiterung aussehen kann. Anhaltspunkte hierfür bieten die Multimediadokumentmodelle ZyX [BOLL und KLAS 2001] oder SMIL [W3C 2001], die solche zeitlichen Beziehungen explizit unterstützen. In diesen Bereich der Erweiterung des Modells der XAM fällt auch die Frage nach der Definition eines präsentationsweiten Zustandsbegriffs. Auf den ersten Blick scheint jedoch die Unterstützung eines solchen Zustandsbegriffs auf Ebene des abstrakten Modells wegen der sehr unterschiedlichen Umsetzung in den einzelnen Zielplattformen als schwierig. Dies sind aber Erweiterungen der XAM, die für die in dieser Arbeit betrachteten ξML-Präsentation unnötig sind und daher nicht weiter bearbeitet wurden.

Wünschenswert im Rahmen einer Implementierung der XAM wäre sicher auch, dass eine graphische Oberfläche zur Verfügung gestellt wird, die etwa benötigte Einstellungen komfortabel ermöglicht. Falls die XAM eigenständig und nicht im Zusammenhang mit dem ξML-System verwendet werden soll, wäre es auch angemessen, das Eingabedokument als Baum zu visualisieren, an dem etwa syntaktische Fehler hervorgehoben werden. Bei einer solchen eigenständigen Verwendung der XAM müsste auch eine semantische Analyse der Eingabe vorgenommen werden, da diese im ξML-System durch den vorgelagerten Generator erfolgt. Die Steuerung einer solchen semantischen Analyse könnte ebenfalls in einer einheitlichen graphischen Oberfläche der XAM integriert werden.

Im Hinblick auf das eigentliche Anwendungsgebiet der XAM, nämlich das ξML-System, wäre es sicher interessant, wenn neben einer Implementierung der XAM auch eine Implementierung des in [SCHÜTTE 2002] vorgestellten Generators zur Verfügung stünde. Zusammen bilden beide Implementierungen ein rudimentär lauffähiges ξML-System, an dem erste Projekte zur Validierung der Konzepte durchgeführt werden könnten. Obwohl die in diesem Kapitel vorgestellten

Überlegungen zeigen, dass noch viel Arbeit in die XAM investiert werden kann, ist im Entwurf der XAM, wie er in dieser Arbeit vorgestellt wurde, die gesamte für einen solchen realen Einsatz im ξML-System benötigte Funktionalität berücksichtigt worden.

Abschließend kann neben den Möglichkeiten zur Erweiterung ein kurzes Fazit gezogen werden. Wie der Titel der Arbeit „XAM – Eine abstrakte Laufzeitumgebung für ξML“ besagt, habe ich mit der Konzeption der XAM in dieser Arbeit die abstrakte Laufzeitumgebung des ξML-Systems entwickelt. Dabei wurde mit der Definition des abstrakten Modells und der XAM-Komponenten ein klarer Erweiterbarkeitsbegriff der XAM formuliert. Die vorgestellte Konzeption ermöglicht es, einen erweiterbaren und abstrakten Generator für das ξML-System zu erstellen, mit dem eine darstellungsneutrale Beschreibung einer ξML-Präsentation in anzeigbare Formate überführt werden kann, wie am Beispiel der giMLi-Präsentation gezeigt wurde. Zur Umsetzung dieser Funktionalität wurde ein komponentenorientierten Entwurf unter der Verwendung objektorientierten Methoden und geeigneter Architektur- und Entwurfsmuster vorgenommen. An Stellen, an denen sich eine Realisierung der Konzepte des Entwurfs nicht offensichtlich ergibt, wurden Hinweise zur Umsetzung geben. Einer Implementierung der XAM auf Grundlage des hier vorgestellten Entwurfs sollte also nichts mehr im Wege stehen.

A Die DTD für die Manifestdatei der XAM-Komponenten

```
<!-- Das Wurzelelement -->
<!ELEMENT manifest (systemcomponent?, xamcomponent*) >
  <!ATTLIST manifest
  >

<!-- Die Systemkomponente -->
<!ELEMENT systemcomponent (objectbroker+, addressresolver+, medialocator+) >
  <!ATTLIST systemcomponent
  >
<!-- Implementierung fuer den ObjectBroker-Dienst -->
<!ELEMENT objectbroker EMPTY >
  <!ATTLIST objectbroker
      type             CDATA #REQUIRED
      targetplatform   CDATA #IMPLIED
      file             CDATA #REQUIRED
  >
<!-- Implementierung fuer den AddressResolver-Dienst -->
<!ELEMENT addressresolver EMPTY >
  <!ATTLIST addressresolver
      type CDATA #REQUIRED
      file CDATA #REQUIRED
  >
<!-- Implementierung fuer den MediaLocator-Dienst -->
<!ELEMENT medialocator EMPTY >
  <!ATTLIST medialocator
      type CDATA #REQUIRED
      file CDATA #REQUIRED
  >

<!-- Die XAM-Komponente -->
<!ELEMENT xamcomponent (xamlfrag, abstractelement, codegeneration) >
  <!ATTLIST xamcomponent
      name CDATA #REQUIRED
  >
<!-- das dtd-Fragment fuer die DTD der Eingabesprache -->
<!ELEMENT xamlfrag EMPTY >
  <!ATTLIST xamlfrag
      file CDATA #REQUIRED
```

```
39   >
40 <!-- das Abstrakte Element fuer die Zwischendarstellung -->
41 <!ELEMENT abstractelement EMPTY >
42   <!ATTLIST abstractelement
43       file  CDATA #REQUIRED
44   >
45 <!-- Dateien fuer die Codeerzeugung, nach Zielplattformen unterschieden -->
46 <!ELEMENT codegeneration (codegenerator,metadata?) >
47   <!ATTLIST codegeneration
48       targetplatform  CDATA #REQUIRED
49   >
50
51 <!-- der Codeerzeuger -->
52 <!ELEMENT codegenerator EMPTY >
53   <!ATTLIST codegenerator
54       file  CDATA #REQUIRED
55   >
56 <!-- das optionale Metadatenobjekt fuer die Codegenerierung -->
57 <!ELEMENT metadata EMPTY >
58   <!ATTLIST metadata
59       file  CDATA #REQUIRED
60   >
```

