Ein System zur endnutzergesteuerten Linkdefinition in Webkontexten

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker(FH)

an der
Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Betreuer: Prof. Dr. H. Hansen
2. Betreuer: Dr. T. Schmidt

Erstellt von: Michael Nam Engelhardt

Berlin, 4. November 2002

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Entwicklung der letzen Jahre hat das Internet zu einer umfangreichen Informationsquelle gemacht. Aus dem ehemaligen ARPANET, das nur wenige Knoten umfaßte, konzipiert, um nach einem möglichen Atomschlag kommunikationsfähig zu bleiben, entwickelte sich am Ende des zwanzigsten Jahrhunderts ein weltumspannendes Kommunikationsnetz mit mehreren Millionen Servern (vergleiche [1]). Firmen und öffentliche Institutionen entdeckten das Internet als Plattform zur Publizierung und zum Marketing. Mit den Schlagworten „eCommerce“ und „eGovernment“ wurde auch der Begriff „eLearning“ geboren, der sowohl Techniken als auch Methoden des computergestützten oder computerunterstützten Lernens zusammenfaßt. Trotzdem existiert keine genaue Definition von dem, was mit eLearning bezeichnet wird. So werden unter diesem Begriff sowohl die technischen Umsetzungen, als auch neue Wege Wissen zu erschließen, gesammelt. Im Rahmen dieser Arbeit bezeichnet eLearning Techniken zur Implementierung internetbasierter Lernumgebungen.

Die Einleitung zum zweiten Kapitel formuliert einige Anforderungen an eLearning–Anwendungen, die dann als Ausgangspunkt für die Betrachtungen, das Internet als Basis für Lernumgebungen zu verwenden, der nachfolgenden Kapitel dient. Hierzu wird zunächst die historische Entwicklung von Hyperreferenz-Systemen reflektiert und auf zwei grundlegende Modelle eingegangen. Vor diesem Hintergrund werden die bestehenden Techniken, wie beispielsweise HTML und sein Verweismodell vorgestellt und untersucht, inwieweit diese den in der Einleitung des Kapitels genannten Kriterien gerecht werden. Da das MIR–System den Projektrahmen dieser Diplomarbeit vorgibt und auch für die Implementierung von Bedeutung ist, werden abschließend die Ideen und Konzepte von „MIR“ vorgestellt.

Aufbauend auf der Erkenntnis, daß Hyperreferenzen als Mittel zur Gestaltung der Interaktivität einer Online–Lernanwendung eine zentrale Rolle einnehmen, betracht Kapitel 3 detailliert dieses Problemfeld. Aus der Diskussion der inhaltlichen Anforderungen sowohl seitens des Autors, als auch des Lernenden an solche Applikationen, wird die Idee der Verweiskontexte entwickelt.

Abschließend werden die aus dem vorangegangen Diskurs herausgearbeiteten technischen Anforderungen erläutert.

Die eingehende Betrachtung des im Rahmen dieser Arbeit entwickelten Konzeptes zur Umsetzung der Verweiskontexte und eines entsprechenden Verweismodells erfolgt in Kapitel 4. Das Konzept ist grundsätzlich unabhängig von einem bestimmten System, einige detaillierte Ausführungen werden aber durch den vorgegebenen Projektrahmen des MIR–Systems beeinflußt.

Bestandteil des darauffolgenden Kapitels ist die prototypische Umsetzung des entwickelten und vorgestellten Konzeptes auf Basis des MIR–Systems. Um die konzeptionelle Unabhängigkeit vom MIR–System aufzuzeigen, endet dieser Teil der Arbeit mit einer Beschreibung einer alternativen Implementierung.

Um Möglichkeiten und Nutzen der Verweiskontexte zu demonstrieren wird in Kapitel 6 ein denkbares Anwendungsszenarium vorgestellt.

Ein Fazit und ein Ausblick schließen diese Arbeit zusammenfassend ab und zeigen Anknüpfungspunkte für nachfolgende Projekte auf.

Zielsetzung dieser Diplomarbeit ist die Entwicklung eines Konzeptes, das über die Definition semantischer Beziehungen zwischen Verweisen dem Anwender eine gezielte Auswahl der darzustellenden Verweise ermöglicht und die prototypische Implementierung im vorgegebenen Projektrahmen. Auf Ebene der Präsentation der Daten dem Nutzer gegenüber sollen als Front–End Webbrowser agieren, so daß also Kompatibilität zu diesen erhalten bleiben muß.

Die Möglichkeit hypermediale Inhalte zu verknüpfen, wird als Chance begriffen, neue Wege des interaktiven Wissenerwerbs zu gestalten, wobei die Sichtweise auf mögliche Probleme weniger vom pädagogischen Standpunkt aus, als vielmehr vor dem Hintergrund der softwaretechnischen Umsetzung solcher Systeme betrachtet werden.

Kapitel 2

Das Internet als Basis für Lernumgebungen

2.1 Einleitung

Mit der Entwicklung des Internets, insbesondere des World–Wide–Webs, zum Massenmedium in den 90ziger Jahren des vergangenen Jahrhunderts, wurde diese Technologie zunehmend für die Erstellung von Lernumgebungen interessant. Die Bedienbarkeit und Medienintegration in Browser verbessert sich ständig. Somit sind internetbasierte Lernumgebungen mittlerweile eine interessante Alternative zum klassischen Lehrbuch geworden.

Was zeichnet jedoch ernstzunehmende – lerntaugliche – Anwendungen aus? Zwei Punkte spielen hierfür eine entscheidende Schlüsselrolle: die Qualität des Inhalts und die Interaktionsmöglichkeiten. Qualitativ hochwertige Inhalte zeichnen sich einerseits durch eine didaktisch an die entsprechende Lernsituation angepaßte Darstellung und Gliederung aus, andererseits sollten diese Inhalte viele Ausdrucksmöglichkeiten (*Multivitalität*), also einen hohen Grad an Wiederverwendbarkeit, bieten. Anforderungen an die Interaktivität sind die Unterstüzung der Inhaltsgliederung und des zu Grunde liegenden didaktischen Modells. Unterstützung heißt hierbei, daß die jeweiligen Konzepte der Lernanwendung einfach und nachvollziehbar in das zu verwendende System abgebildet werden können. Um einen hohen Grad an Multivitalität zu ermöglichen, sollte das System eine feingranulare Speicherung von Inhalten ermöglichen, sowie deren Wiederverwendbarkeit, von einzelnen Inhaltsstücken bis hin zu kompletten Inhalten, unterstützen. Letztendlich dürfen die Aspekte der Unterstützung des Autors bei der Erstellung von Inhalten und die Anwendbarkeit eines solchen Systems nicht vernachlässigt werden. So sollten sowohl dem Nutzer, als auch dem Autor leistungsfähige Werkzeuge oder Hilfsmittel zur Verfügung gestellt werden (vergleiche [EHK+02]). Unter Beachtung der Interaktivität als Schlüsselgröße

und der Hyperreferenz als Interaktionsmoment im Word–Wide–Web, folgen hieraus insbesondere bestimmte Anforderungen an die Fähigkeiten und Umsetzung der Hyperreferenzen, im zu Grunde liegenden System.

Hypertext-/Hypermedia Systeme im allgemeinen erfüllen die obengenannten Anforderungen unterschiedlich gut. Interaktion erfolgt in Hypermedia Systemen durch die Verknüpfung von Inhalten durch Hyperreferenzen. Um eine Grundlage für den Vergleich unterschiedlicher Systeme zu schaffen, wird in Abschnit 2.3.1 das Dexter Hypertext Referenz Modell und in 2.3.2 das Amsterdam Hypermedia Modell, insbesondere hinsichtlich seiner Dexter-Kritik, vorgestellt.

Bezugnehmend auf die Anforderungen eines *Online*–Lernsystems auf Basis der World–Wide–Web Technologien werden diese im Abschnitt 2.4 vorgestellt und hinsichtlich ihrer Fähigkeiten der Inhaltsverarbeitung und Interaktion diskutiert. In diesem Zusammenhang wird auf die Trennung von Struktur, Inhalt und Layout, die durch die Verwendung von XML ermöglicht wird, sowie auf die Notwendigkeit der Beschreibung von Verweiszielen und Inhalten, eingegangen. Diese Betrachtungen führen letztendlich zu den Konzepten des „semantischen Webs“, wofür stellvertretend die Technologien RDF und TopicMaps vorgestellt werden.

Mit dem Media Information Respository (MIR) zeigt Abschnitt 2.6 ein fortgeschrittenes System zur Inhaltsspeicherung und -bearbeitung auf. Unter dem Gesichtspunkt der granularen Speicherung und Verarbeitung von Inhalten erlaubt das MIR System komplexe Inhaltstrukturen in einzeln adressierbare Teile, sogenannte Inhaltszellen (*Content Cells*), zu zerlegen. Für jede dieser Content Cells besteht die Möglichkeit, diese mit Metainformationen zu versehen und wiederzuverwenden. Aus vorhandenen Inhaltszellen können wiederum neue komplexe, adressierbare und verarbeitbare Inhaltsstrukturen erstellt werden.

In seiner Gesamtheit erörtert dieses Kapitel Möglichkeiten und Defizite vorhandener Technologien, um im folgenden Kapitel eine fundierte Schwerpunktdiskussion hinsichtlich der Adaption, Hyperreferenz und kontextabhängigen Darstellung von Inhalten zu führen, da diese wiederum Ausdruck der Interaktion für Online–Lernsysteme darstellen.

2.2 Historische Entwicklung

Bereits 1945 entwickelte Bush [Bus45] die Idee, Informationen verknüpft zu verarbeiten. Ausgehend von der Erkenntnis eines stetig wachsenden Informationsbestandes und der daraus resultierenden Gefahr des Verlustes von Informationen, entwirft er den Memory Extender *Memex*:

> *»A Memex is a device in which an individual stores all his books, records, and communictations, and which is mechanized so that it my be consulted with exceeding speed and flexibility. [...] It affords an immediate step, however, to associative*

> *indexing, the basic idea of which is a provision whereby any item my be caused at will to select immediately and automatically another.«*

Aufbauend auf diesen Gedanken entwickelten Anfang der 60ziger Jahre Douglas C. Engelbart, William K. English und John F. Rulifson am Stanford Research Institute, das *oN-Line System* [Eng63], als erstes auf Ebene eines Wordprozessors existierendes Hypertextsystem.

Ted Nelson prägte 1965 die Begriffe *Hypertext* und *Hypermedia* im Zusammenhang mit seinem Design eines Universal Hypermedia Systems (*Xanadu*), dessen Grundidee in der Umsetzung eines generischen Repositories liegt, indem beliebige Daten von jedermann gespeichert und untereinander verknüpft werden können. Xanadu wurde allerdings nie vollständig implementiert (vergleiche [Nie95] Seite 37f).

Das File Retrieval and Editing System *FRESS*, welches zur Dokumentation der Apollo Mondlandungen benutzt wurde, war das Ergebnis erster umfangreicher Implementierungen an der Brown University unter Andries van Dam [Dam88] im Verlauf der 60ziger Jahre. Grundlage hierfür war das von Andries van Dam und Ted Nelson 1968 entwickelte System *HES* (Hypertext Editing System). Später begann hier die Entwicklung von *Intermedia* (1985).

Das Verweismodell (*Linking Model*) von Intermedia bestand aus zwei Ankern (*Anchors*) zur Subadressierung, verbunden durch einen bi–direktionalen Verweis (*Link*). Adressiert werden konnten hierbei Texte und Grafiken, die Verweise wurden dabei getrennt vom Inhalt gespeichert. Intermedia wird seit Version 3.0 (1989) von IRIS kommeriziell vermarktet.

Zwei Jahre nach dem Beginn der Entwicklung von Intermedia an der Brown University lieferte Apple das kommerzielle *HyperCard* System kostenlos mit jedem verkauften PC aus. HyperCard verfolgt die Metapher des „Kartenstapels“. Die Karten können in beliebiger Reihenfolge durchgeblättert oder aus dem Stapel „gezogen“ werden, wobei auf jeder einzelnen Karte beispielsweise Texte und Grafiken angeordnet werden. Ein Nachteil des HyperCard Systems war das Verweismodell, das keine Verweise im fortlaufendem Text erlaubt, sondern nur an sogenannten Schnittstellenobjekten, wie zum Beispiel Buttons (vergleiche [2]).

Zwischenzeitlich wurden die unterschiedlichen hyperreferentiellen Systeme und ihre Ansätze zu einem eigenen Fokusthema. 1988 wurde in zwei kleinen Workshops im Dexter Inn in New Hamshire das sogenannte *Dexter Hypertext Reference Model* (siehe auch Abschnitt 2.3.1 Seite 6) entwickelt.

In der darauffolgenden zweiten Generation der Hypertext/Hypermedia Systeme entwickelten sich neben anderen Systemen Hyper-G (jetzt das kommerzielle Hyperwave)[3], Microcosm[HDH96] und das *World Wide Web*.

Mit dem *Amsterdam Hypermedia Model* (siehe Abschnitt 2.3.2 Seite 9) im Jahre 1993 und seiner Dexter-Kritik hinsichtlich der Notwendigkeit der Einbeziehung von Zeit und Kontext in den

Prozess der Hyperreferenzierung, wurde der Grundstein für die dritte Generation der Hypermedia Systeme gelegt, in der wir uns heute befinden. Neben der Berücksichtigung von Zeit- und Kontextabhängigkeiten, wie beispielsweise konditionelle Links, sind vor allem Themen wie Strukturierbarkeit und einfache Editierbarkeit (Erstellung, Pflege und Archivierung/Versionierung) von Dokumenten von Bedeutung.

2.3 Hyperreferenz Modelle

Wird über Hyperreferenz–Systeme gesprochen, so bezeichnet man damit einer Reihe von Systemen mit unterschiedlichen Eigenschaften und Fähigkeiten. Beispielsweise waren Systeme wie KMS[AMY88] oder Hypercard auf eine feste Bildgröße festgelegt, im Gegensatz zu Intermedia (vergleiche [HS94], Seite 30). Grundsätzlich existieren vielerlei solcher Unterschiede, die es kompliziert machen, angemessene Vergleichskriterien zu finden. Diese nicht neue Erkenntnis führte letztendlich zum Entwurf formaler Modelle, um Hyperreferenz–Systeme zu beschreiben. Zwei grundlegende Modelle sollen im folgenden vorgestellt werden: das Dexter Hypertext Referenz Modell und das Amsterdam Hypermedia Modell, wobei letzteres insbesondere in seiner Kritik am Dexter Modell betrachtet wird.

Weiterhin legen diese Modelle Begriffe fest, die für eine qualifizierte Diskussion hinsichtlich der Online–Lernsysteme und Technologien ihrer Umsetzung grundlegend sind.

2.3.1 Das Dexter Hypertext Referenz Modell

Das Dexter Hypertext Referenz Modell[1] [HS94] wurde 1988 in zwei Workshops unter Beteiligung von Vertretern der wichtigsten damals existierenden Hypertext–Systeme erarbeitet. Das Modell unterteilt ein System in drei Schichten (*Layers*), die vollständig voneinander getrennt sind und über festgelegte Schnittstellen kommunizieren (siehe Abbildung 2.1 auf Seite 7):

Storage Layer: Das Storage Layer beschreibt eine Datenbasis, bestehend aus einer Hierarchie datenbeinhaltender Knoten oder Komponenten, die durch sogenannte Verweise (*Links*) miteinander verbunden sind und deren Speicherung. Die Komponenten können unterschiedliche Datentypen wie beispielsweise Texte oder Bilder enthalten, werden vom Dexter Modell aber als generische Datenkontainer behandelt. Schwerpunkt des Storage Layers ist der Umstand, wie Verweise und Komponenten, die keine Verweisinformationen tragen zu einem Hypertext Netzwerk verbunden werden. Dexter unterscheidet drei Arten von Komponenten:

[1] kurz: Dexter Modell

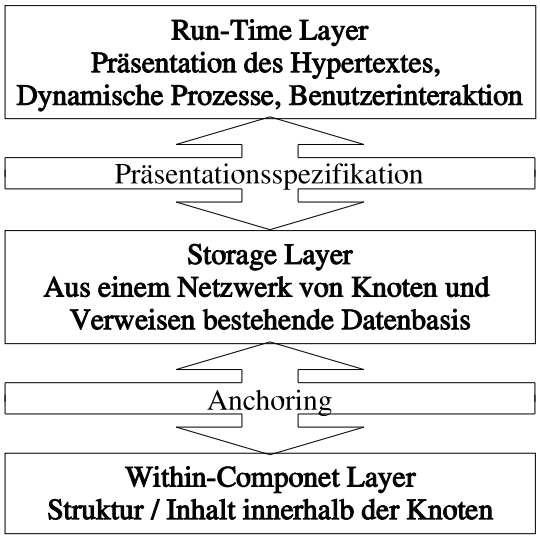

Abbildung 2.1: Dexter Hypertext Referenz Modell
(vergleiche [HS94] Seite 33)

a) *Atomic Components*: nicht zusammengesetzte Komponenten, deren Substruktur im Storage Layer nicht weiter behandelt wird,

b) *Links*: als Repräsentanten von Beziehungen zwischen Komponenten,

c) *Composite Components*: bestehend aus einer Reihe von beliebigen Komponenten, jedoch mit der Einschränkung, daß keine direkten oder indirekten Zyklen erzeugt werden dürfen.

Within–Component Layer: Im Gegensatz zum Storage Layer beschreibt das Within–Component Layer den Umgang mit den Daten innerhalb der Komponenten. Aufgrund der möglichen Vielfältigkeit dieser Daten verweist das Dexter Modell auf spezielle Modelle zur Modellierung und Erzeugung solcher Objekt- und Datenstrukturen.

Run–Time Layer: Die Aufgabe des Run–Time Layers besteht in der Abbildung der aktiven Aspekte eines Hypertext–Systems, wie beispielsweise die Anzeige der Daten oder Nutzerinteraktionen. Auch hier werden aufgrund der unterschiedlichen Präsentations- und Anwendungsmöglichkeiten verschiedener Hypertext–Systeme nur eine Reihe allgemeingültiger Mechanismen definiert.

Als Schnittstelle zwischen dem Storage und dem Within–Component Layer fungiert das *Anchoring*, das den Zugriff auf Daten, die im Storage Layer als abstrakt gelten, erlaubt. Somit ist nicht nur die Verknüpfung von Komponenten möglich, sondern steht darüber hinaus ein Mechanismus zur Subadressierung von Daten innerhalb von Komponenten zur Verfügung.

Links oder Verweise stellen im Dexter Modell eine Verbindung zwischen zwei Komponenten dar. In Hypertext–System im Allgemeinen werden Links zwischen mindestens zwei Endpunkten gebildet. Im Dexter Modell werden solche Endpunkte über sogenannte *Specifier* beschrieben, die aus der Angabe der entsprechenden Komponente (*Component Specification*), einer lokalen Anker Identifikation (*Anchor ID*) und zwei zusätzlichen Feldern Richtung (*Direction*) und Präsentationsspezifikation (*Presentation Specification*) bestehen.

Die Richtung eines Verweises kann vier unterschiedliche Werte annehmen (vergleiche [4] Seite 329):

BIDIRECT: der Specifier kann sowohl Startpunkt als auch Zielpunkt sein.

FROM: kennzeichnet einen Startpunkt, bietet zudem aber die Möglichkeit Verweise in beide Richtungen zu verfolgen, wenn der Zielpunkt das Attribut TO besitzt (Rücksprung vom Zielpunkt).

NONE: dient zum Aufbau von unidirektionalen Verweisen, der Startpunkt erhält den Wert NONE, der Zielpunkt das Attribut TO. Nach der Traversion des Verweises ist der Startpunkt nicht mehr feststellar.

TO: dient zur Definition von Zielpunkten.

Die Präsentationsspezifikation wird von der gleichlautenden Schnittstelle zwischen Storage und Run–Time Layer verwendet, um einen Anker dem Nutzer gegenüber sichtbar zu machen. Die durch das Run–Time Layer visualisierten Anker werden als *Link Marker* bezeichnet.

Obwohl das Dexter Modell auch heutzutage als Referenzmodell akzeptiert wird, existieren einige Kritikpunkte. Es wird zwar die Erstellung von zusammengesetzen Komponenten (Composite Components) vorgesehen, aber hierauf keine Kontextabhängigkeit von Links erkannt und konzipiert. Der Kontext gibt den Zusammenhang vor, in dem Komponenten zueinander stehen. So kann anhand des Kontextes beispielsweise entschieden werden, welche Präsentationsattribute zur Darstellung einer Komponente zu benutzen sind.

Des weiteren geht Dexter nicht auf zeitliche Beziehungen zwischen Komponenten ein und bietet auch keine Semantik zur Definition von fortgeschrittenen Präsentationsattributen. Zeitliche Abläufe spielen in Hypermedia–Systemen aber eine wesentlich Rolle, so daß die Frage nach der Definition von Linkmechanismen hierfür, für Hypermedia–System im Dexter Modell ungelöst bleibt.

Auf diese Defizite geht das Amsterdam Hypermedia Modell in Abschnitt 2.3.2 ein, welches sich selbst als Erweiterung des Dexter Modells versteht ([HBR94] Seite 50).

2.3.2 Das Amsterdam Hypermedia Modell

Das Amsterdam Hypermedia Modell(AHM)[HBR94] entstand auf Grundlage des Dexter Hypertext Referenz Modells und des CMIF (CWI Multimedia Interchange Format) Multimedia Modells [BRL91]. Außerdem werden Erweiterungen vorgenommen, um den Anforderungen von Hypermedia, wie beispielsweise zeitlichen Abläufen, gerecht zu werden. Der Begriff Hypermedia wird hier als eine Vermischung der Konzepte von Hypertext und von Multimedia verstanden. Das AHM begreift sich als ein generelles Hypermedia Modell, das sowohl der herkömmliche Link–basierten Navigation durch Hypertextelemente, als auch den komplexen Zeitabläufen und Darstellungsformen in klassischen Multimedia–Anwendungen Rechnung trägt.

Für diese Arbeit sind jedoch weniger die Aussagen zu zeitlichen Abläufen interessant als vielmehr die Erkenntnis der Notwendigkeit eines sogenannten Link Kontextes (*link context*). Wie zuvor bemerkt, sieht das Dexter Modell zwar zusammengesetzte Komponenten vor, bietet aber keinen Mechanismus, diese in einen Zusammenhang (*Kontext*) zueinander zu setzen. Auch die Ausführung eines Links erfolgt ohne Beachtung des Kontextes. In der Regel wird beim Traversieren eines Verweises entweder der aktuell angezeigte Inhalt durch das Ziel des Verweises ersetzt oder dieser in einem neuem Fenster angezeigt. Dieses ist jedoch in vielerlei Hinsicht unbefriedigend, da zum einem in Hypermedia Anwendungen ein flexibleres Verhalten erwünscht ist, zum anderen der Kontext vollkommen außer Acht gelassen wird. Die Autoren des AHM schreiben hierzu:

> *»Composite components do not, however, provide information on how each component behaves when a link is followed out of that component or within a composite component. In order to describe this type of behavior, the AHM defines the notion of a* ***link context****. A context is a (typically composite) component that contains a collection of composite or atomic components affected by a linking operation.«*([HBR94] Seite 60)

Des weiteren werden die Begriffe des Quellkontextes (*Source Context*) und des Zielkontextes (*Destination Context*) wie folgt definiert:

Definition 2.3.1 (Quellkontext)
Der Quellkontext eines Verweises ist der Teil einer Hypermedia Präsentation, der von der Initiierung eines Verweises betroffen ist (siehe [HBR94], Seite 60).

Definition 2.3.2 (Zielkontext)
Der Zielkontext eines Verweises ist der Teil der Präsentation, der beim Erreichen des Verweisziels angezeigt wird (siehe [HBR94], Seite 60).

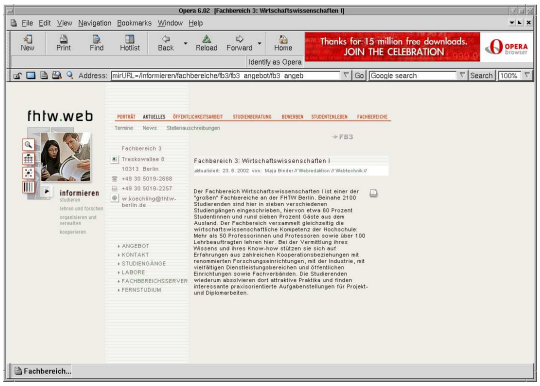

Abbildung 2.2: Fachbereichseite im Kontext „Informieren“

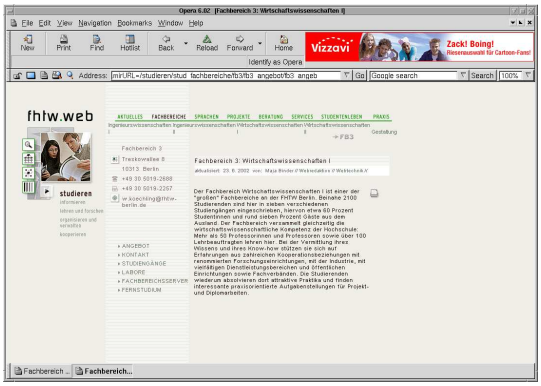

Abbildung 2.3: Fachbereichseite im Kontext „Studieren“

Der Link Kontext des AHM erlaubt die Auswertung von Parametern zur Anzeige in Abhängigkeit von Quell- bzw. Zielkontext, wie beispielsweise das Ersetzen einer Komponente durch das Ziel eines Verweises und wirkt sich somit direkt auf die Verweistraversierung („ausführen einer URI“) aus. Hiervon ist der später in dieser Arbeit auftretende und definierte Begriff des → Link Context bzw. Verweiskontext (siehe Abschnitt 3.3) klar abzugrenzen. Dieser bezeichnet im Gegensatz zum AHM Link Kontext den Kontext der Verknüpfung, also eines Verweises selbst und nicht Kontext in dem ein Objekt angezeigt wird. Da der Link Kontext Bestandteil der zusammengesetzten Komponenten ist, werden von einer Link Verfolgung nur Komponenten auf niederen Hierarchieebenen betroffen.

Um den Unterschied zwischen dem AHM Link Kontext und dem Verweiskontext besser zu verdeutlichen, sollen die folgenden beiden Bespiele betrachtet werden:

Beispiel 2.3.1

AHM Link Kontext: *Anhand des Quell– bzw. des Ziel–Kontextes wird entschieden, welche Komponenten einer Multimediapräsentation bei einer Verweisverfolgung auszutauschen sind. Es wird beispielsweise eine zusammengesetzte Komponente bestehend aus einer Audioinformation und ein Text dargestellt. Initiert der Anwender die Traversierung eines Verweises im Text, so ist es denkbar, daß nur die textuelle Information ausgetauscht wird, die Audioinformation aber bestehen bleibt.*

Verweiskontext: *Die Präsentation der Komponenten wird in Abhängigkeit ihres Aufrufkontextes vorgenommen. In den Abbildungen 2.2 und 2.3 wird dieselbe Komponente in unterschiedlichen Kontexten, „Studieren“ und „Informieren“, dargestellt. Die Unterschiede in der Präsentation liegen zum einen in der unterschiedlichen Farbgebung als auch in der Auswahl der angebotenen Menüpunkte.*

2.4 Technologien des World Wide Webs

Das World–Wide–Web (*WWW*) ist ein auf Basis des Internets angebotener Dienst, der die assoziative Verknüpfung von sogenannten Knoten durch Verweise ermöglicht. Ein Knoten kann Daten, wie beispielsweise Texte oder Grafiken, aufnehmen. Das WWW stellt heutzutage neben dem Versand von eMails vermutlich den meistgenutzten Dienst im Internet dar. Aus diesem Grund werden umgangssprachlich die Begriffe WWW und Internet häufig fälschlicherweise synonym benutzt. Das Internet an sich umfaßt aber weit mehr als nur das WWW und eMails, so gehört beispielsweise auch das USENET (Newsgroups) zur Gruppe der auf Basis der Infrastruktur des Internets realisierten Dienste. Der Grundgedanke des WWWs ist die assoziative Verknüpfung von Informationen, die den Versuch darstellt, die Informationsverknüpfung unseres Gehirns auf Ebene der Informationspräsentation nachzuempfinden.

2.4.1 Entstehungsgeschichte

Die Entstehungsgeschichte des World–Wide–Webs beginnt bereits in den 80ziger Jahren am internationalen Hochenergieforschungszentrum CERN in der Schweiz. Dort schrieb und benutzte Tim Berners–Lee ein kleines Hypertext Programm namens *Enquire*. Enquire erlaubte es, Daten, die in sogenannten Konten (*Nodes*) gespeichert wurden, untereinander durch Verweise zu verknüpfen. Befanden sich zwei Knoten innerhalb derselben Datei, stellte das System automatisch eine bi–direktionale Verknüpfung her.

1988 entschloß sich Tim Berners–Lee Enquire zu einem netzwerktransparenten System weiterzuentwickeln und reichte im darauffolgenden Jahr einen entsprechenden Projektantrag am CERN ein. Sein Kollege Ben Segal hatte ihn zudem überzeugt, die Möglichkeiten des Internets für sein Projekt zu verwenden. Im Jahr 1990 bekam das World Wide Web seinen heutigen Namen und Tim Berners–Lee entwickelte die erste Version seines auf drei Säulen aufbauenden Konzeptes, das immer noch Grundlage des WWWs ist:

1. *HTTP* (Hypertext Transfer Protocol): als Spezifikation zur Kommunikation zwischen Client und Server,
2. *URI* (Universal Resource Identifier): zur Adressierung von *Ressourcen*,
3. *HTML* (Hypertext Markup Language): als Spezifikation einer Auszeichnungssprache zum Erstellen von entsprechenden Dokumenten.

Weihnachten 1990 war dann die erste WWW-Seite im Internet verfügbar (siehe [5]).

2.4.2 Hypertext Markup Language (HTML)

Das Akronym HTML [6] steht für Hypertext Markup Language und ist die „lingua franca“ im World Wide Web. HTML basiert auf SGML (Structured Generalized Markup Language) [sgm86] und benutzt somit ebenfalls Tags, um die Struktur eines Dokumentes festzulegen. Im Gegensatz zu SGML legt HTML darüber hinaus nicht nur die Struktur, sondern auch das Aussehen eines Dokumentes fest.

HTML erlaubt die Verknüpfung von Ressourcen durch Verweise. Ressourcen können zum einen komplette Dokumente, aber auch nur bestimmte Teile von Dokumenten, beispielsweise ein Wort eines Textes oder ein polygonaler Ausschnitt eines Bildes, sein. Subadressierungen werden immer über Anker vorgenommen, die entweder unter Verwendung des `<a>` Tags oder durch das zusätzliche Tag–Attribut `id` bei Standard HTML Tags, gekennzeichnet werden. Die Definition eines Verweises erfolgt ebenfalls unter Verwendung des `<a>` Tags, wobei zur Unterscheidung von Ankern zusätzlich das Tag–Attribut `href` mit der URI des Verweisziels angegeben wird. HTML Verweise sind grundsätzlich uni–direktional und bestehen immer aus genau einer Quelle und einem Ziel. Anker und Verweise werden nicht separat, sondern im eigentlichen Dokumenteninhalt gespeichert. Daraus resultieren zwei Einschränkungen hinsichtlich der Verweiserstellung. Zum einen ist es nicht möglich, Subadressierungen als Verweisziel anzugeben, wenn die dafür benötigten Anker noch nicht im Zieldokument vorhanden sind. Zum anderen können keine Verweisquellen in Dokumente eingefügt werden, auf die man keine Schreibrechte besitzt. Ebenfalls nicht vom Verweismodell unterstützt wird eine kontext-sensitive Auswahl von Verweisen, so daß beispielsweise nur Verweise angezeigt werden, die mit Fußnoten oder Navigationselementen verknüpft sind.

Hinsichtlich der in der Einleitung aufgeführten Forderung nach wiederverwendbaren Inhaltszellen, weist HTML einige Defizite auf. So können zwar externe Komponenten, wie Bilder oder Animationen problemlos wiederverwendet werden, will man aber Texte wiederverwenden, so müssen diese seitenweise gespeichert werden. Seitenweise Speicherung bedeutet, daß immer eine vollständige HTML Seite unter Verwendung der Tags `HEAD` und `BODY` deklariert werden muß. Betrachtet man eine solche Seite als eine Inhaltszelle, so muß diese dann in *Frames* eingebettet werden. Will der Autor beispielsweise einen Paragraphen eines Textes in einer anderen Seite wiederverwenden, so muß zum einen diese Seite aus dem entsprechendem *Frameset* bestehen, zum anderen muß dieser Paragraph in einer vollständig spezifizierten Seite gespeichert werden[2].

