

Informationssysteme auf der Basis Aktiver Hypertextdokumente

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)

durch den Fachbereich Wirtschaftswissenschaften der
Universität-Gesamthochschule Essen

Vorgelegt von:
Eckhart Köppen (Geburtsort Essen)

Gutachter:
Prof. Dr. Gustaf Neumann
Prof. Dr. Stefan Eicker

Tag der mündlichen Prüfung: 5. Januar 2001

Inhaltsverzeichnis

Abschnitt A		
Grundlagen		1
1	Einleitung	2
2	Motivation und Aufgabenstellung	3
2.1	E-Commerce	3
2.2	Web-basierte Informationssysteme und E-Commerce	4
2.3	Aufgabenstellung	5
2.4	Das AHDM als Instrument der Wirtschaftsinformatik	6
2.5	Begründung der Vorgehensweise	7
3	Web-basierte Informationssysteme	7
3.1	Erweiterungen	8
3.2	Aufgabengebiete	10
4	Informationsstrukturierung	11
4.1	Standardized General Markup Language	11
4.2	Hypertext Markup Language	16
4.3	Extensible Markup Language	17
4.4	XML-Namensräume	20
5	Informationsvernetzung	23
5.1	Uniform Resource Identifier	23
5.2	XLink	25
5.3	XPath und XPointer	28
6	Kommunikation	30
6.1	Hypertext Transport Protocol	30
6.2	Common Gateway Interface	33
7	Informationsdarstellung	33
7.1	Cascading Style Sheets	34
7.2	DSSSL	36

7.3	XSLT und XSL	37
8	Interaktion	38
8.1	Intrinsic Event Model	38
8.2	Document Object Model	39
9	Diskussion	43

Abschnitt B Modell

		45
10	Aufgabe des Modells	46
11	Anforderungen an das Modell	46
11.1	Identifikation	46
11.2	Analyse	50
12	Verwandte Bereiche	52
12.1	Implementierungsebene	52
12.2	Infrastrukturebene	65
12.3	Informationsebene	68
12.4	Zusammenfassung verwandter Bereiche	70
13	Entwurf	72
13.1	Design-Entscheidungen	72
13.2	Aktive Hypertextdokumente	73
13.3	Allgemeine Eigenschaften aktiver Hypertextdokumente	75
13.4	Informationsmodell	78
13.5	Objektmodell	83
13.6	Laufzeitumgebung	85
13.7	Kommunikationsmodell	91
13.8	Semantiken	96
14	Zusammenfassung	100
14.1	Erfüllung der Anforderungen	100
14.2	Charakteristika des Modells	102
14.3	Charakteristika aktiver Hypertextdokumente	103

Abschnitt C	
Methode	107
15 Aufgabe der Methode	108
16 Anforderungen	108
16.1 Anforderungen an die Methode	108
16.2 Anforderungen an die Systembestandteile	108
17 Beschreibung der Methode	109
17.1 Notation	109
17.2 Vorgehen bei der Systementwicklung	111
17.3 Richtlinien	113
17.4 Musterorientiertes Vorgehen	113
18 Konstrukte und Muster	113
18.1 Architekturmuster	114
18.2 Entwurfsmuster	118
18.3 Idiome	125
19 Diskussion	132
19.1 Erfüllung der Anforderungen	133
19.2 Besondere Charakteristika	133
 Abschnitt E	
Anwendungen	135
20 Auswahl von Anwendungen	136
20.1 Eigenständige Anwendungen	136
20.2 Kooperative Anwendungen	136
20.3 Agentensysteme	136
21 Kontaktmanager	137
21.1 Aufgabenstellung	137
21.2 Strukturierung der Information	137
21.3 Gruppierung zusammengehöriger Komponenten	139
21.4 Definition der Verteilung der Komponenten	140
21.5 Definition der Kommunikationsstrategie	140

21.6	Realisierung der Komponenten	140
21.7	Auswertung	146
22	Bookmark-Verwaltung	149
22.1	Aufgabenbeschreibung	149
22.2	Strukturierung der Information	149
22.3	Gruppierung zusammengehöriger Komponenten	152
22.4	Definition der Verteilung der Komponenten	152
22.5	Definition der Kommunikationsstrategie	152
22.6	Realisierung der Komponenten	154
22.7	Erweiterungen	156
22.8	Auswertung	160
23	Beschaffungsprozeß	161
23.1	Aufgabenbeschreibung	162
23.2	Strukturierung der Information	162
23.3	Gruppierung zusammengehöriger Komponenten	168
23.4	Definition der Verteilung der Komponenten	169
23.5	Definition der Kommunikationsstrategie	169
23.6	Realisierung der Komponenten	170
23.7	Erweiterungen und Modifikationen	176
23.8	Auswertung	179
24	Diskussion	181

Abschnitt D

Werkzeuge

25	Aufgabenstellung	186
26	Implementierung eines Prototyps	186
26.1	Systemgrenze	187
26.2	Subsysteme	187
26.3	Module	188
26.4	Gruppierung der Module	194
26.5	Anwendung der Bibliothek	194
27	Erweiterung bestehender Werkzeuge	195

27.1	Vorhandene Werkzeuge	196
27.2	Systemgrenze	199
27.3	Subsysteme	199
27.4	Klassen	199
27.5	Gruppierung der Klassen und Systemarchitektur	201
27.6	Anwendung der Klassenbibliothek	201
28	Diskussion	202
Abschnitt F		
Ausblick		205
29	Zusammenfassung	206
29.1	Vorgehensweise in der Arbeit	206
29.2	Kontext des Modells	206
29.3	Submodelle des AHDM	207
29.4	Lose Kopplung der Systembestandteile	208
29.5	Infrastruktur	209
30	Offene Punkte	209
30.1	Sicherheitsaspekte	209
30.2	Plattformunabhängigkeit	212
30.3	Laufzeitverhalten	213
31	Fazit	214
Anhang		215
I	Document Type Definitions	216
II	IDL-Beschreibungen	216
III	Stylesheets	217
IV	Wiederverwendbare Komponenten	218
Literaturverzeichnis		221

Abschnitt A

Grundlagen



1 Einleitung

Eine der wichtigsten Entwicklungen auf dem Gebiet der Informationssysteme dürfte unbestritten das Internet sein, welches gerade in letzter Zeit in immer mehr Bereichen die technische Grundlage für vielfältige Anwendungen liefert. Darunter fallen insbesondere die elektronische Geschäftsabwicklung sowie das Suchen und Anbieten von Informationen. Einen wesentlichen Beitrag dazu leisten die sogenannten *Web-Techniken*, d.h. die Methoden, Techniken sowie die formellen und informellen Standards, die die Basis für das *World Wide Web* (WWW) [19] darstellen. Besonders hervorzuheben sind in diesem Zusammenhang die *Hypertext Markup Language* (HTML) [148], und das *Hypertext Transport Protocol* (HTTP) [17]. Beide Techniken werden vornehmlich eingesetzt, um Web-basierte Anwendungen und Informationssysteme zu implementieren. Die dabei zugrundeliegende Systemarchitektur ist die *Client-Server-Architektur*, bei dem mehrere Dienstanutzer von einem Dienstanbieter über HTTP Inhalte, die in HTML verfasst sind, anfordern (siehe Abbildung 1). Ohne die Erläuterung der technischen Details vorwegzunehmen, läßt sich sagen, daß mit HTML und HTTP die Aufbereitung, Vernetzung und Distribution der unterschiedlichsten Datentypen wesentlich vereinfacht wurde.

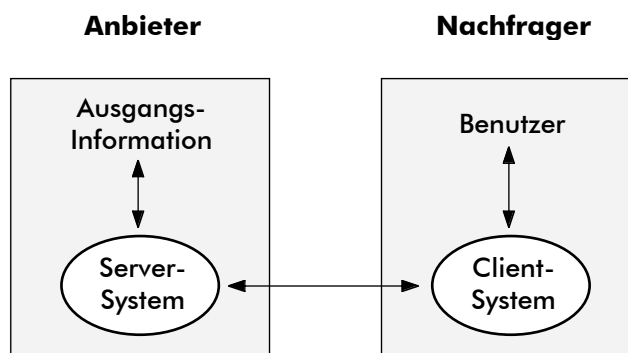


Abb. 1: Client-Server-Architektur

Nachdem die Entwicklung des WWW und der zugrundeliegenden Techniken am Anfang von Pragmatismus und schnellem Wandel geprägt war, ist nun eine Konsolidierung und gezielte Verbesserung der Kernstandards sowie die kontrollierte Definition neuer Standards zu erkennen. Als wichtigstes Gremium existiert hier das *World Wide Web Consortium* (W3C), ein Zusammenschluß von Firmen, Organisationen und akademischen Einrichtungen mit dem Ziel, die weitere Entwicklung der Web-Techniken zu steuern. Als wohl bislang wichtigster neuer Standard wurde die *Extensible Markup Language* (XML) [31] als im Vergleich zu HTML generellerem Mechanismus zur Strukturierung und Annotation von Information definiert.

In der vorliegenden Arbeit soll auf die besonderen Anforderungen bei der Implementierung von Informationssystemen auf der Basis von Web-Techniken eingegangen werden. Dabei spielt die Information innerhalb der Anwendungen eine zentrale Rolle, was zu einer Verlagerung des Schwerpunkts weg von der verbreiteten Client-Server-Architektur hin zu einer Architektur gleichberechtigter Netzwerkknoten führt. Kern der Arbeit ist die Entwicklung eines Modells, welches diesen informationszentrischen Ansatz unterstützt. Als Hauptbestandteil des Modells wird die Information mittels XML-basierter *aktiver Hypertextdokumente* (kurz AHD) gespeichert und per

HTTP transportiert, weshalb es als *Modell für aktiven Hypertext* (AHDM) bezeichnet wird. Aktive Hypertextdokumente transportieren neben anwendungsspezifischer Information auch aktive Komponenten in Form von Programmskripten. Durch die Verwendung aktiver Hypertextdokumente wird so die Realisierung der Anwendungssteuerung in Web-basierten Systemen dezentralisiert, was zur angesprochenen *Peer-to-Peer*-Architektur gleichberechtigter Netzwerkknoten führt.

2 Motivation und Aufgabenstellung

Die Motivation und Aufgabenstellung für diese Arbeit ergibt sich grundsätzlich aus zwei Bereichen. Einerseits erwachsen durch den Einsatz von Web-Techniken zur Implementierung von Informationssystemen bereits bestimmte Konsequenzen hinsichtlich Skalierbarkeit, Komplexität, Wartbarkeit oder realisierbarer Funktionalitäten. Andererseits ist im Rahmen dieser Arbeit neben diesen rein technischen Fragen auch der zunehmende Einsatz Web-basierter Informationssysteme für die elektronische Geschäftsabwicklung (*Electronic Commerce* oder kurz *E-Commerce*) von besonderem Interesse, da es sich hierbei um ein wesentliches Aufgabenfeld der Wirtschaftsinformatik handelt. Aus diesem Grund soll im folgenden zuerst versucht werden, die Schnittstellen zwischen Unternehmensprozessen und Informationssystemen im allgemeinen und Web-basierten Informationssystemen im speziellen aufzuzeigen und daraus ersichtliche mögliche Defizite oder Bereiche für Verbesserungen aufzudecken, um daran anschließend die Motivation und Aufgabenstellung für diese Arbeit zu definieren.

2.1 E-Commerce

Der Begriff des Electronic Commerce umfaßt ein weites Gebiet von möglichen Anwendungen zur Realisierung betriebswirtschaftlicher Abläufe mittels elektronischer Medien. Je nach Betrachtungsweise kann etwa zwischen der Verknüpfung von Endkunden und Unternehmen (*Business-to-Consumer* oder kurz *B2C*), Unternehmen untereinander (*Business-to-Business*, kurz *B2B*) und Verknüpfungen innerhalb von Unternehmen unterschieden werden. Legt man bei der Abwicklung von Transaktionen in diesem Zusammenhang einen elektronischen Markt zugrunde, so kann weiter nach Umfang der unterstützten Marktprozesse (Anbahnung, Vereinbarung, Abwicklung), der Marktbetreiber oder der Einbindung der Märkte in die beteiligten Instanzen differenziert werden [142].

Die Gründe für die elektronische Abwicklung von Geschäftsprozessen sind dementsprechend vielfältig. Generell lassen sich durch den Einsatz von E-Commerce Ziele wie Kostenreduktion, schnellere Produktzyklen, schnellere Reaktion auf Kundenanforderungen oder verbesserte Servicequalität erreichen [92]. Hierbei greifen insbesondere die geringeren Transaktionskosten und die höhere Automatisierbarkeit der Abläufe. Konkrete Vorteile durch E-Commerce ergeben sich allerdings nicht allein durch die Automatisierung von Geschäftsprozessen, sondern vielmehr durch deren Verbesserung und die Umsetzung neuer, vorher nicht durchführbarer Abläufe [100]. Ist es etwa möglich, durch den Einsatz elektronischer Medien zusätzliche Informationen wie z.B. spezielle Kundenwünsche oder genauere Produktdaten zu transportieren und auch zu verarbeiten, so kann sich dies in einer Verbesserung der Angebotsqualität und Kundenzufriedenheit niederschlagen. Ebenso kann die Mög-

lichkeit der direkten Ansprache von Kunden auf elektronischem Wege und die Abwicklung von Aufträgen (Produktauswahl, Auftragseingang, Zahlungsabwicklung) ohne Mittelsmänner zu einer Kostenreduktion führen.

Durch die Nutzung von elektronischer Transaktionsabwicklung zwischen Unternehmen ist es daneben möglich, daß sich die Geschäftsprozesse über die Unternehmensgrenzen hinweg erstrecken. Dies ist insbesondere bei vertikalen Verbindungen zwischen Unternehmen der Fall, beispielsweise beim *Supply Chain Management* (Zulieferer-Hersteller-Kette). Denkbar ist in diesem Fall, daß die Zuliefererbetriebe Informationen über die aktuellen Absatzzahlen des Endprodukteherstellers erhalten und dementsprechend die Herstellung von Vorprodukten effizienter disponieren können [174]. In Gegenrichtung könnte es dem Hersteller durch Bereitstellung detaillierter Produktdaten auf elektronischem Weg möglich sein, zwischen unterschiedlichen Anbietern von Vorprodukten auszuwählen, ohne dazu konventionelle Produktkataloge vergleichen zu müssen [73]. Dieses letzte Beispiel für elektronische Produktkataloge zeigt, daß es aus einer bestehenden „physikalischen Interoperabilität“ wie der Herstellung von Gütern aus unterschiedlichen Vorprodukten heraus notwendig sein kann, eine „virtuelle Interoperabilität“ anzustreben, welche die Effizienz der bestehenden Geschäftsprozesse erhöht bzw. neue Geschäftsprozesse ermöglicht. Gerade auf Geschäftsfeldern, bei denen eine Menge von unterschiedlichen Unternehmen miteinander kooperieren wie etwa dem Logistikbereich ist eine solche Interoperabilität unabdingbar.

Ähnliche Tendenzen sind auch innerhalb von Unternehmen erkennbar. Aus dem Bestreben, flexibler auf Marktgegebenheiten reagieren zu können und unternehmensinterne Abläufe zu optimieren, resultiert eine stärkere Modularisierung von Unternehmen und Veränderung von Organisationsformen, etwa die Umstellung von Geschäftsprozessen auf rein projektbasierte Abläufe (siehe auch hierzu [142]). Daraus ergeben sich aber auch erhöhte Anforderungen an die Kommunikations- und Koordinationsmechanismen in der Unternehmung. Diese können in bestimmten Grenzen durch den Einsatz von Informationstechnologie befriedigt werden, wobei auch hier die Interoperabilität zwischen den Unternehmenseinheiten im Vordergrund steht. Beispiele für eine solche Unterstützung sind etwa Dokumentenmanagementsysteme, Workflowsysteme oder Messaging-Systeme.

2.2 Web-basierte Informationssysteme und E-Commerce

Die fortschreitende Verbreitung von Informationstechnik im Endanwenderbereich erlaubt den Einsatz von E-Commerce zwischen Unternehmen und Endkunden. In diesem Zusammenhang spielen besonders das Internet und das World Wide Web eine zentrale Rolle. Sie schaffen eine für den Benutzer einfach zu nutzende und konsistente Schnittstelle zu Geschäftsabläufen wie dem Einkaufen per Internet, Homebanking oder Bezug von Informationsangeboten. Diese Schnittstelle realisiert die oben bereits angesprochene Interoperabilität zwischen Kunde und Unternehmen.

Daneben verstärkt sich die Tendenz, solche auf Web-Techniken aufbauenden Informationssysteme auch für E-Commerce-Anwendungen zwischen Unternehmen und innerhalb von Unternehmen einzusetzen. Allerdings ist eine solche Nutzung nur indirekt möglich, d.h. es ist nur mit einem gewissen Aufwand möglich, Geschäftsprozesse direkt miteinander zu verknüpfen. Dies liegt an der Ausrichtung der Web-Techniken an den Endanwender und nicht auf die automatisierte Kommunikation

zwischen Informationssystemen. So ist es beispielsweise nicht möglich, automatisiert beliebige Online-Kataloge im WWW nach Produktdaten exakt zu durchsuchen, da solche Kataloge für die Betrachtung durch Endanwender aufbereitet sind. Kurz gesagt sind Web-basierte Informationssysteme für den Business-to-Business-Bereich weniger gut geeignet als für den Business-to-Consumer-Bereich.