B Die DTD für giMLi

```
<!--
     ************************************************************
     ******* gemeinsame Elemente aller xaml-Auspraegungen *******
     ************************************************************
 -->

<!--
    ************************************************
    Listen der Spezialisierungen der Basisprimitive
    ************************************************
 -->
<!-- BOX-Spezialisierungen -->
<!ENTITY % box-spez "simplebox | scrollbox">

<!-- MEDIUM-Spezialisierungen -->
<!ENTITY % medium-spez "background | scene | text | indexedpictures
     | picture">

<!-- POSITION-Spezialisierungen -->
<!ENTITY % pos-spez "absolutposition">

<!-- LINK-Spezialisierungen -->
<!ENTITY % link-spez "simplelink">

<!-- ANCHOR-Spezialisierungen -->
<!ENTITY % anchor-spez "textanchor">

<!-- ACTION-Spezialisierungen -->
<!ENTITY % action-spez "changetheme | changemedium">

<!-- EVENT-Spezialisierungen -->
<!ENTITY % event-spez "onclick">

<!-- STYLE-Spezialisierungen -->
<!ENTITY % style-spez "simplestyle">

<!-- ADDRESS-Spezialisierungen -->
<!ENTITY % addr-spez "simpleaddress">
<!-- ******** Ende der Spezialisierungslisten ********* -->

```

```
<!--
    ************************************************
    Attribute der einzelnen Basisprimitive, die an die
    Abstrakten Elemente quasi vererbt werden
    ************************************************
-->
<!-- Box -->
<!ENTITY % box-attr "
  id      ID    #REQUIRED
  width   CDATA #REQUIRED
  height  CDATA #REQUIRED
  postype CDATA #REQUIRED
  ">

<!-- Medium -->
<!ENTITY % medium-attr "
  id    ID    #REQUIRED
  style IDREF #IMPLIED
  ">

<!-- Link -->
<!ENTITY % link-attr "
  id       ID       #REQUIRED
  source   IDREF    #REQUIRED
  target   IDREF    #REQUIRED
  trigger  NMTOKEN #REQUIRED
  action   IDREF    #REQUIRED
  style    IDREF    #IMPLIED
  addrtype CDATA    #REQUIRED
  ">

<!-- Style-->
<!ENTITY % style-attr "
  id ID #REQUIRED
  ">

<!-- Position -->
<!ENTITY % position-attr "
  id ID #REQUIRED
  ">

<!-- Anchor -->
<!ENTITY % anchor-attr "
  id ID #REQUIRED
  ">

<!-- Addresss -->
<!ENTITY % address-attr "
  id ID #REQUIRED
```

```
89    ">
90
91  <!-- Action -->
92  <!ENTITY % action-attr "
93    id ID #REQUIRED
94    ">
95
96  <!-- Event -->
97  <!ENTITY % event-attr "
98    id ID #REQUIRED
99    ">
100 <!-- ****** Ende der Attribute der Basisprimitve ****** -->
101
102 <!--
103   ************************************************
104   generelle Struktur aller XAML-Dokumente
105   ************************************************
106 -->
107 <!-- Wurzelelement -->
108 <!ELEMENT xaml (structure, navigation, interaction, layout) >
109   <!ATTLIST xaml
110   >
111
112 <!-- die Struktur der Praesentation (logisch + Inhalt) -->
113 <!ELEMENT structure (%box-spez;)+ >
114   <!ATTLIST structure
115   >
116
117 <!-- die Links der Praesentation -->
118 <!ELEMENT navigation ((%addr-spez;)*,(%link-spez;)*) >
119   <!ATTLIST navigation
120   >
121
122 <!-- die moeglichen Ereignisse der Praesentation -->
123 <!ELEMENT interaction ((%event-spez;)*, (%action-spez;)*) >
124   <!ATTLIST interaction
125   >
126
127 <!-- die typografischen Informationen fuer die Praesentation -->
128 <!ELEMENT layout (%style-spez;)* >
129   <!ATTLIST layout
130   >
131 <!-- ********** Ende Struktur von xaml ********** -->
132
133 <!--
134    ************************************************
135    Hilfselemente, fuer alle XAML-Dokumente
136    ************************************************
137 -->
```

```
138
139 <!--
140       Allgemeines Schluessel-Wert-Paar, um Metadaten fuer ein
141       Element definieren zu koennen.
142       Hiermit koennen Informationen abgelegt werden, die
143       ausserhalb des eigentlichen XAM-Typsystems stehen (wie etwa
144       Inhalte von komplexen Medienelementen) oder nicht explizit
145       modelliert werden sollen ( Stil -Informationen)
146 -->
147 <!ELEMENT pair (pair*) >
148   <!ATTLIST pair
149       id    ID    #REQUIRED
150       key   CDATA #REQUIRED
151       value CDATA #REQUIRED
152   >
153 <!-- ********** Ende Hilfselemente ********** -->
154
155 <!-- ***** Ende gemeinsame Elemente aller xaml-Auspraegungen ***** -->
156
157 <!--
158       ************************************************************
159       ******** Spezielle Elemente fuer die gimli-Praesentation ********
160       ************************************************************
161 -->
162
163 <!--
164     ************************************************
165     Fragmente der einzelnen abstrakten Elemente
166     ************************************************
167 -->
168 <!-- abstraktes Element SimpleBox -->
169 <!ELEMENT simplebox ((%pos-spez;), (%medium-spez;), (%box-spez;)*) >
170   <!ATTLIST simplebox
171       %box-attr;
172   >
173
174 <!-- abstraktes Element ScrollBox -->
175 <!ELEMENT scrollbox ((%pos-spez;), (%medium-spez;), (%box-spez;)*) >
176   <!ATTLIST scrollbox
177       %box-attr;
178   >
179
180 <!-- abstraktes Element Background -->
181 <!ELEMENT background EMPTY >
182   <!ATTLIST background
183       %medium-attr;
184   >
185
186 <!-- abstraktes Element Scene -->
```

```
<!ELEMENT scene EMPTY >
  <!ATTLIST scene
     %medium-attr;
  >

<!-- abstraktes Element Picture -->
<!ELEMENT picture EMPTY >
  <!ATTLIST picture
     %medium-attr;
      file  CDATA #REQUIRED
  >

<!-- abstraktes Element Text -->
<!ELEMENT text (textanchor)* >
  <!ATTLIST text
     %medium-attr;
      file  CDATA #REQUIRED
  >

<!-- abstraktes Element indexedpictures -->
<!ELEMENT indexedpictures (pair)+ >
  <!ATTLIST indexedpictures
     %medium-attr;
  >

<!-- abstraktes Element AbsolutPosition -->
<!ELEMENT absolutposition EMPTY >
  <!ATTLIST absolutposition
     %position-attr;
     x CDATA #REQUIRED
     y CDATA #REQUIRED
  >

<!-- abstraktes Element SimpleAddress -->
<!ELEMENT simpleaddress EMPTY >
  <!ATTLIST simpleaddress
     %address-attr;
     boxref IDREF #REQUIRED
  >

<!-- abstraktes Element TextAnchor -->
<!ELEMENT textanchor EMPTY >
  <!ATTLIST textanchor
     %anchor-attr;
      start  CDATA #REQUIRED
     end   CDATA #REQUIRED
  >

<!-- abstraktes Element OnClick-->
```

```
<!ELEMENT onclick EMPTY >
  <!ATTLIST onclick
     %event-attr;
  >

<!-- abstraktes Element ChangeTheme -->
<!ELEMENT changetheme EMPTY >
  <!ATTLIST changetheme
     %action-attr;
  >

<!-- abstraktes Element ChangeMedium -->
<!ELEMENT changemedium EMPTY >
  <!ATTLIST changemedium
     %action-attr;
     refindex  CDATA #REQUIRED
  >

<!-- abstraktes Element SimpleLink -->
<!ELEMENT simplelink EMPTY >
  <!ATTLIST simplelink
     %link-attr;
  >

<!-- abstraktes Element SimpleStyle -->
<!ELEMENT simplestyle (pair)+ >
  <!ATTLIST simplestyle
     %style-attr;
  >
<!-- ********** Ende der abstrakten Elemente ********** -->

<!-- ********************************************************** -->
```