Aufgrund der erst spät einsetzenden Standardisierungsbestrebungen existieren eine Vielzahl von untereinander abweichenden und insbesondere nicht standardkonformen Implementierungen von HTML. Mit *XHTML* (Extensible Hypertext Markup Language) [7] wird seitens des

[2]Diese Betrachtung geht davon aus, daß diese Funktionalität mit reinen HTML Mitteln ohne Verwendung von Server Side Includes oder Scriptsprachen mit Datenbankunterstüzung realisert wird.

W3C (World Wide Web Consortium) der Versuch unternommen, eine standardisierte, moderne Variante von HTML zu etablieren. Im allgemeinen wird HTML 4.0 als XHTML konform angesehen [7]. XHTML ist eine XML konforme Auszeichnungssprache und versucht somit HTML in die neu entstandene XML Welt zu übernehmen.

2.4.3 Extensible Markup Language (XML)

XML (Extensible Markup Language) [8] ist eine Empfehlung des World Wide Web Consurtium (W3C) zur Erzeugung von Auszeichnungssprachen beliebigen Umfangs für nahezu beliebige Zweck (vergleiche [Mic99]).

Wie auch HTML hat XML seine Ursprünge in SGML. Im Gegensatz zu HTML ist XML aber eine echte Teilmenge von SGML. XML wurde insbesondere in Hinblick auf die Verwendung als Auszeichnungssprache für Inter–/Intranet Anwendungen konzipiert und stellt sozusagen einen Kompromiß zwischen der Einfachheit von HTML und der Funktionsvielfalt von SGML dar. Gegen die Verwendung von HTML sprachen dessen zunehmend stärkere proprietäre Entwicklung und die konzeptbedingte, eingeschränkte Erweiterbarkeit. Die Nutzung von SGML als technologische Alternative wäre aufgrund der Komplexität und Funktionsvielfalt mit unaktzeptablen Aufwänden für Implementierung und Betrieb verbunden (siehe [Mic99], Seite 20f).

Zielstellung war, eine leicht verwendbare Sprache zu erschaffen und die Interoperabilität mit SGML [sgm86] und HTML [6] zu erhalten. Ein kurzer Vergleich der Fähigkeiten und Konzepte von XML und HTML kann der Tabelle 2.1 auf Seite 15 entnommen werden. Ein solcher Vergleich wird zwar oft verwendet, beispielsweise in „XML Kompakt“ [Mic99], geht aber immer mit Problemen einher, da dabei zwei unterschiedliche Konzepte miteinander verglichen werden.

Auf Basis des Metakonzeptes von XML existieren bereits eine Reihe an unterschiedlichen XML konformen Sprachen, wie beispielsweise XHTML oder *XSL* (Extensible Stylesheet Language). Eine XML konforme Sprache muß zumindest dem Kriterium der *Wohlgeformtheit* folgen. Die XML Spezifikation [8] legt hierzu fest:

Definition 2.4.1 (Wohlgeformtheit)
Ein textuelles Objekt ist ein wohlgeformetes XML Dokument, wenn:

1. es als Gesamtheit betrachtet zu den spezifizierten Produktionsregeln paßt. Dies ist immer der Fall, wenn es zumindest ein Element – das sogenannte Wurzelelement – enthält, welches in keinem anderen Element enthalten ist und alle folgenden Elemente, begrenzt durch ein Start– und ein End–Tag, korrekt ineinander verschachtelt sind.
2. es alle Wohlgeformtheitsbeschränkungen der W3C XML Spezifikation [8] erfüllt.
3. jede seiner Parsed Entities, die direkt oder indirekt referenziert werden, wohlgeformt sind.

Darüber hinaus ist ein XML Dokument *gültig*, wenn es weiteren Einschränkungen genügt.

Definition 2.4.2 (Gültigkeit)
Ein XML Dokument ist gültig, wenn eine zugehörige Dokumenttyp-Deklaration (DTD) existiert und sich das Dokument an die darin formulierten Beschränkungen hält.

Gültigkeit wird von der XML Spezifikation nicht explizit gefordert.

Ein Verweismodell ist, wie in Tabelle 2.1 auf Seite 15 aufgeführt, nicht direkter Bestandteil der XML Spezifikation. Vielmehr existieren hierzu weitere vom W3C standardisierte und veröffentlichte Spezifikationen, nämlich *XPath*, *XLink* und *XPointer*, auf die in den folgenden Abschnitten gesondert eingegangen wird.

Kriterium	HTML	XML
Versionsvielfalt	hoch, durch erst spät einsetzende Standardisierung. Häufig bestehen Inkompatibilitäten zwischen verschiedenen Sprachversionen und Browsern.	Es existiert nur eine offizielle, vom W3C standardisierte Version.
Sprachumfang	Der Sprachumfang von HTML ist fest definiert und nicht erweiterbar. Da HTML keine Modellierungssprache ist, lassen sich Datenstrukturen unter Verwendung von HTML nur schwer modellieren.	XML ist auf Grund seines Metakonzeptes frei erweiterbar und eignet sich somit sehr gut zur Datenmodellierung. Zudem können Tags mit signifikanten Namen versehen werden.
Struktur & Layout	Inhalt (Struktur) und Layout sind nicht ausreichend voneinander getrennt. Die Tag–Namen treffen im allgemeinen keine Aussage über den Dateninhalt. HTML verfolgt als primäres Ziel die Präsentation von Dokumenten.	Struktur und Layout sind in XML klar getrennt, ein bestimmtes Tag trifft in der Regel keine Aussage über seine Formatierung, sondern über seinen Dateninhalt. Formatierungen werden über XSL vorgenommen.
Verweiskonzept	Fest in die Sprache eingebettet und ist zudem ausdrucksarm. (siehe hierzu Abschnitt 2.4.2).	XML besitzt kein fest eingebettes Verweismodell. Im Standardisierungsprozess befinden sich aber momentan *XLink* und *XPointer*, die weitreichende Verweisfähigkeiten mit sich bringen (siehe dazu Abschnitt 2.4.3.2 und 2.4.3.3).

Tabelle 2.1: Vergleich der Spracheigenschaften HTML und XML

2.4.3.1 XPath

XPath [9] bietet die Möglichkeit, Teile von XML Dokumenten zu adressieren. Verwendung findet diese Fähigkeit unter anderem in der Transformationssprache XSL und zur Adressierung von Daten in XML Dokumenten mittels XPointer.

XPath operiert auf einer Baumstruktur, die eine abstrakte Repräsentation eines XML Dokumentes darstellt. Einzelne Knoten des Baumes werden, ähnlich der Pfadbeschreibung in URLs, ausgehend vom Wurzelelement durch die mit einem Slash ('/') getrennte Aufzählung der Elternknoten adressiert. Neben der Adressierung von Elementen stellt XPath Funktionen zur sogenannten Mustererkennung (*Pattern Matching*) für Knoten und eine Reihe weiterer Funktionen bereit. Eine ausführliche Auflistung aller Funktionen ist unter [9] nachzulesen.

Grundlage der Element–Adressierung sind sogenannte Lokalisierungspfade, die aus mindestens einem Lokalisierungsschritt bestehen. Ein Lokalisierungsschritt wiederum besteht aus der Angabe einer Achse und einem Knotentest. Dieser kann optional um beliebig viele Prädikate ergänzt werden.

Eine Achse definiert das (genealogische) Verhältnis zwischen dem (oder den) Knoten und dem Schritt. Hierfür stehen eine Reihe von Schlüsselwörtern, wie beispielsweise `child`, das den direkten Nachfahren selektiert, zur Verfügung. Eine Auflistung aller Schlüsselwörter ist [9] zu entnehmen. Die Angabe einer Achse und damit die Auswertung dieses Ausdrucks erfolgt immer relativ vom *Kontext–Knoten* aus. Dieser kann entweder absolut oder relativ, beispielsweise durch einen vorangegangen Lokalisierungschritt, adressiert werden.

Knotentests filtern Knoten, die entlang der Achse gefunden werden, entsprechend ihrem Namen oder ihrem Typ, heraus. Durch die Angabe von Prädikaten kann die Ergebnismenge zusätzlich eingeschränkt werden, indem sie Eigenschaften von Knoten abfragen (vergleiche [WM02] Seite 65).

Beispiel 2.4.1
Der folgende XPath–Ausdruck würde ausgehend vom hier nicht näher spezifizierten Kontext–Knoten das letzte `paragraph`*–Element wählen:*

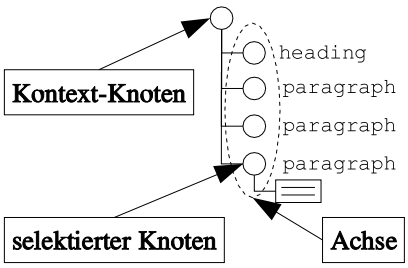

Abbildung 2.4: Baumdarstellung zum XPath Beispiel

$$\underbrace{\texttt{child}}_{\text{Achse}} :: \underbrace{\texttt{paragraph}}_{\text{Knotentest}} \underbrace{[\texttt{position}() = \texttt{last}()]}_{\text{Prädikat}}$$

Die kleinste über einen XPath-Ausdruck adressierbare Entität ist ein Element. Eine Markierung von Daten innerhalb eines Elementes kann nicht vorgenommen werden. Außerdem bietet XPath noch keine Möglichkeit, unterschiedliche Dokumente miteinander zu verknüpfen, um ein Linking zu realisieren. Diese Problematik wird von zwei weiteren Standards, XPointer und XLink, aufgegriffen und dort separat behandelt. Dabei definiert XPointer einen Mechanismus zur Subadressierung innerhalb von Elementen unter Verwendung von XPath. XLink schlägt ein Modell zur Erzeugung und Verwendung von Hyperreferenzen in XML Dokumenten vor. Der folgende Abschnitt stellt diese beiden Standards vor und zeigt die konzeptionellen Unterschiede und Neuerungen gegenüber HTML auf.

2.4.3.2 XLink

Die XLink Spezifikation[10, 11] erlaubt es, Elemente, die Links zwischen → Ressourcen ermöglichen und beschreiben, in XML Dokumente einzufügen. Hierzu wird ein XML konformer Syntax benutzt, der den herkömmlichen Link Mechanismus von HTML abbilden kann und diesen sowohl konzeptionell als auch in seinem Funktionsumfang, erweitert.

Zum Verständnis von XLink sind jedoch zuerst einige Begriffsbestimmungen nötig (vergleiche [10, 11]):

Link: Ein Link ist eine explizite Beziehung zwischen →Ressourcen oder Teilen von Ressourcen.

Link–Element: Ein Link–Element ist ein XLink konformes XML Element, das die Existenz eines Links ausdrückt.

Ressource (*Resource*): Eine Ressource ist eine adressierbare Informations- oder Diensteinheit.

Hyperlink: Ein Hyperlink ist ein Link, der hauptsächlich für die Präsentation für einen menschlichen Benutzer gedacht ist.

Traversieren (*Traversal*): Traversieren heißt das Benutzen oder Verfolgen eines Linkes zu einem beliebigen Zweck.

Start–Ressource (*Starting Resource*): Die Start–Ressource ist die Quelle eines Links von dem aus die Traversierung startet.

End–Ressource (*Ending Resource*): Die End–Ressource ist das Ziel eines Links, das nach der Traversierung des Links erreicht wird.

Kante (*Arc*): Eine Kante gibt Informationen an, wie ein Paar von Ressourcen zu traversieren ist, inklusive der Traversierungsrichtung und eines möglichen Anwendungsverhaltens.

Lokale Ressource (*Local Resource*): Eine lokale Ressource ist ein XML Element, das dadurch an einem Link teilnimmt, daß entweder es selbst oder sein Vater ein Link–Element ist.

Entfernte Ressource (*Remote Resource*): Eine Ressource oder ein Teil einer Ressource, die an einem Link dadurch teilnimmt, daß sie durch eine URI Referenz adressiert wird, nennt man entfernte Ressource.

Ausgehend (*Outbound*): Eine Kante, die eine lokale Start- und eine entfernte End–Ressource hat, heißt ausgehend.

Eingehend (*Inbound*): Wenn die End–Ressource einer Lante lokal ist, aber die Start–Ressource entfernt, dann heißt diese Kante eingehend.

Third–Party: Wenn weder Start- noch die End–Ressource lokal sind, dann wird die Kante bzw. der Link als Third–Party Kante respektiv Third–Party Link bezeichnet.

Link–Bank (*Linkbases*): Dokumente, die Sammlungen von eingehenden Links und Third–Party Links enthalten, werden Link–Datenbanken oder kurz Link-Banken genannt.

Schon aus der Reichhaltigkeit der Begriffsdefinitionen läßt sich erahnen, daß der XLink–Standard komplexe Linkdefinitionen erlaubt. Grundsätzlich unterscheidet XLink zwei Arten von Links, sogenannte einfache Links (*Simple Links*) und erweiterte Links (*Extended Links*).

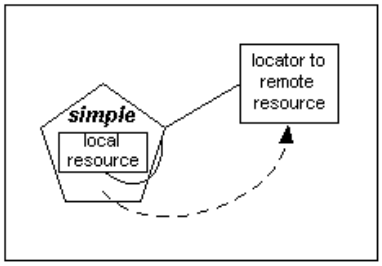

Abbildung 2.5: Einfacher Link
(Vergleiche [10])

Einfache Links

Einfache Links sind die XLink–Analogie zu den klassischen HTML Hyperlinks. Sie sind uni–direktional, bestehen genau aus einer lokalen und einer entfernten Ressource, verbunden durch eine vom der lokalen zur entfernten Ressource gerichteten Kante. Der Link wird wie bei HTML auch im Inhalt der beteiligten Ressource angegeben. Abbildung 2.5 zeigt eine schematische Darstellung eines einfachen Links. Jeder einfache Link kann in einen erweiterten Link überführt werden.

Erweiterte Links

Erweiterte Links bieten die volle in XLink definierte Funktionalität. Hierzu gehören die Fähigkeiten, multi–direktionale, mit mehreren beteiligten Ziel– bzw. Startressourcen verknüpfte eingehende, ausgehende oder Third–Party Links zu bilden.

Interessant ist zudem die Möglichkeit, Ressourcen zu verlinken, auf die man keine Schreibzugriffe hat, also Links zu definieren, ohne den Inhalt der beteiligten Ressourcen zu verändern. Anwendung könnte dieses in Linkverzeichnissen oder bei der Indexerstellung finden. Die Funktionalität für solche, auch häufig als *out–of–line* Links bezeichneten Verweise zwischen Ressourcen werden durch die Verknüpfung von *Locator*–Elementen durch Kanten ermöglicht. Ein *Locator*–Element gibt die Lokalisierung einer Ressource an. Ein Locator spezifiziert zumindest einen eindeutigen Identifikator (ID) und die Lokalität der Ressource in URI Notation (HREF). Zusätzlich kann jeder Locator mit einem Namen (TITLE) und einer Rolle (ROLE) versehen werden.

Der tatsächliche Link wird durch die Verknüpfung von Locator Elementen mittles einer Kante realisiert. Vorgeschriebene Attribute zur Definition umfassen zumindest die Spezifikation der Start–Ressource (FROM) und der End–Ressource (TO). Optional kann noch das Verhalten zur Anzeige über das SHOW Attribut und das Verhalten beim Traversieren der Kante angegeben

werden (ACTUATE). Gültige Werte für die Anzeigeoption sind NEW, um die Anzeige in einem neuem Fenster zu erzwingen, REPLACE, um den aktuellen Inhalt des Fenster mit dem der End–Ressource auszutauschen und EMBED, um nur den Teil der Startressource zu ersetzen. Des weiteren stehen noch OTHER und NONE zur Verfügung. Gültige Werte des ACTUATE Attributes sind ONLOAD, ONREQUEST, OTHER und NONE. Die Angabe von ONREQUEST veranlaßt, daß die End–Ressource erst bei der expliziten Aktivierung des Links durch Nutzerinteraktion geladen wird. Im Gegensatz dazu wird der Link bei ONLOAD beim Laden der Start–Ressource traversiert. Die Attribute OTHER und NONE erlauben ein nicht von dieser Spezifikation angegebenes Verhalten zu erzwingen, mit dem Unterschied, daß bei OTHER die Anwendung eine weitere Spezifikation des Verhaltens innerhalb des Link Elementes erwartet.

2.4.3.3 XPointer

Wie oben bereits ausgeführt, dient XPath zur Adressierung von Elementen in XML konformen Dokumenten. XLink nutzt diese Fähigkeit zum Verweis auf Teile von XML Dokumenten. Für ein fortgeschrittenes Verweismodell ist aber eine Einschränkung auf Elemente als kleinste adressierbare Entität unter Umständen unzureichend. Die Subadressierung innerhalb von Elementen wird unter Verwendung des XPointer–Standards [12] des W3C realisiert. XPointer erlaubt die Analyse hierarchischer Dokumentenstrukturen und die Selektion interner Daten auf Subelement–Ebene und darf daher nur auf Daten mit dem MIME–Type `text/xml`, `application/xml`,`text/xml-external-parsed-entity` oder `application/xml-external-parsed-entity` angewendet werden.

Die XPointer Spezifikation darf nicht losgelöst von XPath betrachtet werden, da sie auf den Konzepten und Funktionen von XPath aufbaut und diese erweitert. XPointer definiert die Dokumentenwurzel, d.h. das Wurzelelement eines XML Dokumentes als Ausgangskontext eines jeden XPath– bzw. XPointer–Ausdrucks.

XPath kennt nur Knoten (*Nodes*) bzw. Knotenmengen als Ergebnismenge. XPointer erweitert diese Ergebnismenge um Punkte (*Points*) und Bereiche (*Ranges*). Ein Punkt ist eine durch einen Index ausgezeichnete Stelle in einem diesen Punkt umgebenenen Knoten (*Container Node*). Ein Bereich ist ein durch einen Start- und Endpunkt begrenzter Abschnitt eines XML Dokumentes. XPointer faßt nun diese drei möglichen Ergebnisse (Konten, Punkte und Bereiche) in einem abstrakten Typ der sogenannten *Location* zusammen. Weiterhin werden die Funktionen von XPath durch XPointer um solche erweitert, die Locations als Ergebnis zurückgeben.

Ein XPointer–Ausdruck besteht aus mindestens einem relativen Term, der auf einem absoluten Term operiert. Die Bedeutung des absoluten Terms ist vergleichbar mit der des Kontext–Knotens in der XPath Spezifikation. Mehrere relative Ausdrücke werden mit eine Slash ('/') getrennt und der Reihe nach im Bezug zum Ergebnis des vorangegangen Ausdrucks ausgewertet.

Beispiel 2.4.2

Der folgende XPointer-Ausdruck würde im XML Dokument `beispiel.xml` *das dritte Vorkommen der Zeichenkette* `'und'`*, vom Wurzelelement aus betrachtet, auswählen. Hierzu wird die Funktion* `string-range` *verwendet, die als Parameter eine Menge vom Locations, den gesuchte Zeichenkette und optional die Startposition und die Länge der zu liefernden Zeichenkette, entgegennimmt.*

$$\underbrace{\texttt{beispiel.xml}}_{\text{absoulter Term}}\ \#xpointer(\underbrace{\texttt{string}-\texttt{range}(/,'\texttt{und}')}_{\text{relativer Term}})\ \underbrace{[3]}_{\text{Prädikat}}$$

2.4.3.4 Die XML Konzepte und Lernsystem-Anforderungen

Die XML–Technologien erlauben es, Inhalte wesentlich granularer zu speichern und auch zu selektieren, als das mit HTML der Fall ist. Zum anderen unterstützen die XML–Konzepte eine klare Strukturierung der Inhalte und die Trennung dieser von optischen Gesichtspunkten (Trennung von Inhalt und Layout). Somit wird dem Autor die Erstellung von wiederverwendbaren Materialien ermöglicht, da diese zum einen durch die Trennung von Layout und Inhalt nicht an eine bestimmte Präsentationsform gebunden sind, zum anderen eine Wiederverwendbarkeit durch granulare Speicherung von Inhalten gefördert wird.

Auch das Verweismodell von XML, bestehend aus XLink, XPath und XPointer, nähert sich den Bedürfnissen von guten Online-Lernsystemen an. So werden Verweise zwischen Ressourcen, auf die man keinen Schreibzugriff hat, unterstützt. Durch die out–of–line und Third–Party Links stehen zudem leistungsfähige Konstrukte zur Erzeugung von Verweissammlungen und Indices zur Verfügung. Zudem bieten die Präsentationsattribute in XLink dem Autor weitere Gestaltungsmöglichkeiten.

Obwohl XLink ein wesentlich flexibleres und ausgestaltbareres Verweismodell bietet, fehlt genau wie bei HTML die Möglichkeit, semantische Beziehungen zwischen Dokumenten auszudrücken. Dieser Forderung nach dem „semantischen Web“ tragen die in dem folgendem Abschnitt beschriebenen Technologien Rechnung.

2.4.4 Neue Technologien - Semantisches Web

Mit der wachsenden Datenmenge im Internet steigt gleichsam nicht nur das Bedürfnis nach leistungsfähigen Suchmachinen, sondern darüber hinaus die Forderung semantische Zusammenhänge (hierzu zählen auch Ontologien als Ausgestaltungsmittel) abbilden und automatisiert verarbeiten zu können. Im allgemeinen wird hierbei von der „dritten Generation“ der Hyperreferenzsysteme, dem sogenannten *semantischen Web*, gesprochen.

Ursprünglich wurde das World–Wide-Web zur Nutzung durch den Menschen konzipiert. Alle Informationen liegen zwar in einen machinenlesbaren Format vor, werden von diesen in der Regel aber nicht „verstanden“. Eine mögliche Lösung hierfür ist die Verwendung von *Metadaten*, um die vorhandenen Daten zu beschreiben (vergleiche [13]). Metadaten könnten zum einen nähere Angaben, beispielsweise den Autor oder das Datum der letzten Modifikation, aber auch das semantische oder ontologische Verhältnis zwischen verschiedenen Daten angeben.

Die in den folgenden Abschnitten beschrieben Technologien beschreiben zwei „konkurrierende“ Ansätze, sowohl zur Beschreibung von Daten, als auch zur Modellierung von Beziehungen und Zusammenhängen zwischen diesen. Da das Ziel dieser Arbeit jedoch nicht der Vergleich von Technologien zur Annotation von Webseiten ist, wird auf eine gegenüberstellende Betrachtung verzichtet und statt dessen lediglich kurz auf die Kernideen eingegangen.

2.4.4.1 Resource Description Framework (RDF)

Das RDF (Resource Description Framework) [13, 14] Modell ist eine Empfehlung des W3C zur (maschinen–interpretierbaren) Beschreibung von Ressourcen und anderen Informationen wie beispielsweise Beziehungen zwischen Ressourcen. Das Modell stützt sich hierbei auf drei Säulen, das Datenmodell (*RDF Model*), eine Syntaxbeschreibung (*RDF Syntax*) und ein Schema (*RDF Schema*) zur Definition eines anwendungsspezifischen Vokabulars.

RDF Datenmodell

Das RDF Datenmodell ist eine syntax–unabhängige Darstellungsform von RDF Ausdrücken. Zwei syntaktisch unterschiedliche Ausdrücke sind gleichwertig, wenn sie in ihrer Darstellung im RDF Datenmodell übereinstimmen. Das Datenmodell definiert drei grundlegende Objekttypen (vergleiche [13]):

Ressourcen (*Resources*): Ressourcen sind alle Entitäten, die durch RDF Ausdrücke beschrieben werden. Der Begriff Ressource wird wie in Abschnitt 2.4.3.2 auf Seite 18 verwendet, kann also sowohl ganze Webseiten, Teile von Webseiten, als auch eine ganze Kollektion von Webseiten beschreiben. Zusätzlich werden im Zusammenhang mit RDF auch

Daten als Ressourcen bezeichnet, die nicht direkt online abrufbar sind, wie beispielsweise gedruckte Bücher. Ressourcen werden anhand eines eindeutigen *Resource Identifier* identifiziert. Ein solcher Identifier besteht zumindest aus einer URI plus einer optionalen Angabe, der Anker ID.

Eigenschaften (*Properties*): Eigenschaften beschreiben einen bestimmten Aspekt, ein Charakteristikum, ein Attribut oder eine Relation von Ressourcen. Jede dieser Eigenschaften hat eine bestimmte Aufgabe und definiert erlaubte Werte, um diese Ressource oder einen Zusammenhang mit anderen Ressourcen zu erklären. RDF Schemata legen hierzu fest, wie die bestimmten Eigenschaften syntaktisch ausgedrückt werden. Beispielsweise kann hier auf bestehende Verfahren zur Annotation wie das Dublin Core Set [15] (siehe Abschnitt 2.5.2) oder die Learning Object Metadata Spezifikation [16] (siehe Abschnitt 2.5.1) zurückgegriffen werden.

Aussagen (*Statements*): Eine Eigenschaft und deren Wert werden als Aussage bezeichnet. Jede Aussage besteht analog zu einem Satz aus drei Komponenten, einem Subjekt, das die zu beschreibende Ressource angibt, einem Prädikat, das die zu beschreibende Eigenschaft enthält und einem Objekt oder Literal, nämlich dem Wert dieser Eigenschaft. Das Objekt einer Aussage kann zum einem ein einfaches Zeichenkettenliteral oder ein vergleichbarer primtiver XML Datentyp, andereseits aber auch eine Ressource sein.

Eine mögliche Darstellungsform des RDF Datenmodells ist ein gerichteter Graph, dessen Knoten Ressourcen und Literale sind. Diese werden verknüpft durch Kanten, die die Eigenschaften (Prädikate) zwischen einem Subjekt und Objekt ausdrücken.

Beispiel 2.4.3
Angenommen man möchte folgende Aussage modellieren: „Ora Lassila ist der Autor der Ressource http://www.w3.org/Home/Lassila.“ So ergibt sich:

Subjekt (Ressource)	*http://www.w3.org./Home/Lassila*
Prädikat (Eigenschaft)	*Autor*
Objekt (Literal)	*„Ora Lassila“*

Tabelle 2.2: Aufspaltung einer RDF Aussage in ihre Bestandteile

Für das gleichlautende RDF Datenmodell ergibt sich der folgende Graph (vergleiche [13]):

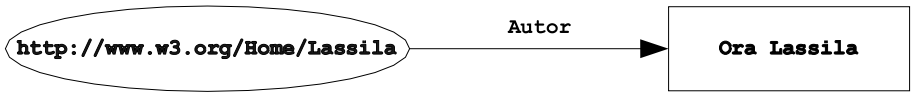

Abbildung 2.6: Darstellung als gerichteter Graph

Häufig ist es nötig, nicht nur auf einzelne Ressourcen, sondern auf eine ganze Sammlung von Ressourcen zu verweisen. RDF sieht hierfür drei unterschiedliche Aufzählungstypen vor:

Unsortierte Liste (*Bag*): Unsortierte Listen enthalten Ressourcen oder Literale, wobei die Reihenfolge unerheblich ist. Mehrfachnennungen von Ressourcen oder Literalen sind erlaubt. Bags werden benutzt, um eine Eigenschaft mehreren Objekten zuzuordnen, wobei deren Reihenfolge unerheblich ist (beispielsweise die Aufzählung aller Studenten in einem Kurs).

Sortierte Liste (*Sequence*): Eine sortierte Liste enthält Ressourcen oder Literale, deren Reihenfolge wichtig ist. Wie bei der unsortierten Liste sind Dublikate erlaubt. Die Verwendung erfolgt analog zur sortierten Liste (beispielsweise die Aufzählung von Spiegelservern, auf denen eine bestimmte Software erhältlich ist, in Abhängigkeit ihrer Bandbreite.)

Alternative List (*Alternativ*): Die alternative Liste enthält Literale oder Ressourcen, die Varianten eines Objektes sind (beispielsweise eine Liste von lokalisierten Versionen einer Index–Seite).

RDF Syntax

Die RDF Syntax Spezifikation definiert eine XML basierten Syntax zur Umsetzung des RDF Datenmodells. Eigentlich werden zwei Syntaxformen spezifiziert, die sogenannte serialisierte Syntax (*Serialization Syntax*) und die verkürzte Syntax (*Abbreviated Syntax*), die eine kompaktere Darstellung erlaubt. Standardkonforme RDF Interpreter sollten sowohl die serialisierten Syntax als auch die verkürzte Syntax verarbeiten können. Eine detaillierte Syntaxbeschreibung ist unter [13] zu finden.

Beispiel 2.4.4

Die Aussage, „Ora Lassila ist der Autor der Ressource http://www.w3.org/Home/Lassila.“, würde im serialisierten Syntax folgende Darstellung haben:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator xmlns:s="http://description.org/schema/">
      Ora Lassila
    </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Im Gegensatz dazu die Darstellung im verkürzten Syntax:

```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org/Home/Lassila"
  s:Creator="Ora␣Lassila" xmlns:s="http://description.org/schema"/>
</rdf:RDF>
```

RDF Schema

Die RDF Schema Spezifikation[17] schreibt vor, wie ein bestimmtes Vokabular zur Beschreibung von Ressourcen gebildet wird. Ein Vokabular ist hierbei eine Menge von wohldefinierten Eigenschaften. Somit ist es möglich, eine stark an den Nutzungsbedürfnissen und der Thematik orientierte Beschreibung von Ressourcen vorzunehmen. Ein Schema gibt hierbei nicht vor, was beschrieben wird, sondern definiert, wie etwas beschrieben wird, indem strukturelle Festlegungen und Einschränkungen vorgenommen werden.

Im Beispiel 2.4.4 wird schon implizit das Element `Creator` aus einem solchen Schema verwendet. Neben der Definition von eigenen Vokabularen besteht auch die Möglichkeit, bereits bewährte Standards, wie bespielsweise das Dublin Core Set, zu verwenden.

2.4.4.2 TopicMaps

Der Begriff *Topic Maps* bezeichnet eine Reihe von Konzepten und Ideen einer Arbeitsgruppe des ISO/IEC JTC1 SC34[3] [WM02], deren Ziel es ist, eine intelligente Informationssuche und Informationsverarbeitung zu ermöglichen und über ein semantisches Netzwerk auf Wissen zuzugreifen. Topic Maps dienen also dazu, Hypertext Dokumente semantisch miteinander zu verknüpfen.

Um verstehen zu können, was Topic Maps sind, müssen zuerst ein paar grundlegende Begriffe und Konzepte erläutert werden. Themen (*Topics*) sind die elementare Entität einer jeden Topic Map. Jedes Thema kann einen oder mehere Typen (*Topic Types*) haben, die selbst wieder als Themen auftreten. Durch eine Verkettung (Schachtelung) von Typen können Typhierarchien gebildet werden. Zur eindeutigen Identifizierung werden sogenannte *Public Subject Descriptor* an ein dem Thema zugehöriges Identitätsattribut (*Identity Attribute*) vergeben. Ein möglicher Public Subject Descriptor für ein Thema, das ein Buch beschreibt, wäre die ISB Nummer. Um Überschneidungen von Topics in unterschiedlichen Bereichen zu vermeiden, kann ein Gültigkeitsbereich (*Scope*) festgelegt werden.

Jedes Thema muß benannt werden. Hierfür werden drei unterschiedliche Varianten angeboten (vergleiche [WM02] Seite 8):

Base Name: Jedes Thema muß zumindest einen Base Name, seinen eigentlichen Namen besitzen.

Display Name: Ein Display Name kann optional, um die Zeichenfolge zur Darstellung anzugeben, vergeben werden. Wird kein Display Name gesetzt, so wird hierfür der Base Name verwendet.

[3] International Organization for Standardization / Internatial Electrotechnical Commission Joint Technical Committee 1, Subcommittee 34

Sort Name: Der Sort Name ist jener Name eines Themas, der zur Sortierung in beliebigen sortierten Listen herangezogen wird. Die Angabe des Sort Names ist ebenfalls optional; ist kein Sort Name vergeben, so wird der Base Name verwendet.