Dennoch stellen Web-basierte Informationssysteme eine wichtige Basis für E-Commerce-Anwendungen dar. Die Infrastruktur solcher Systeme ist flexibel genug, beliebige Inhalte zu transportieren, es existieren eine Reihe Standards die eine offene Kommunikation erlauben und schließlich sind die Kosten für den Einsatz von Web-Techniken im Regelfall vergleichsweise gering. Andersherum stellt der Bereich der elektronischen Geschäftsprozeßabwicklung den zweiten großen Anwendungsbereich für Web-Techniken nach dem ursprünglich angestrebten Ziel des vereinfachten Informationsaustauschs dar. Deswegen sollen die sich daraus ergebenden Anforderungen in die Motivation und Aufgabenstellung einfließen.

2.3 Aufgabenstellung

Wie im vorigen Unterkapitel dargestellt, sind Web-Techniken nur bedingt für die Implementierung von betrieblichen Informationssystemen geeignet. Sie erlauben die Modellierung der in solchen Systemen relevanten Informationen nur eingeschränkt, sondern stellen die Benutzerinteraktion in den Vordergrund. Zum anderen ist die Realisierung Web-basierter Informationssysteme stark an das Client-Server-Modell gebunden. Eine lose Kopplung der Systembestandteile und demnach eine Angleichung der Aufgaben von Client und Server ergänzt die Forderung nach stärkerer Betonung der Systeminformation.

Daraus ergibt sich die grundlegende Aufgabenstellung für diese Arbeit: Schaffung eines Modells, welches die Basis für Systeme ist, die die Vorteile Server- und Client-basierter Architekturen vereinigen, die entsprechenden Nachteile aber möglichst nicht aufweisen. Wenn Nachteile nicht vermieden werden können, so sollen zumindest Mechanismen definiert werden, mittels derer die Nachteile abgemildert werden können. Zentraler Bestandteil des Modells soll zudem die Beschreibung von Informationsstrukturen im Gegensatz zu Prozessen werden, d.h. die Daten sollen im Vergleich zu herkömmlichen Web-basierten Informationssystemen eine wesentlich stärkere Rolle spielen. Die Vertiefung dieser Anforderungen erfolgt bei der Anforderungsanalyse für das zu entwickelnde Modell in Kapitel 11 und bei Diskussion der unterschiedlichen Architekturen und verwandter Bereiche zugrundeliegenden Techniken (Kapitel 12). Neben der Entwicklung eines Architekturmodells für verteilte, Web-basierte Systeme muß dieses Modell auch auf Anwendbarkeit hin überprüft werden. Demnach gliedert sich die Arbeit in sechs Abschnitte:

Grundlagen: Beschreibung der Aufgabenstellung, der benötigten Basistechniken und verwandter Bereiche.

Modell: Identifikation und Analyse der Anforderungen an ein Architekturmodell, Definition des Modells.

Methode: Beschreibung einer Entwicklungsmethode, welche das vorgestellte Modell benutzt und die systematische Erstellung verteilter, Web-basierter Systeme zum Ziel hat.

Anwendungen: Beschreibung von beispielhaften Anwendungen von Modell und Methode.

Werkzeuge: Vorstellung von Werkzeugen, welche das Modell und die Methode für die Entwicklung und Benutzung verteilter, Web-basierter Systeme unterstützen.

Ausblick: Bewertung der erreichten Ergebnisse und Schlußbetrachtung.

2.4 Das AHDM als Instrument der Wirtschaftsinformatik

Die Wirtschaftsinformatik als Wissenschaft und dementsprechend das Forschungsgebiet der Wirtschaftsinformatik kann auf unterschiedliche Art und Weise eingeordnet werden. Entsprechend der Definition aus [78] beschäftigt sich die Wirtschaftsinformatik mit der Untersuchung von Informations- und Kommunikationssystemen (IKS), Isolierung von Teilsystemen und deren Integration. Mit den Begriffen *Informations-* und *Kommunikationssystem* wird die Bedeutung der Information für die Steuerung betriebswirtschaftlicher Abläufe und die Kommunikation zwischen den daran beteiligten Aufgabenträgern in offenen Systemen hervorgehoben. Die o.g. Definition definiert weiterhin als Ziel wissenschaftlicher Untersuchungen der Wirtschaftsinformatik die „Gewinnung von Theorien, Methoden, Werkzeugen und intersubjektiv nachprüfbaren Erkenntnissen über/zu IKS und die Ergänzung des ‚Methoden- und Werkzeugkastens‘ der Wissenschaften ...“. Vergleicht man die bis hierher erarbeitete Aufgabenstellung mit diesem Ziel, so ist zu klären, ob die Entwicklung eines Architekturmodells und einer Entwicklungsmethode für Web-basierte Informationssysteme dem Ziel entspricht. Dazu ist eine nähere Definition des Theoriebegriffs und dem in dieser Arbeit verwendeten Modellbegriffs notwendig.

Einen Überblick liefert hierzu [105], an den sich auch die weiteren Ausführungen in diesem Unterkapitel anlehnen. Demnach kann unter einer Theorie ein konsistentes System von Aussagen über Sachverhalte oder Problemstellungen verstanden werden. Je nach Verwendungszweck einer Theorie kann weiter festgehalten werden, das Theorien auch ein Aussagensystem über die Lösung von Problemstellungen umfassen können. Eng mit dem Theoriebegriff ist der Modellbegriff verwandt. Die Definition dieses Begriffs umfaßt im traditionellen Sinn die Abbildung der Realität nach bestimmten Kriterien wie z.B. struktureller, funktionaler oder Verhaltensähnlichkeit. Im weiteren Sinn können auch Theorien als Modelle angesehen werden, da sie ja einen bestimmten Sachverhalt der Realwelt in ein Aussagensystem überführen. Die Verbindung zwischen Modell- und Theoriebegriff in dieser Arbeit soll darin bestehen, daß das zu entwickelnde Modell als formale Struktur einer Theorie über die Architektur Web-basierter Informationssysteme, in deren Zentrum die bearbeitete Information steht, dient.

Betrachtet man die mögliche Funktion von Theorien und Modellen, so ist eine Einteilung in erklärende und gestaltende Theorien möglich. Sie hängen eng mit einem pragmatischen und theoretischen Wissenschaftsziel zusammen. Das theoretische Wissenschaftsziel umfaßt vornehmlich die Erklärung, Prognose und Überprüfung von Sachverhalten, Theorien oder Modellen mit Hilfe von Empirie und Falsifizierung. Davon abzugrenzen ist das pragmatische Wissenschaftsziel, das weniger an der Erklärung als an der Gestaltung der Realität interessiert ist. In der Wirtschaftsinformatik dominiert das pragmatische Wissenschaftsziel (vgl. auch [94] zu dieser Einteilung).

Demnach soll auch das zu entwickelnde Modell gestaltenden Charakter haben, d.h. es soll „planvolles Handeln zur Lösung realer Probleme“ ermöglichen. Das Modell

für aktive Hypertextdokumente erfüllt diese Forderung insofern, als daß die Entwicklung des Modells von Anforderungen ausgeht, die aus der Anwendung Web-basierter Informationssysteme erwachsen. Das AHDM liefert im Kern Konstrukte, um die gefundenen Anforderungen und Aufgabenstellungen zu erfüllen. Darüberhinaus wird wie oben erwähnt eine Methode zur Anwendung der AHDM vorgeschlagen, so daß von einer konkreten Handlungsanweisung gesprochen werden kann.

2.5 Begründung der Vorgehensweise

Eine Theorie oder ein Modell sind dann als wissenschaftlich zu erachten, wenn sie begründbar, allgemeingültig und kritisierbar sind. Dementsprechend wird in dieser Arbeit versucht, das Modell aufbauend auf einer Analyse bestehender Sachverhalte und Anforderungen aufzubauen. Die gewonnen Erkenntnisse sollen dabei dazu dienen, neue Lösungsansätze mit bekannten Techniken zu kombinieren. Konkret wird in der Arbeit ein neuer Ansatz zur Verknüpfung von menschlich verstehbarer (textuell repräsentierter) Information mit automatisierbarer Logik (Programmen) vorgestellt, für den eine Modellierungssprache vorgeschlagen wird. Diese Sprache wird konstruktiv eingesetzt, um beispielhaft ausgewählte Anwendungsmöglichkeiten des Ansatzes aufzuzeigen und Prototypen zu erstellen und somit die Gültigkeit der Eigenschaften des Modells zu demonstrieren [164]. Die vorgestellten Anwendungen und die dabei benutzten Werkzeuge sollen auch zur Überprüfbarkeit und Wiederholbarkeit der gewonnenen Ergebnisse dienen. Die Arbeit trägt somit zu einem gewissen Ausmaß konstruktivistische Züge, beschränkt sich dabei jedoch nicht auf den reinen Konstruktivismus [163], sondern erprobt die Ergebnisse in der Realwelt, was eine Falsifikation ermöglicht. Die Kritisierbarkeit soll durch eine möglichst formale Beschreibung des Modells, der Entwicklungsmethode und damit zusammenhängender Techniken gewährleistet werden.

Es kann diskutiert werden, ob nicht entweder eine Ableitung des Modells aus einer vollständigen Anforderungsanalyse oder das vollständige Überprüfen des Modells mittels praktischer Anwendungen ausreichen würde. Es ist jedoch offensichtlich, daß einerseits die Entwicklung des Modells aus einer Anforderungsanalyse stark von deren Qualität und Umfang abhängig ist, andererseits wie erwähnt eine Überprüfung des Modells anhand von Anwendungen im Umfang dieser Arbeit nicht vollständig erfolgen kann. Durch die hier gewählte Kombination aus induktivem Herleiten und deduktivem Überprüfen soll dennoch versucht werden, die Allgemeingültigkeit und Begründbarkeit des Modells abzusichern.

3 Web-basierte Informationssysteme

Bevor die Entwicklung des Modells begonnen werden kann, sollen zuerst die grundlegenden Begriffe definiert werden. Dazu zählt an erster Stelle der Bereich der schon angesprochenen Web-basierten Informationssysteme. Sie entstehen aus der Anwendung von Web-Techniken in einer Client-Server-Konfiguration, wie sie in [193] im Detail erläutert ist. Ein solches Web-basiertes Informationssystem besteht danach aus einem Informationsanbieter und einem Informationsnachfrager, welche auf der technischen Ebene in der einfachsten Form durch ein Server-System (den Web-Server) und ein Client-System (den Web-Browser) vertreten sind [162].

Die Verbindung zwischen beiden Systembestandteilen ist bidirektional, so daß nicht nur Daten vom Anbieter zum Nachfrager übertragen werden können, sondern auch in die umgekehrte Richtung, etwa um Daten auf dem Server abzulegen oder bearbeiten zu lassen. Die Ausgangsinformation kann im Normalfall vom Server-System ohne hohen Aufwand (also beispielsweise per Dateizugriff oder über eine lokale Netzverbindung) abgerufen werden. Ähnlich verhält es sich mit dem Transport vom Client-System (typischerweise ein Web-Browser) zum Benutzer. Der Aufwand der Übertragung der Daten vom Server zum Client hingegen ist meist höher, wobei hier Zeitverbrauch, Ressourcenverbrauch oder auch Kosten entstehen.

Legt man eine typische Anwendungsarchitektur zugrunde, wie sie in [77] beschrieben ist, und die von einer Dreiteilung in die Ebenen „Datenhaltung“, „Anwendungssteuerung“ und „Präsentation“ ausgeht, ergibt sich das in Abbildung 2 dargestellte Bild. Zu beachten ist dabei, daß die Anwendungssteuerung innerhalb des Server-Systems nur sehr beschränkte Aufgaben hat, da an dieser Stelle nur einfache Anfragen des Client-Systems bearbeitet werden (siehe dazu auch die Ausführungen zu HTTP in Unterkapitel 6.1). Dabei handelt es sich um die einfachste Variante Web-basierter Informationssysteme.

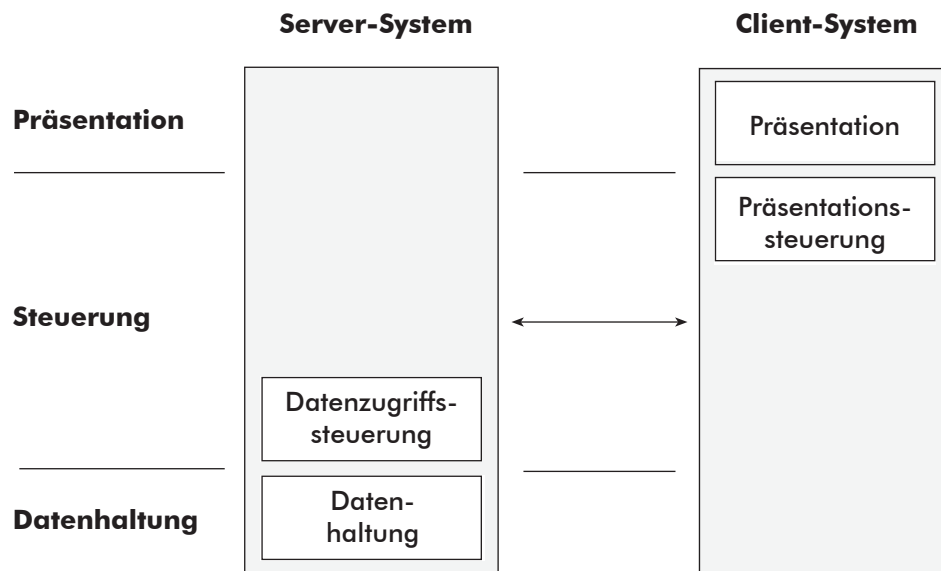


Abb. 2: Einfachste Variante der Schichtenarchitektur in Web-Server und -Client

3.1 Erweiterungen

Die oben dargestellte einfachste Form eines Web-basierten Informationssystem, welche nur aus einem Server- und einem Client-System besteht, kann auf unterschiedliche Art und Weise erweitert werden. Dazu zählen beispielsweise Zugriffs-kontrollmechanismen und Verminderung des Kommunikationsaufwands durch Zwischenspeicherung von Daten im Client-System. Die wichtigste Erweiterung besteht aber in der Möglichkeit, mittels Web-Techniken vorhandene Information nicht nur zu statisch zu verteilen, sondern auch neue Information dynamisch zu generieren oder bestehende Information zu modifizieren. Dadurch kann u.a. aus verschiedenen

Quellen stammende Ausgangsinformation (z.B. aus Datenbanken) für die Verwendung in einem Web-basierten System dynamisch umgeformt werden.

Zu diesem Zweck sind in letzter Zeit mehrere unterschiedliche Ansätze entstanden. Grundsätzlich lassen sie sich in zwei große Kategorien unterteilen:

- die Server-basierten Systeme, welche die dynamische Aufbereitung der Informationen auf Seiten des Informationsanbieters vornehmen.
- die Client-basierten Systeme, bei denen die Information dynamisch vom Informationsnachfrager weiterverarbeitet wird.

Die dynamische Verarbeitung von Informationen unterscheidet sich insofern von der rein statischen Distribution durch das Server-System, als sie die Basis für eine übergeordnete Anwendungssteuerung formen kann. Zudem ist es in den meisten Fällen möglich, nicht nur eine, sondern mehrere Anwendungen auf diese Weise zu realisieren. In einer solchen Konstellation ist die Steuerung für die Anwendung in den Erweiterungen von Server- und bzw. oder Client-System implementiert. Abbildung 3 zeigt die Architektur und die Bestandteile solcher erweiterter Systeme.

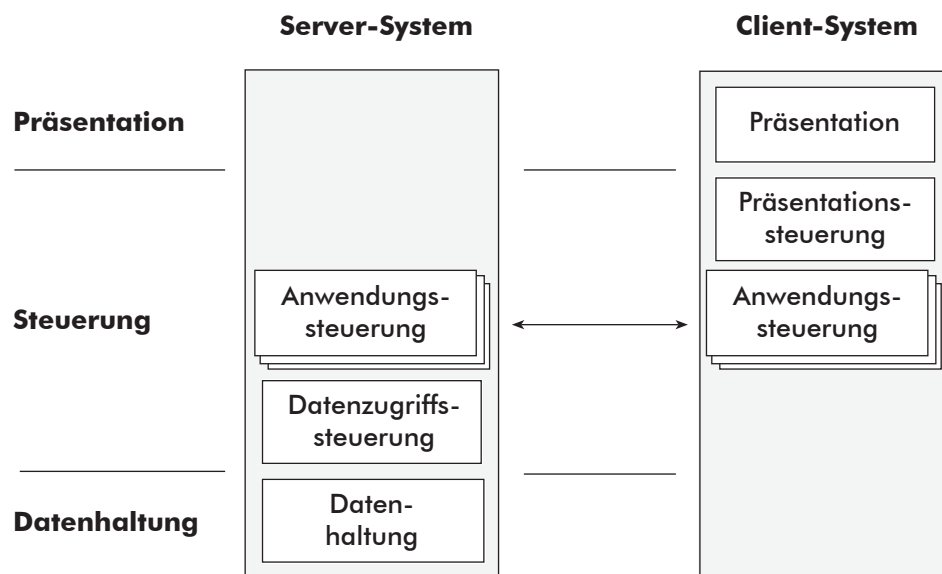


Abb. 3: Erweiterte Architektur Web-basierter Informationssysteme

In Zusammenhang mit der einfachen Betrachtung des Aufwands für die Informationsübermittlung und der Einbeziehung weiterer Kriterien lassen sich für die Realisierung als Server- oder Client-basiertes System bzw. für eine unterschiedliche Wahl der Schwerpunkte auf Client oder Server Vor- und Nachteile bei der Verwendung identifizieren (siehe auch [134] für eine Diskussion unterschiedlicher Ausprägungen solcher Systeme).