C Das XAM-Eingabedokument für die giMLi-Präsentation

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE xaml PUBLIC "-//UniDo//XAMLDTD 1.0//EN" "xaml.dtd">

<xaml>
 <structure>
  <!-- Thema 1: der basikale Querschnitt -->
  <simplebox id="box1000" width="800px" height="600px" postype="absolut">
   <absolutposition id="pos1000" x="0px" y="0px"></absolutposition>
   <scene id="scen1000">
   </scene>
   <!-- Inhalt von Thema 1 -->
   <!-- ... -->
  </simplebox>

  <!-- Thema 2: Der Aufbau der Mittelschiffwand -->
  <simplebox id="box2000" width="800px" height="600px" postype="absolut">
   <absolutposition id="pos2000" x="0px" y="0px"></absolutposition>
   <scene id="scene2000">
   </scene>

   <!-- die Bildseite in gimli: Bilder mit Bildunterschrift -->
   <simplebox id="box2100" width="385px" height="600px" postype="absolut">
    <absolutposition id="pos2100" x="0px" y="0px">
    </absolutposition>
    <background id="bg2100" style="leftside">
    </background>

    <!-- Bilder mit Bildunterschriften als komplexes Medienelement -->
    <simplebox id="box2110" width="365px" height="580px" postype="absolut"
        >
     <absolutposition id="pos2110" x="10px" y="10px"></absolutposition>
     <indexedpictures id="inpic2110" style="gimlipictures">
      <pair id="pa2110_in1" key="index" value="01">
       <pair id="pa2110_11" key="picturefile" value="pic01.gif"></pair>
       <pair id="pa2110_12" key="captionfile" value="cap01.txt"></pair>
      </pair>
      <pair id="pa2110_in2" key="index" value="02">
       <pair id="pa2110_21" key="picturefile" value="pic02.gif"></pair>
```