Jedes Thema kann beliebig viele Ausprägungen (*Occurrences*) besitzen. Ausprägungen können eine bestimmte Rolle (*Occurence Role*) einnehmen, die wiederum ein Topic ist. Ausprägungen sind Verweise zu externen Ressourcen.

Beziehungen zwischen Themen werden durch Assoziationen (*Associations*) beschrieben. Jede Assoziation kann höchstens einen Typ (*Association Typ*) und eine Assoziationsrolle (*Assoziation Role*) haben, die wiederum beide als Themen auftreten.

Themen und Assoziationen können beliebige Eigenschaften (Name–Wert-Paare), sogenannten Facetten (*Facets*), zugeordnet werden. Die Gesamtheit dieser Konstrukte wird als Topic Map bezeichnet.

Abschließend noch ein Beispiel:

Beispiel 2.4.5
Die folgende Graphik zeigt eine Topic Map, deren einzelne Themen als Kreise dargestellt werden. Eine Assoziation zwischen zwei Themen wird durch eine Raute mit dem Assoziationstyp angeben. Alternative Namen für ein Thema sind durch eine Umrahmung, Gültigkeitsbereiche durch ein gestricheltes Oval gekennzeichnet. Ausprägungen der einzelnen Themen werden mittels einer gestrichelten Line zur entsprechenden externen Quelle verbunden.

Dieses Beispiel verfügt nicht über Facetten, ein Beispiel für eine Facette wäre die Zuordnung einer Einwohnerzahl zum Topic Paris.

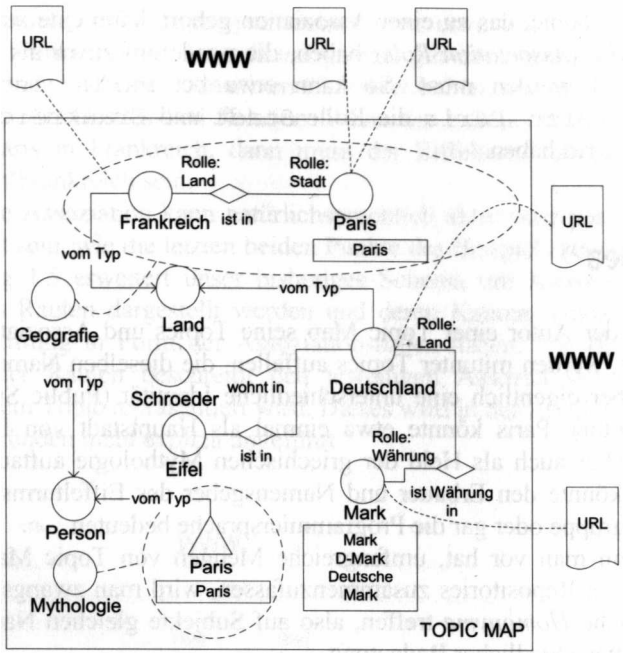

Abbildung 2.7: Graphische Darstellung einer Topic Map (vergleiche [WM02])

2.5 Annotationsverfahren

Verfahren wie RDF benötigen, um ihre volle Funktionalität entfalten zu können, leistungsfähige Schemata zur Beschreibung der Daten. Im Abschnitt 2.4.4.1 wurde schon auf die Bedeutung von Vokabularen zur anwendungsspezifischen Beschreibung von Ressourcen eingegangen. Die folgenden beiden Abschnitte stellen hierfür zwei Kandidaten unterschiedlichen Umfangs und unterschiedlicher Komplexität vor.

2.5.1 Learning Object Metadata (LOM)

LOM (Learning Object Metadata) [16] ist ein in der Entwicklung befinderlicher Standard des IEEE[4] zur Annotation von Objekten für Lehr- und Wissensysteme. Grundlegend für die Entwicklung waren unter anderem das Dublin Core Set[15], das ARIADNE Projekt[18] der Europäischen Union und das IMS Projekt[19], eine Initiative nordamerikanischer Institute und deren Partner in der Industrie [16].

[4] Institute of Electrical and Electronics Engineers

Objekte im Sinne des LOMs sind alle sowohl digitalen als auch nicht–digitalen Entitäten, die zum Lehren oder Lernen benutzt werden. Metadaten bezüglich dieser Objekte beschreiben wichtige charakteristische Eigenschaften. Diese können in neun grundsätzliche Kategorien zusammengefaßt werden:

1. *Generelle Informationen*, die das Objekt als ganzes beschreiben,
2. Informationen zum *Lebenszyklus* eines Objektes,
3. *Meta–Metadata* zur Beschreibung der Metadaten,
4. *Technische Informationen* über die Beschaffenheit und die Anforderungen des Objektes,
5. *Pädagogische Informationen*,
6. *Rechtliche Informationen* wie Urheberrechtsbestimmungen etc.,
7. *Verweise* und Zusammenhänge zwischen diesem und anderen Objekten,
8. freie *Annotation*sfelder
9. Einordnung in andere Standards oder *Klassifikationen*

Die Ziele, die mit diesem Standard verfolgt werden, sind die Akquisition, die Evaluation, das Suchen und Finden sowie die Benutzung und den Austausch von Lern- und Lehrobjekten zu erleichtern. Dieses wird erreicht, durch die Entwicklung von Katalogen und Inventaren, die die vielseitigen kulturellen und sprachlichen Kontexte, in denen solche Objekte verwendet werden, berücksichtigen (vergleiche [16]).

Das *Base Schema* des LOM ist relativ umfangreich. Die oben genannten Kategorien gruppieren Datenelemente, welche in einem hierarchisierten Baum dargestellt werden. Nur Blätter dieses Baumes tragen spezifische Daten. Eine detaillierte Auflistung der Kategorien und deren Elemente kann dem Standard [16] entnommen werden.

2.5.2 Dublin Core Metadata

Dublin Core Metadata [15] ist ein von einer Arbeitsgruppe von Biblothekaren herkömmlicher und digitaler Bibliotheken und Experten für Auszeichnungssprachen entwickelter Standard zur Annotation von Ressourcen. Hierbei wurden die folgenden sechs Ziele verfolgt (vergleiche [15]):

- einfach zu erstellen und zu warten,
- allgemeinverständliches Vokabular,
- Konformität zu vorhandenen und in der Entwicklung befindlichen Standards,
- weltweite Gültigkeit und Anwendbarkeit,
- Erweiterbarkeit,
- Interoperabilität mit Katalog- und Indexsystemen.

Zum grundsätzlichen Umfang des Dublin Core gehören eine Menge von fünfzehn, vom Anwendungsbereich unabhänigen Elementen, die den sogenannten Kern (*Core*) bilden. Diese lassen sich in drei Gruppen einordnen:

1. *Inhaltliche* Beschreibung der Daten,
2. Informationen zum *geistigen Eigentum*,
3. eine nähere Beschreibung der *Instanz*, also der Daten selbst.

Eine ausführliche Beschreibung der Elemente und deren Verwendung ist unter [15] zu finden. Jedes dieser Elemente ist optional und kann beliebig oft vorkommen, wobei der Standard keine Beachtung der Reihenfolge vorschreibt.

Im Gegensatz zum LOM ist Dublin Core weitaus weniger umfangreich und läßt sich problemlos in ersterem abbilden. Eine Vorschrift zur korrekten Abbildung des Dublin Core läßt sich unter [16] finden.

2.6 Das Media Information Respository (MIR)

Das Media Information Respository *MIR* stellt eine Basistechnologie zur Verwaltung und strukturierten Speicherung von Multimedia–Daten zur Verfügung [FKRS01, EKRS01, FS01].

2.6.1 Ziele

Die Konzeption von interaktiven Systemen, seien sie nun netzwerkfähig oder nicht, profitiert in der Regel von einer leistungsfähigen Infrastruktur zur Verwaltung und zum Zugriff auf die verwendeten Daten. Häufig wird diesen ein Datenbankmanagementsystem zugrunde gelegt, in denen ein für die Anwendung spezifischer Datenbankentwurf implementiert wird. Zusätzlich müssen Methoden zum Datenzugriff bereitgestellt werden.

Das MIR–System erlaubt dem Anwendungsentwickler auf Basis einer graphischen Schnittstelle (GUI–Tool), Objektstrukturen zu definieren und bietet zudem über definierte, standardkonforme Schnittstellen einen netzwerktransparenten Zugriff auf die Inhalte, wie beispielsweise über JNDI (Java Naming and Directory Interface). Dadurch wird der Entwickler in die Lage versetzt, sich auf den Entwurf der Informationsstruktur und die Anwendungsfunktionen der entsprechenden Anwendung zu konzentrieren.

Um die feingranulare Speicherung von Inhalten und damit unter anderem eine erhöhte Wiederverwendbarkeit zu erreichen, werden komplexe Informationsstrukturen in einzelne Objektklassen zerlegt. So kann beispielsweise eine Internetseite als eine Kollektion von Paragraphen strukturiert werden. Jeder dieser Paragraphen bildet ein einzelnes, adressierbares Objekt im System, das selbst weitere Objekte referenzieren und selbst Ziel von Referenzen sein kann. So könnte beispielsweise ein Objekt, das eine Internetseite repräsentiert auf eine Reihe solcher Paragraphenobjekte, als Inhalt dieser Seite, verweisen.

Durch eine Klassifizierung von Objekten, verbunden mit Vererbungsmechanismen, können vorhandene Informationsstrukturen einfach erweitert und an die Anforderungen der jeweiligen Anwendung angepaßt werden.

2.6.2 Konzepte

Auf Basis einer herkömmlichen relationalen Datenbank wird ein inhaltliches Objektmodell definiert, das die Speicherung beliebiger Daten erlaubt. Dieses Objektmodell wird datenbankseitig durch ein Metamodell repräsentiert. Auf diesen abstrakten Datenspeicher können verschiedene Sichten angewendet werden. So kann der Inhalt der Datenbank beispielsweise normalisiert, in Form eines virtuellen Dateisystems dargestellt werden. Das MIR–System erlaubt eine Strukturierung dieser Sicht durch die Verwendung von Verzeichnissen. Jedem Objekt wird, wenn es

neu angelegt oder importiert wird, ein bestimmter Speicherplatz in Form einer Pfadangabe im System zugewiesen. Dies ist möglich, da die Darstellung als virtuelles Dateisystem normalisiert ist, d.h. es kommt jedes Objekt nur einmal an einer bestimmten Stelle im System vor. Es sind aber auch Transformationen in nicht–normalisierte Sichten möglich, auf diesen Fakt wird bei der Vorstellung der →Namensräume eingegangen.

Das MIR–System unterscheidet grundsätzlich zwei Arten von Objekten, die konkrete Ausprägungen genau einer →MIR–Klasse sind:

Datenobjekte (*DOBs*): Datenobjekte kapseln den Inhalt eines Objektes. Sie verfügen über eine Reihe vordefinierter Attribute wie beispielsweise die Größe des Inhaltes, dessen MIME-Type oder Informationen über den Besitzer. Es ist jedoch nicht möglich, DOBs um weitere Attribute zu erweitern.

Medienobjekte (*MOBs*): Medienobjekte erlauben die freie Definition von Attributen bzw. Metainformationen. Des weiteren können sie Referenzen auf andere Medien- oder Datenobjekte enthalten.

Ein Rechtemodell bestehend aus Nutzerrechten, Gruppenrechten und Rollen erlaubt eine feingranulare Zugriffsteuerung. So besteht nicht nur die Möglichkeit, Zugriff auf das System zu gewähren bzw. zu verweigern, sondern auch jedes in der Repository gespeicherte Objekt mit speziellen Zugriffsrechten zu versehen.

Jedem Objekt ist eine *MIR–Klasse* zugeordnet. Diese legt die Eigenschaften (*Attribute*) eines Objektes fest, stellt jedoch keine klassenspezifischen Methoden im Sinne von objektorientierten Datenbanken zur Verfügung. Vorhandene MIR–Klassen können über den Mechanismus der Vererbung um zusätzliche Attribute erweitert werden, wobei aber nur Einfach–Vererbung (single inheritance) möglich ist.

Alle DOBs sind eine direkte Ausprägung der MIR–Klasse *GenericDob*. MOBs sind entweder eine direkte Ausprägung der MIR–Klasse *GenericMob* oder einer davon abgeleiteten Kindklasse.

Dadurch entsteht ein Klassenbaum, dessen Wurzelelement die MIR–Klasse GenericClass ist.

Durch die Verknüpfung von Medienobjekten untereinander und von Medien- mit Datenobjekten besteht eine weitere Möglichkeit, Inhalte zu strukturieren und so komplexe Informationsstrukturen abzubilden. Ein Verweis eines MOBs auf ein anderes MOB oder DOB wird als *Target* bezeichnet. Targets können Symbolnamen, sogenannte *Identifiers*, zugeordnet werden. Jeder Identifier stellt eine adressierbare Einheit dar. Die Adressierbarkeit von Objekten über ihre Identifier spannt somit einen weiteren Namensraum im MIR–System auf. Ein Ausdruck zur Adressierung wird als *Namenspfad* bezeichnet.

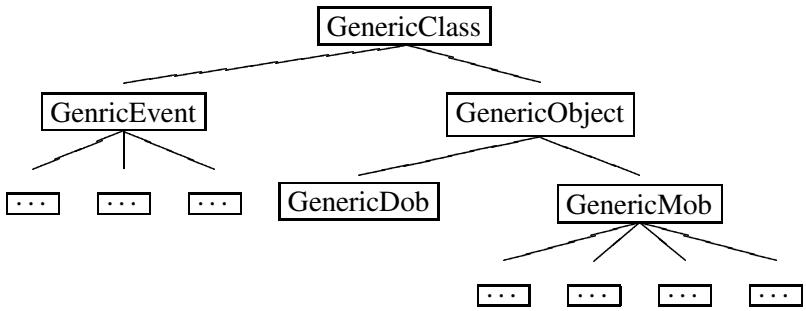

Abbildung 2.8: MIR–Klassenbaum

Diese Sicht ist im Gegensatz zur Sicht des virtuellen Dateisystems nicht normalisiert, d.h. das gleiche Objekt kann über unterschiedliche Namenspfade adressiert werden.

Beispiel 2.6.1
Die in Abbildung 2.9 dargestellte Object „index“ kann über die folgenden Angaben adressiert werden:

1. *Studieren/Fachbereiche/Index (Namenspfad)*
2. *Informieren/Fachbereiche/Index (Namenspfad)*
3. */WWW–Seiten/Struktur/Index (Pfad im virtuellen Dateisystem)*

MOBs können zudem *Events* beinhalten. Events waren ursprünglich als Komponenten zur Zeitsteuerung konzipiert, d.h. sie können die Prozessierung des MOBs oder seiner Targets beeinflussen. In der weiteren Entwicklung des MIR–Systems wurde dieses Konzept über seine temporalen Aspekte hinaus erweitert, so daß Events allgemeine Instruktionen zur Prozessierung der Targets enthalten, die vom entsprechenden Laufzeitsystem verarbeitet werden können. Events sind Ausprägungen der MIR–Klasse *GenericEvent* oder einer von GenericEvent abgeleiteten Kindklasse und sind somit attributbehaftet, analog zu den Medienobjekten. Events existieren nicht als „freie“ Objekte im MIR–System, sondern sind immer an ein Medienobjekt gebunden.

Weitere Bestandteile des MIR–Systems sind graphische Tools zum Anlegen, Bearbeiten und Verwalten von MIR–Objekten, Nutzern und MIR–Klassen.

2.6.3 Implementierung

Das MIR–System ist als eine klassische 3–Tier Architecture implementiert, wobei insbesondere auf den Einsatz plattformunabhängiger und standardisierter Technologien geachtet wird, um die Funktionsfähigkeit in der heterogenen Betriebssystemwelt des Internets zu gewährleisten. Durch

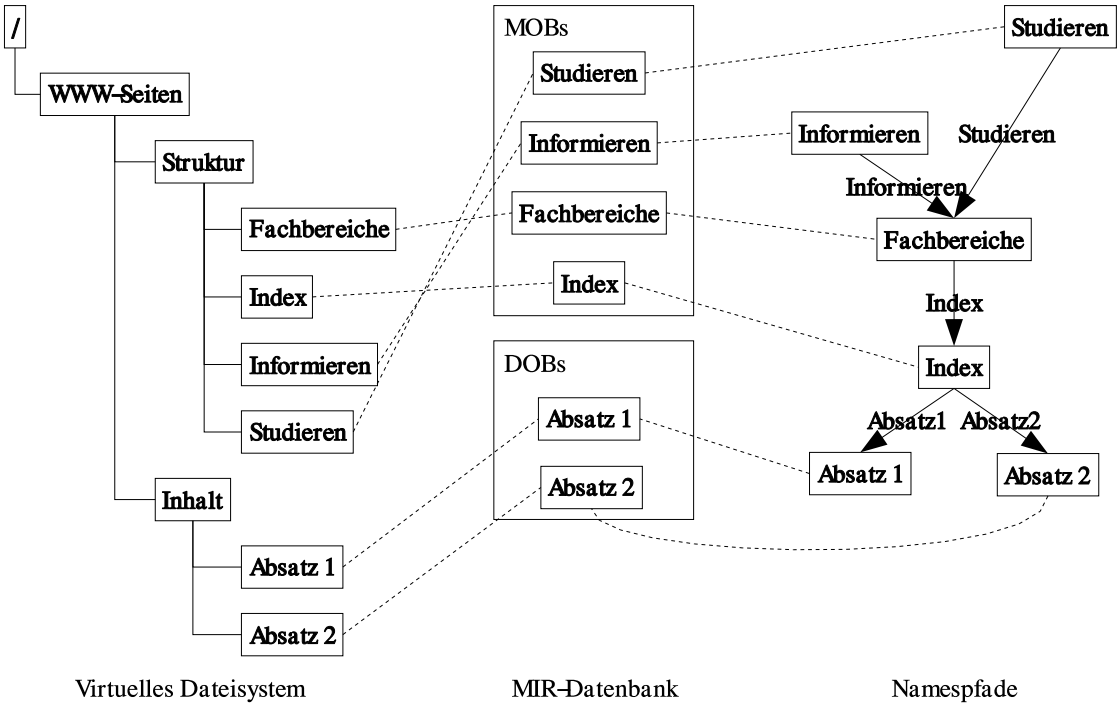

Abbildung 2.9: Beispiel für Namensräume im MIR–System

ein mehrschichtiges Modell, das in seinen Bestandteilen austauschbar ist, wird versucht, dem Anspruch der Plattform- und Produktunabhängigkeit gerecht zu werden. Nachstehend werden die drei Schichten des Modells näher erläutert.

Datenbankschicht Eine relationale Datenbank (in der gegenwärtigen Implementierung der Sybase DataServer) bildet die Datenbankschicht, in der die Objekte auf Grundlage eines Metamodells gespeichert werden. Der Zugriff auf Objekte in der MIR Datenbank wird durch eine API aus Stored Procedures gekapselt, so daß aus darüberliegenden Schichten über Funktionsaufrufe komfortabel auf die Objekte zugegriffen werden kann.

Middlewareschicht Ein Java-Applikationsserver (momentan der Sybase Jaguar) stellt Komponenten zur Manipulation von Klassen (*ClassManager*), Objekten (*ObjectManager*) und Nutzern (*UserManager*) zur Verfügung.

Die Kommunikation zur Datenbank erfolgt hierbei über JDBC, client-seitig steht CORBA zur Verfügung.

Die einzelnen Komponenten der Middleware bieten ein reichhaltiges Set von high-level Methoden, um Operationen auf bzw. mit Objekten der Datenbank durchzuführen. Unter anderem übernimmt die Middleware die Konvertierung der Datenbankobjekte in Java Objekte, die ihrerseits

Methoden, beispielsweise zum Zugriff auf Attribute der MIR–Klasse oder zur Repräsentation eines MIR–Objektes als XML-Text, bereitstellt.

Client Applikationen Es existieren vielfältige Schnittstellen und Applikationen, um mit dem MIR–System zu arbeiten. Grundsätzlich können Client-Anwendungen in jeder Sprache die CORBA unterstützt, implementiert werden. Beispielhaft sollen im folgenden drei Schnittstellen vorgestellt werden, die als Ausgangspunkte für weitere Applikationen dienen können:

MIR–JNDI: JNDI (Java Naming and Directory Interface) [20] bietet einen standardisierten Zugriff auf die Objekte des MIR–Systems an. Neben Zugriffsoperationen wie dem Lesen und Schreiben von Objekten, werden des weiteren Suchfunktionen zur Verfügung gestellt. Sowohl bei den Suchoperationen als auch den Zugriffsoperationen ist die Verwendung beider Namensräume (virtuelles Dateisystem und Namenspfade) möglich.

MAF (MIR Application Framework): Das MAF stellt eine auf Swing/JFC[5] aufsetzende Java API zur Entwicklung netzwerktransparenter, graphischer Client-Applikationen zur Verfügung. Standardmäßig bietet es Unterstützung für das Session–Management, den Respository Zugriff sowie die Nutzer- und Gruppenverwaltung an. Die Netzwerktransparenz wird durch die Verwendung von Java Web Start [21] erreicht, wodurch der Start von Anwendungen aus dem Browser heraus über das Internet ermöglicht wird.

MIRXML Taglib: Mittels der MIRXML Taglib besteht die Möglichkeit des leichtgewichtigen Zugriffs auf Objekte im MIR–System. Eine Taglib (*Tag Library*) stellt im allgemeinen, XML Tags, die mit Java Instruktionen hinterlegt werden, zur Verfügung, um Operationen auszuführen. Zudem werden einige grundlegende Operationen, wie beispielsweise das Auslesen oder Ändern von Eigenschaften von Mobs, bereitgestellt.

Das MIR–System erfüllt die in der Einleitung zu diesem Kapitel formulierten Anforderungen an die feingranulare Speicherung und die Multivitalität von Inhalten in vollem Umfang. Durch seine Werkzeuge zur Informationsstrukturierung und vielfältigen Schnittstellen bietet es eine ideale Infrastruktur zur Realisierung von beispielsweise Online–Lernsystemen. Deshalb wird das MIR–System auch Grundlage der in Kapitel 5 beschriebenen Implementierung sein.

[5] Java Foundation Classes

Kapitel 3

Problemfeld Hyperreferenz

»[. . .] simply linking one text to another fails to achieve the expected benefit of hypermedia and can even alienate the user«
G.P. Landow ([Lan89])

3.1 Einleitung

Nachdem im vorangegangenen Kapitel auf die technischen und konzeptionellen Grundlagen für Hypermediasysteme mit dem Fokus auf das World–Wide–Web eingegangen wurde, soll nun eine Diskussion hinsichtlich der inhaltlichen und der daraus resultierenden technischen Anforderungen für Online–Lernsysteme im World–Wide–Web (eLearing–Systeme, eLearning–Anwendungen) erfolgen.

Wie schon in der Einleitung zum vorigen Kapitel angesprochen, hängt der Erfolg solcher Systeme neben der Qualität der Inhalte, insbesondere von den Interaktionsmöglichkeiten ab, da diese fester Bestandteil des didaktischen Modells sind. Die Beurteilung der Qualität der Inhalte oder der Interaktivität einer Lernanwendung hinsichtlich ihres didaktischen und pädagogischen Wertes kann nicht Aufgabe dieser Arbeit sein, wohl aber die Diskussion von Konzepten und Anforderungen an ein System, um die technischen Voraussetzungen für eine praktikabele und anwenderfreundliche Online–Lernanwendung zu schaffen.

Die Umsetzung von Interaktivität in Hypermedia erfolgt in der Regel auf Basis von Hyperreferenzen.

Definition 3.1.1 (Hyperreferenz)
Eine Hyperreferenz – alternativ (Hyper-)Link oder Verweis – ist eine logische Verknüpfung von mindestens zwei → Ressourcen.

Definition 3.1.2 (Ressource)
Eine Ressource ist eine adressierbare Informations- oder Diensteinheit. (vergleiche [11])

Hyperreferenzen sind ein mächtiges Mittel zur Verknüpfung von Inhalten und zur Gestaltung von Interaktion. Ihre Ausgestaltung ist abhängig von den Fähigkeiten des Hypermediasystems, in dem sie Anwendung finden. Das Spektrum reicht von simplen, strikt vordefinierten Inhaltsverknüpfungen, wie dieses bespielsweise im World–Wide–Web der Fall ist, bis hin zu Anwendungsszenarien mit dynamisch oder automatisch erstellten Verweisen, über mehrwertige Verzweigungen (Indizes oder Glossare) oder der Verarbeitung von Metastrukturinformationen, zum Beispiel RDF Ausdrücken.

Inhalte können in Abhängigkeit der Verwendung von Hyperreferenzen bereichert, aber auch gestört werden, so daß besondere Anforderungen sowohl technischer als auch inhaltlicher Art an eLearning–Systeme gestellt werden müssen.

In Abschnitt 3.2 wird versucht, sich den inhaltlichen Problemen aus Sicht des Autors und des Anwenders anzunähern, um dann einige allgemeine Anforderungen bezüglich der Implementierung von Hyperreferenzen im System abzuleiten. Der darauffolgende Abschnitt stellt sodann eine Lösungsmöglichkeit für die zuvor erläuterten Probleme vor. Anhand dieser Lösung wird abschließend in Abschnitt 3.4 gezielt auf die technischen Anforderungen eingegangen.

Ziel dieses Kapitels ist es, Lösungsmöglichkeiten unter Nutzung bereits beim Anwender vorhandener Technologien zu entwickeln. So wird nicht über die Konzeption einer neuen Benutzerschnittstelle in Form eines neuen Browsers oder ähnlichem nachgedacht, sondern versucht die Möglichkeiten von HTML zu nutzen und serverseitig Verfahren zur angemessenen Prozessierung anzubieten.

3.2 Inhaltliche Anforderungen an Hyperreferenz Systeme

Hyperreferenzen sind Kern der Interaktivität und damit wichtiger Gegenstand der didaktischen Betrachtung. Im Gegensatz zu einem Buch, das sequentiell dargestellt wird, erlauben Hyperreferenzen ein assoziatives Navigieren. Der Nutzer wird mit Netzgraphen, statt mit pseudo–linearen Abläufen konfrontiert, die ihrerseits komplexe Lern- bzw. Interaktionsmöglichkeiten bieten (vergleiche Abbildung 3.1 und Abbildung 3.2).

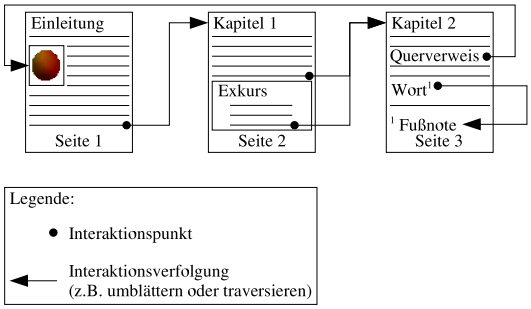

Abbildung 3.1: Buch

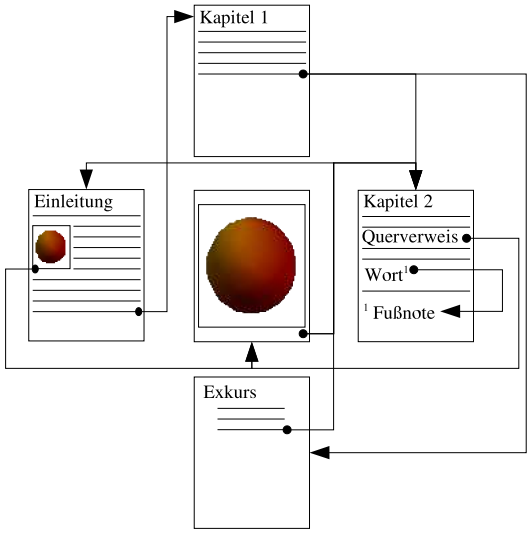

Abbildung 3.2: Webseiten

Vom didaktischen Standpunkt aus betrachtet, ist die Abfolge von Informationen jedoch von Bedeutung. In einem Buch ist dieser „rote Faden“ implizit durch die physikalische Abfolge der Seiten vorgegeben. Im Falle von Hypermedia–Systemen muß der Autor, unterstützt vom System, für eine oder mehrere konsistente und logisch nachvollziehbare Abfolgen der Informationen, durch Verknüpfung der Inhalte mittels Hyperreferenzen, sorgen. Konstruktivistische Lernansätze gehen beispielsweise von zirkulären anstatt von linearen Modellen aus, d.h. der Lernende „kreist“ um ein Thema und kann somit seinen Wissensstand kontinuierlich verfeinern.

Jede Umsetzung, die über ein lineares Übertragen einer jeden Buchseite in ein eigenes Hypermedia–Dokument, verknüpft durch eine Referenz auf seinen Vorgänger bzw. Nachfolger, hinaus geht, verlangt ein inhaltiches Konzept zur Notation von Hyperreferenzen. Ein solches Konzept beinhaltet Regeln über die inhaltliche Ausgestaltung und Funktion von Hyperreferenzen und sollte letztendlich auch für den Lernenden transparent nachvollziehbar sein. Gibt der Autor ein in sich konsistentes und verständliches inhaltliches Verweismodell als Umsetzung des didaktischen Interaktionsmodells vor, so ermöglicht er gleichzeitig dem Nutzer dieses zu erkennen und von einer besseren Orientierung im System zu profitieren.

Bereits George P. Landow erkennt in seiner Arbeit „The Rhetoric of Hypermedia“ [Lan89] die Notwendigkeit „sprechende“ Verweise zu erstellen („A Rhetoric of Hypermeida“), um dem Anwender eine bessere Orientierung im System zu ermöglichen. Sich im System zurechtzufinden, ist die Voraussetzung, um erfolgreich mit dem System zu interagieren.

3.2.1 Die Rolle des Autors

Der Autor ist verantwortlich für die inhaltliche Ausgestaltung der Interaktionsmöglichkeiten und legt damit die „roten Fäden“ fest, die der Anwender verfolgen soll. Da jedoch Hypermedia–Systeme, wie zuvor aufgezeigt, weitaus mehr Interaktionsmöglichkeiten als beispielsweise Bücher besitzen, besteht gleichzeitig die Gefahr, daß der Nutzer diesen vorgegebenen Weg verläßt. Das ist an sich nicht problematisch, wenn der Anwender jederzeit die Möglichkeit hat, den „roten Faden“ wieder aufzunehmen oder zumindest wieder zu einem früheren Ausgangspunkt zurückzukehren.

Besteht diese Möglichkeit für den Anwender nicht, ist er desorientiert. Der Nutzer befindet sich auf irgendeiner Seite, ohne zu wissen woher er kommt, was er hier will und wie er wieder an einen früheren Ausgangspunkt zurückkehrt. Hierfür wurde durch J. Conklin 1987 der Begriff „lost in hyperspace“ [Con87] geprägt.

Auch George P. Landow erkennt in seiner Arbeit [Lan89] dieses Problem und reduziert es im ersten Schritt auf die folgenden drei Aspekte:

1. Navigation durch verschiedene Dokumente (*Navigation*),
2. „Verlassen“ von Dokumenten durch Traversieren von Verweisen (*Departure*),
3. „Ankunft“ auf neuen Dokumenten als Ergebnis einer abgeschlossenen Verweisverfolgung (*Arrival*).

Grundlegend für die anschließenden detaillierten Betrachtungen ist die These Landows, daß der Nutzer immer dann desorientiert ist, wenn er nicht weiß, „wo“ er ist, keine Möglichkeit sieht, zu einem zuvor betrachteten Dokument zurückzukehren oder eine gewünschte bzw. vermutete Information nicht findet. Landows Lösungsvorschlag ist es, „sprechende“ Verweise zu benutzen und somit dem Anwender die Orientierung im System zu ermöglichen. Hierzu entwickelt Landow neunzehn Regeln für Autoren, die dem Anhang A zu entnehmen sind.

3.2.1.1 Navigation

Der Themenkomplex „Navigation“ beschäftigt sich mit dem Problem der Navigation durch eine Reihe mittels Hyperreferenzen verbundener, thematisch zusammengehöriger Ressourcen oder, um mit den Worten des Word–Wide–Webs zu sprechen: Eine Menge verlinkter Seiten.

Um dem Nutzer überhaupt eine Form der Orientierung zu ermöglichen, sollten zumindest die Minimalfunktionalitäten, wie sie uns auch von Büchern bekannt sind, vorhanden sein.