Server-basierte Systeme

Server-basierte Systeme reduzieren durch ihren zentralen Ansatz die Anforderungen an die Client-Systeme, da sie die Datenverarbeitung selbst durchführen und die Client-Systeme die Daten nur darzustellen brauchen. Weiterhin sinkt der Verwaltungsaufwand, da jegliche Ablauflogik zur Generierung und Modifikation von Daten an einem Ort liegt und bei Änderungen leicht zugreifbar ist. Nachteilig ist aber die ge-

ringe Flexibilität, da diese Änderungen nur vom Informationsanbieter vorgenommen werden können. Dieser muß demnach die möglichen Nutzungen der angebotenen Information durch die Nachfrager soweit wie möglich vorausplanen. Zudem ist bei komplexeren Anwendungen der Kommunikationsaufwand hoch, da jede Anfrage in der Übertragung der gesamten modifizierten und neuen Information resultiert.

Client-basierte Systeme

Client-basierte Systeme sind flexibler, da sie es dem Nachfrager erlauben, die Aufbereitung der Ausgangsinformation je nach Bedarf abzuändern. Weiterhin reduzieren sie den Kommunikationsaufwand, da normalerweise nur einmal die Ausgangsinformation übertragen werden muß, die benötigten Modifikationen können dann auf Nachfragerseite durchgeführt werden. Andererseits aber entstehen so höhere Anforderung an die Client-Systeme, welche in der Lage sein müssen, die Verarbeitung der Daten selbst vorzunehmen. Auch steigt der Verwaltungsaufwand für das gesamte Systems, da die Ablauflogik verteilt in den Client-Systemen liegt. Nicht zuletzt sind zudem die unterschiedlichen Techniken, mittels derer Client-basierte Systeme realisiert werden können, zueinander nicht oder nur unvollständig kompatibel, so daß der Informationsnachfrager im ungünstigsten Fall nicht nur ein einzelnes sondern gleich mehrere Systeme benutzen muß.

3.2 Aufgabengebiete

Die Aufgabengebiete, die durch ein Web-basiertes Informationssystem abgedeckt werden können, lassen sich grundsätzlich aus den Aufgaben von Hypertext-Systemen ableiten (beschrieben etwa in [16] oder [20]). Teilweise hängen sie aber auch von der technischen Realisierung (und damit von den Möglichkeiten, die die Basistechniken bieten) und der Verwendung von Erweiterungen ab. Interessant bei der Analyse aktueller Methoden, Techniken und Werkzeuge ist aber, daß eine der ursprünglich im Vordergrund stehenden Aufgaben, nämlich die einfache Erstellung von Hypertexten, in den Hintergrund getreten ist.

An dieser Stelle sollen diese Aufgabengebiete kurz aufgezählt werden, so daß ihnen später die Basistechniken Web-basierter Informationssysteme zugeordnet werden können.

Informationsstrukturierung: Strukturierte Information stellt die Basis für Vernetzung, Darstellung und Interaktion durch Markierung relevanter Bereiche in den im System vorhandenen Daten dar. Zudem kann strukturierte Information Grundlage für die automatische Weiterverarbeitung sein.

Informationsvernetzung: Die Vernetzung von Daten ist ein Hauptcharakteristikum von Hypertextsystemen. Dabei kann zwischen Verknüpfungen als Bestandteil der zu verknüpfenden Daten oder als externen Bestandteil unterschieden werden. Im ersten Fall ist eine Strukturierung- und Annotationsmöglichkeit der Ausgangsinformation notwendig.

Kommunikation: Die Verteilung und der Zugriff auf die vorhandenen Informationen geschieht mittels bestimmter Kommunikationstechniken, die u.a. die Adressierung von Daten und die Übertragungssemantiken festlegen.

Informationsdarstellung: Um die Information für Endbenutzer sinnvoll bereitzustellen, ist eine Aufbereitung für die Darstellung etwa auf dem Bildschirm oder Drucker

notwendig. Auch hier ist eine Strukturierung der Information für eine differenzierte Darstellung Voraussetzung.

Interaktion: Jegliche Form von Aktivität in Web-basierten Systemen wird durch unterschiedliche Arten der Interaktion des Benutzers mit dem System selbst ausgelöst. Weiter gefaßt zählt dazu auch die Interaktion anderer Systeme mit dem System.

In den folgenden Kapiteln sollen die für diese Arbeit wichtigen Basistechniken erläutert werden, welche die zur Zeit verfügbaren Techniken darstellen, mit denen Web-basierte Informationssysteme realisiert werden können. Sie können als Basis des in dieser Arbeit vorgestellten Ansatzes dienen oder zumindest in Teilen darin einfließen. Bei der Beschreibung der Techniken wird nur auf die für das vorgestellte Modell und seine Anwendungen relevanten Eigenschaften, Einschränkungen u.a. eingegangen. Für weitergehende Information sei auf die referenzierte Literatur verwiesen.

4 Informationsstrukturierung

Die am flexibelsten einsetzbaren Daten innerhalb von Web-basierten Informationssystem sind Daten, die als Texte vorliegen. Sie können durch Auszeichnungen annotiert und weiter strukturiert werden. Eine sehr wichtige Anwendung dieses Mittels ist die Annotation von Textstellen mit Vernetzungsinformation. Dazu kommt die Möglichkeit, bei der Erstellung von Dokumenten eine Struktur vorzugeben, um den Dokumentenaustausch und das Verständnis für den Inhalt zu erleichtern.

4.1 Standardized General Markup Language

Die *Standardized General Markup Language*, kurz SGML [71], ist die wichtigste Basistechnik innerhalb des hier vorgestellten Ansatzes. SGML ist eine Metasprache für Auszeichnungssprachen, welche festlegen, wie Textdokumente annotiert und strukturiert werden können. Die Verwendung von SGML besteht aus zwei unterschiedlichen Aufgaben: erstens der Definition von Regeln für die Auszeichnung von Text und zweitens der Anwendung der Regeln bei der Erstellung danach strukturierter Dokumente. SGML-Dokumente werden einerseits von Endbenutzern bearbeitet und gelesen, andererseits aber auch maschinell weiterverarbeitet. Systeme, die SGML-Dokumente verarbeiten, werden als SGML-Prozessoren bezeichnet.

Die Regelfestlegung zur Auszeichnung von Texten, welche mit SGML möglich ist, beschränkt sich grundsätzlich nur auf die Struktur und die Syntax eines Dokuments. Die Semantik der Regeln und der Annotationen ist nicht Bestandteil von SGML und kann bestenfalls nur mittels zusätzlicher, informeller Mechanismen wie beispielsweise Kommentare oder Namenskonventionen erreicht werden. Weiterhin beschränkt sich SGML auf die *deklarative* Auszeichnung von Text, im Gegensatz zur *prozeduralen* Auszeichnung, die festlegt, wie der annotierte Text weiter verarbeitet (angezeigt, gedruckt) werden soll.

Logische Struktur

In der Terminologie von SGML werden die Auszeichnungsregeln in einer *Document Type Definition* (DTD) festgelegt. Die danach erstellten Dokumente sind *Instanzen* dieser DTD. Die Regeln, die in einer DTD enthalten sind, entsprechen einer formalen Sprache. Hauptbestandteil sind dabei *Elemente*. Elemente können verschachtelt wer-

den. Die möglichen Verschachtelungen innerhalb eines Elements werden als *Content Model* bezeichnet, hier können u.a. obligatorische Bestandteile, Sequenzen und Alternativen definiert werden. Die Abkürzung DTD steht im übrigen auch für den Begriff *Document Type Declaration*, mit dem die Verwendung einer DTD in einem Dokument deklariert wird.

Das wichtigste Konstrukt innerhalb von SGML-Dokumenten ist der *Tag*. Ein Tag wird im Text durch eine öffnende spitze Klammer eingeleitet und durch eine schließende spitze Klammer beendet. Sie werden für drei Aufgaben eingesetzt:

Descriptive Markup

Strukturierung, Annotation und Auszeichnung von Textinhalten mittels Elementen. In diesem Fall wird ein Element durch einen *Start-Tag* eingeleitet und durch ein *Ende-Tag* abgeschlossen. Der Start-Tag hat als ersten Bestandteil den *Generic Identifier* (Namen) des Elements, im Ende-Tag wird diesem Bezeichner noch ein Schrägstrich („/“) vorangestellt. Der Start-Tag kann zusätzlich Attribute enthalten (mehr dazu siehe unten).

Zu beachten ist hierbei, daß Elemente in der DTD so deklariert sein können, daß Start- und auch Ende-Tag optional ist. Ist nur der Ende-Tag optional, muß der SGML-Prozessor aufgrund des Kontextes und der in der DTD festgelegten Regeln feststellen, wann das Ende des Elements erreicht ist. Ist auch der Start-Tag optional, so bedeutet dies, daß zwar die Tags nicht innerhalb des Dokuments auftreten müssen, das Element aber trotzdem Bestandteil der logischen Struktur ist. Der Platz innerhalb der logischen Struktur ist dann durch die Content Models der Elemente definiert.

Markup Declaration

Deklaration von Regeln für die Strukturierung von Dokumenten im Rahmen von DTDs. Tags, welche für die Markup Declaration benutzt werden, werden durch ein Ausrufezeichen („!“) vor dem Generic Identifier gekennzeichnet. Für die Markup Declaration existieren vier spezielle Tags, welche im folgenden kurz informell beschrieben werden.

Doctype: leitet eine DTD-Referenz oder -Deklaration ein. Eine Referenz ist durch einen Public- oder System Identifier realisiert (siehe unten). Die dabei referenzierte DTD kann innerhalb des Doctype-Tags durch Element- und Attributdeklaration ergänzt werden (dem sog. *Internal Subset*). Das Internal Subset ist von SGML-Prozessoren vor der externen DTD zu bearbeiten. Zusätzlich wird im Doctype-Tag das Wurzel-Element des Dokuments festgelegt. Der Doctype-Tag ist für ein gültiges SGML-Dokument obligatorisch. Abbildung 4 zeigt eine sehr vereinfachte schematische Darstellung einer DTD-Referenz mit einem Beispiel.

Element: definiert ein Element, dazu gehört der Name des zu definierenden Elements, Information über optionale Start- und Ende-Tags sowie das Content Model. Der Element-Tag kann entweder im Internal Subset des Doctype-Tags oder in der externen DTD benutzt werden.

Attlist: Die Attribute, die für ein Element verwendet werden können, werden mit dem Attlist-Tag deklariert. In der Deklaration werden Attribute eindeutige Namen und Wertebereiche bzw. Datentypen zugewiesen. Attribute präzisieren die Bedeutung des Elementes, dem sie zugeordnet sind. Die Werte von Attributen können Zeichenketten oder einfache Bezeichner (*Token*) sein und können bei der Deklaration mit Vorgabewerten belegt werden. Ein Sonderfall ist die

Schematische DTD-Referenz:

```
"<!DOCTYPE" Wurzel-Element DTD-Referenz "["  
    Internal Subset  
"]>"
```

Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN" [  
    ...  

```

Abb. 4: Aufbau und Beispiel einer DTD-Referenz

Schematische Element-Deklaration:

```
"<!ELEMENT" Elementname Start-Tag? Ende-Tag? Content-Model ">"
```

Beispiel:

```
<!ELEMENT HTML O O (HEAD, BODY)>
```

Abb. 5: Aufbau und Beispiel einer Element-Deklaration

Deklaration eines Attributwerts als *Notation*. In diesem Fall beschreibt der Wert des Attributs, wie der Inhalt des zugeordneten Elements zu interpretieren ist (beispielsweise durch eine Weiterverarbeitung mittels externen Programme). Ein Sonderfall ist auch die Deklaration von Attributen als ID- oder IDREF-Attribute. Ein ID-Attribut hat einen im Dokument eindeutigen Wert, der über ein IDREF-Attribut referenziert werden kann. Somit ist eine einfache Verknüpfung von Elementen in einem Dokument möglich. Werden mehrere Attributlisten für ein Element deklariert, so werden die Attributlisten zusammengelegt. Wird dabei ein Attribut mehr als einmal deklariert, so ist die erste Deklaration bindend. Auch der Attlist-Tag kann im Internal Subset oder einer externen DTD verwendet werden.

Schematische Attributlisten-Deklaration:

```
"<!ATTLIST" Elementname  
    Attributname Wertebereich/Vorgabewert  
    ...  
>"
```

Beispiel:

```
<!ATTLIST FONT  
    size CDATA #IMPLIED  
    color CDATA #IMPLIED  
    face CDATA #IMPLIED  
>
```

Abb. 6: Aufbau und Beispiel einer Attributlisten-Deklaration

Entity: Deklaration von Referenzen auf andere physikalische Bestandteile des Dokuments. Entities sind Bestandteil der physikalischen Struktur eines Dokuments und werden weiter unten definiert.

Schematische Entity-Deklaration:

```
<!ENTITY " Entity-Name
          Entity-Wert | Entity-Identifizier (PUBLIC oder SYSTEM)
">
```

Beispiel:

```
<!ENTITY Autor "Eckhart Köppen">
```

Abb. 7: Aufbau und Beispiel einer Entity-Deklaration

Processing Instructions

Eine grundlegende Idee bei der Verwendung von SGML ist die Trennung von Inhalt und Repräsentation oder auch weitergefaßt jeglicher nicht inhaltlich relevanter Information. Trotzdem kann es wünschenswert sein, zusätzliche Information, die nicht zum Inhalt des Dokuments gehört, in ein Dokument aufzunehmen. Zu diesem Zweck können Processing Instructions eingesetzt werden. Sie werden durch einen speziellen Tag gebildet, bei dem direkt nach der öffnenden spitzen Klammer und vor der schliessenden spitzen Klammer ein Fragezeichen („?“) steht. Die weiteren Bestandteile sind nicht definiert und können von SGML-Prozessoren beliebig verwendet oder auch ignoriert werden. Eine mögliche Verwendung besteht in der Definition von Layout-Angaben.

Attribute versus Elemente

Bei der Erstellung einer DTD ist zu entscheiden, welche Informationen als Elemente in den Instanzen annotiert werden können, und welche Attribute den Elementen zugeordnet. Eine eindeutige Regel gibt es an dieser Stelle nicht, da Attribute auch als Unterelemente eines Elements realisiert werden können, es gibt aber einige Leitlinien für die Verwendung von Attributen und Elementen:

- Elemente tragen zur Struktur des Dokuments bei, Attribute jedoch nicht. Das heißt, daß der Inhalt eines Dokuments immer noch vollständig strukturell beschrieben ist, wenn alle Attribute entfernt werden.
- Attribute präzisieren die Bedeutung eines Elements. Dadurch wird die Anzahl der zu deklarierenden Elemente verringert, da ähnliche Elemente zusammengefaßt und durch Attribute unterschieden werden können.
- Attribute haben Einschränkungen bezüglich des Inhalts und Umfangs ihrer Werte. Der Inhalt von Elementen kann beliebig umfangreich und strukturiert sein.
- Werden zur Erweiterung einer DTD neue Elemente definiert, so ist das Content Model der möglichen umschließenden Elemente zu ändern. Bei der Erweiterung der Attribut-Listen von Elementen wird die Deklaration der neuen Attribute den bestehenden Attributen einfach hinzugefügt.
- Elemente haben neben der hierarchischen Beziehung untereinander auch eine Reihenfolgebeziehung, die sich mit Attributen nicht abbilden läßt.

- Es kann nicht mehr als ein Attribut mit dem gleichen Namen pro Element existieren.

Die Entscheidung, Eigenschaften eines Dokuments in der DTD entweder als Attribut oder als Element zu definieren, hängt zusätzlich in großem Maße vom Einsatzgebiet der Dokumente und der DTD ab. Sind nachträgliche Erweiterungen (z.B. im Internal Subset) wahrscheinlich, so ist das Hinzufügen von Elementen bei strikten Content Models der schon vorhandenen Elemente problematischer als das Hinzufügen neuer Attribute.

Physikalische Struktur

Neben den logischen Strukturen wird in SGML auch eine physikalische Struktur für Dokumente definiert. Basis der physikalischen Struktur sind die *Entities*, welche die physikalischen Bestandteile eines Dokuments enthalten. In Entities sind unterschiedliche Datenformate vorgesehen:

Character Data: *Character Data* (CDATA) besteht aus nicht zu verarbeitenden Zeichen, d.h. die Daten werden nicht von einem Parser weiterverarbeitet.

Parsed Character Data: *Parsed Character Data* (PCDATA) besteht aus Zeichen, die von einem Parser weiterverarbeitet werden, so daß einige Zeichen eine bestimmte Bedeutung haben. Dazu zählen beispielsweise die Zeichen, die eine Markup-Deklaration einleiten oder der Beginn eines Tags oder einer Entity-Referenz (siehe unten).

Entities lassen sich in drei Gruppen einteilen:

Character Entities: Character Entities erlauben es, Zeichen innerhalb eines Dokuments zu benutzen, die nicht im Ausgangssatz des Dokuments enthalten sind, z.B. Umlaute, graphische Zeichen oder mathematische Symbole.

Parameter Entities: Parameter Entities werden innerhalb von DTDs benutzt, um die Deklaration von Elementen und Attributen zu vereinfachen, ähnlich der Verwendung von Makros in Programmiersprachen.

General Entities: General Entities stellen eigenständige Bestandteile eines Dokuments dar.

Entities werden in ein Dokument durch eine Entity-Referenz eingebunden, wobei die Parameter Entities nur innerhalb von DTDs und im Internal Subset referenziert werden dürfen. Die Referenz eines Entities geschieht bei Character und General Entities durch ein kaufmännisches Und („&“), gefolgt vom Namen des Entities und einem optionalen Semikolon. Bei Parameter Entities wird statt des kaufmännischen Und ein Prozentzeichen („%“) verwendet.

Die Deklaration von Entities erfolgt durch eine Markup Declaration mit dem *Entity-Tag*. In der Deklaration wird entweder direkt der Text angegeben, der bei einer Entity-Referenz eingesetzt werden soll, oder es wird auf ein externes Dokument verwiesen, welches den Ersetzungstext enthält. Dazu dienen *Public Identifier* und *System Identifier*. System Identifier geben direkt einen Speicherort für das Entity an (beispielsweise einen Dateinamen), während Public Identifier einen eindeutigen Namen enthalten. Dieser eindeutige Name wird vom SGML-Prozessor über eine *Catalog*-Datei auf einen entsprechenden System Identifier abgebildet. Die Identifikation und Einbindung von Entities wird in der gleichen Art und Weise auch für die Identifikation von DTDs benutzt, die demnach eine spezielle Art von Entities darstellen.

Abbildung 8 zeigt eine mögliche physikalische Aufteilung eines logischen Dokuments. Dort referenziert ein Kerndokument untergeordnete Dokumente (externe Entities) direkt über System Identifier, die DTD und andere Entities werden über Public Identifier eingebunden. Die Public Identifier werden über eine Catalog-Datei in System Identifier überführt.

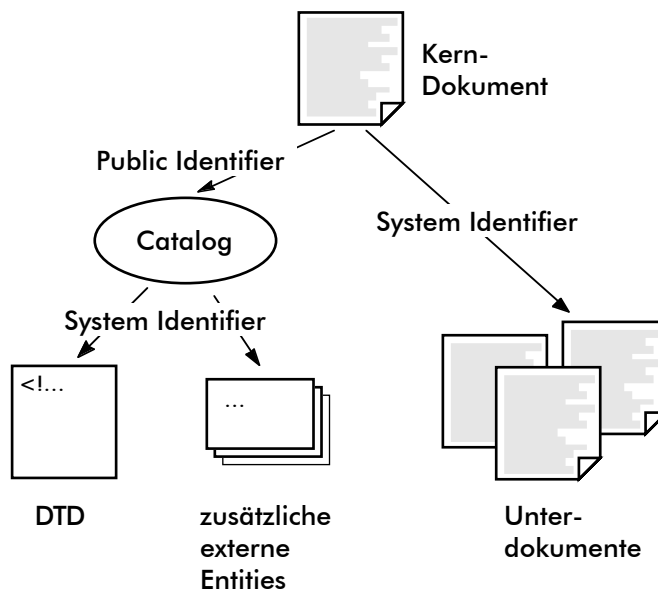


Abb. 8: Mögliche physikalische Aufteilung eines Dokuments

Eine solche physikalische Struktur wird vom SGML-Prozessor bei der Verarbeitung aufgelöst. Entity-Referenzen werden durch den eigentlichen Inhalt des Entities ersetzt, bis der resultierende Text keine Entity-Referenzen mehr enthält. Vergleicht man diese physikalische Struktur mit einem Hypertext-System, so ist zu beachten, daß der SGML-Prozessor die einzelnen Teile über die Entity-Referenzen zu einem sequentiellen Dokument zusammenfügt, während bei Hypertext-Systemen die ursprüngliche Struktur nicht verändert wird. Vielmehr werden die Verbindungen durch Benutzerinteraktion aktiviert.

4.2 Hypertext Markup Language

Die im Kontext von Web-basierten Informationssystemen wichtigste Anwendung von SGML ist die *Hypertext Markup Language* (HTML). HTML ist insofern eine Anwendung von SGML, als mit HTML eine DTD für Hypertextdokumente definiert ist. Zusätzlich werden in HTML aber auch Semantiken für die deklarierten Elemente festgelegt. Diese Semantiken beschreiben zum einen die Bedeutung der Elemente, zum anderen wird dadurch das Erscheinungsbild der Elemente definiert.

Ein großes Problem in HTML ist die fehlende Erweiterungsmöglichkeit. Grundsätzlich sollte es zwar möglich sein, im Internal Subset der DTD-Referenz die im Dokument benötigten Elemente zu deklarieren, allerdings sind die wenigsten HTML-Anzeigewerkzeuge auch SGML-Prozessoren. Zudem ist es nicht möglich, die Semantiken der neu deklarierten Elemente (insbesondere das Erscheinungsbild) derart zu beschreiben, daß die Anzeigewerkzeuge dies berücksichtigen könnten. Erst in

letzter Zeit ist mit den *Cascading Stylesheets* (CSS) ein Beschreibungsmittel für die Darstellung von Elementen festgelegt worden (siehe dazu Unterkapitel 7).

Die wichtigsten Semantiken, die für die HTML-Elemente definiert sind, lassen sich in folgende Gruppen einteilen:

Textstrukturierung: Definition von Überschriftselementen, Absatzelementen, Hervorhebungen, Aufzählungen, Tabellen und Listen.

Informationsvernetzung: Definition von Elementen für Hyperlinks (Verbindungen zwischen und innerhalb von Dokumenten) und andere Vernetzungsmöglichkeiten.

Interaktivität: Elemente, die die Eingabe von Daten durch den Benutzer und eine Weiterverarbeitung ermöglichen, darunter sind beispielsweise Eingabefelder und Auswahlfelder, die in sog. Forms zusammengefaßt werden.

Einbettung: Einbindung von Dokumenten, die einen anderen Inhaltstyp haben (z.B. Bilder) oder deren Inhalt nicht zum Inhalt des einbettenden Dokuments beiträgt sondern eine besondere Bedeutung hat (insbesondere ausführbare Programmskripte).

Präsentation: Einfache Steuerung von Darstellungsaspekten wie etwa Hervorhebungen, Zeilenumbrüche oder Farbauswahl.

4.3 Extensible Markup Language

Die *Extensible Markup Language* (XML) entstand aus dem Bestreben, die Anwendung von SGML und DSSSL im World Wide Web zu fördern [30]. XML ist eine vom W3-Konsortium als Empfehlung verabschiedete vereinfachte Weiterentwicklung von SGML, welche durch eine Reihe weiterer Empfehlungen ergänzt wird.

Ziele von XML

Die Anforderungen an das Design und den Entwurfsprozeß von XML wurden zu Beginn der Entwurfsphase vom *SGML Editorial Review Board* am W3C festgelegt. Ihre nähere Erläuterung nach [30] soll an dieser Stelle zum weiteren Verständnis für die tatsächlichen Eigenschaften von XML aufgeführt werden.

Direkte Einsetzbarkeit im Internet

Das Design von XML soll Internet-Anwendungen direkt unterstützen, d.h. die Anforderungen verteilter Anwendungen auf Basis von Internet-Techniken in großen Netzwerken erfüllen. Es ist allerdings kein Entwurfsziel, XML in existierenden Web-Browsern sofort einsetzen zu können, also XML als Ersatz für HTML zu benutzen.

Unterstützung eines großen Anwendungsbereichs

Obwohl XML hauptsächlich für die Implementierung Internet-basierter Systeme geeignet sein soll, ist die Nutzung von XML nicht nur auf solche Anwendungen beschränkt. So sollen auch andere Anwendungen unterstützt werden, etwa Editoren, Formatierer, Übersetzer oder Filter.

Kompatibilität mit SGML

Die Vorteile von SGML sollen durch Kompatibilität von XML zu SGML erhalten bleiben, so daß existierende SGML-Werkzeuge auch XML verarbeiten können,

XML-Dokumente zugleich gültige SGML-Dokumente sind, zu einem XML-Dokument eine entsprechende SGML-DTD generiert werden kann und schließlich die Ausdrucksstärke von SGML auch in XML vorhanden ist.

Einfache Erstellung von Verarbeitungsprogrammen

Der Erfolg von Datenformaten hängt oft von der Verfügbarkeit zugehöriger Werkzeuge zur Verarbeitung ab, so daß beim Entwurf von XML auf die einfache Implementierbarkeit solcher Werkzeuge Wert gelegt wurde. Dies soll insbesondere für solche Werkzeuge gelten, die eine eventuell vorhandene DTD nicht berücksichtigen.

Minimierung optionaler Bestandteile

Eines der Hauptprobleme von SGML ist die große Anzahl optionaler Dokument-Bestandteile, so daß Probleme beim Austausch unterschiedlicher Dokumente auftreten können. Aus diesem Grund wurde XML so entworfen, daß jedes XML-Werkzeug auch jedes XML-Dokument verarbeiten kann, indem auf optionale Eigenschaften vollständig verzichtet wurde.

Erhöhte Lesbarkeit und Verständlichkeit von XML-Dokumenten

Der Vorteil erhöhter Lesbarkeit und Verständlichkeit von textuellen gegenüber binären Datenformaten soll durch XML erhalten und verstärkt werden. So soll es möglich sein, ein XML-Dokument auch ohne zusätzliche Werkzeuge lesen und verstehen zu können.

Zügige Erstellung des XML-Standards

Beim Entwurf des XML-Standards wurde auf eine möglichst zügige Umsetzung Wert gelegt, da gerade in der Frühphase des World Wide Web viele unterschiedliche, konkurrierende und teilweise inkompatible Techniken verwendet wurden. Mit XML soll hingegen ein offenes, nicht-proprietäres und text-basiertes Austauschformat vorgestellt werden, welches auch hinreichend erweiterbar ist.

Formales und eindeutiges Design

Die Realisierung von Verarbeitungswerkzeugen hängt sehr stark von der Eindeutigkeit der zugrundeliegenden Standards ab, weshalb der XML-Standard möglichst formal die Eigenschaften der Sprache definieren soll.

Einfache Erstellung von XML-Dokumenten

Neben der simplen Begründung, daß mittels XML einfach zu erstellende Dokumente auch tatsächlich erstellt werden, soll das Design von XML auch die Implementierung von Authoring-Systemen erleichtern.

Kompakte Ausdrucksweise nachrangig

Die Komplexität von SGML und die daraus folgende erschwerte Anwendung ist durch die Benutzung von Minimierungsmechanismen, d.h. durch das Streben nach einer kompakten Ausdrucksweise, wesentlich erhöht worden. Um XML weniger komplex zu gestalten, wurde mehr Wert auf klare als auf kompakte Ausdrucksweise gelegt.

Weitere Ziele

In der offiziellen Empfehlung [31] nicht enthaltene aber in [30] erwähnte weitere Ziele umfassen die Internationalisierung von XML und die Berücksichtigung von pragmatischen Implementierungsproblemen, die oft von Programmierern unter Zeit- oder Kostendruck bewältigt werden müssen (den in [30] angesprochenen sog. „*Desperate Perl Hackers*“).

Spezifikation im Vergleich zu SGML

XML verwendet die gleichen Konstrukte wie SGML mit einigen Modifikationen und Vereinfachungen, so existiert eine physikalische und eine logische Struktur für XML-Dokumente.

Logische Struktur

Auch XML benutzt wie SGML unter anderem *Descriptive Markup*, *Markup Declarations* und *Processing Instructions* zur Festlegung der logischen Struktur eines Dokuments. Wichtige Unterschiede im Vergleich zu SGML sind dabei:

- Die Ende-Tags eines Elements sind immer obligatorisch, um die Verarbeitung eines XML-Dokuments durch Minimierung von Deutungsmöglichkeiten zu erleichtern.
- Leere Elemente sind als solche durch das Voranstellen eines „/“ vor der schließenden spitzen Klammer im Start-Tag gekennzeichnet.
- Das Content-Model für Elemente kann entweder gemischten Inhalt beschreiben (d.h. Unterelemente gemischt mit Text), wobei keine Anordnungsrestriktionen definiert werden können, oder es beschreibt nur die Verwendung von Unterelementen, in diesem Fall können Restriktionen bezüglich der Anordnung von Elementen festgelegt werden.
- Groß-/Kleinschreibung bei Element- und Attributnamen wird immer berücksichtigt.

Diese und andere Unterschiede (im Detail in [41] sowie in [30] und [31] beschrieben) dienen insbesondere dem Ziel der Vereinfachung von XML gegenüber SGML.

Physikalische Struktur

Eine interessante Eigenschaft der physikalischen Struktur ist die Festlegung, daß ein *System Identifier* eine URI [18] ist, was die Verarbeitung von XML-Dokumenten, die solche System Identifiers benutzt, gerade in Internet-Anwendung wesentlich vereinfacht und Mehrdeutigkeiten und Inkompatibilitäten bei der Auflösung vermeidet. Eine weitere Vereinfachung ist die Anforderung, daß jedes Werkzeug, welches XML verarbeitet, mindestens die UTF-8- oder UTF-16-Kodierung des ISO/IEC-10646-Standards [86] als Eingabe akzeptieren muß.

Eigenschaften

Durch die Kompatibilität zu SGML besitzt XML viele Eigenschaften dieser Meta-Sprache. Dazu zählen insbesondere die physikalische und die logische Struktur von XML-Dokumenten, welche derjenigen von SGML-Dokumenten vergleichbar ist. Dennoch zeichnet sich XML durch eine Reihe besonderer Charakteristika aus, die hier erläutert werden sollen.

Einfachheit

Die Spezifikation von XML [31] ist zwar umfangreich, allerdings läßt dies nicht auf eine entsprechende Komplexität von XML selbst schließen. Vielmehr besteht XML aus wenigen einfachen Konzepten: Ein XML-Dokument besteht immer aus Elementen mit zugehörigen Attributen, es ist nur in einer einzigen Art und Weise verarbeitbar, jegliche Kontextinformation (Deklaration von Elementen, Attributen, Entitäten usw.) kann gegebenenfalls vernachlässigt werden, ohne daß ein XML-Dokument unbenutzbar wird.

Kern weiterer Spezifikationen

XML steht als Auszeichnungssprache im Kontext von Internet-Anwendungen nicht alleine da. Einerseits existieren weitere Empfehlungen, Techniken und Standards, die XML um wichtige Funktionalitäten erweitern (etwa XML-Namensräume, XLink oder XSLT, s.u.), andererseits ist XML selbst wieder Grundlage für weitere Konzepte, beispielsweise als zugrundeliegendes Datenformat. Es ergibt sich somit eine Schichtenarchitektur, bei der auf XML aufbauend Erweiterungen vorgenommen werden, ohne daß dadurch aber die eigentliche Spezifikation von XML zu modifizieren ist.

Flexible Beschreibungsmöglichkeiten

Eine große Stärke von XML ist die Flexibilität, mit der Informationen strukturiert und annotiert werden können. XML selbst nimmt außer der Verwendung von Elementen und Attributen sowie der Zeichenkodierung keine weitere Festlegung hinsichtlich der Syntax und Semantik von XML-Dokumenten vor. Erst durch die o.g. angesprochenen weiteren Techniken und durch Mechanismen wie Schema-Beschreibungssprachen werden die vorhandenen Daten semantisch mit zusätzlicher Information angereichert. So sieht beispielsweise XML keine Beschränkung von Elementinhalten über Datentypen vor, erst durch zusätzliche Schema-Beschreibungen können Restriktionen wie etwa Beschränkung auf ganze Zahlen, Währungseinheiten oder Ländernamen formuliert werden. Der Vorteil hierbei ist die gegebene Flexibilität hinsichtlich veränderlicher Anforderungen oder Deutungsweisen. Die mittels XML strukturierten Daten können durch die nicht vorhandenen Inhaltsrestriktionen auch in anderen Kontexten gedeutet und verarbeitet werden. Dies ist einerseits hilfreich, wenn Informationen nicht eindeutig annotiert und strukturiert werden können, andererseits erleichtert diese Flexibilität eine spätere Modifikation von aufbauenden Anwendungen.

4.4 XML-Namensräume

Die Informationen innerhalb eines XML-Dokuments können in zwei Dimensionen strukturiert werden, nämlich über die Hierarchie und die Reihenfolge von Elementen. Diese Struktur wird üblicherweise über eine DTD oder andere Mechanismen festgelegt, wodurch die Art der Information und deren Struktur miteinander verbunden sind.

Der Mechanismus von Namensräumen erlaubt es nun, unterschiedliche Informationsarten und unterschiedliche Strukturdefinitionen miteinander zu mischen. Ein Beispiel wäre die Kombination von ansonsten unabhängigen Abrechnungsdaten und Artikeldaten in einem Bestellsdokument. Mittels Namensräumen kann die Her-

kunft sowohl der Informationen selbst als auch die möglichen Strukturen (also verwendbaren Elemente) kenntlich gemacht werden. Es ergeben sich zwei Vorteile:

- Vorhandene Strukturdefinitionen in Form von DTDs oder anderer Schemata können wiederverwendet und miteinander kombiniert werden.
- Informationen können in einem Dokument über Annotationen mit einer entsprechenden Herkunftskennung versehen werden, so daß sie anhängig vom Anwendungskontext entweder ignoriert oder verarbeitet werden.

Der XML-Namespace-Standard [32] stellt eine Umsetzung von Namensräumen dar, die XML wesentlich aufwertet. Zwar ist die Kombination vorhandener DTDs über XML-Namespace noch nicht spezifiziert [28], dennoch können Elemente in XML-Dokumenten bereits mit Namespace-Identifikatoren versehen werden. Eine spätere Validierung dieser Elemente gegen die referenzierten Schemata wird Bestandteil entsprechender W3C-Empfehlungen werden.

Deklaration von Namensräumen

Namensräume sind vor ihrer Verwendung zu deklarieren. Dazu werden vordefinierte Attribute benutzt. Die Werte dieser Attribute können entweder explizit im Dokument angegeben werden, oder sie werden über Attribut-Definitionen in der DTD definiert.

Die Namen der Attribute für die Deklaration von Namensräumen besitzen entweder den Präfix *xmlns:*, oder es wird *xmlns* alleine als Name verwendet. Die Bedeutung dieser unterschiedlichen Deklarationen wird weiter unten erläutert. Wird ein Namensraum über ein Attribut mit Präfix *xmlns:* deklariert, so stellt der Suffix des Attributnamens den eindeutigen Identifizierer für den Namensraum dar. Abbildung 9 zeigt die Bestandteile einer Namensraum-Deklaration.

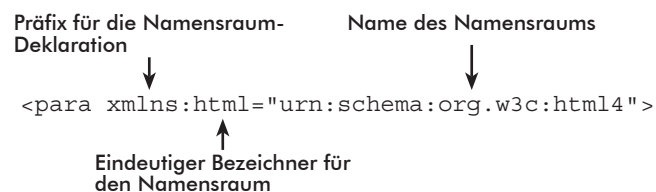


Abb. 9: Bestandteile einer XML-Namespace-Deklaration

Der Name eines Namensraumes ist eine URI, welche einen Namensraum eindeutig identifizieren soll. Wichtig ist dabei, daß die Ressource, welche über diese URI referenziert wird, nicht weiter verarbeitet wird. Insbesondere macht die Namespace-Empfehlung explizit deutlich, daß die URI nicht als Referenz auf ein Schema, eine DTD o.ä. zu sehen ist, sondern nur ein Mittel zu Unterscheidung von Namensräumen ist. Die Einbeziehung von Schemata bleibt späteren Anwendungen und Empfehlungen vorbehalten. Ein Grund für diese konkrete Realisierung ist u.a. die größere Ausdrucksfähigkeit von URIs (insbesondere von URNs) im Vergleich zu den in XML möglichen Attributnamen.

Qualifizierte Namen

Die Aufteilung eines Element- oder Attributnamens in einem XML-Dokument, wie etwa bei der Deklaration eines Namensraumes, führt zum Konzept des *qualifizierten Namens*. Der Präfix eines qualifizierten Namens (d.h. der Teil bis zum Doppelpunkt) kennzeichnet den Herkunfts-Namensraum für das Element oder Attribut. Der Präfix

muß vorher im Dokument deklariert sein. Der Suffix nach dem Doppelpunkt ist der lokale Teil des Namens.

Der Präfix stellt nur einen Platzhalter für den Namensraum-Namen dar, da dieser Zeichen enthalten kann, die in Element- oder Attributnamen nicht erlaubt sind. Eine zwar nicht XML-konforme, aber trotzdem teilweise verwendete Notation ersetzt den Präfix durch den Namensraum-Namen in geschweiften Klammern, so daß gemäß Abbildung 9 ein Elementname auch in der Form {urn:schema:org.w3c:html4}body geschrieben werden kann.

Verwendung von Namensräumen

Wird in einem Element ein XML-Namespace deklariert, so gilt diese Deklaration für diese Elemente und alle darin enthaltenen Unterelemente. Dementsprechend kann für dieses Element und die Unterelemente sowie alle damit verbundenen Attribute der deklarierte Präfix verwendet werden. Es ist auch möglich, mehrere Namensräume in einem Element zu deklarieren, etwa im Wurzelement eines Dokuments.

Stammen viele Elemente in einem Dokument aus einem Namensraum, so kann ein *default namespace* verwendet werden. Dazu wird bei der Deklaration ein Namensraum über ein Attribut mit dem Namen `xmlns` deklariert, d.h. es wird kein expliziter Präfix deklariert. Alle Elemente und Attribute, die keinen Präfix tragen, werden dann diesem Vorgabe-Namensraum zugeordnet. Die unterschiedlichen Anwendungsmöglichkeiten sind in Abbildung 10 in einem Beispiel dargestellt.