```
38        <pair id="pa2110_22" key="captionfile" value="cap02.txt"></pair>
39       </pair>
40       <pair id="pa2110_in3" key="index" value="03">
41        <pair id="pa2110_31" key="picturefile" value="pic03.gif"></pair>
42        <pair id="pa2110_32" key="captionfile" value="cap03.txt"></pair>
43       </pair>
44       <pair id="pa2110_in4" key="index" value="04">
45        <pair id="pa2110_41" key="picturefile" value="pic04.gif"></pair>
46        <pair id="pa2110_42" key="captionfile" value="cap04.txt"></pair>
47       </pair>
48      </indexedpictures>
49     </simplebox>
50    </simplebox>
51
52    <!-- der Farbverlauf in der Mitte -->
53    <!-- Bild als Uebergang -->
54    <simplebox id="box2200" width="30px" height="600px" postype="absolut">
55     <absolutposition id="pos2200" x="385px" y="0px"></absolutposition>
56     <picture id="pic2200" file="edge.gif">
57     </picture>
58    </simplebox>
59
60    <!-- die Textseite in gimli: Inhaltsverzeichnis , Ueberschrift , Fliesstext -->
61    <simplebox id="box2300" width="375px" height="600px" postype="absolut">
62     <absolutposition id="pos2300" x="415px" y="0px">
63     </absolutposition>
64     <background id="bg2300" style="rightside">
65     </background>
66
67     <!-- Inhaltsverzeichnis mit Eintraegen als einzelne Boxen -->
68     <simplebox id="box2310" width="365px" height="90px" postype="absolut">
69      <absolutposition id="pos2310" x="10px" y="10px"></absolutposition>
70      <background id="bg2310" style="toc">
71      </background>
72      <!-- Ueberschrift: Inhaltsverzeichnis -->
73      <simplebox id="box2311" width="365px" height="15px" postype="absolut"
            >
74       <absolutposition id="pos2311" x="0px" y="0px"></absolutposition>
75       <text id="txt2311" file="tocheading.txt" style="tocheading">
76       </text>
77      </simplebox>
78      <!-- Eintrag eins: Der basikale Querschnitt -->
79      <simplebox id="box2312" width="365px" height="15px" postype="absolut"
            >
80       <absolutposition id="pos2312" x="0px" y="15px"></absolutposition>
81       <text id="txt2312" file="toc1.txt" style="tocitem">
82        <textanchor id="ta2312_1" start="1" end="3"></textanchor>
83       </text>
84       </simplebox>
```

```
85      <!-- Eintrag zwei: Der Aufbau der Mittelschiffwand -->
86      <simplebox id="box2313" width="365px" height="15px" postype="absolut"
            >
87       <absolutposition id="pos2313" x="0px" y="30px"></absolutposition>
88       <text id="txt2313" file="toc2.txt" style="tocitem">
89        <textanchor id="ta2313_1" start="1" end="4"></textanchor>
90       </text>
91      </simplebox>
92      <!-- Eintrag drei: Das Mittelschiffgewölbe -->
93      <simplebox id="box2314" width="365px" height="15px" postype="absolut"
            >
94       <absolutposition id="pos2314" x="0px" y="45px"></absolutposition>
95       <text id="txt2314" file="toc2.txt" style="tocitem">
96        <textanchor id="ta2314_1" start="1" end="2"></textanchor>
97       </text>
98      </simplebox>
99      <!-- Eintrag vier: Die Vierung -->
100     <simplebox id="box2315" width="365px" height="15px" postype="absolut"
            >
101      <absolutposition id="pos2315" x="0px" y="60px"></absolutposition>
102      <text id="txt2315" file="toc2.txt" style="tocitem">
103       <textanchor id="ta2315_1" start="1" end="2"></textanchor>
104      </text>
105     </simplebox>
106    </simplebox>
107
108    <!-- Fliesstextueberschrift: Der Aufbau der Mittelschiffwand -->
109    <simplebox id="box2320" width="365px" height="30px" postype="absolut">
110     <absolutposition id="pos2320" x="0px" y="px"></absolutposition>
111     <text id="txt2320" file="heading_th2.txt" style="themeheading">
112     </text>
113    </simplebox>
114
115    <!-- scrollbarer Text -->
116    <scrollbox id="box2330" width="365px" height="460px" postype="absolut">
117     <absolutposition id="pos2330" x="0px" y="px"></absolutposition>
118     <text id="txt2330" file="theme1.txt" style="normaltext">
119      <!-- Mittelschiffwand -->
120      <textanchor id="ta2330_1" start="88" end="88"></textanchor>
121      <!-- außen über die Seitenschiffdächer hinausragt -->
122      <textanchor id="ta2330_2" start="50" end="54"></textanchor>
123      <!-- gemeinsame, durchbrochene Maßwerkfigur -->
124      <textanchor id="ta2330_3" start="143" end="145"></textanchor>
125      <!-- modularen Verkettung dieser Felder -->
126      <textanchor id="ta2330_4" start="169" end="172"></textanchor>
127     </text>
128    </scrollbox>
129   </simplebox>
130
```