Hierzu gehören laut Landow zumindest die Möglichkeiten

a) *die aktuelle Position ermitteln*, beispielsweise durch einen den Seitenzahlen eines Buches nachempfundenen Mechanismus oder anderer geeigneter Identifikationsfaktoren,

b) *Zusammenhänge zu anderen Ressourcen erkennen*, beispielsweise durch geeignete Überblicksformen, Landow führt diese ausführlich aus [Lan89] Seite 49ff,

c) *zum Ausgangspunkt zurückkehren*, hier ist zu diskutieren, in wie weit die „Zurück Knopf" Funktionalität des Webbrowsers ausreichend ist,

d) *Zusammenhänge zwischen Ressourcen, die nicht direkt miteinander verknüpft sind, aber thematisch verwandte Ressourcen darstellen, erkennen*

zu können (vergleiche [Lan89], Seite 44). Das Kapseln mehrerer thematisch ähnlicher Ressourcen in Überblicks- oder sogenannten Gatewaydokumenten, erleichtert unter Umständen das Auffinden eines entsprechenden Einstiegspunktes, schützt den Nutzer aber nicht davor, in der weiteren Verweistraversierung vom durch dem Autor beabsichtigten „roten Faden" abzukommen.

Gerade für eLearning–Systeme ist es aber von Bedeutung, daß der Nutzer sich zumindest im groben von dem Autor durch das didaktische Modell leiten läßt, so daß sowohl der logische als auch der didaktisch–pädagogische Fluß nicht unterbrochen wird. Hierzu muß dieser vorgegebene „rote Faden" aber zuerst einmal vom Nutzer erkannt werden. Das ist aber nur unproblematisch, solange eine sequentielle Verknüpfung der Seiten vorliegt. Versucht man jedoch, das System, beispielsweise mit Querverweisen oder Exkursen zu erweitern, ist dem Anwender nicht in jedem Fall klar, welche Funktion und Bedeutung ein Verweis trägt.

Vor dem Hintergrund der ersten und dritten Regel Landows (siehe Anhang A), wird deutlich, wie Verweise auch Inhalte zerstören können. Diese Regeln besagen, daß schon die Existenz eines Verweises in einem Hypermedium den Anwender motiviert, dahinter eine für ihn interessante und wichtige Information zu erwarten. Wird diese Erwartung nicht erfüllt, empfindet der Nutzer diese Dokumente als zusammenhangslos und unbedeutend (vergleiche [Lan89], Seite 42).

In der verbalen Kommunikation erleichert die gute Rhetorik eines Redners, komplexe Zusammenhänge zu verstehen und Beziehungen zwischen Fakten herstellen zu können. Die Rhetorik angewendet auf Hypermedia–Systeme ist dann die Kunst des Autors, konsistene und bedeutungstragende Hyperreferenzen zu erstellen. Unter Verwendung einer solchen Rhetorik für Verweise kann eine unnötige Verweisverfolgung unter Umständen vermieden werden, da der Nutzer erkennen kann, wohin und warum ein Verweis dorthin führt, bzw. warum er ein bestimmtes Dokument erreicht hat. Außerdem sollte ein Autor immer →bidirektionale oder →multidirektionale Verweise verwenden, da er dem Anwender damit eine definierte Rücksprungbeziehung vorgibt und dieser sich nicht auf die zu oft durch externe Einwirkungen beeinflußte, unbefriedigende Funktionalität des „Zurück–Knopfes" verlassen muß.

3.2.1.2 Departure

Bereits bevor der Anwender eine Hyperreferenz aktiviert und damit deren Traversierung auslöst, sollte es ihm möglich sein, sich über das Verweisziel zu informieren. Wichtig ist, daß für den Nutzer irgendein *nachvollziehbarer* Zusammenhang gegeben ist, beispielsweise der Art: „Ressource B ist eine Spezialisierung von Ressource A“, „Ressource A behandelt die Kultur des Abendlandes, Ressource B die des Orient.“ oder ähnliches. Somit wird der Anwender befähigt, vor der Traversierung eines Verweises zu entscheiden, ob er dort gewünschte Informationen finden wird und so eine desorientierende Verweisverfolgung vermeiden kann. Solche Beziehungen zwischen verschiedenen Ressourcen können häufig nur aus dem Inhalt, also dem Kontext, in dem ein Verweis auftaucht, ermittelt werden. Eine maschinelle Verarbeitung ist somit nicht anwendbar.

Um auch eine automatisierte, maschinelle Verarbeitung zu ermöglichen, ist es nicht ausreichend, semantische Beziehungen im eigentlichen Inhalt zu kodieren. Darüber hinaus müssen auf Basis von Annotationsverfahren für Ressourcen maschinell prozessierbare Semantiken hergestellt und von der Laufzeitumgebung (*Run–Time Layer*) verarbeitet werden.

Weitaus größer wird die Bedeutung solcher, für den Nutzer transparenten Semantiken, wenn ein System zudem →multidirektionale Verweise unterstützt, da zum einen geeignete Präsentationsformen für die möglichen Verweisziele angeboten werden müssen, zum anderen ja durch den Nutzer eine Entscheidung für eines der angebotenen Ziele getroffen werden soll. Um einen qualifizierten Entscheidungsprozeß zu ermöglichen, muß der Benutzer zumindest eine Vorstellung von dem Inhalt hinter einem solchen Verweisziel besitzen und eine Beziehung zu seinem momentanen Aufenthaltsort herstellen können.

In jedem Fall ist der Autor für die Bedeutung der Verweise und deren Klarstellung gegenüber dem Nutzer verantwortlich. So wird der Autor bei der Verwendung von Annotationen gezwungen, sich über die Zusammenhänge der Ressourcen, die er in Beziehung setzen möchte, klar zu werden und diese, in Form der entsprechenden Metainformationen, zu formulieren. Wichtig ist hierbei, daß der Autor eine entsprechende Unterstützung durch das System erfährt. So könnte er beispielsweise beim Anlegen einer neuen Ressource zur Eingabe von Informationen zur Annotation aufgefordet werden.

Unabhängig davon ob ein Verweis aussagekräftig ist, stellt sich die Frage, wie sich die Laufzeitumgebung hinsichtlich der Präsentation verhalten soll, wenn der Nutzer die Verfolgung einer Hyperreferenz initiiert (vergleiche [HBR94] und Abschnitt 2.3.2). Bekannte Ansätze hierfür sind, die dargestellten Inhalten der →Start–Ressource durch die der →End–Ressource zu ersetzen (Ersetzungsstrategie) oder den Inhalt der →End–Ressource in einem neuem Fenster zu öffnen. Wird eine Ersetzungsstrategie verfolgt, besteht die Gefahr, daß der Nutzer den Bezug zu vorangegangen Ressourcen verliert. Im Gegensatz dazu ermöglicht ein Laden jeder →End–

Ressource in einem eigenem Fenster zwar den Bezug zur →Start–Ressource, birgt aber gleichzeitig das Problem der Unübersichtlichkeit. Für dieses Problem kann keine Pauschallösung angeboten werden; grundsätzlich ist jedoch davon auszugehen, daß sowohl der Autor, als auch der Benutzer die Möglichkeit haben sollten, die Verweistraversierung in diesem Sinne zu beeinflussen.

Definition 3.2.1 (Start–Ressource)
Die Start–Ressource ist die Quelle einer Hyperreferenz, von der aus die Traversierung startet (in Anlehnung an [10, 11]).

Definition 3.2.2 (End–Ressource)
Die End–Ressource ist das Ziel einer Hyperreferenz, das nach der Traversierung der Hyperreferenz erreicht wird (in Anlehnung an [10, 11]).

Grundsätzlich erleichtert der Autor dem Nutzer den Umgang mit dem System, wenn er ein konsistentes, in sich schlüssiges und vor allem auch für den Anwender nachvollziehbares Verweisschema benutzt. So kann der Nutzer beispielsweise davon ausgehen, daß ein Verweis, plaziert an einer bestimmten, ausgezeichneten Stelle im einem Dokument, immer eine ähnliche Funktion und Bedeutung einnimmt.

3.2.1.3 Arrival

„Das Ankommen auf neuen Ressourcen“ als Ergebnis einer erfolgreichen Traversierung einer Hyperreferenz, sollte dem Nutzer eine einfache und sofortige Orientierung ermöglichen. Dies ist insbesondere wichtig, wenn die End–Ressource nur einen Teil einer Information, also beispielsweise nur einen Paragraphen einer Webseite und nicht die gesamte Seite selbst darstellt.

Ausgehend von der Annahme, daß der Autor die Rhetorik seiner Verweissetzung dem Nutzer vermitteln konnte und dieser somit zielgerichtet die Hyperreferenz zu dieser Ressource verfolgte, muß der Anwender zwischen dieser Ressource und der Start–Ressource eine Beziehung herstellen können. Folglich darf eine Information, beispielsweise ein Bild, nicht ohne weitere Informationen, ohne Kontext dargestellt werden. Solche Zusammenhänge können einerseits implizit vorgegeben sein, wie beispielsweise ein bestimmtes Wort innerhalb eines Absatzes, oder müssen andererseits explizit angegeben werden, zum Beispiel durch einen Begleittext bei einem Bild. Eine solche Angabe ermöglicht es, dem Benutzer Beziehungen zwischen Ressourcen zu verständlich zu machen.

3.2.2 Verarbeitung von zusammengesetzen Komponenten

Als Inhalte eines eLearning–Systems werden überwiegend zusammengesetzte Komponenten, sogenannte Composite Components im Sinne des Dexter Modells [HS94], Anwendung finden. Wie zuvor gezeigt, ist es bei der Traversierung einer Hyperreferenz, an der Composite Components beteiligt sind, wichtig, diese unter Beachtung ihres Quellkontextes bzw. Zielkontextes zu prozessieren. Dies ist unbedingt notwendig, da die Idee des Kontextes gleichsam bedeutet, daß gleiche Inhalte in verschiedenen Zusammenhängen oder Sichten auftreten können. Hierbei können, wie in den Ausführungen zum AHM in Abschnitt 2.3.2 die folgenden beiden Kontextarten unterschieden werden:

- der *Quellkontext*, der seinerseits wiederum mit dem Departure Problem korrespondiert,
- der *Zielkontext*, der im Zusammenhang mit dem Arrival Problem betrachtet werden muß.

Verweismodelle, deren Komponenten direkt in den Inhalt der beteiligten Ressourcen eingebunden werden, wie dies beispielsweise bei HTML der Fall ist, geben einen entsprechenden Kontext implizit vor. Anderenfalls muß der Kontext ausdrücklich angegeben werden. Vernachlässigt man die Art, wie Graphiken in HTML Dokumente eingebunden werden, so existieren vereinfacht betrachtet HTML Dokumente nur als atomare Komponenten auf Ebene des Storage Layers. Eine Komposition wird erst auf der Ebene der Laufzeitumgebung (Run–Time Layer) durch Hilfskonstrukte wie beispielsweise Frames vorgenommen.

Werden Ressourcen jedoch aus einzelnen Komponenten, beispielsweise skriptgesteuert aus eine Datenbank, dynamisch zusammengesetzt, so können auch im Falle von HTML basierten Hypermediadokumenten „echte“ Composite Components auftreten. Hierbei muß der Kontext zusammen mit der Adressierung der entsprechenden End–Ressource vorgegeben und verarbeitet werden, verwiesen sei hierbei beispielsweise auf die Namenspfade im MIR–System[EHK+02].

3.2.3 Einflußnahme des Rezipienten

Ein für den Nutzer nachvollziehbares, inhaltlich bedeutungtragendes Verweismodell bereichert das diadaktische Modell einer eLearning–Anwendung, ist aber nicht der einzige Faktor für den Erfolg einer solchen Anwendung. Die Nutzer eines Systems unterscheiden sich stark in ihren Anforderungen, Fähigkeiten und Voraussetzungen. Somit steht der Erfolg einer Online–Lernanwendung in starker Abhängigkeit von der Fähigkeit eines Systems, sich an den Anwender anzupassen. Zuerst sollen die hiermit eng verbundenen Begriffe Adaptivität und Adaptierbarkeit erläutert werden, um dann anschließend auf mögliche anwenderorientierte Anpassungen einzugehen. Unerläßlich ist hierbei die Kommunikation des Anwenders mit dem System.

3.2.3.1 Adaptivität und Adaptierbarkeit

Generell lassen sich Benutzer multimedialer Lehr- und Informationssysteme anhand der benötigten Unterstützung, im Umgang mit einem solchen System bzw. zum Erreichen eines Informations- oder Lehrzieles, differenzieren. „Novizen“, Nutzer die sich weder mit dem System noch mit dem jeweiligen Gegenstandsbereich auskennen, und „Experten“, die lediglich spezifische Informationen nachschlagen, bilden die beiden Extreme bezüglich des Unterstützungsbedarfs (vergleiche [Leu97]). Aufgrund des Entwicklungsprozesses des „Novizen“ zum „Experten“ sinkt der Unterstüzungsbedarf laut Leutner [Leu97] kontinuierlich, allein in Abhängigkeit vom Erfolg der Interaktion mit dem System.

Der Erfolg von Lernsystemen im allgemeinen hängt laut Leutner [Leu97] im wesentlichen von zwei Größen ab:

- Benutzerfreundlichkeit (*usibility*),
- Erlernbarkeit (*lernability*).

Beide Faktoren tragen als wesentliche Teilkomponente die Interaktion des Nutzers mit dem System in sich. Wie bereits in der Einleitung bemerkt, stellen Hyperreferenzen das wesentliche Interaktionsmoment dar.

Die Erlernbarkeit erfolgt durch eine angemessene Adaptation des Systems, so daß die Aspekte der Adpativität und Adaptierbarkeit genauer betrachtet werden müssen. Die Frage nach der Adaptivität eines Systems lautet demnach, in wie weit das System selbständig den Unterstützungsbedarf des Lernenden diagnostiziert und schlußfolgernd durch geeignete Maßnahmen reagiert. Im Gegensatz dazu bezeichnet die Adaptierbarkeit die Fähigkeit eines Systems, auf Basis externer Diagnosen, extern so eingestellt werden, das es dem Unterstüzungsbedarf des Lernenden möglichst gut entspricht.

Definition 3.2.3 (Adaptierbarkeit, Makro–Adaptation)
Ein System ist dann *adaptierbar*, wenn es durch externe Eingriffe an veränderte Bedingungen angepaßt werden kann [Leu97].

Im Rahmen der Marko–Adaptation erfolgt eine Anpassung des Lehrsystems in größeren zeitlichen Abständen, zumindest aber zu Beginn der jeweiligen Lehreinheit. Lernprozesse werden als offene Wirkungskreise aufgefaßt und gesteuert, wobei aber unmittelbare Rückkopplungen zwischen Lehrergebnissen und Maßnahmen zur Makro–Adaptation vorgenommen werden.

Die Adaptierbarkeit eines Systems sollte nur auf in Zusammenhang mit einer Lehreinheit konstante Größen angewand werden, wie zum Beispiel die Vorbildung des Lernenden.

Definition 3.2.4 (Adaptivität, Mikro–Adaptation)
Ein System ist dann *adaptiv*, wenn es sich selbständig an veränderte Bedingung anzupassen vermag [Leu97].

Im Gegensatz zur Makro–Adaptation wird die Anpassung des Systems kontinuierlich vorgenommen, d.h. in sehr kurzen zeitlichen Abständen analysiert, überprüft und angepaßt. Der Lernprozeß wird als geschlossener Wirkungskreis aufgefaßt, die Lernergebnisse des Lerndenden wirken sich unmittelbar auf die Lehrmethoden aus.

Leutner betracht die Interaktion als einen wesentlichen Schlüssel für den Erfolg eines eLearning–Systems (vegleiche [Leu97]) und legt hierbei insbesondere auf Anpassung des Systems an die Nutzerbedürfnisse wert. Um auf diese einzugehen, bietet er eine Reihe von Adaptionsregeln an (vergleiche [Leu97]):

- Adaptation des Instruktionsumfangs und der Lernzeit,
- Adaptation der Instruktionssequenz,
- Adaptation der Aufgaben-Präsentationszeit und adaptive Antwortzeitbegrenzung,
- Adaptation der Aufgabenschwierigkeit,
- Adaptive Hilfen beim entdeckenden Lernen,
- Adaptive Definition neu zu erlernender Begriffe
- Adaptiver Informationszugriff in Hypertext–Systemen.

Diese Regeln lassen sich grundsätzlich in zwei Kategorien unterteilen, zum einen Regeln mit zeitlichem Bezug, zum anderen solche die inhaltliche Aspekte behandeln. Für den folgenden Verlauf der Arbeit sind jedoch überwiegend Regeln mit inhaltlicher Bedeutung interessant.

3.2.3.2 Annotationen durch den Anwender

Insbesondere die Darstellung komplexer Zusammenhänge und Abläufe kann zu Verständnisschwierigkeiten auf Seite der Nutzer führen. Hilfreich könnten hierbei Anmerkungen anderer Anwender oder eigene Kommentare an bestimmten Textstellen oder Graphiken sein.

Heutzutage ist es eine relativ weitverbreitete Möglichkeit, Webseiten mit Kommentaren zu versehen. Diese tauchen aber in der Regel nicht als Verweise im Text auf, sondern werden separat am Ende eines Absatzes oder am Ende der Seite angezeigt. Für ein fortgeschrittenes eLearning–System sollte dieser Ansatz, zu Annotationen die den Inhalt referenzieren, weitergeführt werden. Um trotzdem noch Übersichtlichkeit zu gewährleisten, sollte die Anzeige von solchen, durch Kommentare oder Bemerkungen erzeugten Verweise durch den Anwender steuerbar sein.

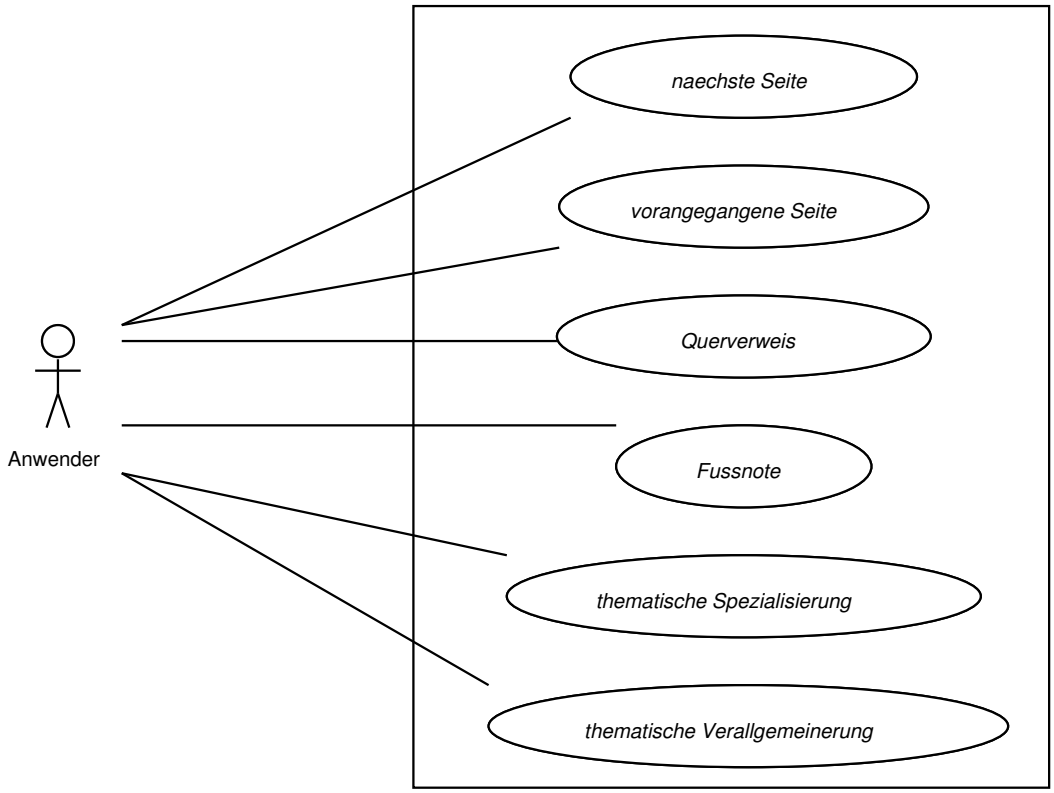

Abbildung 3.3: Beispiele für Anwendungsfälle von Hyperreferenzen

3.2.3.3 Adaptiver Informationszugriff

Unterschiedliche Voraussetzungen, Lern- und Informationsziele der Anwender verlangen eine an den Lernenden angepaßte didaktische und pädagogische Darstellung der Inhalte. Das ist mit den Mitteln, die HTML bietet, unzureichend zu realisieren, da der Anwender, selbst wenn der Autor rhetorische nachvollziehbare Hyperreferenzen verwendet hat, mit einer Menge unterschiedlich klassifizierbarer Verweise konfrontiert wird. Zum einen können diese die Struktur, also die inhaltliche und logische Abfolge, vorgeben. Zum anderen werden sie benötigt, um das Navigieren durch das System, gleichsam dem Blättern in einem Buch, zu ermöglichen. Fußnoten und Querverweise werden ebenfalls mit Hyperreferenzen realisiert (siehe Abbildung 3.3).

Eine solche Vielfalt an inhaltlich unterschiedlichen Hyperreferenzen kann den Nutzer verwirren, zudem wird der didaktische Kontext, also die Lernsituation nicht hinreichend berücksich-

tigt. Ein thematisch versierter Anwender ist vermutlich an einer Spezialisierung seines Wissens interessiert, wogegen ein Novize eher zusätzlichen, erklärenden Hinweisen bedarf. An diesem Punkt setzt das Prinzip des *adaptiven Informationszugriffs* an. Hierbei geht es um die Frage des online–dynamischen Verbindens von Informationseinheiten eines umfangreichen Hypertextes unter Einbeziehung von online–erfaßbaren Eigenschaften wie das Informations–Suchverhalten eines Benutzer. Grundüberlegung ist hierbei, dem Anwender möglichst nur Informationen über das Thema anzubieten, mit dem er sich augenblicklich beschäftigt (vergleiche [RBL]).

Vor diesem Hintergrund ist es wünschenswert, verschiedene Verweisschemata auf einen bestimmten Inhalt anzuwenden. Ein Beispiel soll diese Idee illustrieren:

Beispiel 3.2.1
Ein Student der Psychologie liest einen deutschsprachigen Text über die neuen Konzepte von eLearning–Systemen. Zum besseren Verständnis wäre es für diesen wahrscheinlich hilfreich, wenn Fachbegriffe der Informatik mit einem Verweis zu einer Erläuterung hinterlegt wären. Liest ein Student der Informatik diesen Text, so wäre ihm im Gegensatz dazu am besten mit einer Erklärung der psychologischen Fachbegriffe geholfen. Einem beispielsweise englischem Studenten der ebenfalls um Umfeld der eLearing–Systeme arbeitet, aber Defizite im Verständnis der deutschen Sprache hat, würde am meisten von einem Verweis hinter jedem Wort auf seine englische Übersetzung profitieren.

Da es durchaus möglich ist, eine Ressource oder einen Teil einer Ressource in unterschiedlichen Verweisschemata zu referenzieren, könnte über diesen Mechanismus auch die zuvor angesprochene Steuerung der Anzeige von Kommentaren oder Bemerkungen Dritter realisiert werden.

Kriterium	Erläuterung
Adaptivität	Die Fähigkeit eines Systems, sich an seinen Nutzer selbständig anzupassen.
Adaptierbarkeit	Die Fähigkeit eines Systems, sich aufgrund externer Analysen und Einstellungen an die Bedürfnisse des Anwenders anzupassen.
Annotierbarkeit der Inhalte	Die Fähigkeit eines Systems, die von ihm zur Verfügung gestellten Inhalte mit Metainformationen zu versehen.
Annotierbarkeit durch den Anwender	Die Möglichkeit vom System, zur Verfügung gestellte Inhalte mit Kommentaren, Bemerkungen oder Anmerkungen seitens des Anwenders zu versehen.
Flexibles Interaktionsmodell	Die Fähigkeit eines Systems, sowohl das didaktische Modell als auch strukturierte Inhalte zu unterstützen.
Funktionsabhängige Verweisselektion	Die Fähigkeit eines Systems, Hyperreferenzen anhand ihrer Funktion auszuwählen.
Inhaltsabhängige Verweisselektion	Die Fähigkeit eines System, Hyperreferenzen anhand ihrer inhaltlichen Bedeutung auszuwählen.
Kontextbeachtung	Die Fähigkeit eines Systems, Inhalte unter Beachtung ihres Quell- bzw. Aufrufkontextes darzustellen.
Orientierbarkeit	Die Fähigkeit eines Systems, den Anwender bei der Orientierung durch geeignete Hilfsmittel zu unterstützen.
Präsentationssteuerung	Die Fähigkeit eines Systems, die Anzeige von Inhalten zu steuern.
Strukturierbarkeit	Die Fähigkeit eines Systems, strukturierte Inhalte aufzunehmen, zu verarbeiten und diese angemessen zu präsentieren, sowie Interaktionen auf diesen Strukturen zu definieren.
Unterstüzung des Autors	Die Fähigkeit eines Systems, den Autor sowohl in der Erstellung und Verknüpfung der Inhalte als auch in der Ausgestaltung der Interaktionsmöglichkeiten zu unterstützen.

Tabelle 3.1: Inhaltliche Anforderungen an eLearning–Systeme

3.3 Verweiskontext

Die Kernaussage des vorangegangen Abschnittes war die Notwendigkeit, Verweise in einer bestimmten inhaltlichen Bedeutung und Aussage durch den Autor zu erstellen, die sogenannte Rhetorik der Hyperreferenzen. Des weiteren wurde auf die Rolle des Anwenders als Akteur hingewiesen, sowie die Notwendigkeit verschiedene inhaltliche Verweiskonzepte oder Verweisschema auf den selben Inhalt anzuwenden. Ein Problem hierbei ist, daß HTML keine unterschiedlichen Verweiskonzepte auf einem Inhalt unterstützt.

Der Autor ordnet im Sinne einer inhaltstragenden, bedeutungsreichen Ausgestaltung der Hyperreferenzen diesen bereits implizit einen Schema zu. So kann beispielsweise ein Verweis zu einem weiterführenden Text als Spezialisierung, ein Verweis zum einfachen Weiterblättern als Navigationsfunktion kategorisiert werden. Es fehlt aber eine generelle Abstraktionsschicht, die die Selektion von Verweisen anhand bestimmter Eigenschaften vornimmt, bzw. eine Ausdruckmöglichkeit für ein gewünschtes Schema ermöglicht.

Beispiel 3.3.1
Gegeben sei eine Seite, die einen fremdsprachigen Text enthält. Jedes Wort dieser Seite ist mit einem Verweis auf seine deutsche Übersetzung hinterlegt.

Das Verweisschema im Beispiel 3.3.1 könnte beispielsweise „Wörterbuch“ heißen und wäre eine mögliche Sicht auf diesen Inhalt. Obwohl jeder Verweis ein unterschiedliches Verweisziel besitzt, haben sie alle die gleiche Bedeutung, gehören zu einem *Kontext*. Für dieses Beispiel sind weitere Kontexte denkbar, wie etwa das Verbinden bestimmter Begriffe mit einer näheren Erläuterung oder einer graphischen Darstellung.

Im folgenden soll als Bezeichnung für eine solche Sammlung an Verweisen, die im selben Kontext zueinander stehen, der Begriff →*Verweiskontext* eingeführt werden.

Definition 3.3.1 (Verweiskontext, Link Context)
Der Verweiskontext bildet eine Abstraktionsschicht[1], die die Selektion von Verweisen für bestimmte Inhalte, anhand bestimmter wohldefinierter Eigenschaften erlaubt.

Mit Hilfe des Verweiskontextes ist es möglich, die grundlegenden Forderungen nach einem für den Nutzer transparenten, inhaltlich nachvollziehbaren Verweismodell und der adaptiven Anpassung des Systems durch den Anwender zu erfüllen. Verschiedene Verweise können klassifiziert und zu Verweiskontexten, anhand einer abstrakten Beschreibung, zusammengefaßt werden. Der Anwender hat zur Laufzeit die Möglichkeit, verschiedene Sichten auf die Inhalte zu erhalten, indem er den aktuellen Verweiskontext ändert.

[1] siehe MIRaCLE Schichtenmodell Abbildung 4.3

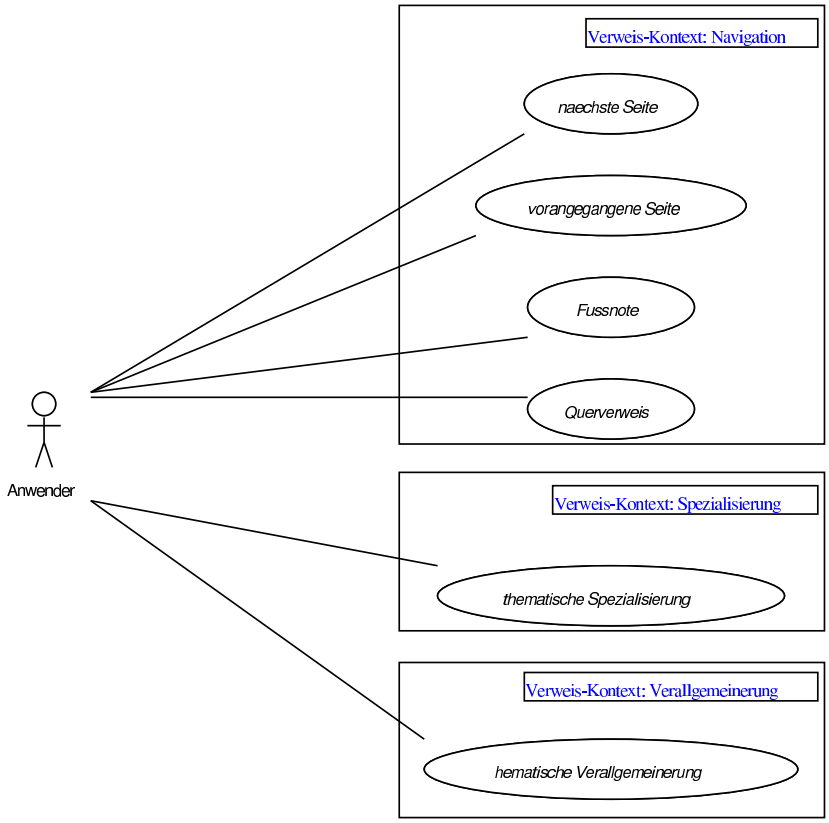

Abbildung 3.4: Beispiel für Verweiskontexte

Abbildung 3.4 zeigt eine mögliche Aufteilung in verschiedene Kontexte, ausgehend von den in Abbildung 3.3 beispielhaft genannten Anwendungsfällen für Hyperreferenzen.

Der Verweiskontext soll sowohl eine statische Definition von beteiligten Verweisen, beispielsweise in Form einer enumerativen Aufzählung, als auch eine dynamische Selektion von Verweisen anhand bestimmter Kriterien erlauben. Solche Kriterien können unter anderem Symbolnamen oder Metainformationen sein, die mit den Verweisen gespeichert werden.

Da der Anwender den Verweiskontext zur Laufzeit jederzeit ändern kann, können weder Anker noch die Verweise selbst Bestandteil des Inhaltes sein. Diese und weitere technische Anforderungen werden im folgenden Abschnitt detailliert erläutert.

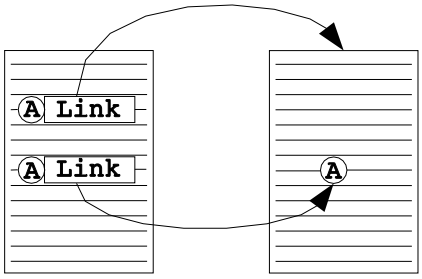

Abbildung 3.5: HTML Verweismodell

3.4 Technische Anforderungen an Hyperreferenz Systeme

In den vorangegangen Abschnitten wurden inhaltliche Anforderungen aufgeführt, ohne dabei auf technische Details einzugehen. Einen kleinen Überblick der technischen Anforderungen bietet die Tabelle 3.2 am Ende dieses Abschnittes.

Im folgenden wird nicht auf jedes Kriterium im einzelnen eingegangen, sondern nur auf jene die durch die inhaltliche Diskussion motiviert wurden.