```
<html xmlns="urn:schema:org.w3c:html4"
      xmlns:person="urn:schema:org.w3c:people">
  <body>
    <h1>Homepage von
      <person:name>Eckhart Köppen</person:name>
    </h1>
  </body>
</html>
```

Abb. 10: Verwendung von XML-Namespaces

Bei der Verarbeitung von XML-Dokumenten, die Namensräume enthalten, sind drei Optionen möglich:

- Die Anwendung wertet alle Namensräume aus und behandelt Elementinhalte und Attributwerte entsprechend.
- Die Anwendung verarbeitet nur die Element und Attribute, deren Namensraum bekannt ist und ignoriert den Rest.
- Die Anwendung behandelt qualifizierte Namen als gewöhnliche Namen im Sinne der XML-Spezifikation und ignoriert somit die Herkunftsangabe über Namensräume.

Insbesondere der letzte Punkt ermöglicht es, Dokumente mit Namensräumen auch in Systemen zu benutzen, die Namensräume nicht auswerten können. Dies ist möglich, da ein qualifizierter Name ein gültiger Name gemäß der XML-Empfehlung ist.

Abschließend ist zu sagen, daß die XML-Namespace-Empfehlung eine Zuordnung von Elementen und Attributen zu einem bestimmten Namensraum ermöglicht, die genauere Bedeutung und Verarbeitung des referenzierten Namensraums erst in späteren Empfehlungen und in Zusammenhang mit Techniken wie Schema-Beschreibungssprachen präzisiert werden kann.

5 Informationsvernetzung

Die Vernetzung der Information stellt für den Anwender den Hauptnutzen dar. Die Anwendungsmöglichkeiten sind sehr weitläufig. Sie reichen von der einfachen Aufteilung großer Informationsmengen über die Möglichkeit der Schaffung von Querverweisen zur Annotation bis zur Einbindung der verschiedenen Datenformate. Als Grundvoraussetzung für die Vernetzung von Informationen wird die Bedeutung von Speicherungsarten für Daten und die Adressierung unterschiedlicher Server-Systeme definiert.

5.1 Uniform Resource Identifier

Die Referenzierung von HTML-Dokumenten und anderen Ressourcen erfordert einen Adressierungs- und Identifikationsmechanismus, welcher beispielsweise bei der Verknüpfung mittels Hyperlinks genutzt werden kann. Aus diesem Grund wurden *Uniform Resource Identifiers* (URIs) standardisiert [18], die einen allgemeinen Mechanismus zur Identifikation und Adressierung von Ressourcen realisieren. Spezielle Varianten sind die weiter unten beschriebenen *Uniform Resource Locators* (URLs) und *Uniform Resource Names* (URNs). Ein URI ist in den meisten Fällen nach dem Muster `<scheme>://<authority><path>?<query>` aufgebaut, allerdings existieren auch andere Ausprägungen. Die Bestandteile, deren Bedeutung und alternative Formen werden im folgenden erläutert.

Uniform Resource Locators

Eine URI wird durch einen Schema-Bezeichner (*scheme*) eingeleitet, der den Identifizierer näher beschreibt und die eigentliche Information zur Adressierung und Identifikation konkretisiert. Schemata können beispielsweise das Übertragungsprotokoll kennzeichnen, über das die referenzierte Ressource angefordert werden kann. In diesem Fall ist die Aufgabe der URIs primär die Adressierung einer Ressource und weniger die eindeutige Identifikation. Eine solche URI fällt unter die Kategorie der *Uniform Resource Locators* (URLs).

Schema-Bezeichner

Die Schema-Bezeichner, welche bei URLs eingesetzt sind, umfassen unter anderem `http` für HTTP, `news` für Usenet-Artikel, oder `telnet` für Telnet-Verbindungen. Die Menge der Schema-Bezeichner ist erweiterbar, es können also neue Übertragungsprotokolle in einer URL genutzt werden, solange der schema-spezifische Teil genügend Informationen zur Adressierung einer Ressource über das Protokoll aufnehmen kann.

Wurzelkomponente

Der schema-spezifische Teil eines URLs besteht aus einer Adresse, die im Verbindung mit dem im Schema angegebenen Protokoll eingesetzt wird. Diese Adresse bezeichnet die Wurzelkomponente (*authority*), welche den Zugang zur eigentlichen Ressource erlaubt. Im Regelfall (etwa bei HTTP-, Telnet oder FTP-URLs) ist diese Komponente eine Internetadresse eines Rechners, andere Protokolle benutzen E-Mail-Adressen oder Newsgruppennamen. Auch können Portnummern zur Identifikation von einzelnen Prozessen auf diesem Rechner angegeben werden.

Eine Besonderheit ist die Möglichkeit, Authentisierungsinformation in einer URL anzugeben, etwa den Benutzernamen oder auch ein Paßwort.

Pfad

Die eigentliche Ressource wird über einen Pfad (*path*) adressiert, der über die Wurzelkomponente aufgelöst wird. Es kann sich hier z.B. um einen Pfad zu einer Datei im Dateisystem des Zielrechners handeln, allerdings kann von der Struktur des Pfades nicht auf die physikalische Realisierung des Zugriffs geschlossen werden. Ein Pfad besteht aus Segmenten, die mittels „/“ getrennt werden und parametrisiert werden können. Die Parameter für ein Segment werden nach dem Segmentwert mit vorangestelltem Semikolon angehängt.

Anfrageparameter

Die über die Wurzelkomponente und den Pfad adressierte Ressource kann neben statischer Information auch eine dynamische Komponente repräsentieren. Ein Zugriff auf eine solche Komponente führt dann zu deren Aktivierung, das Ergebnis der Aktivierung wird an die anfordernde Instanz übertragen. Wie bei der Parametrisierung des Zugriffspfades kann auch die Aktivierung der Komponente mit Anfrageparametern durchgeführt werden. Sie werden im *query*-Teil einer URL übergeben, der mit vorangestelltem Fragezeichen „?“ an den Pfad angefügt wird.

Relative und absolute Adressierung

Da eine URL die Adressierung einer Ressource in den Vordergrund stellt, und diese Adressen häufig einen konkreten Speicherort beinhalten, ergibt sich ein erhöhtes Problem der Integrität von URLs. Ändert sich der Speicherort einer Ressource, so sind alle referenzierenden URLs zu ändern. Eine Milderung dieser Problematik besteht in der relativen Adressierung von Ressourcen, bei der ausgehend von einer Ausgangsadresse dazu relative Adressen eingesetzt werden. Verändert sich die Ausgangsadresse, so sind davon die relativen Referenzen nicht berührt. Der Aufbau einer relativen URI unterscheidet sich insofern von absoluten URIs, als daß bestimmte Bestandteile nicht angegeben werden. Es kann auf das Schema, die Wurzelkomponente und Teile des Pfades verzichtet werden, so daß diese fehlende Information aus der absoluten Basis-URI abgeleitet wird. Dies heißt aber auch, daß eine relative URI nur in Zusammenhang mit einer (impliziten) Basis-URI verwendet werden kann.

Uniform Resource Names

Die oben angesprochene Problematik der Integrität von Adressen kann durch die Verlagerung des Schwerpunkts zur Identifizierung von Ressourcen behoben werden. Zu diesem Zweck können *Uniform Resource Names* (URNs) [117] eingesetzt werden, die eine Ressource eindeutig identifizieren, unabhängig von ihrem Speicherort. Der Schema-Bezeichner für einen URN ist *urn*, welcher im schema-spezifischen Teil eines URN einem Namensraum-Bezeichner folgt. An diesen wird mit Doppelpunkt abgetrennt der namensraum-spezifische Wert des URN angehängt, der den eigentlichen Namen der Ressource enthält.

Die Namensräume, welche in einem URN verwendet werden können, sind in der URN-Spezifikation nicht weiter festgelegt, denkbar ist hier z.B. ISBN für die Identifikation von Büchern, RFC für IETF-Standards oder auch Identifizierer für Firmen und Organisationen.

Abhängig vom Namensraum kann die tatsächliche Adresse der referenzierten Ressource über einen Verzeichnisdienst ermittelt werden. Zwar ist damit eine weitere Instanz zur Ermittlung der Adresse notwendig, betrachtet man aber die Umsetzung der Adressermittlung bei URLs, so ist auch dort im Regelfall eine Suche der Adresse der Wurzelkomponente etwa über einen *Domain Name Service* (DNS) notwendig.

Fragment Identifier

Wird eine URI benutzt, um eine Ressource nicht nur zu referenzieren, sondern auch anzufordern (etwa bei der Nutzung eines Web-Browsers), so können innerhalb der angeforderten Ressource untergeordnete Fragmente identifiziert werden. Die Bezeichner der Fragmente können mit vorangestelltem Doppelkreuz „#“ an eine URI angehängt werden. Die Fragment-Bezeichner (*fragment identifier*) werden von der Wurzelkomponente der Zielressource nicht ausgewertet, stattdessen liegt es im Aufgabenbereich der anfordernden Instanz, diese zu behandeln. Die Bedeutung von Fragment-Bezeichnern wird in der URI-Spezifikation vom Inhaltstyp der referenzierten Ressource abhängig gemacht. So wird bei HTML-Ressourcen der per Fragment-Bezeichner benannte Teil eines Dokuments im Web-Browser dargestellt.

Fragment-Bezeichner sind nicht Bestandteil der URI, welche ja eine Ressource identifizieren und adressieren, sondern Bestandteil einer URI-Referenz, also eines Verweises auf eine Ressource, der mittels einer Ressource formuliert wird.

5.2 XLink

Da XML für den Einsatz im Internet und World Wide Web geeignet sein soll, sind auch Mechanismen zur Verknüpfung und Referenzierung von Ressourcen notwendig, also eine Nachbildung der Anchor- und Link-Mechanismen aus HTML. Zusätzlich existieren noch weitere Anforderungen [48], welche ähnlich wie die Anforderungen an XML auf einfache Verwendbarkeit und Implementierbarkeit zielen. Auch sollen die erweiterten Möglichkeiten von Techniken wie HyTime [85] oder TEI [165] erreicht werden.

Zu diesem Zweck wird vom W3C die *XML Linking Language* (XLink) [49] nach den in [110] definierten Prinzipien entworfen. Grundsätzlich bietet XLink die Möglichkeiten zur Verknüpfung, wie sie auch HTML bietet. Sie werden über einen einfachen Link realisiert. Darüber hinaus beschreibt XLink auch erweiterte Links, die mehrdirektionale Links, externe Links und andere fortgeschrittene Verknüpfungsmöglichkeiten erlauben.

Grundsätzliche Funktionsweise

Die Definition einer Verbindung zwischen Ressourcen gemäß XLink wird ähnlich wie in der XML-Namespaces-Empfehlung über vordefinierte Attribute erreicht. Sie sind im XLink-Namensraum beschrieben, welcher über die URL <http://www.w3.org/XML/XLink/0.9> referenziert werden kann. Durch die Verwendung dieser Attribute ist es möglich, beliebige Elemente als Verknüpfungselemente (*linking elements*) zu nutzen. Zusätzlich sind im XLink-Entwurf aber auch dedizierte Elementtypen definiert, die einzig für die Verknüpfung von Ressourcen vorgesehen sind. Die Kennzeichnung eines Elements als Verknüpfungselement wird hauptsächlich durch das *type*-Attribut erreicht, dessen Wert die Art des Links bezeichnet. Den grundsätzlichen Aufbau von XLink-Verknüpfungen zeigt Abbildung 11.

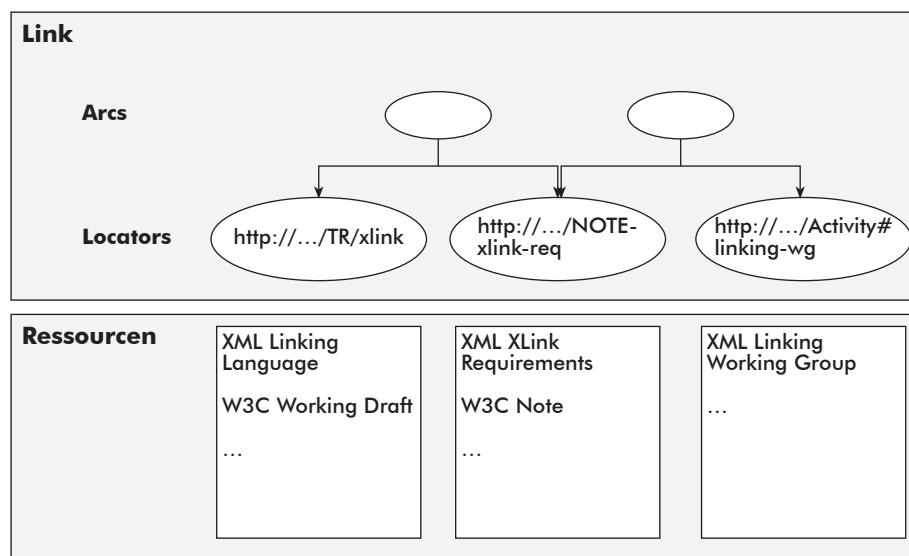


Abb. 11: Aufbau eines XLinks

Ein XLink besteht immer aus mehreren Komponenten, welche unterschiedliche Rollen ausfüllen. Diese Rollen können aber auch zusammengelegt werden, so daß einzelne Komponenten entfallen können. Konkret werden im XLink Ressourcen über *Locators* referenziert, diese Locators werden in *Arcs* paarweise miteinander verbunden, ein oder mehrere Arcs schließlich bilden einen XLink. Die Rolle eines Elements wird wiederum über das *type*-Attribut festgelegt, so daß hierfür schließlich die Werte *simple* und *extended* (Art des Links), sowie *locator* und *arc* (Rolle des Elements) verwendet werden können.

Link-Attribute

Die Semantik von Links kann über Link-Attribute weiter detailliert werden. Dazu werden neben dem Locator-Attribut *href*, welches eine Referenz auf eine Ressource beinhaltet, und den Arc-Attributen *from* und *to* für die Endpunkte eines Arcs noch Verhaltens- und Semantik-Attribute verwendet.

Verhaltens-Attribute werden für die Spezifizierung von Aktionen in Zusammenhang mit der Aktivierung eines Links benutzt:

- Das *show*-Attribut beschreibt, wie die referenzierte Ressource bei der Aktivierung des Links darzustellen ist. Mögliche Werte sind *new* für die Darstellung in einem neuen Kontext, *parsed* für die Einbettung der Ressource an Stelle des Link-Ausgangspunkts sowie *replace* für das Ersetzen der Ressource am Ausgangspunkt durch die referenzierte Ressource.
- Das *actuate*-Attribut definiert, wie der Link aktiviert werden kann, hier stehen die Werte *user* für eine benutzerinitiierte und *auto* für eine automatische (d.h. beim Verarbeiten der Ausgangsressource ausgelöste) Aktivierung.

Der HTML-Anchor würde dementsprechend das *show*-Attribut mit dem Wert *new* oder *replace* und das *actuate*-Attribut mit dem Wert *user* belegen, während eine XML-Entity-Referenz über die Werte *embed* und *auto* für die Verhaltens-Attribute simuliert werden könnte.

Semantische Attribute dienen zur Definition zusätzlicher Eigenschaften einer Verknüpfung. Dazu können das *title*- und das *role*-Attribut verwendet werden. Beide Attribute enthalten nicht näher spezifizierte Aussagen, d.h. ihre Werte sollen primär für den Endbenutzer Bedeutung haben. Mit dem *role*-Attribut könnte der Inhalt der referenzierten Ressource und mit dem *title*-Attribut die Funktion der Verknüpfung näher beschrieben werden.

Einfache XLinks

Ein einfacher XLink reicht für eine Vielzahl von Anwendungen aus, da damit die Semantik von HTML-Links abgebildet werden kann. Im einfachen XLink sind die Rollen des Ausgangs- und Ziel-Locators, der Ausgangsressource und eines Arcs in einem einzigen Element zusammengefaßt und über Link-Attribute spezifiziert (ein Beispiel zeigt Abbildung 12).

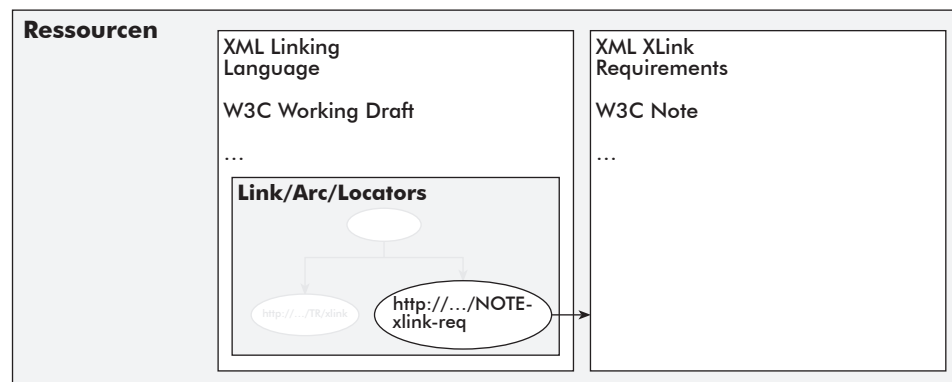


Abb. 12: Aufbau eines einfachen XLinks

Wie generell im XLink-Entwurf möglich wird auch hier das Verknüpfungselement durch Hinzufügen der entsprechenden Attribute definiert. So besitzt ein einfaches Verknüpfungselement mindestens ein *type*-Attribut mit dem Wert *simple* und ein *href*-Attribut für den Ziel-Locator für den Link. Abbildung 13 enthält ein Beispiel dazu. Das dort vorhandene Element stammt nicht aus dem XLink-Namensraum und wird durch die Link-Attribute zu einem einfachen XLink. Zur Vereinfachung könnten die Attribute *type*, *show*, *actuate* sowie die Namensraum-Deklaration als Vorgabewerte in einer DTD festgelegt werden, so daß im Dokument selbst nur noch das *href*-Attribut auftritt.