```
131   </simplebox>
132
133   <!-- Thema 3: das Mittelschiffgewoelbe -->
134   <simplebox id="box3000" width="800px" height="600px" postype="absolut">
135    <absolutposition id="pos3000" x="0px" y="0px"></absolutposition>
136    <scene id="scene3000">
137    </scene>
138    <!-- Inhalt von Thema 3 -->
139    <!-- ... -->
140   </simplebox>
141
142   <!-- Thema 4: die Vierung -->
143   <simplebox id="box4000" width="800px" height="600px" postype="absolut">
144    <absolutposition id="pos4000" x="0px" y="0px"></absolutposition>
145    <scene id="scene4000">
146    </scene>
147    <!-- Inhalt von Thema 4 -->
148    <!-- ... -->
149   </simplebox>
150  </structure>
151
152  <!-- Definition der Navigationsdimension -->
153  <navigation>
154   <!-- Adressen fuer Thema 1 -->
155   <simpleaddress id="addr1000" boxref="box1000"></simpleaddress>
156   <!-- ... -->
157   <!-- Adressen fuer Thema 2 -->
158   <simpleaddress id="addr2000" boxref="box2000"></simpleaddress>
159   <simpleaddress id="addr2110" boxref="box2110"></simpleaddress>
160   <!-- Adressen fuer Thema 3 -->
161   <simpleaddress id="addr3000" boxref="box3000"></simpleaddress>
162   <!-- ... -->
163   <!-- Adressen fuer Thema 4 -->
164   <simpleaddress id="addr4000" boxref="box4000"></simpleaddress>
165   <!-- ... -->
166
167   <!-- Links fuer Thema 1 -->
168   <!-- ... -->
169   <!-- Links fuer Thema 2 -->
170   <!-- Verweise auf andere Themen -->
171   <simplelink id="tl2001" source="ta2312_1" target="addr1000" trigger="
          onclick" action="act01" style="themelink" addrtype="simple">
172   </simplelink>
173   <simplelink id="tl2002" source="ta2313_1" target="addr2000" trigger="
          onclick" action="act01" style="themelink" addrtype="simple">
174   </simplelink>
175   <simplelink id="tl2003" source="ta2314_1" target="addr3000" trigger="
          onclick" action="act01" style="themelink" addrtype="simple">
176   </simplelink>
```

```
177   <simplelink id="tl2004" source="ta2315_1" target="addr4000" trigger="
          onclick" action="act01" style="themelink" addrtype="simple">
178   </simplelink>
179   <!-- Verweise auf Bilder innerhalb der IndexedPicture -->
180   <simplelink id="ml2001" source="ta2330_1" target="addr2110" trigger="
          onclick" action="act02ref01" style="mediumlink" addrtype="simple">
181   </simplelink>
182   <simplelink id="ml2002" source="ta2330_2" target="addr2110" trigger="
          onclick" action="act02ref02" style="mediumlink" addrtype="simple">
183   </simplelink>
184   <simplelink id="ml2003" source="ta2330_3" target="addr2110" trigger="
          onclick" action="act02ref03" style="mediumlink" addrtype="simple">
185   </simplelink>
186   <simplelink id="ml2004" source="ta2330_4" target="addr2110" trigger="
          onclick" action="act02ref04" style="mediumlink" addrtype="simple">
187   </simplelink>
188   <!-- Links fuer Thema 3 -->
189   <!-- ... -->
190   <!-- Links fuer Thema 4 -->
191   <!-- ... -->
192  </navigation>
193
194  <!-- Definition der Interaktionsdimension -->
195  <interaction>
196   <!-- Ereignisse -->
197   <onclick id="ev01"></onclick>
198   <!-- Aktionen -->
199   <changetheme id="act01"></changetheme>
200   <changemedium id="act02ref01" refindex="01"></changemedium>
201   <changemedium id="act02ref02" refindex="02"></changemedium>
202   <changemedium id="act02ref03" refindex="03"></changemedium>
203   <changemedium id="act02ref04" refindex="04"></changemedium>
204  </interaction>
205
206  <!-- Definition der typografischen Dimension -->
207  <layout>
208   <simplestyle id="leftside">
209    <pair id="paleftside_1" key="color" value="blue"></pair>
210   </simplestyle>
211   <simplestyle id="rightside">
212    <pair id="parightside_1" key="color" value="yellow"></pair>
213   </simplestyle>
214   <simplestyle id="gimlipictures">
215    <pair id="pagimlipictures_1" key="bg-color" value="blue"></pair>
216    <pair id="pagimlipictures_2" key="font-color" value="white"></pair>
217    <pair id="pagimlipictures_3" key="font-size" value="12px"></pair>
218   </simplestyle>
219   <simplestyle id="caption">
220    <pair id="pacaption_1" key="color" value="blue"></pair>
```