3.4.1 Ankerfunktionalität

Im Allgemeinen wird eine Hyperreferenz als Verknüpfung von Start–Ressourcen mit End–Ressourcen betrachtet. Die Verknüpfung wird Verweis oder *Link* genannt. Dexter [HS94] erkennt, daß zur Adressierung von Substrukturen ein weiteres Element, der sogenannte Anker notwendig ist. Es existieren eine Reihe von unterschiedlichen Ansätzen, solche Hyperreferenzen zu implementieren. Das Verweismodell von HTML beispielsweise bettet Anker- und Verweisinformationen direkt in den Inhalt der Ressourcen ein (vergleiche Abbildung 3.5).

Das Problem der Verweisimplementierung wie sie HTML benutzt, ist die vermischte Speicherung von Inhalt, Verweisen und Ankern. Diese vermischte Speicherung verlangt, daß der Autor Schreibrechte auf die entsprechende Ressource besitzt, falls mit Ankern innerhalb des Dokumentes gearbeitet wird.

Wie bereits zuvor aufgeführt demonstriert auch dieses Beispiel, daß weder Anker noch Verweise direkte Bestandteile des Inhaltes sein können, sondern als eigene Entitäten im Storage Layer existieren müssen. Somit können diese dann auch von den Zugriffsrechten der Ressourcen getrennt werden, d.h. es können Ressourcen über Anker referenziert werden, ohne dabei Schreibzugriff auf den Inhalt der Ressource zu besitzen. Ein kleines Beispiel soll dieses Problem von Ankern, die mit dem Inhalt der Ressource gespeichert werden, illustrieren:

Beispiel 3.4.1

a) *Der besitzt Schreibzugriff auf eine lokale Seite mit dem Namen „apache-konfiguration.html“. Von dieser Seite soll ein Verweis auf die Index–Seite des Apache Projektes zu finden unter „http://www.apache.org/index.html“ erstellt werden. Dies ist problemlos möglich, da Schreibzugriff auf die lokale Seite besteht, in die der Verweis eingetragen wird und nicht auf Subdokumentenebene addressiert wird.*

b) *In der oben genannten lokalen Seite soll zudem ein Verweis auf einen bestimmten Paragraphen innerhalb der Indexseite des Apache Projektes gesetzt werden. Da kein Schreibzugriff auf diese Seite besteht, und der Autor der Indexseite des Apacheprojektes leider diesen Paragraphen nicht mit einer addressierbaren Entität versehen hat, ist dieses Vorhaben leider unmöglich.*

Zur eindeutigen Unterscheidung sollen im folgenden die Begriffe →*Inline Anker* und →*dekorierender Anker* definiert werden:

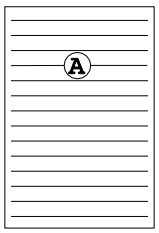

Abbildung 3.6: Inline Anker

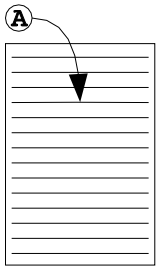

Abbildung 3.7: Dekorierender Anker

Definition 3.4.1 (Inline Anker)
Ein Anker wird als *inline* bezeichnet, wenn der Ankerpunkt direkt im Inhalt der Ressource gesetzt und Anker sowie Inhalt zusammen in einer Ressource gespeichert werden (siehe Abbildung 3.6).

Definition 3.4.2 (Dekorierende Anker)
Ein Anker wird als *dekorierend* bezeichnet, wenn der Anker außerhalb der Ressource und deren Inhalt gespeichert wird. Ein dekorierende Anker referenziert Daten in der mit ihm verknüpften Ressource (siehe Abbildung 3.7).

Durch die Verwendung dekorierender an Stelle von Inline Ankern, ist es auch möglich, unterschiedliche Sichten im Sinne eines Verweiskontextes auf den Inhalt von Ressourcen zu projezieren. Außerdem können nun auch Dritte verschiedene Anker auf denselben Inhalt, wie beispielsweise eine Graphik oder eine Textstelle, definieren. Dieser kann dann Quelle oder Ziel einer Verknüpfung zwischen einer Bemerkung oder einem Kommentar werden.

3.4.2 Verweisfunktionalitäten

Die Verwendung des klassischen HTML Verweises ist aus verschiedenen inhaltlichen und technischen Gründen für fortgeschrittene Hypermedia–Systeme problematisch. So schließt die zuvor besprochene Notwendigkeit, Anker als eigene Entitäten im Storage Layer zu speichern, die Verwendung von HTML verweisen aus, da diese Inline Anker verwenden. Die Komponenten einer HTML Hyperreferenz sind demnach im Storage Layer unbekannt. Darüber hinaus, wurde in den inhaltlichen Anforderungen auf die Bedeutung von → bidirektionalen und → multidirektionalen Verweisen hingewiesen.

Definition 3.4.3 (unidirektionaler Verweis)
Beschreibt ein Verweis die Traversierung in Richtung von genau einer Start–Ressource zu genau einer End–Ressource, wird dieser als unidirektional bezeichnet.

Definition 3.4.4 (bidirektionaler Verweis)
Beschreibt ein Verweis sowohl die Traversierung in Richtung von genau einer Start–Ressource zu genau einer End–Ressource als auch von genau dieser End–Ressource zu genau dieser Start–Ressource, wird dieser als bidirektional bezeichnet.

Definition 3.4.5 (multidirektionaler Verweis)
Ist zumindest eine der an einem Verweis beteiligten Ressourcen mehrwertig, so wird dieser Verweis als multidirektional bezeichnet.

Der XLink Standard [10, 11] des W3C erlaubt, wie in Abschnitt 2.4.3.2 beschrieben, die Definition sowohl von unidirektionalen, bidirektionalen als auch multidirektionalen Verweisen unter Verwendung von dekorierenden Ankern. Somit sollte eine eLearning–Anwendung zumindest die Funktionalitäten der XLink Spezifikation implementieren, um die beschriebenen inhaltlichen und technischen Anforderungen zu erfüllen.

3.4.3 Annotierbarkeit von Ankern und Verweisen

Die Annotierung von Daten, im Sinne einer Auszeichnung mit Metainformationen, eröffnet, wie schon zuvor bemerkt, eine automatisierte, maschinelle Verarbeitung. Eine entsprechend reichhaltige Menge an solchen Metainformationen kann es ermöglichen, Zusammenhänge und Beziehungen zwischen einzelnen Ressourcen automatisiert herzustellen.

Diese Möglichkeit ist nicht nur hinsichtlich der Inhalte, sondern auch für die interaktiven Komponenten einer Hyperreferenz, also Anker und Verweise, interessant. Denkbar wären Szenarien, in denen ein Verweis anhand bestimmter Kriterien die an ihm beteiligten Anker auswählt und somit laufzeit–dynamisch Verknüpfungen verschiedener Inhalte ermöglicht. Verweise wiederum könnten in ihren Annotierungen beispielsweise semantische Beziehungen kodieren, die

ihrerseits von der Laufzeitumgebung ausgewertet und dem Anwender präsentiert werden. Eine weitere Möglichkeit besteht in der automatisierten Erzeugung von RDF Beschreibungen.

Kriterium	Erläuterung
Annotationen	Die Möglichkeit eines Systems, Linkverknüpfungen durch Dritte auf Inhalte zu erzeugen, an denen keine Autorenrechte bestehen.
Annotierbarkeit der Referenzen	Die Möglichkeit eines Systems, Anker und Verweise mit Metainformationen zu versehen.
Bidirektionale Referenzen	Die Fähigkeit eines Systems, neben einseitigen Verknüpfungen (von – nach) auch symetrische Verknüpfungen (zwischen) verarbeiten zu können.
Konditionelle Referenzen	Die Fähigkeit eines Systems, die Gültigkeit von Referenzen an semantische oder Laufzeitkonditionen zu koppeln.
Mehrwertige Referenzen	Die Fähigkeit eines Systems, mehrwertige Linkverknüpfungen zu erzeugen und zu verarbeiten.
Referenzerzeugung durch automatische Relationserzeugung	Die Fähigkeit eines Systems, Verweise etwa durch eine dynamische Inhaltsanalyse selbständig zu erzeugen.
Referenzerzeugung durch manuelles Authoring	Die Eigenschaften eines System, den Autor bei der manuellen Verweisverknüpfung zu unterstützen (eine Deklaration für viele Verknüpfungen).
Referenzkonsistenz	Die Fähigkeit eines Systems, Verweisintegrität selbstständig zu bewahren.
Referenzkontexte	Die Fähigkeit eines Systems, Verweiszusammenhänge zu verarbeiten, darzustellen und anzusteuern.
Referenzprozessierung	Die Fähigkeit eines Systems, Verweise in verschiedenartiger, parametrisierbarer Weise zu verarbeiten.
Verarbeitung impliziter Verweise	Die Fähigkeit eines Systems, neben explizit definierten Verweisen auch implizite Referenzen zu verarbeiten.
Zeitbasierte Referenzen	Die Fähigkeit eines Systems, Verweisverknüpfungen zeitbasiert zu erzeugen und zu verarbeiten.

Tabelle 3.2: Anforderungen an Hyperreferenzsysteme (vergleiche [22])

Kapitel 4

Konzeption

Die vorangegangen Kapitel gaben einen Überblick über die Fähigkeiten einiger zur Realisierung von eLearning–Anwendungen geeigneter Technologien , als auch über die inhaltlichen und technischen Anforderungen an solche Applikationen. Anhand der Betrachtungen der Rolle des Autors und die Anforderungen seitens des Anwenders, kristallisierte sich die Forderung nach einer Möglichkeit heraus, mehrere unterschiedliche Verweisschemata auf den selben Inhalt anzuwenden. Die Definition einer neuen Abstraktionsebene, dem sogenannten Verweiskontext, soll die „high–level“ Selektion von Verweisen anhand bestimmter Kriterien erlauben.

Das zur Umsetzung von Anwendungen die Verweiskontexte nutzen, benötigte Konzept soll in diesem Kapitel erstellt werden. Hierzu wird zunächst der Ist–Zustand beschrieben und mit dem Soll–Zustand verglichen, um dann eine Aufgabendefinition für diese Arbeit zu schaffen. Anschließend wird ein Grobkonzept entwickelt, das die generelle Funktionsweise des Systems erklärt und aus diesem das Feinkonzept abgeleitet, das Grundlage der im nächsten Kapitel vorgestellten Implementierung ist.

4.1 Anforderungsanalyse

Ziel der Anforderungsanalyse ist es, aus dem Vergleich zwischen Soll- und Ist–Zustand die Anforderungen für das zu implementierende System herauszuarbeiten.

4.1.1 Ist–Zustand

Das MIR–System erlaubt die Speicherung beliebiger Daten verbunden mit frei definierbaren Eigenschaften, die zur Erzeugung von Metainformationen genutzt werden können. Die vorhandenen Schnittstellen erlauben einen netzwerktransparenten Zugriff auf die gespeicherten Daten

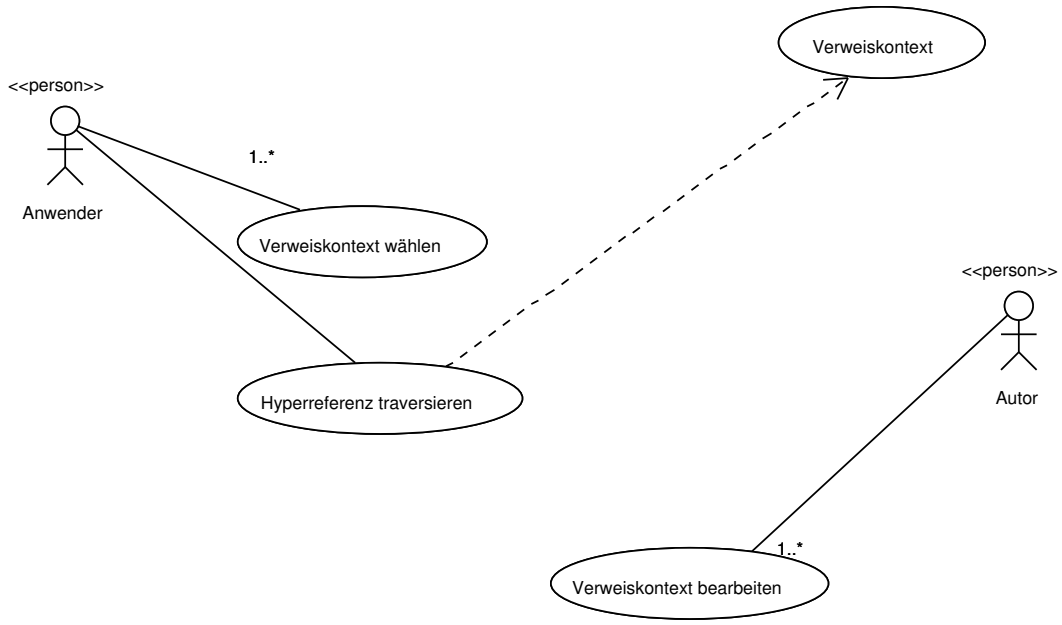

Abbildung 4.1: Anwendungsfälle

einschließlich der Möglichkeiten, im Datenbestand oder den mit den Daten verknüpften Metainformationen zu suchen.

Im Rahmen des Projektes „FHTW.Web “[23] bestehen bereits Erfahrungen und Tools zur Realisierung von Webapplikationen auf Basis des MIR–Systems. Insbesondere existieren bereits ein *DTD* (Document Type Definition)[24, 25] zur Strukturierung und ein Editor[26] zur komfortablen Erstellung und Bearbeitung von Inhalten. Hinsichtlich der Publikation der Inhalte hat sich in diesem Projekt bereits eine Kombination aus dem Apache Tomcat[27] und dem Cocoon Publishing Framework[28] bewährt, um die Transformation von XML[8] unter Verwendung von XSLT (Extensible Stylesheet Language Transformation)[29, 30] nach HTML und somit die Anzeige in einem gewöhnlichen Webbrowser zu ermöglichen.

Das MIR–System unterstützt zur Zeit kein Verweismodell, abgesehen von der Möglichkeit, Informationstrukturen durch Datenverknüpfungen (siehe Abschnitt 2.6 Seite 31) aufzubauen. Ebenso werden bidirektionale oder multidirektionale Verweise von Webbrowsern derzeit nicht unterstützt. Auch wenn die aktuellen Versionen der Browser teilweise bereits die Transformation von XML Inhalten mittels XSLT bereitstellen, werden die über einen einfachen HTML Link hinausgehenden Fähigkeiten des XLink[10, 11] Standards nicht unterstützt (vergleiche [31]).

4.1.2 Soll–Zustand

Das zu implementierende System soll die Anwendung verschiedener durch den Verweiskontext bestimmte Verweise auf dem selben Inhalt ermöglichen. Die Verweiskontexte sollen durch den Anwender frei wählbar sein. Die Abbildung 4.1 zeigt die typischen Anwendungsfälle der Applikation.

Um die Implementierung einer neuen Präsentations- und Laufzeitumgebung zu vermeiden, werden dem Nutzer die Inhalte in einem herkömmlichen HTML Browser präsentiert. Um den Informationsgehalt möglichst ausdrucksreich zu halten, wird erst im Rahmen der Auslieferung der Daten an den Browser eine Transformation der XML Daten nach HTML vorgenommen. Diese Transformation der informationsreichen Sprache (XML) in die informationsärmere Sprache (HTML) betrifft auch die Verweise und Anker. Vor der Transformation nach HTML werden diese konform zum XLink[10, 11] Standard ausgedrückt, um Interoperabilität und Standardkonformität zu erhalten.

Anker und Verweise sind persistente Daten und müssen somit im Storage Layer als eigene Entitäten existieren. Jede dieser Entitäten kann mit Metainformationen annotiert werden. Neben sogenannten statischen Hyperreferenzen, deren Start–Ressourcen bzw. End–Ressourcen von vornherein festgelegt sind, existieren dynamische Hyperreferenzen deren Ressourcen erst zur Laufzeit bestimmt werden. Das Konzept der Hyperreferenz wird über die Fähigkeiten eines HTML Verweises hinaus erweitert, so daß sowohl bidirektionale als auch multidirektionale Verweise sowie weitere präsentationsspezifische Attribute unterstützt werden.

Die zu präsentierenden textuellen Inhalte werden in einer XML konformen Auszeichnungssprache gespeichert, um zum einen die Trennung von Inhalt und Layout, zum anderen die Unabhängigkeit in Bezug auf das Ausgabemedium zu gewährleisten.

Die Beschreibung der Verweiskontexte erfolgt ebenfalls in einer XML konformen Auszeichnungssprache, so daß eine automatisierte, maschinelle Verarbeitung möglich ist. Außerdem werden durch die Kontexte semantische Beziehungen ausgedrückt. Die in Kapitel 2 vorgestellten Sprachen zur Modellierung solcher Beziehungen verwenden entweder XML oder SGML, da aber auch die Verweisproblematik durch den XLink[10, 11] Standard inspiriert wurde, liegt die Verwendung einer XML basierten Auszeichnungssprache nahe. Der Verweiskontext soll so gestaltet werden, daß sowohl dem Autor als auch dem Anwender „high–level“ Mechanismen zur Verweisselektion zur Verfügung stehen.

4.1.3 Aufgabendefinition

Um den Ist–Zustand in den Soll–Zustand zu überführen, wird zunächst ein Konzept entwickelt, das dann die Basis der Implementierung darstellt. Hierbei werden im wesentlichen zwei Ziele verfolgt: Zum einen die Implementierung eines Verweismodells im MIR–System, zum anderen die Umsetzung der Idee des Verweiskontextes auf Basis des MIR–Systems und seinem Verweismodell.

Aufgrund der Beschreibung des Soll–Zustandes lassen sich die folgenden Anforderungen ableiten:

1. Allgemeine Anforderungen:
   - Konzept soll unabhängig von der spezifischen Implementierung des Storage Layers sein.
   - Als Anwenderschnittstelle soll ein herkömmlicher Webbrowser agieren.
2. Anforderungen an das Verweismodell:
   - Anker und Verweise müssen als eigene Entitäten im Storage Layer existieren,
   - Anker und Verweise sollen frei mit Metainformationen annotierbar sein,
   - Unterstützung von Verweisen, die ihre Ressourcen zur Laufzeit bestimmen,
   - Unterstützung von präsentationsspezifischen Attributen.
3. Anforderungen an den Verweiskontext:
   - In einer XML konformen Auszeichnungssprache definierbar,
   - enumerativ aufzählbar,
   - dynamische Verweisselektion anhand bestimmter Kriterien,
   - Verweisselektion erfolgt mit hohem Abstraktionsgrad.

4.2 Grobkonzept

Die folgenden beiden Abschnitte sollen zunächst ein allgemeines Verständnis des zu konzipierenden Systems ermöglichen. Basis der Implementierung wird das zuvor vorgestellte MIR–System (siehe Abschnitt 2.6) sein. Die Erweiterung des MIR–Systems um die Fähigkeiten des adaptiven, kontextsensitiven Linkings wird im folgenden auch als *MIRaCLE* (Media Information Repository adaptive Context Linking Environment) bezeichnet. Es wird aber der Versuch unternommen, den Systementwurf durch die Verwendung geeigneter Mittel so abstrakt wie möglich

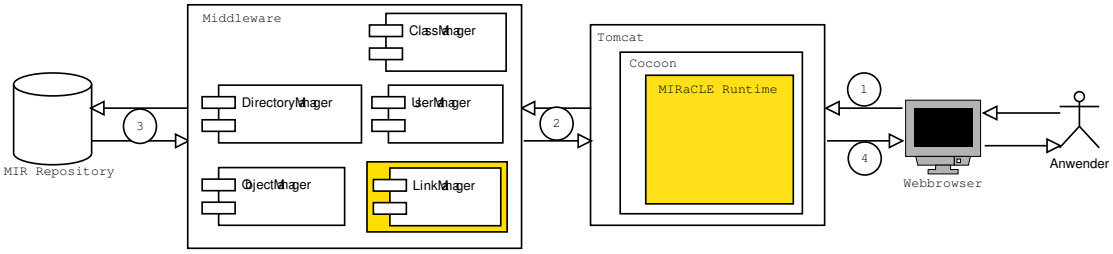

Abbildung 4.2: Allgemeiner Systemüberblick

zu halten, um auch eine Implementierung auf Basis anderer Systeme zu ermöglichen. Einen kleinen Ausblick darauf bietet Abschnitt 5.5.

Das Konzept ist prinzipiell unabhängig vom MIR–System, da für alle das Storage Layer betreffenden Funktionen, Schnittstellen definiert werden. Jede Anwendung, die diese Schnittstellen implementiert, kann demnach dieses Konzept nutzen. Außerdem werden allgemeingültige Standards, wie beispielsweise XLink[10, 11] angewendet, soweit diese sinnvoll einsetzbar sind. Auch das MIR–System benutzt allgemeingültige Standards, so daß sowohl einzelne Bestandteile, als auch das gesamte MIR ausgetauscht werden kann. Dies alles unterstreicht die grundsätzliche Unabhängigkeit dieses Konzeptes vom MIR–System, das nur eine mögliche, sehr kompfortabele Implementierung des Storage Layers darstellt.

Das → MIRaCLE Schichtenmodell (Abbildung 4.3) erlaubt die Zuordnung von Daten (Inhaltsdaten, Anker, Verweise etc.) zu bestimmten Schichten, auf deren Grundlage dann die notwendigen persistenten Daten und Schnittstellen entworfen werden können.

4.2.1 Allgemeiner Systemablauf

Der Systemablauf untergliedert sich in vier Stufen, die nacheinander durchlaufen werden müssen, da jede einzelne Stufe nur seine unmittelbaren Nachbarn kennt. Da die innere Struktur der beteiligten Elemente anonym und nur deren Schnittstellen bekannt sind, können diese vollkommen transparent ausgetauscht werden, solange Schnittstelle und Funktionalität gleich bleibt.

Die grobe Funktionsweise des Systems läßt sich am besten ausgehend vom Anwender erläutern (vergleiche Abbildung 4.2). Der Benutzer betrachtet ein diesem System entsprungenes HTML Dokument im Webbrowser. Durch das Anklicken eines Verweises aktiviert er das Traversieren einer Hyperreferenz. Vom Webbrowser wird ein *HTTP Request* ① an den Tomcat, der hier in der Rolle eines Webservers agiert, abgesetzt. Dieser wertet die Anfrage aus und weist sie der entsprechenden Webapplikation (MIRaCLE Runtime) zu. Die Webapplikation bestimmt aufgrund der *HTTP Session* Information den gültigen Verweiskontext, mit Hilfe dessen die gültigen Hyperreferenzen im angefragten Dokument bestimmt werden. Hierzu werden Funktionen der Middle-

ware verwendet. Die Kommunikation zwischen Middleware und Webapplikation ② kann über CORBA erfolgen. Die benötigten Daten werden über Methodenaufrufe der Middleware, letztere verwendet ihrerseits JDBC ③, um auf die Datenbank (MIR–Repository) zuzugreifen, geladen. Durch das Zusammenspiel von Webapplikation, Middleware und Datenbank wird ein vollständiges XML Dokument erstellt, wobei die Hyperreferenzen in Abhängigkeit vom Verweiskontext mit dem Inhalt des Dokumentes verknüpft werden. Bevor dieses als *HTTP Response* an den Webbrowser ④ zurückgesandt wird, erfolgt noch eine Transformation nach HTML unter Verwendung eines *extensible Stylesheets*.

4.2.2 Das MIRaCLE Schichtenmodell

Das MIRaCLE Schichtenmodell (siehe Abbildung 4.3) besteht aus vier Abstraktionsebenen mit der Zielstellung, die unterschiedlichen Prozesse zu gliedern und bestimmten Schichten zuzuordnen. Jede dieser Schichten besitzt persistente Daten und kommuniziert nur mit seinen unmittelbaren Nachbarn. Die folgende Beschreibung der einzelnen Schichten des Modells soll nur einen Überblick bieten; die Bestandteile der einzelnen Schichten werden im Feinkonzept vertiefend behandelt.

Verweiskontextschicht: Der Verweiskontext erlaubt die Selektion ausgezeichneter Verweise für bestimmte Inhalte anhand wohldefinierter Eigenschaften. Diese Eigenschaften können sowohl statisch, wie beispielsweise eine enumerative Aufzählung der beteiligten Verweise, als auch dynamisch sein. Dynamische Definitionen werden zur Laufzeit ausgewertet. Anhand dieser Auswertung wird dann eine Menge von passenden Verweisen selektiert.

Die Verweiskontextschicht enthält die verschiedenen Verweiskontexte. Sie stellt Funktionen zur Kommunikation mit der →Verweisschicht bereit.

Verweisschicht: Die Verweisschicht nutzt die in ihr gespeicherten Verweise, um Verknüpfungen zwischen Ankern herzustellen. Damit eine vollständige Hyperreferenz dargestellt werden kann, muß ein Verweis zumindest zwei Anker, die Start- und die End–Ressource, auswählen, wobei die Anzahl der an einem Verweis beteiligten Anker konzeptionell nicht begrenzt ist. Analog zur Verweiskontextschicht können auch bei der Selektion von Ankern statische und dynamische Prozesse unterschieden werden. Ein statischer Verweis zeichnet sich durch eine bereits vor der Laufzeit bestehende, feste Verknüpfung mit konkreten Ankern. Im Gegensatz dazu ermitteln dynamische Verweise ihre Ankerkomponenten erst zur Laufzeit.

Die Verweise besitzen Schnittstellen zum Datenaustausch mit Ankern.

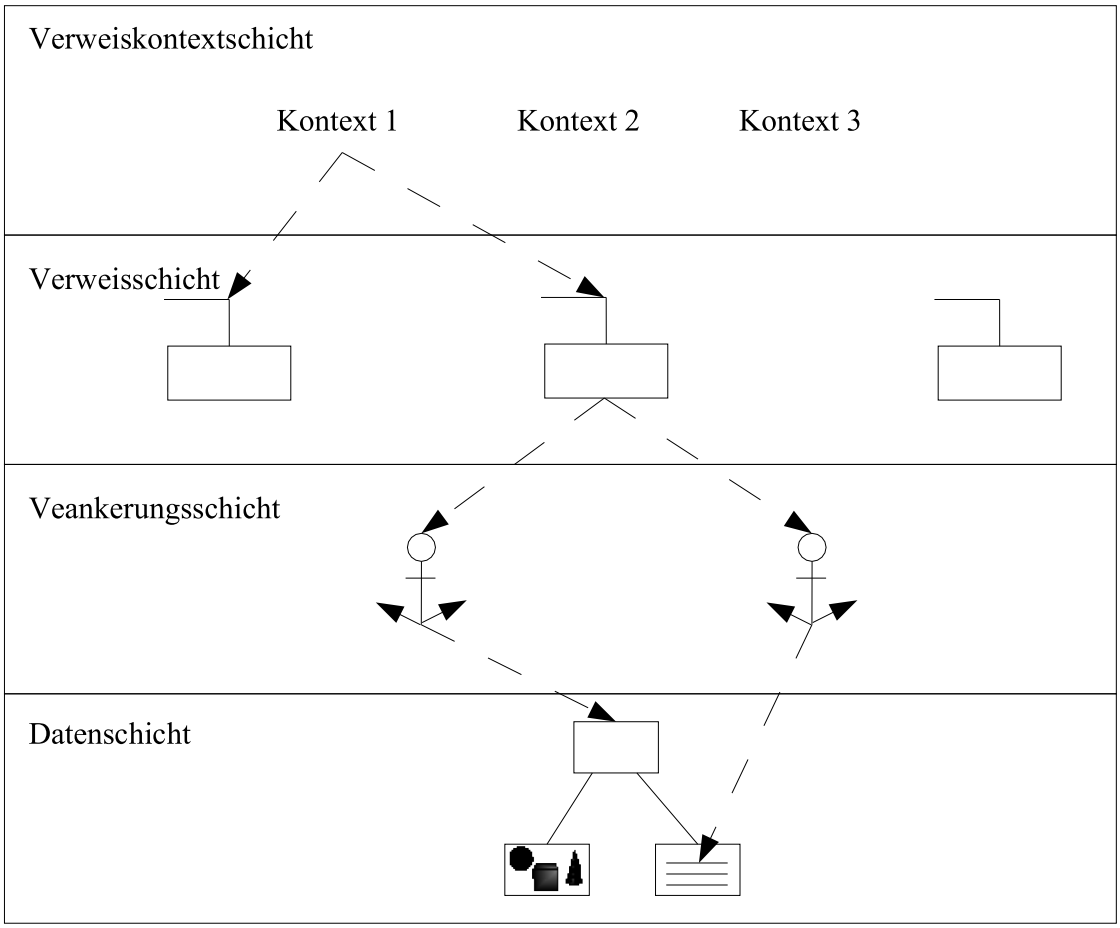

Abbildung 4.3: Das MIRACLE Schichtenmodell

Verankerungsschicht: Die Aufgabe von Ankern ist, wie in Kapitel 3 gezeigt, in der Regel die Subadressierung von Daten. Auf Grundlage dieses Modells sollen zwei Arten von Ankern, statische und dynamische, unterschieden werden. Die von statischen Ankern zu markierenden Inhalte sind bereits bekannt und müssen somit nicht gesondert ermittelt werden. Im Gegensatz dazu bieten dynamische Anker die Möglichkeit anhand der ihnen übergebenen Parameter, die Markierung entsprechender Daten erst zur Laufzeit vorzunehmen. Dies bedeutet, daß diese Art der Anker zuvor zur Laufzeit durch das System verarbeitet werden müssen.

Anker opererieren zudem direkt auf der Datenschicht und erlauben die so markierten Daten an die nächst höhere Schicht zur Weiterverarbeitung durchzureichen.

Datenschicht: Innerhalb dieser Ebene werden die Daten sowie die damit verknüpften Meta-Informationen gespeichert. Hier sind die Methoden zum Zugriff (Input/Output Operatio-

nen und Suchfunktionalitäten) auf die eigentlichen Daten sowie deren Meta-Informationen implementiert.

Das MIRaCLE Schichtenmodell verfügt somit über vier zentrale Bestandteile, die Daten, die Anker, die Verweise und die Verweiskontexte.

4.3 Feinkonzept

Auf Basis der Anforderungsanalyse und des MIRaCLE Schichtenmodells kann nun detailliert auf die einzelnen Elemente und ihre Schnittstellen zu benachbarten Schichten eingegangen werden. Hierbei wird das Modell von der logisch höchsten Schicht, der Verweiskontextschicht, nach unten durchlaufen, wobei auf die Betrachtung der Datenschicht verzichtet wird, da diese durch das MIR–System vorgegeben und nicht Bestandteil dieser Arbeit ist.

Wichtig für das Grundverständnis dieses Konzeptes ist die Erkenntnis, daß alle Bestandteile (Anker, Verweise und Verweiskontexte) in Form von RDF Aussagen[13] (siehe Abschnitt 2.4.4.1) beschreibbar sind. Um Aussagen über Anker bzw. Verweise treffen zu können, wird folgendes Transformationsschema verwendet:

- Der Name des Ankers bzw. Verweises ist das *Subjekt*,
- der Metadeskriptor wird zum *Prädikat*, im Sinne von „hat die Eigenschaft . . . “,
- der Wert des Metadeskriptors wird zum *Objekt*.

So lassen sich prinzipiell für jeden Anker bzw. Verweise zumindest Aussagen der Form *„Anker A bzw. Verweis A hat die Eigenschaft Name=Wert“* formulieren. Hinsichtlich der Eigenschaft von Verweisen, Anker in Form von Start–Ressourcen bzw. End–Ressourcen zu verknüpfen, wird die Fähigkeit von RDF, Aussagen über Aussagen zu machen, genutzt. Ein Verweis kann somit durch drei RDF Statements beschrieben werden. Je ein Statement repräsentiert die Eigenschaften der Start–Ressource bzw. der End–Ressource, und ein Statement stellt die Verknüpfung zwischen Start–Resource und End–Ressource sowie die konkrete Ausgestaltung der Hyperreferenz dar. Die Beschreibung der Verknüpfung trifft hierbei die genannte Aussage über eine Aussage, nämlich „Anker A ist Startressource des Verweises, wobei Anker A die Eigenschaft . . . besitzt“.

Auf die Verwendung von RDF zur Formulierung von → Verweiskontexten wird im folgenden Abschnitt gesondert eingegangen, da diese eine zentrale Rolle in diesem Konzept einnehmen.

Die Ausgestaltung der tatsächlichen Implementierung muß auf Anker- bzw. Verweisebene (siehe Abbildung 4.3) nicht zwangsläufig mit RDF Aussagen arbeiten, sondern kann eine interne Repräsentation wählen.