```
<link
  xmlns:xlink="http://www.w3.org/XML/XLink/0.9"
  xlink:type="simple"
  xlink:href="http://www.uni-essen.de"
  xlink:role="homepage"
  xlink:title="Universität Essen"
  xlink:show="replace"
  xlink:actuate="user">
  Startseite der Universität GH Essen
</link>
```

Abb. 13: Beispiel für einen einfachen XLink

Erweiterte XLinks

Der Ausgangs-Locator ist im Falle eines einfachen XLinks implizit mit der Referenz auf die aktuelle Ressource belegt, weiterhin ist der einfache XLink ein fester Bestandteil dieser Ressource, d.h. ein einfacher XLink ist ein sog. *inline link*.

Mehr Möglichkeiten bietet im Vergleich dazu der erweiterte XLink. Ein erweiterter XLink ist zwingend dann nötig, wenn

- Verbindungen zwischen mehr als zwei Ressourcen beschrieben werden sollen, oder
- die Ressourcen für die Verknüpfung nicht mehr modifiziert werden können, also keine Verknüpfungselemente in einer der zu verknüpfenden Ressourcen eingefügt werden können (etwa bei CD-ROMs oder externen Ressourcen).

Im letzten Fall werden erweiterte XLinks als sog. *out-of-line links* eingesetzt, bei denen die Verknüpfungselemente in einem separaten Dokument ähnlich einer Link-Datenbank verwaltet werden. Wird dieses Dokument zentral verwaltet, reduziert sich auch der Aufwand für die Konsistenzprüfung bei sich ändernden Links.

Ein erweiterter XLink wird durch Elemente gebildet, von denen jedes genau eine Rolle entsprechend Abbildung 11 einnimmt. Es existiert also ein Element, welches den Link bildet, sowie untergeordnete Locator- und Arc-Elemente. Sie werden wie beim einfachen XLink entweder über Attribute gekennzeichnet oder über den XLink-Namensraum identifiziert. Das `type`-Attribut enthält den Wert `extended` für das umschließende Link-Element, `locator` für die Locator-Elemente und `arc` für die Arc-Elemente. Neben den bereits vorgestellten Link-Attributen können für alle im Link enthaltenen Arcs Vorgaben für das Verhalten (d.h. für die `show`- und `actuate`-Attribute) über die Attribute `showdefault` und `actuatedefault` festgelegt werden. Sie können durch explizite Angaben im Arc selbst präzisiert werden.

Die Zuordnung zwischen Arcs und Locators erfolgt über `ID`- und `IDREF`-Attribute. Die Locator-Elemente werden über ein `ID`-Attribut eindeutig gekennzeichnet, dieses kann über die Werte der `from`- und `to`-Attribute, welche als `IDREF`-Attribute deklariert sind, referenziert werden.

Ein erweiterter XLink enthält also eine gewisse Anzahl von Verknüpfungen (Arcs) zwischen jeweils zwei Ressourcen. Auf diese Weise können in einem erweiterten XLink Verknüpfungen zwischen mehr als zwei Ressourcen abgebildet werden. Das Beispiel aus Abbildung 11 würde dementsprechend nach Abbildung 14 in einem XML-Dokument eingebunden werden können.

Die XLink-Spezifikation befindet sich allerdings zur Zeit noch in der Entwurfsphase, weshalb die oben beschriebenen Mechanismen noch Änderungen unterliegen können. Allerdings ist davon auszugehen, daß die in [48] beschriebenen Anforderungen in jedem Fall erfüllt werden und somit auch zumindest äquivalente Mechanismen Bestandteil der endgültigen Empfehlung werden.

5.3 XPath und XPointer

Die Mechanismen zur Referenzierung von Ressourcen (URI, URN oder URL) sind nicht geeignet, um einzelne Bestandteile dieser Ressourcen zu adressieren. Es existieren zwar mit den `ID`-/`IDREF`-Attributen und den Fragment-Identifizieren zwei einfache Mechanismen zur Verknüpfung, allerdings fehlen ihnen wichtige Funktionalitäten wie etwa die Adressierung mehrerer Elemente oder die Berücksichtigung der

```
<extended
  xmlns="http://www.w3.org/XML/XLink/0.9"
  type="extended"
  showdefault="new"
  actuatedefault="user">

  <locator href="http://.../TR/xlink" id="xlink"/>
  <locator href="http://.../NOTE-xlink-req" id="xreq"/>
  <locator href="http://.../Activity#linking-wg" id="wg"/>