```
221   </simplestyle>
222   <simplestyle id="toc">
223    <pair id="patoc_1" key="color" value="yellow"></pair>
224   </simplestyle>
225   <simplestyle id="tocheading">
226    <pair id="patocheading_1" key="font-color" value="gray"></pair>
227   </simplestyle>
228   <simplestyle id="tocitem">
229    <pair id="patocitem_1" key="font-size" value="10px"></pair>
230   </simplestyle>
231   <simplestyle id="themeheading">
232    <pair id="pathemeheading_1" key="font-color" value="black"></pair>
233    <pair id="papathemeheading_2" key="font-size" value="24px"></pair>
234    <pair id="papathemeheading_3" key="font-family" value="'Times New
          Roman', Times, serif"></pair>
235   </simplestyle>
236   <simplestyle id="normaltext">
237    <pair id="panormaltext_1" key="font-color" value="black"></pair>
238   </simplestyle>
239   <simplestyle id="themelink">
240    <pair id="pathemelink_1" key="font-color" value="black"></pair>
241   </simplestyle>
242   <simplestyle id="mediumlink">
243    <pair id="pamediumlink_1" key="font-color" value="black"></pair>
244   </simplestyle>
245  </layout>
246
247 </xaml>
```

Literaturverzeichnis

[ADLER et al. 2001] ADLER, SHARON, A. BERGLUND, J. CARUSO, S. DEACH, T. GRAHAM, P. GROSSO, E. GUTENTAG, A. MILOWSKI, S. PARNELL, J. RICHMAN und S. ZILLES (2001). *Extensible Stylesheet Language (XSL) Version 1.0*. W3C Recommendation 20011015, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3.org/TR/2001/REC-xsl-20011015/.

[AHO et al. 1988a] AHO, ALFRED V., R. SETHI und J. D. ULLMAN (1988a). *Compilerbau, Teil I*. Addison-Wesley Verlag, Bonn.

[AHO et al. 1988b] AHO, ALFRED V., R. SETHI und J. D. ULLMAN (1988b). *Compilerbau, Teil II*. Addison-Wesley Verlag.

[ALFERT 1999] ALFERT, KLAUS (1999). *Developing the Altenberger Dom Presentation — Integrating Content Provider and Software Developers*. In: HAHN, WILFRIED, E. WALTER-KLAUS und J. KNOOP, Hrsg.: *Euromedia '99*, S. 70–77, München, Deutschland.

[ALFERT et al. 1999] ALFERT, KLAUS, E.-E. DOBERKAT und C. KOPKA (1999). *Towards Constructing a Flexible Multimedia Environment for Teaching the History of Arts*. SWT-Memo 101, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[ALFERT et al. 2001] ALFERT, KLAUS, U. HENSEL, J. SCHRÖDER und S. SCHÜTTE (2001). *ξML — Konzepte eines komponentenbasierten Szenarios für multimediale Präsentationen*. interner Bericht, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[APACHE XML PROJECT 2000] APACHE XML PROJECT (2000). *Xerces2 Java Parser Readme*. Apache Software Foundation. Im Internet erhältlich: http://www.xml.apache.org/xerces2-j/.

[APPARAO et al. 1998] APPARAO, VIDUR, S. BYRNE, M. CHAMPION, S. ISAACS, I. JACOBS, A. LE HORS, G. NICOL, J. ROBIE, R. SUTOR, C. WILSON und L. WOOD (1998). *Document Object Model (DOM) Level 1 (Second Edition)*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3c.org/TC/1998/REC-DOM-Level-1-19981001.

[BEN-NATAN 1995] BEN-NATAN, RON (1995). *A Guide to Common Object Request Broker Architecture*. McGraw-Hill.

[BOLL und KLAS 2001] BOLL, SUSANNE und W. KLAS (2001). *ZyX — A Multimedia Document Model for Reuse and Adaption of Multimedia Content*. IEEE Transactions on Knowledge and Data Engineering, 13(3).

[BOLL et al. 1999] BOLL, SUSANNE, W. KLAS und U. WESTERMANN (1999). *A Comparison of Multimedia Document Models concerning advanced Requirements*. Technischer Bericht–Ulmer Informatik-Berichte Nr. 99-01, Universität Ulm.

[BORDEGONI et al. 1997] BORDEGONI, M., G. FACONTI, S. FEINER, M. MAYBURY, T. RIST, S. RUGGIERI, P. TRAHANIAS und M. WILSON (1997). *A Standard Reference model for Intelligent Multimedia Presentation Systems*. Computer Standards and Interfaces, 18(6,7):477–496.

[BOX 1998] BOX, DON (1998). *Essential COM*. Addison-Wesley Publishing Company, Boston.

[BRAY et al. 2000] BRAY, TIM, E. MALER, J. PAOLI und C. M. SPERBERG-MCQUEEN (2000). *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3c.org/TC/2000/REC-xml-20001006.

[BULTERMAN et al. 1991] BULTERMAN, DICK C.A, G. VAN ROSSUM und R. VAN LIERE (1991). *A Structure for Transportable, Dynamic Multimedia Documents*. In: *Proceedings of the Summer 1991 USENIX Conference*, S. 137–155, Nashville, TN.

[BUSCHMANN et al. 1996] BUSCHMANN, FRANK, R. MEUNIER, H. ROHNERT, P. SOMMERLAD und M. STAL (1996). *Pattern-oriented Software Architecture Vol 1: A System of Patterns*, Kap. 2.5 Adaptable Systems. John Wiley & Sons Ltd., Chichester, England.

[CARSON 1993] CARSON, GEORGE W. (HRSG.) (1993). *Introduction to the Computer Graphics Reference Model*. Computer Graphics, 27(2):108–119.

[CLARK 1999] CLARK, JAMES (1999). *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3c.org/TC/1998/REC-xslt-19991116.

[DEROSE et al. 2001] DEROSE, STEVE, E. MALER und D. ORCHARD (2001). *XML Linking Language (XLink) Version 1.0*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3.org/TR/xlink.