4.3.1 Verweiskontext

Der Verweiskontext erlaubt die Selektion von Verweisen anhand bestimmter Kriterien zur Laufzeit. Die Definition des Verweiskontextes erfolgt durch den Autor, die Auswahl des anzuwendenden Verweiskontextes durch den Anwender (vergleiche Abbildung 4.1).

Der Verweiskontext soll durch die Verwendung einer XML konformen Auszeichnungssprache definiert werden. Da er semantische Beziehungen ausdrückt, ist der Rückgriff auf die Standards des „semantischen Webs“ angebracht. Insbesondere erscheint hier die Verwendung des *Resource Description Frameworks (RDF)* sinnvoll. RDF ist zum einen eine XML konforme Auszeichnungssprache und wurde zum anderen explizit zur Beschreibung von Ressourcen und Zusammenhängen zwischen Ressourcen entwickelt.

Alle Verweise, auf denen operiert wird, sind zur Laufzeit bekannt. Der Verweiskontext erzeugt demnach keine neuen Verweise oder Anker, da er gemäß dem MIRaCLE Schichtenmodell (siehe Abbildung 4.3) keinen direkten Zugriff auf die Datenschicht besitzt.

Die Definition eines Verweiskontextes gliedert sich ein zwei Teile: Die deskriptive Beschreibung und die Auszeichnung der partizipierenden Verweise.

Zur deskriptiven Beschreibung des Kontextes wird das *Dublin Core* (DC) Set verwendet. Um den dargestellten inhaltlich–deskriptiv Kontext zu beschreiben, sollten zumindest die folgenden Angaben gemacht werden:

DC Tag	Bedeutung	Beispiel
`<dc:creator>`	Autor des Kontextes	`<dc:creator> Al Capone </dc:creator>`
`<dc:description>`	Kurze Erklärung des Kontextes	`<dc:description> Alles rund um Kriminalität </dc:description>`
`<dc:title>`	Titel des Kontextes	`<dc:title> Kriminalität </dc:title>`

Tabelle 4.1: Deskriptive Minimalanforderung zur Beschreibung des Inhaltes eines Verweiskontextes

Neben diesen inhaltlichen Deskriptioren muß das Selektionsschema für die Verweise beschrieben werden. Gemäß der Anforderungsdefinition soll sowohl eine enumerative Aufzählung, als auch eine dynamische Selektion von Verweisen möglich sein. Die Beschreibung des Selektionsschemas erfolgt in Form von RDF[13]. Das Tag `<base-set>` bestimmt die Menge aller

Verweise (*Grundmenge*), die für diesen Kontext definiert sind. Der folgende Quellcode zeigt das Grundgerüst für einen Verweiskontext. Alle mit dem Verweiskontext zusammenhängenden Tags werden dem Namensraum `miracle` zugordnet:

```
1  <rdf:RDF
2    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3    xmlns:dc="http://purl.org/metadata/dublin_core#"
4    xmlns:miracle="http://www.rz.fhtw-berlin.de/MIR/miracle#">
5    <rdf:Description about="Verweiskontext1">
6      <!-- Grundmenge -->
7      <miracle:base-set>...</miracle:base-set>
8
9      <!-- Verweisselektion -->
10     <miracle:set>...</miracle:set>
11
12     <!-- Deskriptoren -->
13     <dc:creator>...</dc:creator>
14     <dc:title>...</dc:title>
15     <dc:description>...</dc:description>
16   </rdf:Description>
17 </rdf:RDF>
```

Listing 4.1: Grundgerüst eines Verweiskontextes

Gemäß der RDF Spezifikation (siehe Abschnitt 2.4.4.1) muß dem Prädikat `<miracle:set>` ein Objekt folgen. Dieses Objekt kennzeichnet die *Menge (Set)* der durch den Verweiskontext selektierten Verweise. Für die Definition eines solchen Sets stehen vier Varianten zur Verfügung (vergleiche Listing 4.2):

(a) Es wird die Adresse eines Verweises als `rdf:resource` übergeben.

(b) Es wird eine bereits definierte Menge an Verweisen referenziert. Die Referenz wird ebenfalls an `rdf:resource` übergeben.

(c) In Literal der Form *Eigenschaft=Wert* bestimmt die Menge alle Verweise.

(d) Es wird der Aufzählungstyp `<rdf:Bag>` verwendet, dessen Listenelemente `<rdf:li>` wiederum des Typs (a), (b) oder (c) entsprechen.

```
1  ...
2  <!-- Fall (a) -->
3  <rdf:Description ID="defined_set">
4    <miracle:set rdf:resource="/links/verweis1"/>
5  </rdf:Description>
6
7  <!-- Fall (b) -->
8  <rdf:Description ID="another_set">
9    <miracel:set>label=Maya</miracle:set>
10 </rdf:Description>
11
12 <!-- Fall (c) -->
13 <rdf:Description ID="yas">
14   <miracle:set rdf:resource="#defined_set"/>
15 </rdf:Description>
16
17 <!-- Fall (d) -->
18 <rdf:Description ID="super_set">
19   <miracle:set>
20     <rdf:Bag>
21       <rdf:li resource="/links/verweis1" />
22       <rdf:li>label=Maya</rdf:li>
23       <rdf:li resource="#defined_set"/>
24     </rdf:Bag>
25   </miracle:set>
26 </rdf:Description>
27 ...
```

Listing 4.2: Definition einer Verweismenge

Die Verwendung des `ID` Attributes ermöglicht die eindeutige Adressierung jeder Menge von Verweisen. Referenzen können aufgelöst werden, indem man die RDF Aussagen ineinander schachtelt.

Da zuvor bereits von der Selektion einer *Menge* von Verweisen gesprochen wurde, soll für die Definition der Prädikate die Mengenoperationen darstellen, von den Begriffen der Mengenlehre ausgegangen werden (siehe Tabelle 4.2). Diese Operationen sind wohldefiniert und vollständig, was im Falle von eigenen Operatoren zu beweisen wäre. Außerdem ist die Wahl selbsterklärender Namen schwierig, da Intuition immer sehr persönlich ist, so daß schnell Mißverständnisse auftreten können.

Neben der Definition der Grundmenge wird auch der Begriff der *leeren Menge* übernommen. Letztere beschreibt eine Menge, die keine Verweise enthält. Hierbei gilt analog zur klassischen Mengelehre, daß das Komplement der leeren Menge bei erklärter Grundmenge, die Grundmenge selbst ist. Ebenso gelten alle weiteren Grundgesetze der Mengenalgebra (vergleiche [BSMM99] Seite 290).

Prädikat	Bedeutung	Erklärung
union	Vereinigungsmenge	Subjekt A vereinigt mit Objekt B. Die Ergebnismenge enthält alle Verweise der Mengen A und B ($E = A \cup B$).
intersection	Schnittmenge	Subjekt A geschnitten mit Objekt B. Die Ergebnismenge enthält nur die Verweise, die in den Mengen A und B vorkommen ($E = A \cap B$).
complement	Komplementärmenge	Subjekt A ist Komplement bezüglich Objekt B. Die Menge B bezeichnet die Grundmenge M. Die Ergebnismenge enthält alle Verweise der Grundmenge B die nicht in der Menge A vorkommen ($E = \overline{A} = C_M(A)$).
weitere Operationen:		
difference	Differenzmenge	Die Differenz bezüglich des Subjektes A und des Objektes B. Die Ergebnismenge enthält alle Verweise, die in der Menge A, nicht aber in der Menge B vorkommen ($E = A \setminus B$).

Tabelle 4.2: Prädikate

Jede Mengenoperation wird als RDF Statement formuliert, wobei das Ergebnis durch die Angabe eines Identifikators als Wert des Attributes `bagID` des `<rdf:Description>` Tags eindeutig adressierbar ist. Da die Objekte dieser Aussagen wiederum Mengen sind, entspricht der gültige Wertebereich dem des Tags `<miracle:set>`.

Über diesen Mechanismus lassen sich Relationen zwischen Verweismengen, die als Subjekt bzw. Objekt auftreten, modellieren. Möchte man beispielsweise aus der Menge aller Verweise nur diejenigen auswählen, die über „die Architektur der Maya“ Auskunft geben, so bildet man den Durchschnitt aller Verknüpfungen zum Thema Maya und Architektur (vergleiche Listing 4.4).

Komplexe Beziehungen können durch die Schachtelung oder Referenzierung von Verweismengen dargestellt werden. Ein Beispiel hierfür ist die Differenzmenge, die wie folgt erklärt ist:

$$A \setminus B = \{x \mid x \in A \wedge x \notin B\}$$

Bei definierter Grundmenge, die durch die Menge aller möglichen Verweise in einem Kontext gegeben ist, läßt sich die Differenz auch unter Verwendung von Komplement und Durchschnitt darstellen:

$$A \setminus B = A \cap \overline{B} = \{x \mid x \in A \wedge x \notin B\}$$

Um die Definitionsmöglichkeiten für Verweiskontexte komfortabler und besser lesbar zu gestalten, können weitere Operatoren erklärt werden, wie dies im Falle der Differenz bereits geschehen ist.

In der inhaltlichen Problemdiskussion wurde auf die Notwendigkeit der „Einflußnahme des Rezipienten“ und damit verbunden die Adaptierbarkeit eines Systems (siehe Abschnitt 3.2.3) eingegangen. Hierfür ist es notwendig, auf Laufzeitinformationen über den Anwender zuzugreifen. Diese Funktionalität bietet die Verwendung des Tag `<miracle:user-pref>` als Prädikat einer RDF Aussage. Als Objekt dieses Statements wird die gewünschte Eigenschaft angegeben. Für die Existenz und Pflege der entsprechenden Daten ist die Laufzeitumgebung verantwortlich. Da ein solches RDF Statement ebenfalls eine Menge repräsentiert, können diese Informationen auch wieder als Bestandteil von Mengenoperationen verwendet werden.

Konditionen sind auf Ebene des Verweiskontextes zunächst einmal Schnittmengen, da die Bedingung: „Selektiere alle Verweise aus der Menge B, wenn sie sich auch in der Menge A befinden“ aus Sichtweise einer Mengenoperation genau die Schnittmenge „A geschnitten mit B“ darstellt. Benutzt man nun die Möglichkeit auf Zustandsinformationen über den Anwender zuzugreifen, lassen sich beispielsweise Verweismengen in Abhängigkeit von Nutzerverhalten erstellen. Der folgende Quellcode wählt alle Verweise zum Thema „Maya“ aus, wenn ein Verweis zu diesem Thema traversiert wurde:

```
1        ...
2    <!-- Menge aller besuchten Verweise  definieren  -->
3    <rdf:Description ID="visited_links">
4        <miracle:user-pref>visited</miracle:user-pref>
5    </rdf:Description>
6
7    <!-- Menge aller Verweise zum Thema Maya -->
8    <rdf:Description ID="maya">
9        <miracle:set>label=maya</miracle:set>
10   </rdf:Description>
11
12   <!-- Kondition: wenn ein Verweis zum Thema Maya besucht -->
13   <!-- wurde, dann alle Verweise zu diesem Thema selektieren -->
14   <rdf:Description about="#visited_links">
15       <miracle:intersection rdf:resource="#maya" />
16   </rdf:Description>
17       ...
```

Listing 4.3: Beispiel einer Kondition

Abschließend wird als Beispiel der vollständige Quellcode zur Definition eines Verweiskontextes geben. Hierbei werden zunächst zwei Verweismengen erzeugt, die dann für Mengenoperationen verwendet werden. Die Grundmenge wird als Vereinigung dieser beiden Mengen erklärt.

Der Verweiskontext selektiert alle Verweise, die Informationen über die Architektur der Maya enthalten.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#"
  xmlns:miracle="http://www.rz.fhtw-berlin.de/MIR/miracle#">

  <rdf:Description ID="maya">
    <miracle:set>label=Maya</miracle:set>
  </rdf:Description>

  <rdf:Description ID="arch">
       <miracle:set>label=Architektur</miracle:set>
  </rdf:Description>

  <rdf:Description about="#maya" bagID="maya-arch">
    <miracle:intersection rdf:resource="arch"/>
  </rdf:Description>

  <rdf:Description about="#maya" bagID="base-set">
    <miracle:union rdf:resource="arch">
  </rdf:Description>

  <rdf:Description about="Verweiskontext1">
    <miracle:base-set rdf:resource="#base-set"/>

    <miracle:set rdf:resource="#maya-arch"/>

    <dc:title>Architektur der Maya</dc:title>
    <dc:creator>Michael Engelhardt</dc:creator>
    <dc:description>Die Architektur der Maya</dc:description>
  </rdf:Description>
</rdf:RDF>
```

Listing 4.4: dynamische Kontextdefinition

4.3.2 Verweis

Verweise bilden Hyperreferenzen, indem sie Anker miteinander verknüpfen. Wie aus der Anforderungsanalyse und den vorangegangenen Diskussionen erkenntlich wurde, sollen sie eine eigene Entität im Storage Layer bilden. Zudem sollen sie mit Metainformationen annotierbar sein.

Neben den Metainformationen müssen Verweise die in Tabelle 4.3 angegebenen Daten persistent speichern. Der Entwurf der Datenstruktur der Verweise orientiert sich stark am XLink[10, 11] (vergleiche auch 2.4.3.2):

Attribute	Anzahl	Bedeutung
actuate	1	Interaktionsverhalten
bidirektional	1	Traversionsrichtung
endType	1	Selektionsart der End–Ressource (statisch oder dynamisch)
label	1..∗	Symbolische Bezeichner
from	1..∗	Start–Ressource
role	1	Rolle der Verknüpfung
show	1	Präsentationsverhalten
to	1..∗	End–Ressource
startType	1	Selektionsart der Start–Ressource (statisch oder dynamisch)

Tabelle 4.3: Persistente Daten eines Verweises

Anker werden nicht als integraler Bestandteil des Verweises gespeichert, sondern über die *from* bzw. *to* Angaben referenziert. Da die Felder, die die Start–Ressource und die End–Ressource spezifizieren, mehrere Wertangaben erlauben, können multidirektionale Verweise abgebildet werden. Ob eine Hyperreferenz bidirektional zu traversieren ist, entscheidet das Feld *bidirektional*. Zur Adressierung der an einem Verweis beteiligten Anker wird das folgende Schema verwendet:

$$\underbrace{identifier}_{\text{Anker}}\# \quad \underbrace{specifier}_{\text{Selektionsmethode}} \quad ?\underbrace{parameters}_{\text{Parameter}}$$

Durch die Angabe des *Identifiers* wird der auszuwertende Anker ausgewählt. Der *Specifier* bestimmt die durch den Anker zu benutzende Selektionsmethode. Des weiteren ist es möglich, Parameter zur Prozessierung durch den Anker zu übergeben. Die Kombination von *Identifier* und *Specifier* muß einen Anker bzw. den durch den Verweis selektierten Anker eindeutig bestimmen.

Hinsichtlich der Umsetzung der Verweisstrukturen im MIR–System bieten sich Medienobjekte an. Diese bilden eine eigene Entität im Storage Layer und können frei mit Metainformationen erweitert werden. Zudem können über Targets weitere Objekte referenziert werden, die zur Verknüpfung der Anker mit dem Verweis verwendet werden.

Die Implementierung von Verweisen durch Medienobjekte entspricht der in Abbildung 4.4 schematischen Darstellung. Jeder referenzierte Anker erscheint in der Targetliste des Medienobjektes, das den Anker implementiert, unabhängig davon, ob dieser Anker eine Start–Ressource oder eine End–Ressource darstellt.

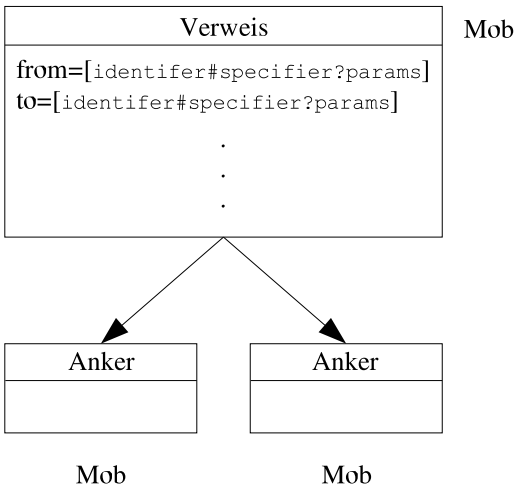

Abbildung 4.4: Verweisumsetzung auf Basis von MOBs

Da jedem Target ein bezüglich des Medienobjektes eindeutiger Symbolname, der sogenannte *Identifier*, zugeordnet wird, kann dieser als Wert von Eigenschaften verwendet werden, um den entsprechenden Target zu referenzieren. Jedes Target repräsentiert dabei einen Anker, unabhängig von seiner Funktion als Start–Ressource oder End–Ressource. Die Festlegung hinsichtlich ihrer Funktionalität als Start- oder Zielpunkt eines Verweises wird durch die Zuweisung des Identifiers des Ankers an eine *from* bzw. *to* Eigenschaft (*Property*) des verknüpfenden Verweises zugewiesen.

Die folgende Tabelle gibt einen Überblick über die im Rahmen der Implementierung dieses Konzeptes auf Basis des MIR–Systems gültigen Werte für Specifier:

Specifer	Bedeutung
property=*Name*	bezeichnet den Zugriff auf den Eigenschaftswert eines Ankers (*Beispiel: property=author*).
event=*Symbolname*	bezeichnet den Zugriff auf ein Datum mittels der Selektionslogik des durch den Symbolnamen ausgwählten Events (*Beispiel: event=textselector*).

Tabelle 4.4: Gültige Werte für den Specifier

Die Darstellung von Verweisen durch MOBs erfüllt auch die Forderung nach bidirektionalen oder multidirektionalen Verweisen, da die Zahl der Targets in einem Medienobjekt nicht begrenzt ist.

In der Anforderungsanalyse wurde die dynamische Erzeugung von Hyperreferenzen genannt. Auf Ebene der Verweise besteht die Dynamik in der Selektion der an einem Verweis beteiligten

Anker. Dieser Vorgang soll im folgenden als *dynamische Ankerselektion* bezeichnet werden. Die dynamische Ankerselektion erfolgt durch die Auswertung eines Ausdruckes zur Laufzeit. Die Entscheidung, ob eine dynamischen Selektion nötig ist, wird anhand der *startType* bzw. *endType* Eigenschaften getroffen. Liegt der Fall einer dynamischen Ankerselektion vor, wird das Adressierungsschema wie folgt erweitert:

1. Neben der statischen Adressierung eines Ankers mittels `identifer#specifier` wird die Angabe von Ausdrücken der From `"path#specifier"` erlaubt.
2. Zur Auflösung des Ausdruckes werden nur MIR Objekte unterhalb des angegebenen Pfades (*path*) untersucht. .

Im Gegensatz zu einer statischen Verknüpfung von Ankern und Verweisen, wird bei der dynamischen Ankerselektion nicht die im Verweis gespeicherte Liste aller referenzierten Anker, sondern der durch den Ausdruck *path* angegebene Gültigkeitsbereich untersucht. Diese Erweiterung ist notwendig, da es für ein Autorensystem einen enormen Aufwand bedeuten würde, für die Erstellung eines jeden Ankers, die Auswirkungen auf mögliche dynamische Selektionsprozesse zu überprüfen und die entsprechenden Anker in den Referenzen des Verweises nachzutragen.

Die Verweise müssen über Schnittstellen zum Datenaustausch mit den Ankern verfügen, um Parameter an Anker übergeben und Rückgabewerte erhalten zu können.

4.3.3 Anker

Die Aufgabe von Ankern ist, wie zuvor besprochen, die Adressierung von Daten. Um auf bereits atomar vorliegende Daten zuzugreifen, werden Anker nicht benötig. Wie aus der Anforderungsanalyse ersichtlich, sollen die Anker als eigene Entitäten im Storage Layer gespeichert werden. Folgende Daten sind dabei persistent zu halten:

Attribute	Anzahl	Bedeutung
label	1..∗	symbolischer Bezeichner
selector	1..∗	referenzierender Ausdruck

Tabelle 4.5: Persistente Daten eines Ankers

Das Storage Layer wird im MIR–System durch das MIR–Repository gebildet. Zur Datenspeicherung wird die Kapselung der Daten in ein Medienobjekt (MOB) oder ein Datenobjekt (DOB) angeboten. Medienobjekte verfügen bereits über eine Reihe von Eigenschaften, die sie zur Umsetzung der Ankerfunktionalitäten prädestinieren. Werden Anker, als separate Daten, getrennt vom Inhalt einer Ressource gespeichert, so müssen diese die Ressource referenzieren. Medienobjekte können über Targets auf weitere Medien- oder Datenobjekte verweisen. Über ihre Events

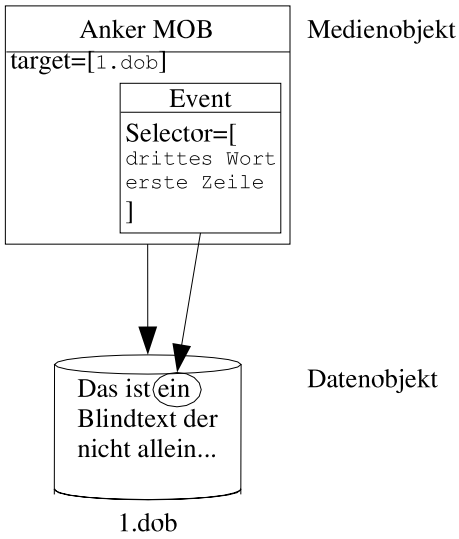

Abbildung 4.5: Ankerumsetzung auf Basis des MOB–Event Konstruktes

ist es möglich, der Laufzeitumgebung Instruktion zu erteilen, wie das entsprechende Target prozessiert wird. Die von den Ankern benötigten Informationen zur Subadressierung (*Selector*) der referenzierten Daten kann dem Medienobjekt als Event zur Prozessierung eines Targets mitgegeben werden (vergleiche Abbildung 4.5). Jeder Selektor in Form eines Events erhält einen im Medienobjekt eindeutigen Symbolnamen.

Problematisch ist die Veränderung von Daten, da hierbei referenzierte Daten geändert werden könnten. So wäre es denkbar, daß die ausgezeichnete Stelle entfernt wird. Der Anker würde damit zwar unter Umständen seine rein formale Gültigkeit behalten, jedoch werden inhaltliche Bedeutungen zerstört. Diese Tatsache ist aber kein spezielles Problem dieses Konzeptes, sondern tritt beispielsweise auch bei der Bearbeitung von HTML Seiten auf. Da dieser Effekt im wesentlichen während des Bearbeitens von Daten auftritt, soll die Lösung dieses Problems in die Autorenumgebung verlagert werden.

Alle Medienobjekte implementieren eine generische Ankerschnittstelle, so daß von der Verweisschicht eine einheitliche Schnittstelle angesprochen wird. Diese ermöglicht es, sowohl atomare, als auch durch Anker subadressierte Daten anzusprechen. Somit können auch komplette Medienobjekte selbst oder Eigenschaften (Properties) von Medienobjekten Start- bzw. Endpunkt eines Ankers sein.

Die Subadressierung von Daten durch einen Anker ist MIME–Type abhängig. Da die Beschreibung dieser Adressierung in die Events ausgelagert wurde, sind Medienobjekte als Anker zunächst MIME–Type unabhängig. Erst der entsprechende Selektionsausdruck im Selektor hängt vom MIME–Type des entsprechendem verknüpften Datums ab. So muß sich beispielsweise der Selektionsausdruck für einen Text von dem für ein Video unterscheiden.

Für Selektion in Texten[1] wird hierbei auf die Funktionalität von XPath[9] und XPointer[12] zurückgegriffen, da diese zum einem im Rahmen von XLink[10, 11] für solche Zwecke konzipiert wurden, zum anderen einfach XLink konforme Verweisdefinitionen erzeugt werden können. Selektoren für andere MIME–Types können nicht näher spezifiziert werden, da die Vielfalt der verschiedenen Formate und Codecs den Rahmen dieser Arbeit sprengen würden. Grundsätzlich ist es aber sinnvoll auf vorhandene Standards zur Datenselektion zurückzugreifen.

Ein Charakteristikum von Medienobjekten ist die freie Erweiterbarkeit um Metainformationen. Hinsichtlich der dynamischen Erzeugung von Hyperreferenzen wurde die Forderung nach der Annotierbarkeit der Anker und Verweise aufgestellt. Diese wird durch die Umsetzung der Anker als Medienobjekte erfüllt. In diesen Zusammenhang sind zwei Arten von Ankern zu unterscheiden, zum einen sogenannte statische Anker und zum anderen dynamische Anker.

Statische Anker referenzieren immer genau ein Datum. Dieses kann beispielsweise ein einzelnes Zeichen, aber auch eine Zeichenkette, ein Dokumentenfragement oder ein Polygon sein. Ein statischer Anker entspricht seinem Wesen nach der Ankerauffassung von Dexter (vergleiche [HS94]). Dynamische Anker nehmen eine dynamische Subadressierung von Inhalten vor, das heißt der Zielpunkt bzw. die Zielpunkte eines Ankers werden erst zur Laufzeit ermittelt. Allgemeiner ausgedrückt, ersetzt ein dynamischer Anker mehrere gleichartige statische Anker.

4.3.4 Statisches Modell

Das statische Modell beschreibt die Eigenschaften und Aufgaben sowie die Beziehungen zwischen Schnittstellen und Klassen einer Applikation.

4.3.4.1 Paket (Packages)

Alle hier vorgestellten Pakate befinden sich im Namensraum `de.fhtw.mir`. Insofern werden die Paketnamen im folgenden nur noch verkürzt angegeben. Anstatt von z.B. `de.fhtw.mir.miracle.core` wird nur noch von `miracle.core` gesprochen. Die Zuordnung der Klassen zu unterschiedlichen Paketen erfolgt nach funktionalen Gesichtspunkten. Durch die gestrichtelten Pfeile werden Abhängigkeiten der Pakete untereinander dargestellt.

[1] sowohl einfacher Text, als auch strukturierter Text wie XHTML etc.

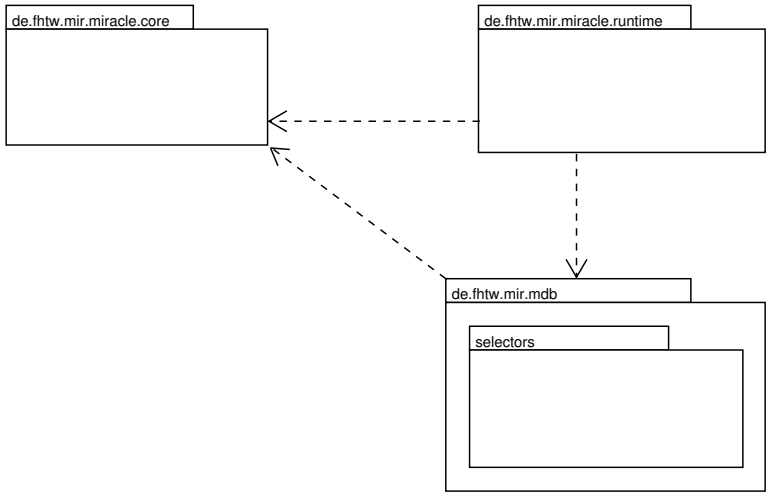

Abbildung 4.6: statisches Modell der Pakete und Klassen

miracle.core: Das Paket `miracle.core` umfaßt alle Schnittstellen und Klassen, deren Implementierung unabhängig vom verwendeten Storage Layer sind.

miracle.runtime: Hier ist die Beispielanwendung zur Demonstration der Fähigkeiten der MIRaCLE Applikation enthalten. Klassen dieses Packetes benutzen sowohl die Schnittstellen vom `miracle.core`, als auch die konkreten Implementierungen aus `mdb` zur Prozessierung der Verweise und des Verweiskontextes.

mdb: Unterhalb des Paketes `mdb` befinden sich die MIR–System spezifischen Implementierungen der Schnittstellen des Paketes `miracle.core`. Sie stellen die konkrete Funktionalität zur Verfügung.

mdb.selectors: Dieses Packet enthält die MIME–Type und Storage spezifischen Implementierungen der „Selector“ Schnittstelle des Paketes `miracle.core`.

4.3.4.2 Klassen und Schnittstellen

Schnittstellen (Interfaces) beschreiben die Minimalfunktionalität, die für eine vollständige Implementierung umzusetzen sind. Im Rahmen der MIRaCLE Anwendung sind diese Kernfunktionalitäten Bestandteil des Paketes `miracle.core`. Da diese Schnittstellen Grundlage der im nächsten Kapitel vorgestellten Implementierung sind, sollen zuvor die Ideen und Konzepte erläutert werden. Hierzu wird getrennt auf die Schnittstellen zur Umsetzung eines Verweismodells und die Schnittstelle bzw. Klasse zur Verwendung eines Verweiskontextes eingegangen.

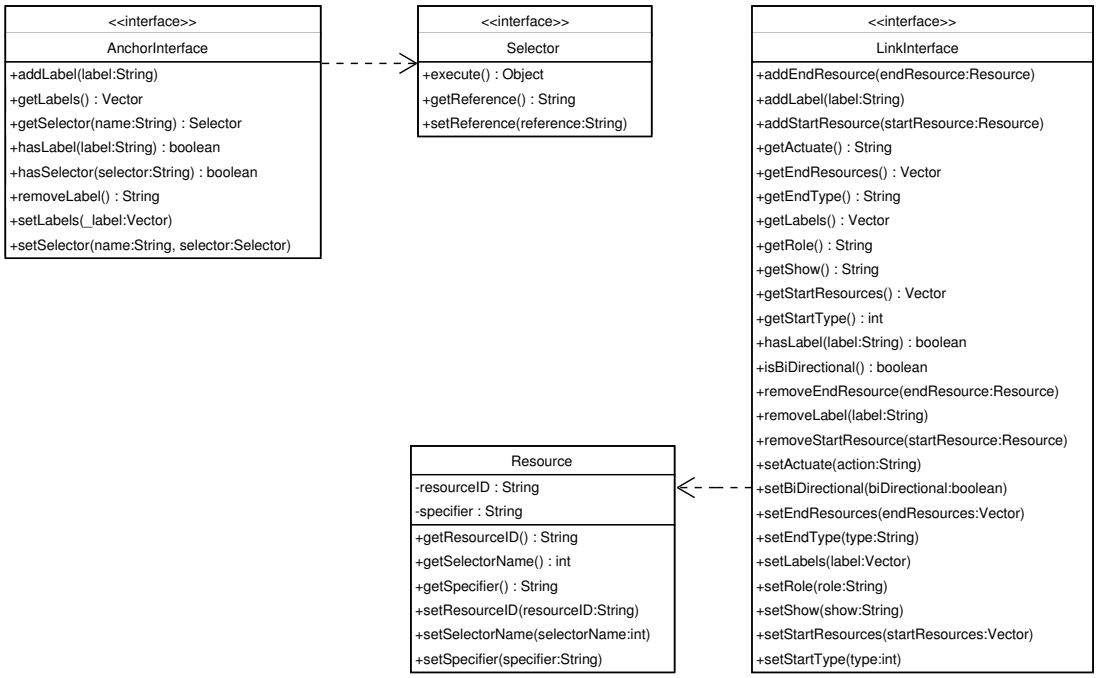

Abbildung 4.7: Schnittstellen des Verweismodell

Verweisfunktionalitäten

Zur Umsetzung des besprochenen Verweismodells werden drei Schnittstellen zur Repräsentation von Verweis (`LinkInterface`), Anker (`AnchorInterface`) und Selektionsmethode (`Selector`) sowie die Klasse `Resource` benötigt. Abbildung 4.7 stellt diese in Form eines UML Klassendiagrammes dar.

`LinkInterface`: Das `LinkInterface` repräsentiert einen Verweis und stellt Methoden zum Zugriff auf die an ihm beteiligten Ressourcen zur Verfügung. Des weiteren erlaubt das Interface die Beeinflußung von Attributen zur Anzeige sowie mehrere Symbolnamen (Labels) für einen Verweis.

`Resource`: Die Klasse `Resource` modelliert eine Start- bzw. End–Ressource. Sie identifiziert zum einen den beteiligten Anker und zum anderen die entsprechende Selektionsmethode, womit das durch den Verweis ausgezeichnete Datum eindeutig bestimmt wird.

`AnchorInterface`: Das `AnchorInterface` bildet die Funktionalität eines Ankers ab. Wird eine Subadressierung von Daten vorgenommen, so stehen Methoden bereit, um auf den entsprechenden Selektor zu zuzugreifen. Wie auch den Verweisen können Ankern Symbolnamen zugeordnet werden.

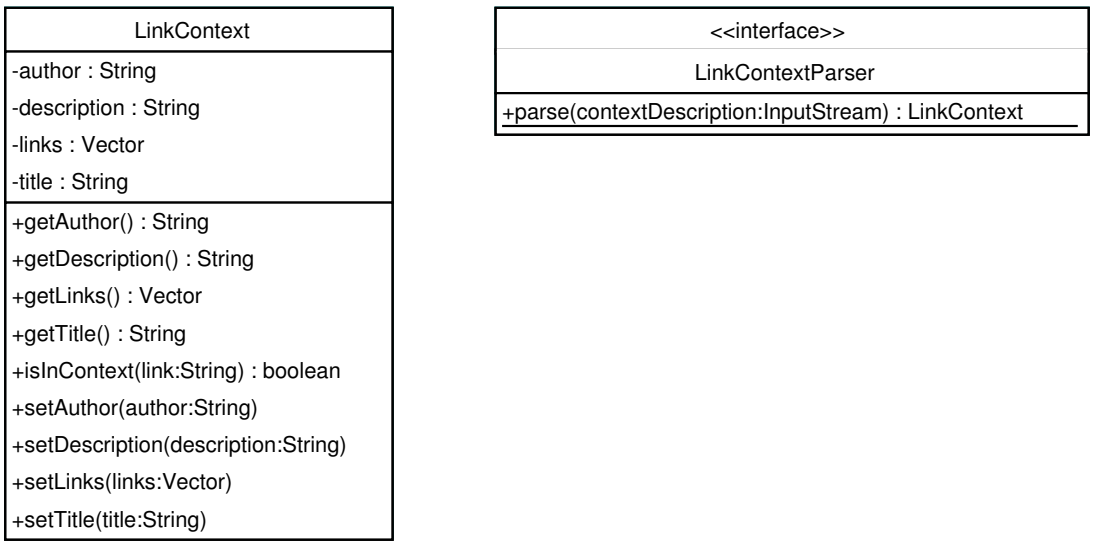

Abbildung 4.8: Klassen zur Behandlung des Verweiskontextes

`Selector`: Die `Selector` Schnittstelle ist Grundlage aller MIME–Type abhängigen Selektor Implementierungen. Mit Hilfe eines referenzierenden Ausdruckes wird das Datum festgelegt, auf dem der Selektor operiert.

4.3.4.3 Verweiskontext

Im folgenden werden eine Klasse und eine Schnittstelle vorgestellt, die als Ausgangspunkt für die Verwendung von Verweiskontexten dienen. Auch diese Klasse bzw. Schnittstelle ist Bestandteil des Paketes `miracle.core`.

`LinkContextParser`: Dieses Interface bietet eine statische Methode, um die XML konforme Beschreibung eines Verweiskontextes in ein Verweisobjekt umzuwandeln.

`LinkContext`: Die Klasse `LinkContext` kapselt einen Verweiskontext in Form eines Javaobjektes. Neben den deskriptiven Eigenschaften wie Autor, Titel und einer kurzen Beschreibung des Verweiskontextes stellt diese Klasse den Zugriff auf die Verweise innerhalb eines Kontextes bereit.

Kapitel 5

Implementierung

Das folgende Kapitel gibt einen kurzen Überblick über die prototypische Implementierung des vorgestellten Konzeptes auf Basis des MIR–Systems. Die Einbettung der Verweisfunktionalität in das MIR erfogt in zwei Schritten, zum einen wird eine neue Middlewarekomponente, der `LinkManager`, geschaffen, auf deren Funktionalität Client–Anwendungen mittels CORBA zugreifen können. Zum anderen müssen die Klassen `GenericAnchor` und `GenericLink` umgesetzt werden, um mit den Verweisstrukturen zu operieren. Die Implementierung erfolgt unter Verwendung der Programmiersprachen Java zur Umsetzung der Funktionalität und IDL (Interface Definition Language), um die Schnittstellen der `LinkManager` Komponente zu erzeugen.

Das MIR–System wird als Grundlage dieser Implementierung genutzt, da es bereits einen Großteil der benötigten Infrastruktur zur Umsetzung des Konzeptes bereitstellt. Es bietet bereits ein vollständiges Storage Layer mit entsprechenden Werkzeugen und Schnittstellen, um Datenstrukturen zu erzeugen, zu bearbeiten und auf diese zuzugreifen (siehe Abschnitt 2.6). Somit kann die aufwendige Umsetzung eines Storage Layers entfallen.

5.1 Erweiterung der Middleware

Die Middleware des MIR–Systems muß um eine Komponente – den `LinkManager` – erweitert werden. Hierzu wird zuerst eine Definition der Schnittstellen unter Verwendung von IDL[1] vorgenommen. Aus dieser Schnittstellendefinition werden dann mit Hilfe eines IDL zu Java Compilers die entsprechenden Java Klassen und Interfaces generiert. Die Implementierung dieser Schnittstelle heißt im konkreten Fall `LinkManagerImpl`.

[1] CORBA IDL