  <arc from="xlink" to="xreq"/>
  <arc from="xreq" to="wg"/>

</extended>
```

Abb. 14: Beispiel für einen erweiterten XLink

Hierarchiebeziehungen in einem XML-Dokument. Dies soll mit der XPath-Empfehlung [43] möglich sein, die sich beim W3C in der finalen Spezifikationsphase befindet. Darauf baut die XPointer-Spezifikation auf, die in [50] beschrieben ist.

XPath

XPath beschreibt Ausdrücke, mit deren Hilfe einzelne Bestandteile oder Gruppen von Bestandteilen eines XML-Dokuments referenziert werden können. Die Bestandteile entsprechen dabei nicht der physikalischen Repräsentation, etwa in einer Datei, sondern der logischen Struktur des Dokuments. XPath versucht, diese Bestandteile ähnlich wie Pfadangaben in Dateisystemen zugreifbar zu machen und so die hierarchische Struktur von XML-Dokumenten zu berücksichtigen.

Datentypen

Die Datentypen, die in XPath benutzt werden, orientieren sich am XML Information Set [46]. Die dort spezifizierten Bestandteile eines XML-Dokuments haben jeweils eine direkte Entsprechung in der XPath-Empfehlung. So existieren Datentypen für Elemente, Attribute, Dokumente, aber auch für Zeichenketten, Zahlen oder Wahrheitswerte.

Pfade

Kern von XPath sind Pfade, die als Ausdrücke über die unterschiedlichen Datentypen formuliert werden. Pfade bestehen aus einer Kette von Einzelschritten durch die Elementhierarchie. Jeder Schritt bezieht sich auf einen assoziierten Kontextknoten. Knoten sind in XPath dabei u.a. Elemente und Attribute.

Mittels einer Achsenangabe (*AxisSpecifier*) wird spezifiziert, welche Gruppe von Knoten, ausgehend vom Kontextknoten, adressiert wird, eine Knotenspezifikation (*NodeTest*) bestimmt, welche Knoten adressiert werden sollen, diese wiederum kann über Prädikate (*Predicate*) näher verfeinert werden. Das Format eines Schrittes ist in der ausgeschriebenen Form also *AxisSpecifier::NodeTest[Predicate]*.

Achsen

Die Achsen, welche die Beziehung zwischen dem Kontextknoten und den zu adressierenden Elementen beschreiben, beinhalten unter anderem folgende Ausprägungen:

- *child* und *parent* für die Adressierung von direkten Vorgänger- und Nachfolgeknoten des Kontextknotens,
- *following-sibling* und *next-sibling* für Nachbarn des Kontextknotens,
- *attribute* für die Attribute des Kontextknotens oder
- *namespace* für die XML-Namespace-Attribute des Kontextknotens.

Knotenspezifikationen

Die über eine Achse adressierte Gruppe von Knoten kann durch eine Knotenspezifikation weiter eingeschränkt werden. Darüber hinaus ist es möglich, alle Knoten mit einem bestimmten Element- oder Attributnamen oder auch alle Knoten abhängig von ihrem Typ (Text, Element, Kommentar usw.) zu selektieren.

Prädikate

Die Menge der adressierten Knoten kann durch Prädikate gefiltert werden. Prädikate sind Ausdrücke, die Literale, aber auch Variablen und Funktionsaufrufe enthalten können. Prädikate selektieren über Positionsangaben hauptsächlich einen oder mehrere bestimmte Knoten in der durch Achsenangabe und Knotenspezifikation adressierten Menge von Knoten.

Funktionen

Jede XPath-Implementierung muß eine Reihe von Funktionen implementieren, die in Ausdrücken verwendet werden können. Dazu zählen Funktionen, die auf Zeichenketten und Zahlen operieren, oder Funktionen, die auf eine Menge von Knoten angewendet werden können.

XPointer

XPointer soll XPath so erweitern, daß die Adressierung von Bereichen mit beliebigen Begrenzungspunkten in einem XML-Dokument möglich wird. Dies soll durch eine *range*-Achse erreicht werden. Zusätzlich beschreibt die XPointer-Spezifikation, wie XPath-Ausdrücke als Fragment Identifier von URIs eingesetzt werden können, die auf XML-Ressourcen verweisen.

6 Kommunikation

Eine der grundlegenden Aufgaben von Web-basierten Informationssystemen ist die Distribution von Information über Kommunikationskanäle. Im Gegensatz zu anderen Informationssystemen kann diese Information aber in den unterschiedlichsten Formaten vorliegen. Hierzu existieren Regeln, nach denen entschieden wird, wie die unterschiedlichen Repräsentationen zu behandeln sind. Weiterhin sind Semantiken definiert, welche die Verteilung von Daten beschreiben.

6.1 Hypertext Transport Protocol

Die Grundlage für die Kommunikation in Web-basierten Informationssystemen ist das TCP/IP-basierte *Hypertext Transport Protocol* (HTTP), welches in den Versionen 1.0 [17] und 1.1 [60] eingesetzt wird. Beiden Versionen liegt die Idee zugrunde, daß Ressourcen mittels Methoden bearbeitet und übertragen werden, vergleichbar mit objektorientierten Systemen. Ressourcen werden im HTTP über einen Uniform Resource Identifier angesprochen. Der Zugriff auf diese Ressourcen wird von

HTTP-Servern gewährleistet, wobei von den anfragenden Instanzen (Clients) ein Satz von vordefinierten Methoden benutzt werden kann. Methoden werden nach einem Anfrage-Antwort-Schema abgearbeitet, so daß sich insgesamt eine Client/Server-Architektur gemäß Abbildung 15 ergibt.

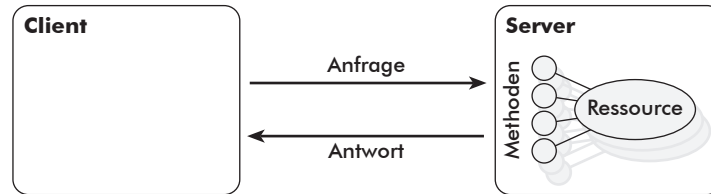


Abb. 15: Architektur für HTTP-Verbindungen

Ressourcen

Die über eine URI adressierbaren Ressourcen decken einen sehr weiten Bereich ab. HTTP verwendet neben der Adresse bzw. dem Identifikator einer Ressource deswegen noch weitere Metadaten, um den Anforderungen verteilter heterogener Informationssysteme gerecht zu werden. Diese Metadaten werden bei Anfragen und Antworten mit übertragen. Wichtige Metadaten sind im HTTP beispielsweise die Zeit der letzten Modifikation, der MIME-Inhaltstyp [25] der Ressource, deren mögliche zusätzliche Codierungen oder auch die Größe der Ressource.

Methoden

Die Methoden, welche auf eine Ressource angewandt werden können, zielen hauptsächlich auf Informationsübertragung und administrative Aufgaben ab. Die wichtigsten der in HTTP/1.1 verfügbaren Methoden sollen im folgenden kurz charakterisiert werden.

GET: Die GET-Methode ist die am häufigsten genutzte HTTP-Methode. Sie dient der Übertragung des Inhalts einer Ressource von Server zum Client. Die Ressource selbst bleibt unverändert, auch deren Adresse oder Identifikation verändert sich nicht.

PUT: Legt entweder eine neue Ressource mit dem übertragenen Inhalt an oder ersetzt den Inhalt einer unter der angegebenen URI bereits bestehenden Ressource damit.

POST: Die POST-Methode stellt einen Sonderfall dar, da sie nicht direkt auf der angegebenen Ressource operiert, sondern stattdessen Daten an diese Ressource übermittelt, die diese dann verarbeitet. Die POST-Methode stellt also u.a. eine Möglichkeit dar, auf dem Server Operationen ausführen zu lassen.

HEAD: Mittels der HEAD-Methode können Metadaten über eine Ressource angefordert werden.

OPTIONS: Um die Charakteristika der Übertragungsstrecke einschließlich des Servers abzufragen, kann die OPTION-Methode eingesetzt werden.

DELETE: Die DELETE-Methode weist den Server an, die angegebene Ressource zu löschen.

Anfragen und Antworten

Anfragen (*Requests*) und Antworten (*Responses*) realisieren den Ablauf einer Methodenaktivierung auf einer Ressource. Eine Anfrage resultiert immer in einer direkten Antwort, so daß bei der Anfrage-Antwort-Kette von einer in sich abgeschlossenen und nicht unterbrechbaren Operation gesprochen werden kann (dies wird in HTTP/1.1 etwas aufgeweicht, siehe unten). Es können allerdings durchaus mehrere Anfragen gleichzeitig durch einen HTTP-Server verarbeitet werden.

Eine Anfrage besteht immer aus der Angabe, welche Methode auf welcher Ressource ausgeführt wird. Sie wird in der *Request Line* übermittelt, die immer die erste Zeile einer Anfrage ist. Zusätzlich können weitere Meta- oder Steuerdaten in Form von *Header*-Feldern übergeben werden, die als weitere Zeilen nach der Request-Zeile übergeben werden. Header-Felder werden durch ein Schlüsselwort gefolgt vom Wert des Headers gebildet. Abgeschlossen werden diese Daten immer durch eine leere Zeile. Auch ist es bei einigen Methoden möglich, zusätzliche Daten im *Message Body* zu übertragen, etwa bei der POST- oder der PUT-Methode.

Die Antwort des Servers wird durch eine Statuszeile (*Status Line*) eingeleitet, die Informationen über die Durchführung der Methode enthält. Weiter können wiederum zeilenweise Metadaten (*Response Header*) übergeben werden, die das Ergebnis der Methode näher beschreiben. Je nach Methode folgt das eigentlichen Ergebnis im Message Body, z.B. der Inhalt der angeforderten Ressource bei einer GET-Anfrage. Ein Beispiel für Anfrage und Antwort zeigt Abbildung 16.

Anfrage:

<i>Methode + URI</i>	PUT /index.html HTTP/1.0
<i>Zusätzliche Angaben</i>	Content-length: 688
<i>Zu verarbeitende Daten</i>	<HTML>...

Antwort:

<i>Statusmeldung</i>	HTTP/1.0 201 Created
<i>Zusätzliche Angaben</i>	Date: Tue, 16 Nov 1999 08:12:31 GMT

Abb. 16: Beispiel für HTTP-Anfrage und -Antwort

Verbindungen

Der Datenaustausch bei HTTP/1.0 wird pro Anfrage-Antwort-Paar über eine einzelne TCP/IP-Verbindung durchgeführt. Dadurch wird HTTP zu einem zustandslosen Protokoll, bei dem im Regelfall eine Anfrage als von anderen Anfragen unabhängiger Vorgang betrachtet wird. Es sind keine Informationen über etwaige anfrageübergreifende Mechanismen wie Sitzungen vorhanden. Für die Implementierung Web-basierter Informationssysteme, die komplexe Abläufe wie etwa das Bestellen von Waren abbilden, die in mehreren Anfragen resultieren, stellt sich diese Eigenschaft natürlich als Nachteil dar. Aus diesem Grund existieren mehrere Ansätze, anfrageübergreifende Zustandsinformationen in den Datenaustausch einzubinden, so daß ein Server je nach Zustand einer Sitzung unterschiedlich auf Anfragen reagieren kann. Ein Beispiel hierfür sind die sog. *Cookies* (beschrieben in [102]), welche es dem Server erlauben, kleine Dateneinheiten in der Client-Software abzulegen, die bei späterer

ren Anfragen des Clients mit übertragen wird. Generell sind solche Ansätze auf der Anwendungsebene angesiedelt, so daß sie hier nicht weiter betrachtet werden sollen. Ein anderer Nachteil der Verbindungen nach HTTP/1.0 ist der hohe Aufwand, welcher bei der Anfrage vieler Ressourcen anfällt, da für jede Ressource eine Verbindung etabliert werden muß. Aus diesem Grund sieht HTTP/1.1 vor, daß über eine Verbindung mehrere Anfragen hintereinander abgewickelt werden können. Wichtig ist aber, daß dies nicht als Mittel vorgesehen ist, um Zustandsinformationen über Anfragen hinweg zu nutzen. Eine Diskussion der Erweiterungen von HTTP/1.1 gegenüber HTTP/1.0 befindet sich beispielsweise in [101].

6.2 Common Gateway Interface

Die bei der HTTP-POST-Methode (und im Grunde auch bei der GET-Methode) gegebene Möglichkeit, auf dem Server Prozesse zu aktivieren, wird durch das *Common Gateway Interface* (CGI) [45] näher spezifiziert. Die Prozesse werden dabei durch das Starten von Programmen erzeugt, die durch die Anfrage-URI näher bezeichnet sind. Die zu aktivierende Ressource wird im Regelfall als vom Web-Server getrennter Prozeß gestartet und ihre Ausgabe als Antwort an den anfragenden Web-Client zurückgesandt.

Bei einer Aktivierung einer als ausführbar gekennzeichneten Ressource können zusätzliche Parameter übergeben werden. Im Fall der POST-Methode ist dies der Nachrichteninhalte, bei der GET-Methode werden die Parameter an die URI angehängt. Parameter werden über einen Namen identifiziert, d.h. im Gegensatz zu vielen Programmiersprachen werden keine positionalen Parameter verwandt. Da alle Parameter in einer einzigen Zeichenkette übertragen werden, die im Fall der Aktivierung über einen GET-Request zudem noch in der Anfrage-URI enthalten ist, müssen Sonderzeichen und reservierte Zeichen maskiert werden. Dazu wird die MIME-Kodierung *application/x-www-form-urlencoded* verwendet. Parameter werden im Regelfall aus HTML-Form-Elementen ausgelesen und dem CGI-Programm übergeben.

Die Programme, die durch CGI über einen Web-Server genutzt werden können, decken einen sehr weiten Aufgabenbereich ab. Da der Verbund von Web-Server und CGI-Anwendung konkrete Applikationen ermöglicht, spricht man auch von *Applikations-Servern*. Die CGI-Schnittstelle bildet hier die programmtechnische Grundlage für die in Abbildung 3 dargestellte Erweiterung einfacher Web-basierter Systeme auf der Server-Seite, die aber in unterschiedlichen Variationen implementiert ist. So werden aus Effizienzgründen häufig benötigte CGI-Programme nicht nur bei Bedarf gestartet, sondern können auch ständig aktiv sein. Allen Varianten ist aber die Ansteuerung über GET- oder POST-Methode und die oben beschriebene Parameterübergabe gemein.

7 Informationsdarstellung

Die Information, die mittels Web-basierter Informationssysteme verteilt wird, kann zum einen maschinell weiterverarbeitet, zum anderen aber auch direkt vom Endbenutzer konsumiert werden. Im ersten Fall ist für die Weiterverarbeitung eine Vereinbarung über die Struktur der Information wichtig. Im zweiten Fall kommt noch eine adäquate Aufbereitung für den Endbenutzer dazu. Hierunter fallen beispielsweise die

Möglichkeiten, Informationen am Bildschirm zu lesen, auszudrucken oder auch vorzulesen.

7.1 Cascading Style Sheets

Die Verwendung von HTML im WWW hat gezeigt, daß die Trennung von Inhalt und Aussehen von HTML-Dokumenten, welche ursprünglich angestrebt wurde, nicht praktikabel von HTML unterstützt wird. Vielmehr werden viele Elemente in HTML entgegen ihrem ursprünglichen Einsatzzweck für die Steuerung des Seitenlayouts mißbraucht, etwa Tabellen zur Positionierung von Elementen oder Zitatelemente für Einrückungen. Zudem ist die Darstellung von HTML-Dokumenten nur für die Bildschirmausgabe definiert, wichtige Steuerungsmöglichkeiten für die Druckausgabe oder die Möglichkeit der Sprachausgabe fehlen gänzlich.

Um diese Nachteile zu beheben, wurden am W3C die *Cascading Style Sheets* (CSS) in einer Empfehlung [27] beschrieben. Sie greifen die Idee der Trennung von Inhalt und Form auf und schaffen einen formalen Rahmen für die Beschreibung der Form von SGML- und XML-basierten Dokumenten. Weitere Ziele beim Entwurf von CSS waren:

- Wartbarkeit durch Trennung der Nutzdaten von den Formatvorlagen, so daß bei einer Änderung des Layouts diese Änderung für mehrere Dokumente gleichzeitig gilt.
- Unabhängigkeit von Herstellern, Plattformen oder Geräten, um Inhalte möglichst universell verbreiten zu können.
- Flexibilität durch die definierte Kombination mehrerer Formatvorlagen.
- Zugang zu Inhalten auch für Benutzer mit Behinderungen wie etwa Seh- oder Hörgeschädigte, indem diese Benutzergruppen eigene Formatvorlagen mit den vom Autor vorgesehenen Formatvorlagen kombinieren oder ein anderes Ausgabemedium wählen können.

Layout-Modell

Das Layout-Modell, welches CSS zugrundeliegt, stützt sich auf ein Block-Modell, in dem Blöcke zur Strukturierung des Layouts genutzt werden. Die Blöcke können Text, Bilder, andere eingebettete Objekte (z.B. Eingabefelder) oder wiederum Blöcke enthalten. Jedes Element eines XML- oder HTML-Quelltextes wird in einem entsprechenden Block abgebildet. Zusätzlich werden in CSS noch Pseudo-Elemente definiert, die bestimmte, im Ursprungsdokument nicht vorhandene Bestandteile für eine Layout-Beschreibung verfügbar machen. Dazu gehören Elemente für die erste Zeile eines Absatzes, Aufzählungsmarkierungen oder Numerierungen vor Absätzen in einer Liste oder auch generell Text vor oder nach einem Absatz. Durch diese Pseudo-Elemente wird erreicht, daß semantische Information, die im Elementnamen verborgen ist, aber nicht im Elementinhalt wiedergegeben wird, auch für die Darstellung zu nutzen ist. So kann etwa ein Element mit dem Namen *name* und dem Inhalt *Eckhart Köppen* durch Nutzung des Pseudo-Elements *before* bei der Ausgabe mit einem Beschreibungstext versehen werden. Dadurch können die wesentlichsten Nachteile der rein deskriptiven Natur von CSS-Layoutvorlagen ausgeglichen werden.

Layout-Eigenschaften

Jedes Element hat gemäß der CSS-Definition einen Satz von Layout-Eigenschaften, die bestimmte Werte annehmen können. Diese Eigenschaften steuern die Darstellung (oder Ausgabe) der Elemente und umfassen alle dafür notwendigen Bereiche wie etwa Farben, Zeichensätze, Abstände, Positionen oder Längenangaben. Wichtig für das Layout sind insbesondere zwei Eigenschaften, *display* und *float*. Sie definieren, ob ein Element frei im Layout des umschließenden Blocks fließen kann, ob es an einer bestimmten Position einzufügen ist, und in welcher Art und Weise der restliche Inhalt um das Element fließt.

Die Werte der Eigenschaften entsprechen einem einfachen Typsystem, das die möglichen Ausprägungen beschreibt. Werte können absolut oder in vielen Fällen auch relativ zu anderen Werten angegeben werden. Auch können Werte von umschließenden Elementen geerbt werden, was beispielsweise bei Zeichensatzwahl und Zeichengröße sinnvoll ist.

Layout-Regeln

Die Zuordnung von Werten für bestimmte Eigenschaften zu den Elementen erfolgt über Layout-Regeln. Sie bestehen aus einem Selektor und einer Deklaration. Der Selektor beschreibt, auf welche Elemente die nachfolgende Deklaration anzuwenden ist. Dazu wird eine Beschreibungsform gewählt, die sehr ähnlich zu XPath ist. Es erscheint sinnvoll zu betrachten, ob nicht XPath die geeignetere (und universellere) Form der Selektion von Elementen in einem Dokument ist. Die CSS-Spezifikation berücksichtigt XPath allerdings noch nicht.

Selektoren erlauben die Spezifikation unterschiedlichster Kriterien, nach denen eine Deklaration angewandt werden soll. Einfachste Fälle sind beispielsweise die Angabe des Elementnamens oder des Werts für ein bestimmtes Attribut, komplexere Fälle umfassen die Angabe einer Verschachtelungsreihenfolge oder Nachbarbeziehung für Elemente. Insbesondere für die Anwendung in HTML-Dokumenten wird das Attribut *class* für die Variation von Layout-Angaben bei gleichem Elementnamen eingesetzt, etwa um einfache Absätze von hervorgehobenen Absätzen zu unterscheiden. Abbildung 17 zeigt einige Beispiele.

Style Sheets, Kaskadierung und Medientypen

Mehrere Layout-Regeln können in einem Style Sheet zusammengefaßt werden. Typischerweise existieren mehrere dieser Formatvorlagen:

- eine Vorgabevorlage des Benutzers,
- eine Vorlage, die der Autor des HTML- oder XML-Dokuments erstellt hat und per HTML-LINK-Element oder nach [42] mit dem Dokument verknüpft,
- sowie eine Vorlage in der Client-Software, die das Aussehen der Elemente grundsätzlich definiert (etwa die Definition für HTML-Elemente).

Diese Formatvorlagen werden gemäß einer Kaskadierungsordnung miteinander verknüpft, um die Anpassung der Ausgabe an unterschiedliche Gegebenheiten zu ermöglichen. Den höchsten Stellenwert nehmen die Layout-Regeln des Autors ein, gefolgt von den Angaben des Benutzers. Diese Reihenfolge kann durch die Annotierung einer Regel als *important* verändert werden, so daß ein Benutzer bestimmte Layout-Regeln wie vergrößerte Zeichensätze oder erhöhte Lautstärke fest vorgeben kann.

Selektion über den Elementnamen:

```
BODY {font-size: 12pt; color: black}
```

Selektion über Attributwerte (beide Regeln sind äquivalent):

```
P[CLASS=example] {margin: 2em; font-style: italic}  
P.example {margin: 2em; font-style: italic}
```

Berücksichtigung der Elementhierarchie:

```
EM > EM {font-style: roman}
```

Berücksichtigung eines direkten Nachbarn:

```
H1 + P {text-indent: 0}
```

Abb. 17: Beispiele für CSS-Selektoren

In einem Style Sheet können Layout-Regeln auch bestimmten Ausgabemedien zugeordnet werden. Es ist demnach möglich, zwischen der Ausgabe auf einem normalen Bildschirm, einem Drucker oder einer Braille-Zeile zu unterscheiden und die Charakteristika der Ausgabegeräte zu berücksichtigen. Die Angabe des entsprechenden Medientyps erfolgt bei der Verbindung zwischen dem Dokument und den Style Sheets.

7.2 DSSSL

Die Ausgabe von SGML-Dokumenten wird in vielen Fällen mittels der *Document Style and Semantics Specifications Language* (DSSSL) [87] realisiert. DSSSL ist in vielen Punkten leistungsfähiger als CSS. Zu den Hauptvorteilen gehören die Transformationsmöglichkeiten und das umfangreichere Layout-Modell. Zudem bietet DSSSL nicht nur eine deklarative Spezifikation des Layouts, sondern auch die Verwendung imperativer Komponenten. Die Verarbeitung von SGML-Dokumenten mit DSSSL ist ein zweistufiger Prozeß, der in Abbildung 18 dargestellt ist.

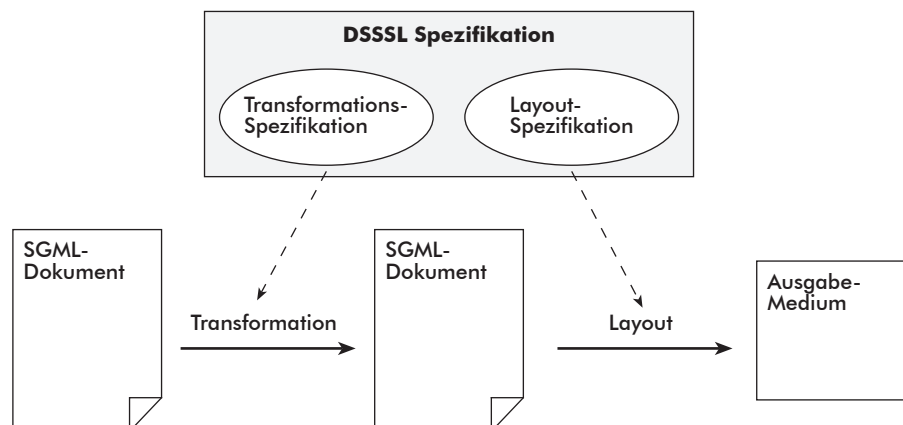


Abb. 18: Verarbeitung von SGML-Dokumenten mit DSSSL

Sowohl dem Transformations- als auch dem Layoutprozeß liegt hierbei eine an die Programmiersprache *Scheme* [57] angelehnte Spezifikationssprache zugrunde.

Transformation

Die Transformation eines Ausgangsdokuments ist dann notwendig, wenn nicht alle Elemente des Dokuments verarbeitet, neue Elemente erzeugt (Verzeichnisse, Indizes, Beschreibungen) oder Elemente umgestellt werden sollen. Die Regeln für die Transformation werden in Assoziationen festgehalten, die aus drei Teilen bestehen:

- ein Auswahl Ausdruck, der eine Liste von zu bearbeitenden Elementen zurückliefert,
- eine Transformationsanweisung, die die ausgewählten Elemente bearbeitet und
- eine Prioritätsangabe, die eine Rangfolge für mehrere Assoziationen herstellt.

Layout

Der Layoutprozeß ähnelt dem Transformationsprozeß insofern, als daß auch hier eine Umwandlung des Dokuments stattfindet. Die Elemente des Dokuments werden in *flow objects* überführt, welche die Basis für das eigentliche Layout sind. In DSSSL ist eine Reihe von flow objects vordefiniert, darunter solche für Seiten, Absätze oder Spalten. Diese besitzen per Layout-Spezifikation festlegbare Eigenschaften, ähnlich CSS. Die Festlegung wird über Konstruktionsregeln durchgeführt. Eine Konstruktionsregel wird durch die Auswahl eines Elements über den Elementnamen eingeleitet, gefolgt von Anweisungen zur Konstruktion eines oder mehrerer flow objects. Der Rückgabewert der Anweisungen zur Konstruktion der flow objects bestimmt, in welcher Form der Inhalt des ausgewählten Elements bearbeitet werden soll. Im Beispiel in Abbildung 19 wird für ein Element mit dem Namen *emph* ein entsprechendes flow object in kursiver Schrift erzeugt, welches den Inhalt des Elements enthält.