[DOBERKAT 1996] DOBERKAT, ERNST-ERICH (1996). *A Language for Specifying Hyperdocuments*. Software - Concepts and Tools, 17(4):163–172.

[D'SOUZA und WILLS 1999] D'SOUZA, DESMOND FRANCIS und A. C. WILLS (1999). *Objects, Components and Frameworks with UML: The Catalysis Approach*. Object Technology Series. Addison-Wesley Publishing Company, Boston.

[EGENHOFER und FRANZOSA 1991] EGENHOFER, MAX J. und R. FRANZOSA (1991). *Point-Set Topological Spatial Relations*. International Journal of Geographic Information Systems, 5(2).

[Eli 1998] ELI (1998). *Guide for New Eli Users*. Compiler Tools Group, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO, USA.

[ENCYCLOPAEDIA BRITANNICA 1978] ENCYCLOPAEDIA BRITANNICA, Hrsg. (1978). *The New Encyclopaedia Britannica, Macropaedia*, Bd. 1, Kap. „Architecture“, S. 1088–1115. Encyclopaedia Britannica Inc., Chicago.

[ENGLANDER 1997] ENGLANDER, ROBERT (1997). *Developing Java Beans*. Java Series. O'Reilly & Associates, Inc.

[EPSTEIN 1998] EPSTEIN, BRUCE A. (1998). *Lingo in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[FLANAGAN 2001] FLANAGAN, DAVID (2001). *JavaScript: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 4. Aufl.

[FOEGEN und BATTENFELD 2001] FOEGEN, MALTE und J. BATTENFELD (2001). *Die Rolle der Architektur in der Anwendungsentwicklung*. Informatik Spektrum, 24(5):290–301.

[FRONK 2001] FRONK, ALEXANDER (2001). *Algebraische Semantik einer objektorientierten Sprache zur Spezifikation von Hyperdokumenten*. Doktorarbeit, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[GAEDKE et al. 1999] GAEDKE, MARTIN, D. SCHEMPF und H.-W. GELLERSEN (1999). *WCML: An enabling technology for the reuse in object-oriented Web Engineering*. In: *Poster-Proceedings of the 8th International World Wide Web Conference (WWW8)*.

[GAMMA et al. 1996] GAMMA, ERICH, R. HELM, R. JOHNSON und J. VLISSIDES (1996). *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Professionelle Softwareentwicklung Serie. Addison-Wesley Verlag, Bonn. Übersetzung aus dem Amerikanischen von Dirk Riehle.

[GARZOTTO et al. 1993] GARZOTTO, FRANCA, P. PAOLINI und D. SCHWABE (1993). *HDM - A Model-Based Approach to Hypertext Application Design*. Information Systems, 11(1):1–26.

[GHEZZI et al. 1991] GHEZZI, CARLO, M. JAZAYERI und D. MANDRIOLI (1991). *Fundamentals of Software Engineering*. Prentice-Hall, Englewood Cliffs, New Jersey.

[GOSLING et al. 2000] GOSLING, JAMES, B. JOY, G. STEELE und G. BRACHA (2000). *Te $Java^{TM}$ Language Specification*. The $Java^{TM}$ Series. Addison-Wesley Publishing Company, Boston, 2. Aufl.

[GRIFFEL 1998] GRIFFEL, FRANK (1998). *Componentware: Konzepte eines Softwareparadigmas*. dpunkt-Verlag, Heidelberg.

[GROSS et al. 2000] GROSS, PHIL, F. ELLEY und K. TUCKER (2000). *Director 8 and Lingo Authorized*. Macromedia Press.

[GÖTZE 1995] GÖTZE, RAINER (1995). *Dialogmodellierung für multimediale Benutzerschnittstellen*. Teubner Verlag, Leipzig.

[HALASZ und SCHWARTZ 1994] HALASZ, FRANK und M. SCHWARTZ (1994). *The Dexter Hypertext Reference Model*. Communications of the ACM, 37(2):31–39.

[HARDMAN et al. 1998] HARDMAN, LYNDA, D. C. A. BULTERMAN, J. JANSEN, K. MULLENDER und L. RUTLEDGE (1998). *GRiNS: A GRaphical INterface for creating and playing Smil Documents*. In: *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, Melbourne, Australien.

[HARDMAN et al. 1994] HARDMAN, LYNDA, D. G. A. BULTERMAN und G. VAN ROSSUM (1994). *The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model*. Communications of the ACM, 37(2):50–62.

[HARDMAN et al. 1993] HARDMAN, LYNDA, G. VAN ROSSUM und C. A. BUTLERMAN (1993). *Structured Multimedia Authoring*. In: *Computer Graphics (Proceedings of the ACM Multimedia '93)*, S. 283–289. ACM Press.

[HEIDUCK 1999] HEIDUCK, MATTHIAS (1999). *Konzeption einer Spezifikationssprache zur prototypischen Beschreibung von Multimediapäsentationen*. Diplomarbeit, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[HELIO 1999] HELIO (1999). *SOJA „Cherbourg 2"*. Im Internet erhältlich: `http://www.helio.org/products/smil/`.

[HERMAN und DUKE 1997] HERMAN, I. und D. J. DUKE (1997). *Programming paradigms in an object-oriented multimedia standard*. CWI research report INS-R9705, CWI.

[HERMAN et al. 1996] HERMAN, IVAN, G. J. REYNOLDS und J. VAN LOO (1996). *PREMO: An Emerging Standard for Multimedia Presentation*. IEEE MultiMedia, 3(3):83–89.

[ISO-SGML 1986] ISO-SGML (1986). *Imformation Processing - Text and Office System - Standard Generalized Markup Language (SGML)*. ISO. ISO-IS 8879.

[JOURDAN et al. 1998] JOURDAN, MURIEL, N. LAYAÏDA, C. ROISIN, L. SABRY-ISMAIL und L. TARDIF (1998). *Madeus, an Authoring Environment for Interactiv Multimedia Documents*. In: *Proceedings of the sixth ACM international conference on Multimedia 1998*, S. 267–272, Bristol, UK.

[KERNIGHAN und RITCHIE 1988] KERNIGHAN, BRIAN W. und D. M. RITCHIE (1988). *The C Programming Language*. Prentice-Hall International, Englewood Cliffs, New Jersey, ANSI C, 2. Aufl.