```
1  #include "de/fhtw/mir/mdb/DatabaseError.idl"
2  module de{
3    module fhtw {
4      module mir {
5        module mdb {
6          interface LinkManager {
7            ...
8           de.fhtw.mir.mdb.GenericAnchor getAnchor (
9              in string path
10          ) raises (::de::fhtw::mir::mdb::DatabaseError);
11           ...
12         };
13       };
14     };
15   };
16 };
```

Listing 5.1: IDL Schnittstelle

Der Quellcode 5.1 zeigt die Deklaration der Methode `getAnchor` im Paket `de.fhtw.mir.mdb`.

Zeile 1: Bindet eine weitere Schnittstelle (`DatabaseError`) aus dem selben Paket ein.

Zeilen 2-4: Legen den Namensraum fest, in dem sich das Interface befindet.

Zeilen 8-10: Definieren die Methode `getAnchor` mit einer Zeichenkette als Übergabeparameter und der Klasse `GenericAnchor` als Rückgabewert. Im Fehlerfall wird die Ausnahme `DatabaseError` erzeugt.

Analog zu diesem Beispiel kann man diese Schnittstelle um weitere Methoden ergänzen.

Die Umsetzung der konkreten Funktionalität erfolgt in der Klasse `LinkManagerImpl`. Die Aufgabe dieser Klasse ist es, alle datenaufwendigen Operationen zu übernehmen, da sie direkten Zugriff auf die Datenbank besitzt. Hierzu gehören sowohl Mechanismen, um Verweise und Anker aus dem MIR–Repository zu laden und wieder zurückzuspeichern, als auch Methoden zur gesteuerten Selektion von Ankern und Verweisen. Weiterhin werden einige Hilfsfunktionen bereitgestellt, um die Eigenschaften von Ankern und Verweisen komfortabel abzufragen.

Das Laden und Speichern von MIR–Objekten ist grundsätzlich Aufgabe des `ObjectManagers`. Um in ihrer Funktion redundante Methoden zu vermeiden, greift die `LinkManagerImpl` Klasse ebenfalls unter Verwendung des `ObjectManagers` auf das Repository zu. Da beide Schnittstellen und deren implementierende Klassen als Komponenten Bestandteil der Middleware (siehe 2.6) sind, muß ein sogenannter *Intercomponent Call* ausgeführt werden.

```
1  ...
2  public class LinkManagerImpl {
3     ...
4    // needed by intercomponent call
5    private Properties props = null;
6    private ORB orb = null;
7    private ObjectManager objectManager = null;
8
9    public LinkManagerImpl() {
10
11     props = new Properties();
12     props.put("org.omg.CORBA.ORBClass", "com.sybase.CORBA.ORB");
13
14     orb = ORB.init((String[]) null, props);
15     objectManager = ObjectManagerHelper.narrow(
16                     orb.string_to_object("de.fhtw.mir.mdb/ObjectManager"));
17      ...
18   }
19    ...
20 }
```

Listing 5.2: Intercomponent Call

Listing 5.2 zeigt auszugsweise den Quellcode der `LinkManager` Implementierung, wobei nur die für den Intercomponent Call relevanten Statements dargestellt werden.

Zeile 11-14: Initalisieren den ORB (Object Request Broker).

Zeile 15: Erzeugt eine Instanz des `ObjectManagers`, wobei die gewünschte Middlewarekomponente durch die Angabe des Paketnames und dem Namen der Komponente, getrennt durch einen Slash („/“), addressiert wird.

Die Methode `getAnchor` (siehe Listing 5.3) beispielsweise verwendet den `ObjectManager`, um MOBs aus dem MIR–Repository zu laden.

Das nachfolgende Listing 5.3 demonstriert, wie man aus den MOBs im MIR–Repository das korrespondierende Java Objekt erzeugt und mit Werten füllt. Verläuft das Laden eines MOBs fehlerfrei, werden die Eigenschaften des MOBs in den zu erzeugenden Anker kopiert, anderenfalls wird eine Ausnahme geworfen. Die etwas ungewöhnliche Initalisierung eines Objektes mit konkreten Werten außerhalb des Konstruktors, liegt in der Tatsache begründet, daß die zu erzeugenden Objekte keine eigene Datenbankverbindung besitzen. Insofern muß diese Aufgabe von anderen Objekten, in diesem Fall von der Middlewarekomponente `LinkMangager` übernommen werden, da diese wie zuvor beschrieben, Zugriff auf die Daten des MIR–Repositories besitzt.

```
1  public GenericAnchor getAnchor( String path )
2    throws DatabaseError {
3
4    // get MOB from repository
5    GenericMob genericMob = (GenericMob) objectManager.getMob(path);
6
7    // construct new anchor object
8    GenericAnchor genericAnchor = new GenericAnchor();
9
10   // set all properties inherit from GenericObject (Java class)
11   genericAnchor.setAccessRights(genericMob.getAccessRights());
12   genericAnchor.setAccessTime(genericMob.getAccessTime());
13   genericAnchor.setCreationTime(genericMob.getCreationTime());
14   genericAnchor.setGlobalPath(genericMob.getGlobalPath());
15   genericAnchor.setGroupName(genericMob.getGroupName());
16   genericAnchor.setID(genericMob.getID());
17   genericAnchor.setModificationTime(genericMob.getModificationTime());
18   genericAnchor.setUserName(genericMob.getUserName());
19
20   // set all properties inherit form ComplexObject (Java class)
21   genericAnchor.setMirClass(genericMob.getMirClass());
22   genericAnchor.setSettings(genericMob.getSettings());
23   genericAnchor.setValid(genericMob.getValid());
24
25   // set all properites inherit from GenericMob (Java class)
26   genericAnchor.setEventList(genericMob.getEventList());
27   genericAnchor.setTargetList(genericMob.getTargetList());
28
29   // return *complete* GenericAnchor object
30   return genericAnchor;
31 }
```

Listing 5.3: Die `getAnchor` Methode der Klasse `LinkManagerImpl`

Zeile 4: Laden eines MOBs aus dem MIR–Repository unter Verwendung der `ObjectManagers` (vergleiche auch 5.2).

Zeile 7: Erzeugen eines neuen Ankerobjektes durch Aufruf des Konstruktors.

Zeile 10-26: füllen des Ankerobjektes mit Werten. Hierfür müssen alle Elternklassen der Vererbungshierarchie durchlaufen und deren Eigenschaften konkrete Werte zugewiesen werden.

Analog zum Laden eines Ankers bzw. eines Verweises, funktioniert das Speichern eines Ankers bzw. Verweises. Das Umkopieren einer jeden Eigenschaft erweist sich hierbei jedoch als unnötig, da ein einfacher Downcast nach `GenericMob` diese beibehält. Es ist nur noch erforderlich, die Membervariablen der Klasse als MIR Properties persistent zu machen.

5.2 Anker und Verweise

Anker und Verweise spielen, wie zuvor gezeigt, eine zentrale Rolle im Verweismodell. Der folgende Abschnitt gibt einen Überblick über die für die Implementierung interessanten Details. Hierbei wird zuerst auf die Anker und anschließend auf die Verweise eingegangen.Die generelle Beschreibung der umzusetzenden Schnittstellen und ihrer Funktionalitäten ist Bestandteil des vorangegangen Kapitels.

Das Klassendiagramm (Abbildung 5.2 Seite 87) gibt einen Einblick in die Struktur, das Zusammenspiel und die Abhängigkeiten der Klassen und Interfaces dieser Anwendung. Die gestrichelten Pfeile stellen Abhängigkeiten zwischen Klassen oder Schnittstellen dar. Dieses Modell soll als Grundlage aller weiteren Betrachtungen dienen.

5.2.1 Die Klasse `GenericAnchor`

Die Klasse `GenericAnchor` repräsentiert einen Anker und implementiert die Schnittstellen `AnchorInterface` und `Serializable`. Sie stellt dabei eine Spezialisierung der Klasse `GenericMob` (vergleiche Abbildung 5.2) dar.

Zur Erzeugung von Ankern stehen grundsätzlich zwei Wege zur Verfügung. Zum einen kann der Konstruktor von `GenericAnchor` aufgerufen werden, um ein mit Standardwerten initialisiertes Objekt zu erzeugen. Zum anderen kann die Methode `getAnchor` des `LinkManagers` ausgeführt werden. In diesem Fall wird ein Objekt mit den entsprechenden Daten des korrespondierenden MOBs erzeugt. Wie schon in den Erläuterungen zum `LinkManager` ausgeführt, besitzen weder Anker noch Verweise Zugriffsmöglichkeiten auf die Daten des MIR–Repositories. Deshalb ist eine Initialisierung der (Anker)Objekte mit den Daten des korrespondierenden MOBs im Konstruktor nicht möglich, sondern muß vom `LinkManager` übernommen werden.

`GenericAnchor` besitzt keine eigenen Membervariablen. Die Methoden, die aufgrund der Schnittstellenspezifikation `AnchorInterface` implementiert werden, bilden Wrapper um Methoden von Elternklassen. Sinn dieser Wrappermethoden ist es, ein kompfortables Arbeiten mit den Klassen zu ermöglichen. Natürlich besteht jederzeit die Möglichkeit, diese Eigenschaften über die Mehtode `getValue` der Elternklassse `ComplexObject` abzufragen. Der

folgende Auschnitt aus dem Quellcode von `GenericAnchor` soll diesen Sachverhalt verdeutlichen:

```
1  ...
2  public class GenericAnchor extends GenericMob implements AnchorInterface, Serializable
3     ...
4    public Vector getLabels() {
5      // use Object GenericMob(String) from ComplexObject
6      return( (Vector) this.getValue("labels") );
7    }
8
9    public void setLabels(Vector labels) throws PropertyVetoException {
10     // use void setValue( String , Object ) throws PropertyVetoException from
11     // ComlexObject
12     this.setValue("labels", labels);
13   }
14    ...
15 }
```

Listing 5.4: Auschnitt `GenericAnchor` Klasse

Zeile 6: Benutzt die von `ComplexObject` geerbte Methode `getValue`, um auf die Eigenschaften eines MIR–Objektes zuzugreifen.

Zeile 12: Verwendet die ebenfalls von `ComplexObject` geerbete Methode `setValue`, um die Eigenschaften eines MIR–Objektes zu verändern.

Gleiches trifft für die `getSelector` und `setSelector` Methoden zu. Eine Besonderheit hierbei ist die Art, in der die speziellen Selektoren erstellt werden. Es wurde bereits darauf hingewiesen, daß die Selektoren MIME–Type abhängig sind. Jede Implementierung des Interfaces $\rightarrow$ `Selector` ist demnach MIME–Type spezifisch. Umgekehrt muß jeder konkrete Selektor zumindest über die Methoden der Schnittstelle `Selector` verfügen, so daß diese aufgerufen werden können. Diese Eigenschaft macht sich die Klasse `GenericAnchor` zunutze.

Die Methode `getSelector` des `GenericAnchors` benutzt die Hilfsklasse `Selector-Factory` (siehe Abbildung 5.1), um anhand der MIR–Klasse des MIR–Events die konkrete Selektorimplementierung zu bestimmen.

```
1   ...
2   public Selector getSelector( String name)
3     throws
4       InstantiationException,
5       IllegalAccessException,
6       ClassNotFoundException,
7       NullPointerException {
8
9       Selector concreteSelector = null;
10
11      // get events from MOB
12      Vector events = this.getEventList();
13
14      GenericEvent selectorEvent = null;
15
16      // find matching event
17      for( Iterator it = events.iterator(); it.hasNext(); ) {
18        if( ( selectorEvent = (GenericEvent) it.next()).getLocalName().equalsIgnoreCase( name ) ){
19          continue;
20        }
21        selectorEvent = null;
22      }
23
24      // if selectorEvent equals null than there is no matching selector ;
25      // throw exception , otherwise create instance
26      if( selectorEvent != null ) {
27        concreteSelector = SelectorFactory.getSelector( selectorEvent );
28      }
29      else {
30        throw new NullPointerException();
31      }
32
33      return( concreteSelector );
34  }
35  ...
```

Listing 5.5: Die `getSelector` Methode

Zeilen 17-22: Ermitteln des zum übergebenen Namen zugehörigen MIR–Event, der die Selektionslogik enthält.

Zeilen 26-31: Wurde ein passender Selektor ermittelt, so wird die öffentliche, statische Methode `getSelector` der Hilfklasse `SelectorFactory` verwendet, um ein neues Selektorobjekt zu instanzieren. Andernfalls wird eine Ausnahme vom Typ `NullPointer-Exception` erzeugt.

Die interne Funktionsweise der Methode `getSelector` der Hilfsklasse `SelectorFactory` ist dem Abschnitt 5.4.1 zu entnehmen.

5.2.2 Die Klasse `GenericLink`

Die Klasse `GenericLink` ist das Pendant zu `GenericAnchor` auf der Ebene eines Verweises. Sie stellt ebenfalls eine Spezialisierung der Klasse `GenericMob` dar und implementiert die Schnittstellen `LinkInterface` und `Serializable`.

Es stehen auch hier zwei grundsätzliche Möglichkeiten zur Verfügung, um ein Objekt vom Typ `GenericLink` zu erhalten. Zum einen können diese durch den Aufruf des Konstruktors erzeugt werden, zum anderen stellt der `LinkManager` die Methode `getLink` bereit. Die Begründung für dieses Vorgehen wurde bereits in den vorangegangen Abschnitten „Erweiterung der Middleware“ und „Die Klasse `GenericAnchor`“ gegeben.

Wie in Abschnitt 5.2.1 ausgeführt, besitzt auch diese Klasse keine eigenen Membervariablen, sondern stellt Wrappermethoden zur Verfügung. Eine Sonderfunktion nehmen hierbei die Methoden mit dem Suffix `EndRessource` bzw. `StartResource` ein. Hierbei wird anstatt von Zeichenkettenliteralen die Klasse `Resource` verwendet, die diese Literale bereits in einer aufgespaltenen Form enthält, um eine einfache Weiterverarbeitung zu ermöglichen.

5.3 Der Verweiskontext

Der Verweiskontext als abstrakte Organisationsschicht für Verweise wird durch die Klasse `LinkContext` in der vorliegenden Implementierung ausgedrückt. Da die Beschreibung des Verweiskontextes durch RDF vorgenommen wird, muß diese zuvor durch einen Parser eingelesen, um dann in die interne Darstellung des Storage Layers überführt zu werden. Diese Aufgabe übernimmt die Klasse `LinkContextParser`. Die interne Darstellung in Form der Klasse `LinkContext` ist dann Ausgangspunkt für alle Verweisselektionen.

5.3.1 Die Klasse `LinkContextParser`

Die Klasse `LinkContextParser` übernimmt die Aufgabe, die in Form von RDF vorliegende Beschreibung eines Verweiskontextes in ein entsprechendes Java Objekt der Ausprägung `LinkContext` zu transformieren. Diese Funktionalität wird von der öffentlichen, statischen Methode `parse` angeboten. Sie erlaubt es, die Beschreibung eines Verweiskontextes zu parsen, ohne dafür separat ein Objekt zu instanzieren.

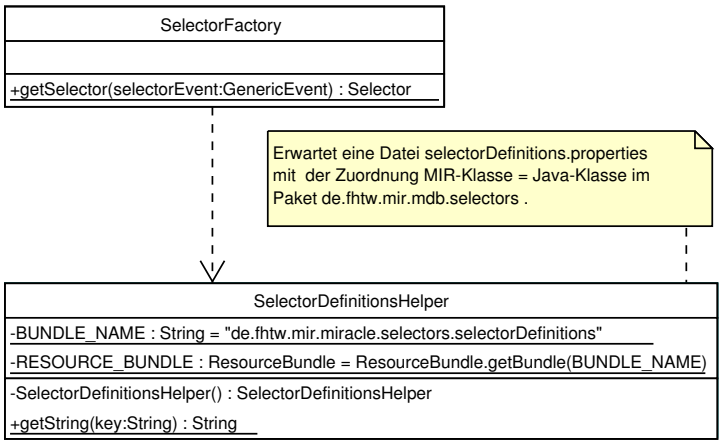

Abbildung 5.1: `SelectorFactory` und `SelectorDefinitionsHelper`

5.3.2 Die Klasse `LinkContext`

Die Abbildung der Verweiskontexte in Form von Java Objekten erfolgt durch Instanzen der Klasse `LinkContext`. Analog zur Verweisimplementierung speichert auch diese nicht Referenzen auf Objekte der Ausprägung `GenericLink`, sondern ebenfalls Zeichenkettenliterale, die MOB des MIR–Repositories addressieren. Eine Instanz enthält alle zu einem Kontext gehörenden Verweise.

Objekte der Klasse `LinkContext` können auf zwei Arten erzeugt werden. Zum einen ist dies durch den Aufruf des Konstruktors, zum anderen durch das Ausführen der öffentlichen, statischen `parse` Methode der Klasse `LinkContextParser` möglich. Das Parsen der Verweiskontexte ist nicht Bestandteil des Konstruktors, da die Ausgestaltung und Implementierung des Parsers unabhängig vom Verweiskontext gehalten werden soll. Somit können Erweiterungen an der Grammatik des Verweiskontextes vorgenommen werden, ohne die Java Klassenrepräsentation zu ändern.

Über die Methode `isInContext` kann leicht überprüft werden, ob ein Verweis Bestandteil der durch dieses Objekt ausgezeichneten Menge an Verweisen ist.

5.4 Subaddressierung von Daten

Die Subaddressierung der Daten ist, wie oben gezeigt, in Abhängigkeit des entsprechendem MIME–Types vorzunehmen. Das vorgestellte Konzept überträgt diese Aufgabe den Selektoren, wobei jeder Selektor zumindest die Schnittstelle `Selector` realisieren muß.

Die Zuordnung der in den MIR–Events gespeicherten Selektionslogik zu einer entsprechenden Java Klasse erfolgt durch die Datei *selectorDefinitions.properties*. Diese enthält *Schlüssel=Wert* Paare, wobei der Schlüssel der vollständig qualifizierte Name der MIR–Klasse und der Wert der ebenfalls vollständig qualifizierte Name der Java Klasse ist.

Dieser Mechanismus ermöglicht die Erweiterung der Anwendung um zusätzliche Selektoren (siehe hierzu Abschnitt 5.4.2), ohne dabei bestehende Klassen oder Schnittstellen zu verändern.

5.4.1 Die Klasse `SelectorFactory`

Die Hilfsklasse `SelectorFactory` erlaubt es, unter Verwendung ihrer öffentlichen, statischen Methode `getSelector` aus den MIR–Events Java Objekte zu erzeugen, die die Schnittstelle `Selector` implementieren:

```
1  ...
2  public class SelectorFactory {
3    public static Selector getSelector(GenericEvent event)
4      throws
5        InstantiationException,
6        IllegalAccessException,
7        ClassNotFoundException {
8
9      // get MIR class definition from selector
10     String classDefinition = event.getMirClass();
11
12     // use selectorDefinition . properties to get concrete mime-type specific
13     // selector conforming to the de.fhtw.mir.miracle.core.Selector interface
14     Selector selector =
15       (Selector) Class.forName(
16         SelectorDefinitionsHelper.getString(classDefinition)
17        ).newInstance();
18
19     // set data
20     selector.setSelector( event );
21
22     return (selector);
23   }
24 }
```

Listing 5.6: Die `parse` Methode der Klasse `SelectorFactory`

Zeile 10: Ermittelt den Namen der MIR–Klasse des betreffenden MIR–Events.

Zeilen 14-17: Instanziiert ein neues Objekt in drei Schritten:

1. Ermitteln der Java Klasse auf Basis der MIR–Klasse unter Verwendung der öffentlichen, statischen Methode `getString` der Klasse `SelectorDefinitions-Helper`, die wiederum die Datei *selectorDefinitions.properties* verwendet.
2. Anfordern einer Instanz der Java Klasse `Class`, die den Klassendeskriptor abbildet, durch die öffentliche, statische Methode `forName` von `Class`.
3. Ausführen der Methode `newInstance` des zuvor erzeugten anonymen Objektes vom Typ `Class`, die dann ein Objekt der aus der MIR–Klasse ermittelten Java Klasse erzeugt.

Zeile 20: Füllen des erzeugten Objektes mit Daten.

5.4.2 Definition eines neuen Selektors

Um der Anwendung einen weiteren Selektor hinzuzufügen, sind die folgenden Schritte notwendig:

1. Anlegen der entsprechenden Datenstruktur im MIR–System in Form einer MIR–Klassendefinition für einen MIR–Event.
2. Implementieren der Selektionslogik und der Schnittstelle `Selector`.
3. Kompilieren der erstellten Selektionsklasse und bereitstellen des Kompilates im Klassensuchpfad(Classpath) der Java Laufzeitumgebung.
4. Erweitern der Datei *selectorDefinitions.properties*, um einen Eintrag `MIR-Klasse=Java Klasse`, so daß aus dem Namen der MIR–Klasse das zu instanziierende Java Objekt abgeleitet werden kann.

Der neu definierte Selektor steht jetzt der Anwendung zur Verfügung, d.h. Anker können nun die Selektionslogik dieses Selektors verwenden[2].

[2]Unter Umständen ist ein erneutes Starten der Anwendung notwendig, um den Klassen Cache der Java Laufzeitumgebung erneut zu initialisieren.

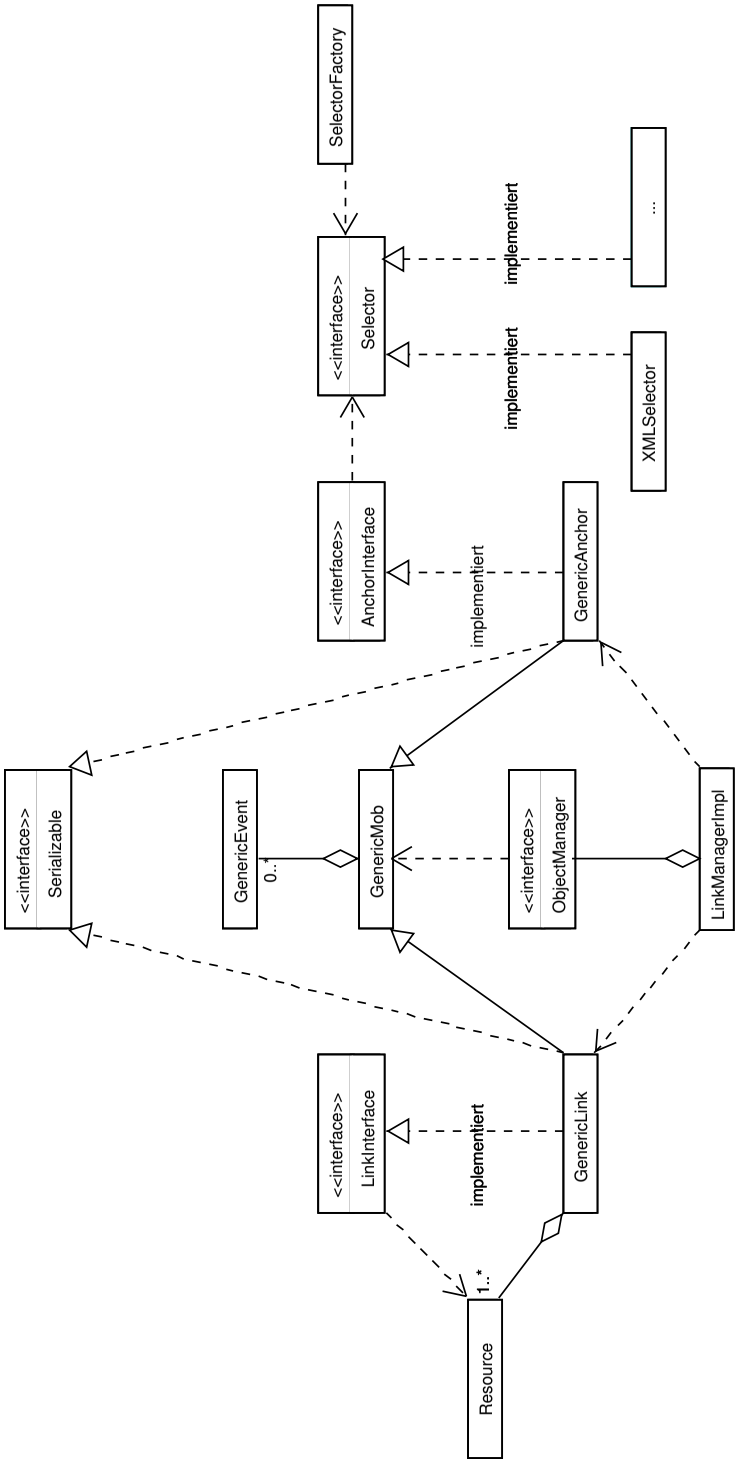

Abbildung 5.2: Java Klassendiagramm

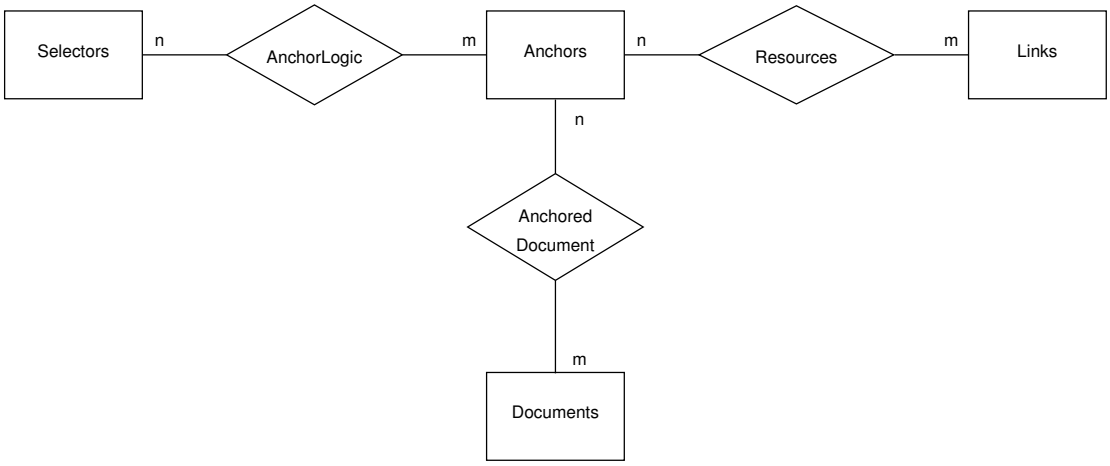

Abbildung 5.3: ER–Modell des Verweismodells

5.5 Alternative Implementierung

Um die Unabhängigkeit dieses Konzeptes vom MIR–System aufzuzeigen, wird hier ein alternativer Implementierungsansatz vorgestellt.

Grundsätzlich sind verschiedene Implementierungen des Storage Layers (Speicherungsschicht) möglich. Denkbar wäre hier beispielsweise eine relationale Datenbank oder auch die Abbildung in ein Dateisystem. Somit kann das Storage Layer als abstrakt angenommen werden. Wichtig für die Umsetzung dieses Konzeptes ist jedoch, das sich die Daten des Storage Layers unter Verwendung von RDF beschreiben lassen. Die so formulierten Aussagen über die Eigenschaften von Ankern oder von Verweisen, bilden dann die Grundlage für weitere Operationen, wie beispielsweise die Verknüpfung von Ankern durch einen Verweis zu einer Hyperreferenz. Jede Anwendung kann natürlich intern eigene Repräsentationen für beispielsweise Anker oder Verweise wählen.

Ein weiteres seitens des Storage Layer zu unterstützendes Feature ist, wie aus der Anforderungsanalyse erkenntlich (siehe Abschnitt 4.1.3), das Speichern von Ankern und Verweisen als eigenständige Entitäten. Ebenso muß durch das Storage Layer die referentielle Integrität der in Abbildung 5.3 dargestellten Verknüpfungen gewährleistet werden. Bei der Umsetzung der Speicherungsschicht ist außerdem auf die weiteren, in der Aufgabendefinition (siehe Abschnitt 4.1.3) herausgestellten Anforderungen, wie beispielsweise die Annotierbarkeit durch Metainformationen, zu achten.

Der nächste Schritt besteht darin, die Schnittstellen `LinkInterface`, `AnchorInterface` und `Selector` zu implementieren. Gleichzeitig müssen Klassen oder Methoden zum Zugriff auf die Daten des Storate Layers erstellt werden. Die Klasse `LinkContext` verläßt sich lediglich auf die korrekte und folgerichtige Umsetzung dieser Schnittstellen.

Die Transformation der Verweisinformationen in eine durch einen HTML Browser verarbeitbare Ausgabe ist Aufgabe der Präsentationsschicht und wird im Presentation Layer (siehe Abschnitt 2.3.1) vorgenommen.

Anwendungen, die den Verweiskontext nutzen wollen, können dann auf dieser relativ unabhängigen Implementierung aufsetzen.

Kapitel 6

Anwendungsszenarium

Um den Nutzen des hier entwickelten Konzeptes und der Implementierung zu demonstrieren, erfolgt in diesem Kapitel eine schematische Beschreibung eines typischen Anwendungsfalles im Rahmen einer eLearning–Anwendung. Ein weiteres denkbares Szenarium könnte die Verwendung in einem durch den Nutzer personalisierbares Informationssystems sein.

Als Ausgangssituation werden Lernende angenommen, die alle einen Vortrag über die Kultur der Maya halten sollen, wobei jeder einen unterschiedlichen Schwerpunkt betrachtet:

- Student A soll über die Architektur der Maya,
- Student B über die Siedlungsgebiete der Maya,
- Student C über die wissenschaftlichen Entdeckungen der Maya

referieren. Des weiteren wird angenommen, daß die Anwendung über umfangreiches Material zu diesen Themen verfügt.

Ausgangspunkt der Recherche dieser Studenten könnte ein Dokument sein, das einen einleitenden Text über die Kultur der Maya enthält (Abbildung 6.1). Dieses Dokument beinhaltet für jeden der Studenten einen interessanten Einstiegspunkt, um weitere spezialisierte Informationen abzurufen. Jedoch können die Verweise nicht unbedingt eindeutig dem jeweiligen Thema zugeordnet werden. So könnte sich hinter dem Link *„observatories“* sowohl die Beschreibung des Bauwerkes, als auch ein Dokument mit wissenschaftlichen Entdeckungen verbergen. Die Verwendung bereits definierter Verweiskontexte erlaubt es, Zweideutigkeiten zu eleminieren, so daß jeder Verweis im dargestellten Kontext eindeutig ist.

Um aus der Menge aller möglichen Verweise nur die themenbezogenen zu selektieren, wählt jeder der Studenten in Abhängigkeit von seinem Thema einen bestimmten Verweiskontext aus:

- Student A wählt den Kontext: „Architektonischer Hintergrund“ (Abbildung 6.2),
- Student B den Kontext: „Geographische Einordnung“ (Abbildung 6.3),
- Student C den Kontext „Wissenschaftliche Entdeckungen“ (Abbildung 6.4)

Jedem Studenten werden nun nur noch die in seinem Kontext relevanten Verweise angezeigt. Hinter dem Verweis „Observatories“ sind tatsächlich sowohl Informationen zur Architektur als auch zur Wissenschaft der Maya hinterlegt, jedoch wird nach der Traversierung des Verweises bei Student A und C jeweils ein anderes Dokument angezeigt. Da der ausgewählte Verweiskontext über das Verfolgen einer Hyperreferenz hinaus besteht, werden jedem Studenten auf neuen Seiten nur themenbezogene Verweise angeboten, die Ausgangspunkt für weitere Seitenaufrufe sind.

Natürlich kann der durch den Anwender bestimmte Verweiskontext jederzeit geändert werden. So möchte sich Student A vielleicht über den architektonischen Hintergrund hinaus informieren, etwa um eine Verbindung zwischen Bauweise und in der Region vorhandenen Rohstoffen herzustellen. In diesem Fall wählt Student A den entsprechenden Kontext und kann seine gezielte Informationssuche unter einem anderem Gesichtspunkt fortsetzen. Die nun dargestellten Verweise gehören alle dem Kontext „Geographische Einordnung“ an, bis der Verweiskontext wieder gewechselt oder kein Kontext gewählt wird. Ist kein Kontext ausgewählt, werden in dieser Anwendung alle mit dem Dokument verknüpften Verweise angezeigt. Sollte ein Verweis mehrere Ziele haben, wie im Fall von „Observatories“, so sorgt die Präsentationsschicht für eine angemessene Darstellung um eines der Ziele auszuwählen. Dieses könnte beispielsweise in Form einer Auswahlbox geschehen.

Der Vorteil dieses Verfahrens ist die Fokussierung auf bestimmte Verweise eines vorgegebenen Kontextes. Dies unterstützt den Lernenden, sich zielgerichtet in der Lernanwendung zu bewegen.

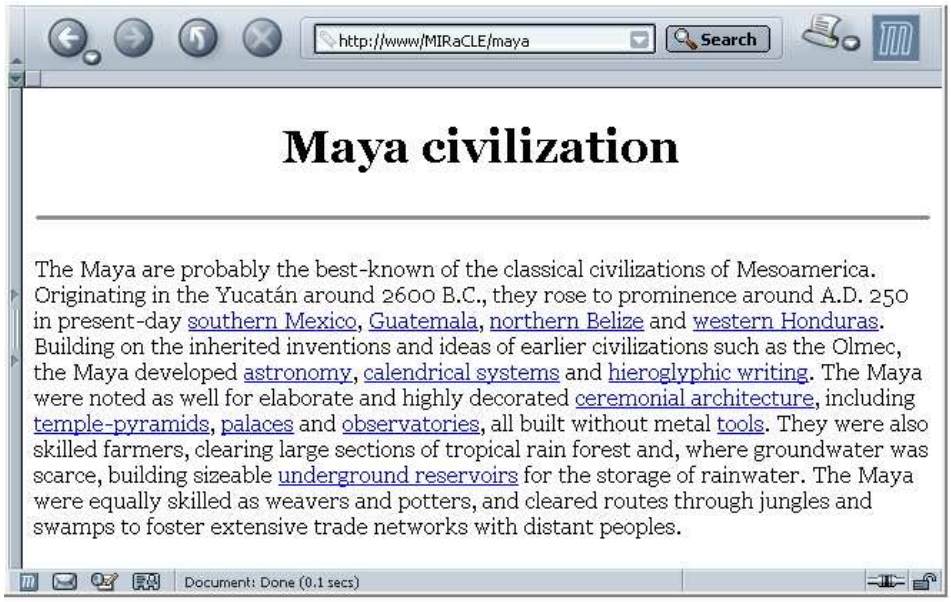

Abbildung 6.1: Dokument ohne gewählten Verweiskontext

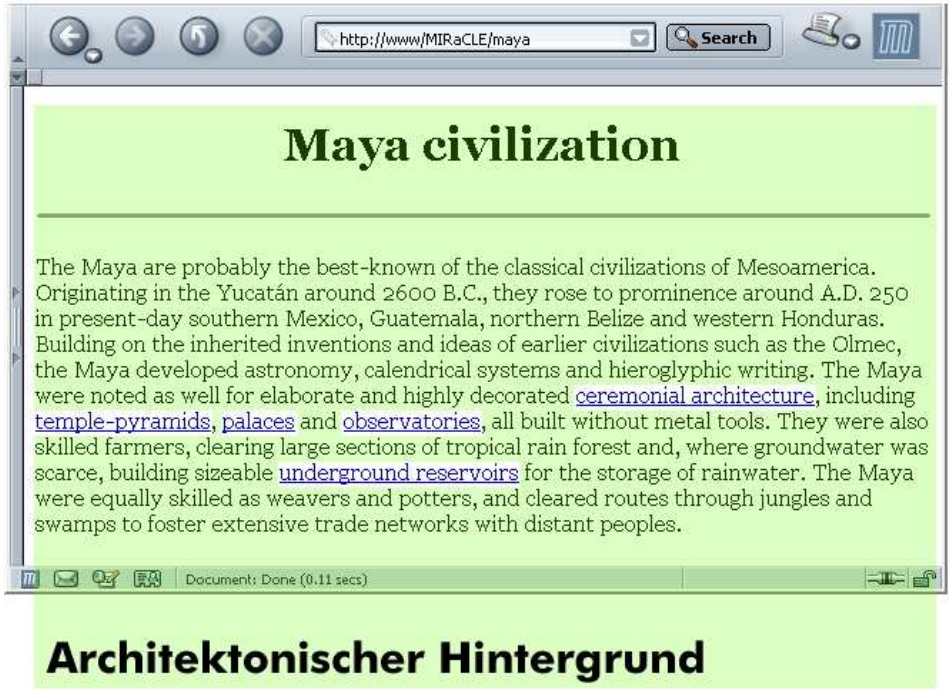

Abbildung 6.2: Dokument mit gewählten Verweiskontext „Architektonischer Hintergrund“

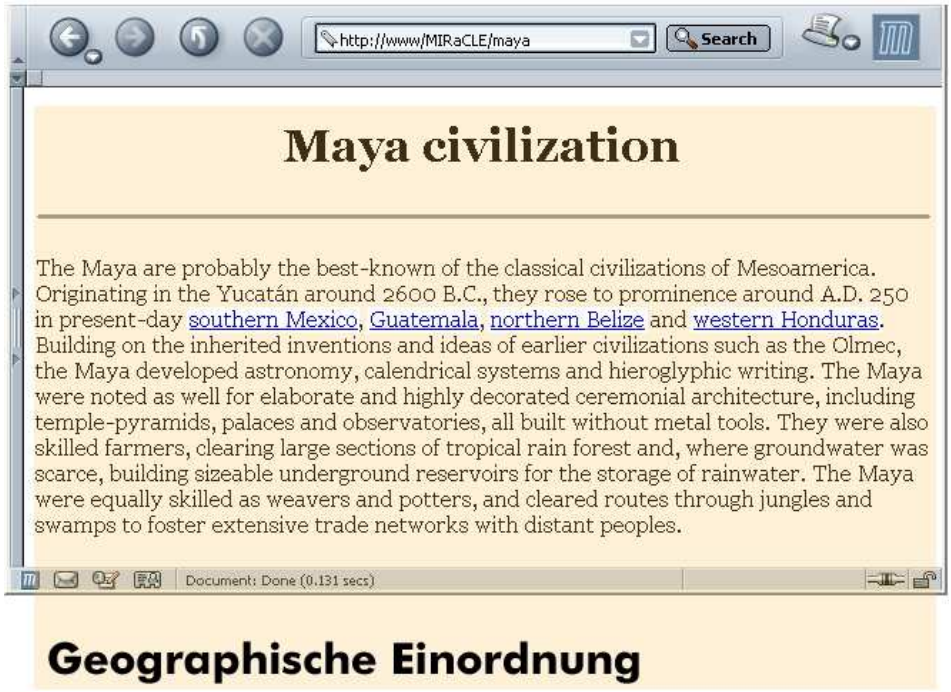

Abbildung 6.3: Dokument mit gewählten Verweiskontext „Geographische Einordnung“

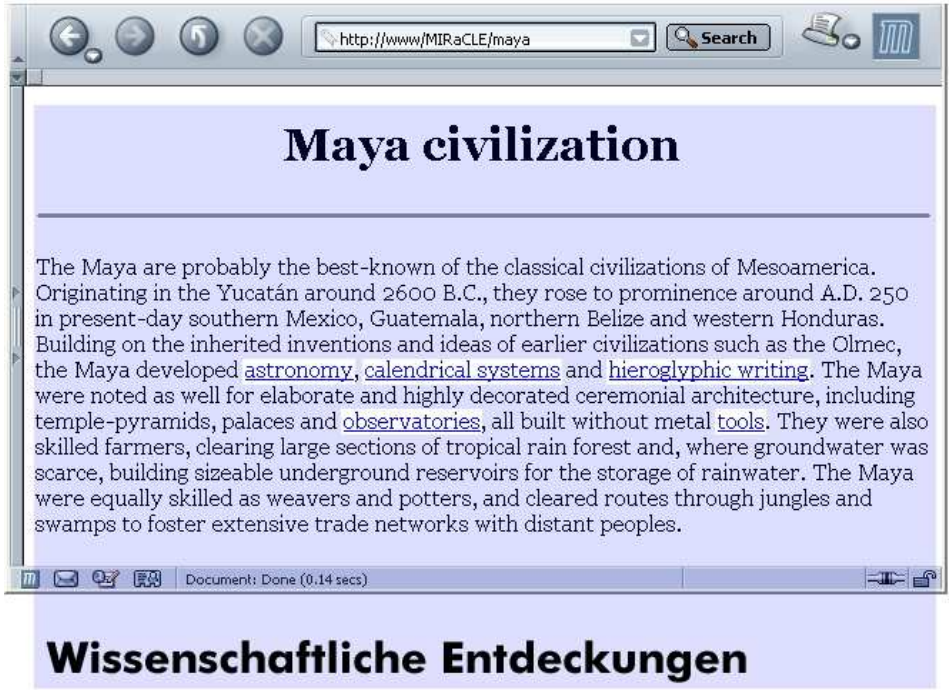

Abbildung 6.4: Dokument mit gewählten Verweiskontext „Wissenschaftliche Entdeckungen“

Kapitel 7

Fazit und Ausblick

In der Einleitung des 2. Kapitels „Das Internet als Basis für Lernumgebungen“ wurde festgestellt, daß der pädagogische Wert einer eLearning–Anwendung stark von zwei Kriterien abhängt: Zum einen von der Qualität der Inhalte, zum anderen von den Interaktionsmöglichkeiten. eLearning–Anwendungen nutzen Hyperreferenzen als Mittel der Interaktion, womit sich ihre Verwendung durch den Autor direkt auf die Qualität der Inhalte und auf das didaktische Modell auswirkt.

Internetbasierte Lernanwendungen bieten durch die Verbindung verschiedener Medien (Bild, Text, Annimationen, etc.) und die Möglichkeit, Inhalte beliebig miteinander zu verknüpfen, neue Wege, Wissen zu erschließen. Sie erforden aber gleichermaßen einen hohen Grad an Strukturiertheit und Transparenz, um vom Lernenden bedienbar und ihm verständlich zu sein. In seiner grundlegenden Arbeit „The Rhetoric of Hypermedia“ betrachtet Landow Hyperreferenzen als ein zweischneidiges Schwert, die zum einem dem Nutzer Informationen in einer neuen, sehr effizienten Art präsentieren, zum anderen aber häufig Anlaß zur Verwirrung geben (vergleiche ([Lan89]). Kapitel 3 zeigt, daß „sprechende“ Verweise ein Weg sind, dieses Problem zu vermeiden. Hiervon ausgehend wurde die Idee der Verweiskontexte als abstrakte Organisationsschicht zur „high–level“ Verweisselektion vorgestellt und ein entsprechendes Verweismodell entworfen.

Das in dieser Arbeit entwickelte Konzept der Verweiskontexte erlaubt die Kategoriesierung von Verweisen nach inhaltlichen oder funktionalen Gesichtspunkten, so daß auf einen Inhalt mehrere Sichten verschiedener Verweismöglichkeiten angewendet werden können. Der Lernende kann in Abhängigkeit seiner persönlichen Bedürfnisse, um ein Lehr- oder Informationsziel zu erreichen, aus einer Menge durch Autoren definierter Verweiskontexte auswählen und diese auf den Inhalt anzuwenden. eLearning–Applikationen profitieren hierdurch von erhöhter Übersichtlichkeit und erlauben, dem Anwender ein klar strukturiertes didaktisches Modell zu präsentieren. Dadurch könnten internetbasierte Lernanwendungen einen echten Mehrwert für die Lehre darstellen und in Zukunft vielleicht eine ernst zunehmende Alternative zum klassischen Lehrbuch bieten.

Wie gefordert (siehe Abschnitt 4.1.3 Aufgabendefinition) ist das erstellte Konzept unabhängig von der konkreten Implementierung des Storage Layers. Erreicht wurde diese Unabhängigkeit durch die Verwendung von RDF sowohl zur Beschreibung der Verweiskontexte, als auch optional zur Beschreibung der Verweise und Anker. Das entwickelte Verweismodell erweitert, auf Basis von XLink, die Fähigkeiten des einfachen HTML Verweises um bidirektionale und multidirektionale Verknüpfungen, so daß dem Autor fortgeschrittene Mittel zur Interaktionsgestaltung zur Verfügung stehen. Dynamische Anker und Verweise bieten zudem flexibele Möglichkeiten Informationen zu verknüpfen.

Die Implementierung eines Prototypen auf Basis des MIR–Systems wurde begonnen. Die vollständige Umsetzung aller für den Verweiskontext definierten Mengenoperationen, wird jedoch über den Bearbeitungszeitraum dieser Arbeit hinaus andauern. Des weiteren wurde nur der Selektor für einfachen bzw. strukturierten Text exemplarisch umgesetzt. Problematisch hierbei war insbesondere eine funktionierende und weitgehend vollständige XPointer Implementierung zu finden, da die Java Entwicklungsumgebung der Firma Sun, zum Zeitpunkt dieser Arbeit, keine Unterstützung hierfür beinhaltet.

Der Austausch der XPointer Implementierung ist zu erwägen, sobald Sun eine eigene, in seine Java Entwicklungsumgebung eingebettete Implementierung vorstellt. Somit können Versionskonflikte zwischen externen und internen Bibliotheken, wie zuletzt beim XML Parser beobachtet, vermieden werden.

Für die Zukunft wäre die Entwicklung von Anwendungen, die intensiven Gebrauch von den Fähigkeiten des Verweiskontextes und des Verweismodells machen, wie beispielsweise eine komplexe eLearning–Anwendung, interessant. In diesem Zusammenhang ist auch die Implementierung einer anwenderfreundlichen Autorenumgebung zum Erstellen und Bearbeiten von Verweiskontexten und Verweisen bedeutsam.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Onlineressourcen

[1] *Web Server Survey*. – http://www.netcraft.com/Survey/Reports/ (letzter Zugriff: 03. November 2002)

[2] *HyperCard*. – http://www.apple.com/hypercard (letzer Zugriff: 03. November 2002)

[3] *Hyperwave Homepage*. – http://www.hyperwave.com/ (letzter Zugriff: 03. November 2002)

[4] BOLES, Dietrich: *Begeleitbuch zur Vorlesung Multimedia–Systeme*. 1998. – http://www-is.informatik.uni-oldenburg.de/~dibo/teaching/mm98/buch/main.html (letzter Zugriff: 03. November 2002)

[5] MÜNZ, Stefan: *Entstehung des World Wide Web*. – http://selfhtml.teamone.de/intro/internet/www.htm (letzter Zugriff: 03. November 2002)

[6] W3C ; RAGGETT, Dave (Hrsg.) ; LE HORS, Arnaud (Hrsg.) ; JACOBS, Ian (Hrsg.): *HTML 4.01 Specification*. – http://www.w3.org/TR/html4/ (letzter Zugriff: 03. November 2002)

[7] W3C: *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*. – http://www.w3.org/TR/xhtml1/ (letzter Zugriff: 03. November 2002)

[8] W3C ; BRAY, Tim (Hrsg.) ; PAOLI, Jean (Hrsg.) ; SPERBERG-MCQUEEN, C. M. (Hrsg.) ; MALER, Eve (Hrsg.): *Extensible Markup Language (XML) 1.0*. Oktober 2000. – http://www.w3.org/TR/REC-xml (letzter Zugriff: 03. November 2002)

[9] W3C ; CLARK, James (Hrsg.) ; DEROSE, Steve (Hrsg.): *XML Path Language (XPath) Version 1.0*. November 1999. – http://www.w3c.org/TR/xpath (letzter Zugriff: 03. November 2002)

[10] W3C ; DEROSE, Steve (Hrsg.) ; MALER, Eve (Hrsg.) ; ORCHARD, David (Hrsg.): *XML Linking Language (XLink) Version 1.0*. – http://www.w3.org/TR/xlink (letzter Zugriff: 03. November 2002)

[11] W3C ; DEROSE, Steve (Hrsg.) ; MALER, Eve (Hrsg.) ; ORCHARD, David (Hrsg.): *XML Linking Language (XLink) Version 1.0*. – http://www.edition-w3c.de/TR/2001/REC-xlink-20010627/ (letzer Zugriff: 20. August 2002)

[12] W3C ; GROSSE, Paul (Hrsg.) ; MALER, Eve (Hrsg.) ; MARSH, Jonathan (Hrsg.) ; WALSH, Norman (Hrsg.): *XPointer Framework*. – http://www.w3.org/TR/xptr-framework/ (letzter Zugriff: 03. November 2002)

[13] W3C ; LASSILA, Ora (Hrsg.) ; SWICK, Ralph R. (Hrsg.): *Resource Description Framework (RDF) Model and Syntax Specification*. – http://www.w3.org/TR/REC-rdf-syntax (letzer Zugriff: 24. August 2002)

[14] KLYNE, Graham (Hrsg.) ; CAROLL, Jeremy (Hrsg.) ; MCBRIDE, Brain (Hrsg.): *Resource Description Framework (RDF): Concepts and Abstract Data Model*. – http://www.ninebynine.org/wip/RDF-basics/Current/Overview.htm (letzter Zugriff: 03. November 2002)

[15] WEIBEL, S. (Hrsg.) ; KUNZE, J. (Hrsg.) ; LAGOZE, C. (Hrsg.) ; WOLF, M. (Hrsg.): *Dublin Core Metadata for Resource Discovery*. – http://www.ietf.org/rfc/rfc2413.txt (letzter Zugriff: 03. November 2002)

[16] IEEE: *Draft Standard for Learning Object Metadata*. – http://ltsc.ieee.org/ (letzter Zugriff: 03. November 2002)

[17] W3C ; BRICKLEY, Dan (Hrsg.) ; GUHA, R.V. (Hrsg.): *RDF Vocabulary Description Language 1.0: RDF Schema*. – http://www.w3.org/TR/rdf-schema/ (letzter Zugriff: 03. November 2002)

[18] *ARIADNE Homepage*. – http://www.ariadne-eu.org (letzer Zugriff: 15. September 2002)

[19] *IMS Global Learning Consortium, Inc*. – http://www.imsproject.org (letzter Zugriff: 03. November 2002)

[20] *Java Naming and Directory Interface (JNDI)*. – http://java.sun.com/products/jndi/ (letzter Zugriff: 03. November 2002)

[21] *Java^{TM} Web Start*. – http://java.sun.com/products/javawebstart/ (letzter Zugriff: 03. November 2002)

[22] SCHMIDT, Thomas C.: *Konzepte zur Hyperreferenz*. – http://www.rz.fhtw-berlin.de/mirdoc/mir-db/hyperhist.xml (letzter Zugriff: 03. November 2002)

[23] *FHTW.Web: Internetauftritt der Fachhochschule für Technik und Wirtschaft Berlin*. – http://webtest.rz.fhtw-berlin.de/fhtw.web (letzter Zugriff: 03. November 2002)

[24] ENGELHARDT, Michael ; HILDEBRAND, Arne: *Styleguide FHTWDOC DTD*. – http://www.rz.fhtw-berlin.de/mirdoc/app-web/design-guide.xml (letzter Zugriff: 03. November 2002)

[25] HILDEBRAND, Arne: *Zerlegte Dokument-Typ-Definition für fhtw.web*. – http://www.rz.fhtw-berlin.de/mirdoc/app-web/divided_DTD.xml (letzter Zugriff: 03. November 2002)

[26] RACK, Torsten: *fhtw.web Autorenwerkzeug*. – http://www.rz.fhtw-berlin.de/mirdoc/app-web/authoring-1.xml (lezter Zugriff: 03. Oktober 2002)

[27] *The Jakarta Site – Apache Tomcat*. – http://jakarta.apache.org/tomcat/index.html (letzter Zugriff: 03. November 2002)

[28] *Apache Cocoon*. – http://xml.apache.org/cocoon (letzter Zugriff: 03. November 2002)

[29] CLARK, James (Hrsg.): *XSL Transformations (XSLT) Version 1.0*. – http://www.w3.org/TR/xslt (letzter Zugriff: 03. November 2002)

[30] CLARK, James (Hrsg.): *XSL Transformations (XSLT) Version 1.0*. – http://www.edition-w3c.de/TR/xslt (letzter Zugriff: 03. November 2002)

[31] *XML Linking Implementations*. – http://www.w3.org/XML/2000/09/LinkingImplementations.html (letzter Zugriff: 03. November 2002)

Literaturverzeichnis

[AMY88] AKSCYN, R. ; MCCRACKEN, D.L. ; YODER, E.A.: KMS: A distributed hypertext for managing knowledge in organizations. In: *Communications of the ACM* 37 (1988), July, Nr. 7, S. 820–835

[BRL91] BULTERMAN, Dick C A ; VAN ROSSUM, Guido ; VAN LIERE, Robert: A Structure for Transportable, Dynamic Multimedia Documents. In: *USENIX conference*, 1991, S. 137–155

[BSMM99] BRONSTEIN, I.N. ; SEMENDJAJEW, K.A. ; MUSIOL, G. ; MÜHLING, H.: *Taschenbuch der Mathematik*. 4. überarbeitete und erweiterte Auflage. Frankfurt am Main, Thun : Verlag Harri Deutsch, 1999

[Bus45] BUSH, Vannever: As We May Think. In: *The Atlantic Monthly* 176 (1945), July, Nr. 1, S. 101 – 108

[Con87] CONKLIN, Jeff: Hypertext: An Introduction and Survey. In: *IEEE Computer* 20 (1987), Nr. 9, S. 17–41

[Dam88] VAN DAM, Andries: Hypertext'87 Keynote Address. In: *Communications of the ACM* 31 (1988), Nr. 7, S. 887 – 895

[EHK+02] ENGELHARDT, M. ; HILDEBRAND, A. ; KÁRPÁRTI, A. ; RACK, T. ; SCHMIDT, T.C.: Educational Content Management – A Celluar Approach. In: AUER, Michael E. (Hrsg.) ; AUER, Ursula (Hrsg.): *International Workshop Interactive Computer Aided Learning ICL 2002*. Villach (Austria), 2002

[EKRS01] ENGELHARDT, M. ; KÁRPÁRTI, A. ; RACK, T. ; SCHMIDT, T.C.: A Virtual Knowledge Marketplace. In: AUER, Michael E. (Hrsg.) ; AUER, Ursula (Hrsg.): *International Workshop Interactive Computer Aided Learning ICL2001*. Villach (Austria), 2001

[Eng63] ENGELBART, Douglas C.: A Conceptual Framework for Augmentation of Man's Intellect. In: *Vistas of Information Handling* Bd. 1, 1963

[FKRS01] FEUSTEL, B. ; KÁRPÁRTI, A. ; RACK, T. ; SCHMIDT, T.C.: An Environment for Processing Compound Media Streams. In: *Informatica* 25 (2001), July 2001

[FS01] FEUSTEL, B. ; SCHMIDT, T.C.: Media Objects in Time - a multimedia streaming system. In: *Computer Networks* 37 (2001), Nr. 6, S. 729 – 737

[HBR94] HARDMAN, Lynda ; BULTERMAN, Dick C. A. ; VAN ROSSUM, Guido: The Amsterdam Hypermedia Model. In: *Communications of the ACM* 37 (1994), February, Nr. 2, S. 50–62

[HDH96] HALL, Wendy ; DAVIS, Hugh ; HUTCHINGS, Gerard: *Rethinking Hypermedia. The Microcosm Approach*. Bosten, Dordrecht, London : Kluwer Academic Publishers, 1996

[HS94] HALASZ, Frank ; SCHWARTZ, Mayer: The Dexter Hypertext. In: *Communications of the ACM* 37 (1994), February 1994, Nr. 2, S. 32–39

[Lan89] LANDOW, George P.: The Rhetoric of Hypermedia: Some Rules for Authors. In: *Journal of Computing in Higher Education* 1 (1989), Frühjahr, Nr. 1, S. 39 – 64

[Leu97] LEUTNER, Detlev: Adaptivität und Adaptierbarkeit multimedialer Lehr- und Informationssysteme. In: ISSING, Ludwig J. (Hrsg.) ; KLIMSA, Paul (Hrsg.): *Information und Lernen mit Multimedia*. 2. überarbeitete Auflage. Weinheim : Psyschologie Verlags Union, 1997, S. 139–149

[Mic99] MICHEL, Thomas: *XML kompakt - Eine praktische Einführung*. Carl Hanser Verlag München Wien, 1999

[Nie95] NIELSEN, Jakob: *Multimedia and Hypertext. The Internet And Beyond*. San Diego (USA) : Academic Press, 1995

[RBL] REPGES, R. ; BUCHNER, H. ; LEUTNER, D. *Untersuchung von Verfahren zum dynamisch-kontextabhängigen Zugriff auf Wissenbasen: Informations- und Lehrsystem klinische Elektroneurophysiologie*

[sgm86] GOLDFARB ET AL. (Hrsg.). *Text and Office Systems – Standard Generalized Markup Language (SGML)*. 1986

[WM02] WIDHALM, Richard ; MÜCK, Thomas: *Topic Maps*. Springer-Verlag Berlin Heidelberg, 2002

Anhang A

Landows Regeln für Autoren

Überblick der 19 von George P. Landow in seiner Arbeit „The Rhetoric of Hypermedia: Some Rules for Authors“[Lan89] definierten Regeln:

Rule 1. The very existence of links in hypermedia conditions the reader to expect purposeful, important relationships between linked materials.

Rule 2. The emphasis upon linking materials in hypermedia stimulates and encourages habits of reational thinking in the reader.

Rule 3. Since hypermedia systems predispose users to expect such significant relationships among documents, those documents that disappoint these expectations appear particularly incoherent and nonsignificant.

Rule 4. The author of hypermedia materials must provide devices that stimulate the reader to think and explore them.

Rule 5. The author of hypermedia must employ stylistic devices that permit readers to navigate materials easily and enjoyably.

Rule 6. Devices of orientation permit readers (a) to determine their present location, (b) to have some idea of that location's relation to other materials, (c) to return to their starting point, and (d) to explore materials not directly linked to those in which they presently find themselves.

Rule 7. Authors should consider employing several overviews to organize the same body of material and to assist readers to gain easy access to it.

Rule 8. Never place link markers independent of accompanying text or image.

Rule 9. When creating a link or positioning a link marker that indicates the presence of a link, remember that all links are bidirectional.

Rule 10. Avoid linking to words or phrases that only provide appropriate points of arrival but give the reader no suggestsion of where the link might lead on departure.

Rule 11. Place the link marker in close proximity to a text that indicates the probable nature of the link destionation.

Rule 12. When creating documents, assist readers by phrasing statements or posing question that provide obvious occasions for following links.

Rule 13. When possible provide specific information about a link destination by directly drawing attention to it.

Rule 14. Linked graphic materials must appear with appended texts that enable the user to establish a relation betwwen a point of departure and that of arrival.

Rule 15. The entire text accompanying visual material and not just the opening sentence or two serves as introduction.

Rule 16. The text accoumpanying an image does not have to specify all relevant information the author wihes the reader to have; rather emphasizing that a relationship exists at all may be enough.

Rule 17. Texts serve not only to provide information but also to reassure the reader that the link embodies a significant relationship and to provide some hint, however, incomplete, of how that relationship can be formulated by the reader.

Rule 18. When creating documents for hypermedia, conceive the text unit as brief passages in order to take maximum advantage of the linking capacities of hypermedia.

Rule 19. When adapting for hypermedia presentation documents created according to book technology, do not violate the original organization.

Eigenständigkeitserklärung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfaßt habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 4. November 2002

. . . . . .
Michael Nam Engelhardt