[LIE und BOS 1996] LIE, HÅKON WIUM und B. BOS (1996). *Cascading Style Sheets, Level 1*. W3C Recommendation REC-CSS1-19990111, World Wide Web Consortium (W3C). Im Internet erhältlich: http://www.w3.org/TR/1999/REC-CSS1-19990111.

[LINDHOLM und YELLIN 1997] LINDHOLM, TIM und F. YELLIN (1997). *$Java^{TM}$ Tutorial – Die Spezifikation der virtuellen Maschine*. *$Java^{TM}$* Series. Addison-Wesley-Longman, Bonn. Deutsche Übersetzung aus dem Amerikanischen von Dorothea Heymann,Birgit Krehl, Ralf Lübeck, Arnulf Mester, Michael Sczittnick, Dirk Steinkamp, Frank Wegmann.

[MAYBURY 1993] MAYBURY, MARK T., Hrsg. (1993). *Intelligent Multimedia Interfaces*. AAAI Press / The MIT Press.

[MEGGINSON 1998] MEGGINSON, DAVID (1998). *Simple API for XML (SAX)*. Im Internet erhältlich: http://www.saxproject.org.

[MEYER 1992] MEYER, BETRAND (1992). *Applying Design by Contract*. IEEE Computer; Special Issue on Inheritance and Classification, 25(10):40–52.

[MONSON-HAEFEL 2001] MONSON-HAEFEL, RICHARD (2001). *Enterprise JavaBeans*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3. Aufl.

[NANGA et al. 2002] NANGA, CHAK, S. LOGAN und C. SEAWOOD (2002). *Mozilla Runtime Environment (MRE)*. The Mozilla Organization. Im Internet erhältlich: http://www.mozilla.org/projects/embedding/MRE.html.

[NEWCOMB et al. 1991] NEWCOMB, STEVEN R., N. A. KIPP und V. T. NEWCOMB (1991). *The "HyTime" Hypermedia/Time-based Document Structuring Language*. Communications of the ACM, 34(11):67–83.

[OPERA SOFTWARE 2002] OPERA SOFTWARE (2002). Im Internet erhältlich: http://www.opera.com.

[OPÉRA 1997] OPÉRA (1997). *THOT, A structured document editor*. INRIA, Monbonnot, Frankreich. Erhältlich nur im Internet: http://www.irialpes.fr/opera/Thot.en.html.

[ORATRIX DEVELOPMENT BV 2001] ORATRIX DEVELOPMENT BV (2001). *GRiNS*. Im Internet erhältlich: http://www.oratrix.com/GRiNS/.

[ORFALI et al. 1996] ORFALI, ROBERT, D. HARKEY und J. EDWARDS (1996). *The Essential Distributed Objects Survival Guide*. John Wiley & Sons Ltd., Chichester, England.

[PLEUMANN 2000] PLEUMANN, JÖRG (2000). *dodl2html – Ein Generator zum Erzeugen von graphspezifizierten Hyperdokumenten*. Diplomarbeit, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[QUINT und VATTON 1997] QUINT, V und I. VATTON (1997). *An Introduction to Amaya*. World Wide Web Journal, 2(2):39–46.

[RAY 2001] RAY, ERIK T. (2001). *Einführung in XML*. O'Reilly Verlag GmbH & Co. KG, Köln. Deutsche Übersetzung von Olaf Brodacki.

[ROISIN 1998] ROISIN, CECILE (1998). *Authoring Structured Multimedia Documents*. In: ROVAN, B., Hrsg.: *25th Conference on Current Trends in Theory and Practice of Informatics*, S. 222–239. Springer-Verlag.

[RUTLEDGE et al. 1998] RUTLEDGE, LLOYD, L. HARDMAN, J. VAN OSSENBRUGGEN und D. C. BULTERMAN (1998). *Structural Distinctions Between Hypermedia Storage and Presentation*. In: *Proceedings of the sixth ACM international Conference on Multimedia*, S. 145–150, Bristol, UK.

[SCHÜTTE 2002] SCHÜTTE, SEBASTIAN (2002). *Entwurf und Bewertung eines Kompositionskonzeptes für ξML-Komponenten*. Diplomarbeit, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund.

[SCHWABE und ROSSI 1995] SCHWABE, DANIEL und G. ROSSI (1995). *The object-oriented hypermedia design model*. Communications of the ACM, 38(8):45–46.

[SILBERSCHATZ und GALVIN 1994] SILBERSCHATZ, ABRAHAM und P. B. GALVIN (1994). *Operating Systems Concepts*. Addison-Wesley Publishing Company, Reading, Massachusetts, 4. Aufl.

[Sun 1999] SUN (1999). *Java Media Framework Api Guide (JMF 2.0 FCS)*. Sun Microsystems, Inc. Im Internet erhältlich: `http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html`.

[TOLKIEN 1983] TOLKIEN, JOHN RONALD REUEL (1983). *The Lord of the Rings*. Unwin Paperbacks.

[VAN ROSSUM et al. 1993] VAN ROSSUM, GUIDO, J. JANSEN, K. S. MULLENDER und D. C. BULTERMAN (1993). *CMIFed: A Presentation Environment for Portable Hypermedia Documents*. In: *Computer Graphics (Proceedings of the ACM Multimedia '93)*, S. 183–188. ACM Press.

[VILLARD et al. 2001] VILLARD, LIONEL, C. ROISIN und N. LAYAÏDA (2001). *A XML-Based Multimedia Document Processing Model for Content Adaptation*. In: *Eighth International Conference on Digital Documents and Electronic Publishing*. Technische Universität München, Springer-Verlag.

[W3C 1999] W3C (1999). *HTML 4.01 Specification*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: `http://www.w3.org/TR/html4`.

[W3C 2001] W3C (2001). *Synchronized Multimedia Integration Language (SMIL 2.0)*. W3C Recommendation, World Wide Web Consortium (W3C). Im Internet erhältlich: `http://www.w3.org/TR/2001/REC-smil20-20010807/`.

[WAITE und GOOS 1985] WAITE, WILLIAM MCCASTLINE und G. GOOS (1985). *Compiler Construction*. Texts and Monographs in Computer Science. Springer-Verlag, New York.

[WILLIAMSON und EPSTEIN 2001] WILLIAMSON, HEATHER und B. A. EPSTEIN (2001). *Dreamweaver in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.