```
(element emph (make sequence font-posture: 'italic
                    (process-children) )
```

Abb. 19: Beispiel für eine DSSSL-Konstruktionsregel

Das Ergebnis des Layout-Prozesses, der *flow object tree*, muß von einem entsprechenden Programm ausgewertet werden, um auf einem Ausgabemedium dargestellt werden zu können.

7.3 XSLT und XSL

Da DSSSL sehr mächtig aber auch sehr komplex ist und zudem nicht an die Verwendung in Web-basierten Anwendungen angepaßt ist (so sind die Eigenschaften von Elementen in CSS und DSSSL nicht deckungsgleich und es fehlen Möglichkeiten, Links zu beschreiben), sind mit XSL und XSLT zwei Standards in der Entwurfsphase, die gezielt auf das Layout von XML-Dokumenten im Internet abzielen.

Ursprünglich als ein einziger Mechanismus mit dem Namen *Extensible Style Language* (XSL) [40] vorgesehen, ist nunmehr eine Aufteilung in zwei Spezifikationen vorgesehen, die den zweistufigen Prozeß wie bei DSSSL nachbilden:

- die Transformation von XML-Dokumenten soll mittels der *Extensible Style-sheet Language Transformations* (XSLT) [44] ermöglicht werden,

- die Formatierung von XML-Dokumenten wird anhand der XSL-Spezifikation [47] durchgeführt.

Beide Spezifikationen orientieren sich eng an DSSSL und CSS. Die grundlegende Funktionsweise entspricht der von DSSSL, so werden auch in XSL flow objects eingesetzt, die über *Templates* (analog zu DSSSL-Konstruktionsregeln) erzeugt werden. Die Eigenschaften für die Layout-Elemente umfassen eine Obermenge der CSS-Eigenschaften. XPath wird benutzt, um die Konstruktions- und Transformationsregeln an bestimmte Elemente zu binden, ähnlich den Selektoren in CSS oder den Auswahlausdrücken in DSSSL. Das Beispiel für eine DSSSL-Konstruktionsregel aus Abbildung 19 wird in XSL/XSLT wie in Abbildung 20 formuliert.

```
<template match="emph">
  <inline-sequence font-style="italic">
    <apply-templates/>
  </inline-sequence>
</template>
```

Abb. 20: Beispiel für eine XSL-Template

Transformations- und Layoutbeschreibungen nach XSL und XSLT sind wiederum XML-Dokumente. Dies vereinfacht die Verarbeitung, da so eine einzige Syntax zur Beschreibung von Struktur, Inhalt und Aussehen eines Dokuments nötig ist.

8 Interaktion

Im Zusammenhang mit der Erweiterung von Web-basierten Informationssystemen um dynamische Informationsverarbeitung sind zunehmend auch interaktive Anwendungen möglich. Dazu zählen insbesondere Anwendungen, bei denen nicht nur einfache, in sich abgeschlossene Anfragen an das System gestellt werden, sondern komplexere Abläufe, in deren Verlauf Information durch den Benutzer zusammengestellt, modifiziert und weiterverwendet wird.

8.1 Intrinsic Event Model

Wichtiger Bestandteil von HTML 4.0 ist das *Intrinsic Event Model* (IEM). Es regelt die Interaktionsmöglichkeiten zwischen Benutzer und Dokument bzw. Client-Programm. Die Ereignisse, die dabei ausgelöst werden können, umfassen folgende Gruppen:

Tastaturereignisse: Sie werden ausgelöst, wenn der Benutzer eine Taste drückt oder losläßt.

Mausereignisse: Hierzu zählen Ereignisse bei Mausbewegungen, bei Click- oder Doppelklick.

HTML-Form-Ereignisse: Ereignisse, die im Zusammenhang mit HTML-Forms auftreten können, etwa die Veränderung von Eingabefeldern, das Wechseln des Eingabefokus oder das Absenden eines Forms.

Ereignisse des Client-Programms: Beim Laden und Entladen eines Dokuments im Client-Programm wie etwa einem Web-Browser werden entsprechende Ereignisse ausgelöst.

Die Behandlung der Ereignisse kann in HTML durch einen einfachen Mechanismus erreicht werden. Dazu wird für die Behandlung eines Ereignisses durch ein Element ein Attribut mit dem Namen des zu behandelnden Ereignisses gesetzt. Der Wert des Attributs ist ein Programmskript, welches beim Auftreten des Ereignisses ausgelöst wird.

Das Intrinsic Event Model ist nur sehr rudimentär, erlaubt aber die Implementierung einfacher Interaktionen. Es wird durch das wesentlich leistungsfähigere *Document Object Model* in der Version 2 [192] erweitert, welches sich zur Zeit aber noch in der Spezifizierungsphase befindet (siehe dazu auch das nächste Unterkapitel).

8.2 Document Object Model

Einen anderen Ansatz für die Modifikation von XML- und HTML-Dokumenten als DSSSL und XSLT stellt das *Document Object Model* (DOM) dar. Es ist in Level 1 als Empfehlung des W3-Konsortiums festgelegt [191], Level 2 ist zur Zeit in der Entwurfsphase [192].

Das DOM stellt eine programmiersprachenunabhängige Schnittstelle (*Application Programming Interface*, kurz API) zur Erstellung, Modifikation und Abfrage von XML- und HTML-Dokumenten dar. Es besteht aus einem Kernteil, der auf die Bearbeitung von XML- und HTML-Dokumenten abzielt, sowie einem HTML-Teil, der die Besonderheiten von HTML-Dokumenten abdeckt. Dieser Teil deckt unter anderem die Funktionalität ab, die durch vorher bestehende Browser-Implementierungen realisiert wurden und unter dem (nicht formal definierten) Sammelbegriff *Dynamic HTML* zusammengefaßt werden. Die Programmiersprachenunabhängigkeit wird durch die Verwendung der *Interface Definition Language* (IDL, Bestandteil der Common Object Request Broker Architektur beschrieben in [133]) für die Definition der APIs erreicht.

Dokumente werden im DOM als Baum repräsentiert, der aus einzelnen Knoten besteht. Dies ist allerdings nur eine logische Repräsentation in Form eines strukturellen Modells, das DOM macht keine Annahmen über die tatsächliche Implementierung. Der Begriff *object modell* resultiert aus der Tatsache, daß die Knoten im Strukturmodell als Objekte bestehend aus einer Zusammenfassung von Daten und Verhalten betrachtet werden. Es ist allerdings wichtig zu beachten, daß in der DOM-Spezifikation die konkrete Beschreibung der DOM-Schnittstelle nicht mit objektorientierten Methoden erfolgt, d.h. die dort beschriebenen IDL-Interfaces stellen keine direkte Repräsentation der Knoten in einem Dokument dar. Deren Implementierung kann auf unterschiedlichen Wegen erfolgen, wichtig ist nur, daß die Implementierung konform zu den DOM-Schnittstellen erfolgt.

Neben den Schnittstellen, die für die Verarbeitung von XML- und HTML-Dokumenten spezifiziert werden, werden im DOM noch grundlegende Datentypen benutzt. Der wichtigste darunter ist der *DOMString* für die Zeichenkettenbehandlung. Ein *DOMString* ist immer in UTF-16 kodiert. Der tatsächlich verwendete Datentyp in einer DOM-Implementierung muß allerdings nur zu diesem Datentyp konform sein, so daß beispielsweise bereits vorhandene Zeichenketten-Datentypen transparent verwendet werden können.

Speichermanagement

Eine wichtige Implikation daraus ist, daß das DOM nicht festlegt, wie die Knoten in einem Dokument physikalisch angelegt werden, da die Realisierung der Knoten von deren Schnittstelle abweichen kann. Auch ist das Speichermanagement nicht im DOM geregelt, sondern Aufgabe einer tatsächlichen Implementierung. Dennoch werden in der Spezifikation Beispielrealisierungen der Interfaces in Java [72] und ECMAScript [59] beschrieben. Wichtig ist, daß hier die Mechanismen für die Speicherverwaltung in den jeweiligen Programmiersprachen in einer konkreten Realisierung der DOM-APIs implementiert und genutzt, also Vorkehrungen für eine automatische oder nicht-automatische Speicherverwaltung getroffen werden müssen.

Strukturmodell

Die Struktur eines Dokuments wird im DOM mittels einer Reihe von abstrakten Schnittstellen beschrieben, die einerseits Vererbungsbeziehungen aufweisen, andererseits auch die Aggregation von Elementen widerspiegeln. Die Schnittstellen sind in Tabelle 1 erläutert.

Schnittstelle	Beschreibung
Node	Die Node-Schnittstelle stellt eine allgemeine Schnittstelle für alle Knoten in einem Dokument dar und beinhaltet u.a. Funktionen für den Zugriff auf strukturell benachbarte Knoten, Knotennamen und -inhalt.
Document	Repräsentiert ein gesamtes XML- oder HTML-Dokument und enthält ein untergeordnetes Wurzelement, eine DTD sowie Funktionen zur Erzeugung aller anderen Knotentypen.
DocumentFragment	Um Knoten auch außerhalb eines Dokuments verarbeiten zu können (etwa zum Ausschneiden und Einfügen von Knoten), kann ein DocumentFragment benutzt werden, das Vorgängerknoten für eine Reihe untergeordneter Knoten sein kann.
DocumentType	Enthält den Namen, der in der DTD angegeben wurde, sowie die Entities und Notations, welche im Dokument deklariert wurden.
ProcessingInstruction	Repräsentiert eine Processing Instruction inklusive deren Inhalt.
Element	Die Element-Schnittstelle erweitert die Node-Schnittstelle um die Möglichkeit, auf Attribute eines Elements zuzugreifen. Sie bildet die XML- oder HTML-Elemente in einem Dokument ab.
Text	Erlaubt den Zugriff auf den textuellen Inhalt von Elementen. Der Inhalt kann durch ein oder mehrere Text-Elemente gebildet werden, das Zusammenfassen mehrerer aufeinanderfolgender Text-Elemente wird über das Element-Interface realisiert.
Comment	Steht für Kommentare und ihren Inhalt.
CDATASection	Repräsentiert CDATA-Sektionen.

Schnittstelle	Beschreibung
EntityReference	Obwohl das DOM davon ausgeht, daß ein von einem Parser verarbeitetes Dokument keine Referenzen auf Entities mehr enthält, können diese über die EntityReference-Schnittstelle programmiertechnisch in eine Dokumentstruktur eingebettet werden.
Attr	Enthält den Namen und den Wert eines Attributs.
Entity	Enthält die Public- und System-Identifizier für ein Entity.
Notation	Enthält die Public- und System-Identifizier für eine Notation.

Tabelle 1: Schnittstellen im DOM

Die wichtigste Vererbungsbeziehung zwischen diesen Schnittstellen besteht darin, daß alle Schnittstellen direkt oder indirekt von der Schnittstelle Node abgeleitet sind. Dadurch kann über die Node-Schnittstelle allein ein Dokument abgefragt werden. Die dafür wichtigsten Attribute in der Schnittstelle sind in Tabelle 2 beschrieben.

Attribut	Beschreibung
nodeType	Kennzeichnet den Knotentyp (Document, Element, Text, ...).
nodeName	Enthält den Namen des Knotens, etwa den Element- oder Attributnamen, den Entity-Namen, sowie vordefinierte Namen, falls kein sinnvoller Wert für einen Namen existiert.
nodeValue	Wert des Knotens, nur bei den Schnittstellen Text, Attr, Comment und CDATASection sinnvoll.
parentNode	Verweis auf den Vorgängerknoten, nicht gesetzt für das Attr-, Document- und DocumentFragment-Interface.
childNodes	Liste der Nachfolger des Knotens. Die wichtigsten möglichen Nachfolgerbeziehungen sind in Abbildung 21 dargestellt.
previousSibling	Verweis auf den linken Nachbarn entsprechend den möglichen Nachfolgerbeziehungen.
nextSibling	Verweis auf den rechten Nachbarn.
attributes	Nur für Element-Knoten mit einer Liste der Attribute gesetzt.
ownerDocument	Dokument, mit dem der Knoten assoziiert ist.

Tabelle 2: Wichtige Attribute der Node-Schnittstelle

Auch eine Modifikation ist in vielen Fällen allein über die Node-Schnittstelle möglich. Dazu existieren die Methoden *insertBefore*, *appendChild*, *replaceChild* und *removeChild*, die eine Veränderung der Nachfolgerliste eines Knotens erlauben. Diese alleinige Verwendung der Schnittstelle Node soll die Verwendung in solchen Umgebungen vereinfachen, in denen eine explizite Umwandlung (*cast*) einer Node-Instanz in eine Instanz der konkreten Schnittstelle aufwendig ist. Die Benutzung der abgeleiteten Schnittstellen (Element, Text, Document usw.) ist in allen anderen Fällen vorzuziehen.

Neben den Vererbungsbeziehungen zwischen den Schnittstellen sind insbesondere die strukturellen Beziehungen zwischen den Knoten eines Dokuments von Bedeu-

ting, d.h. deren hierarchische Anordnung. Die wichtigsten Beziehungen zeigt Abbildung 21 unter Verwendung der Notation der *Unified Modelling Language* (UML) [157].

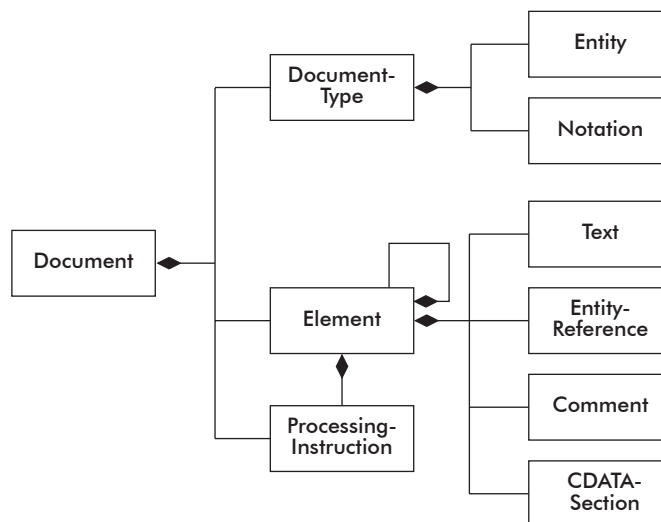


Abb. 21: Strukturelle Beziehungen im Document Object Model

HTML-Modell

Die fundamentalen Schnittstellen reichen aus, um XML-Dokumente zu verarbeiten. Durch die Zielsetzung, bestehende Implementierungen von Client-Programmen sinnvoll abdecken zu können, sind im DOM allerdings noch spezielle Schnittstellen für die Verarbeitung von HTML-Dokumenten definiert. Diese Schnittstellen sind rückwärtskompatibel, so daß ihre Funktionalität auch allein mit den Kernschnittstellen realisiert werden kann. Sie zielen hauptsächlich auf eine Vereinfachung der Erstellung von Programmen im HTML-Umfeld ab. Allerdings ist ihre Verwendung zwingend daran gebunden, daß die zu verarbeitenden Dokumente konform zur HTML-DTD sind.

Der HTML-spezifische Teil des DOM besteht aus folgenden Schnittstellen:

- Die Schnittstelle *HTMLDocument* ist von der Schnittstelle *Document* abgeleitet und erweitert diese um Zugriffsfunktionen für spezielle Attribute (Farben, Hintergrundbild, Liste aller eingebundenen Bilder, Forms usw.) sowie um Funktionen zur dynamischen Modifikation des ursprünglichen Quelltextes.
- Die Element-Schnittstelle wird durch die Schnittstelle *HTMLElement* erweitert, die Zugriff auf Attribute erlaubt, die in allen HTML-Elementen vorhanden sein können (*id*, *name*, *lang*, *dir* und die CSS-Klasse).
- Die innerhalb der HTML-DTD festgelegten HTML-Elemente werden über Spezialisierung des *HTMLElement*-Interfaces angesprochen. So existieren Schnittstellen für HTML-Form-Elemente, Elemente aus dem Kopf eines HTML-Elements oder Verknüpfungselemente.

In Level 1 des DOM sind allerdings keine Funktionalitäten für den Zugriff auf CSS-Layoutinformationen und das in HTML definierte Ereignismodell (siehe Unterkapi-

tel 8) vorhanden. Diese und weitere Punkte werden Bestandteil des DOM Level 2 sein.

DOM Level 2

Die Erweiterung des DOM bezüglich wichtiger fehlender Funktionalitäten wird im DOM Level 2 vorgenommen, welches sich zur Zeit in der Entwurfsphase befindet. Neben den bereits in Level 1 vorhandenen Kern- und HTML-Schnittstellen werden folgende Bereiche abgedeckt:

- ein allgemeiner Mechanismus, um Layoutvorlagen verarbeiten zu können,
- Interfaces für die Benutzung von CSS-Layoutvorlagen (CSS-Eigenschaften sind allerdings nicht direkt an Elemente gebunden, sondern nur indirekt aus den Style Sheets ableitbar),
- ein Ereignismodell, welches auf dem Intrinsic Event Model von HTML aufbaut und dieses stark erweitert,
- Möglichkeiten, unterschiedliche Sichtweisen auf ein Dokument zu definieren,
- Schnittstellen für das Traversieren von Knoten in Dokumenten sowie
- Schnittstellen für die Behandlung von Knotenmengen, wichtig etwa für vom Benutzer selektierte Bereiche eines Dokuments.

Zukünftig sehr wichtig sind Erweiterungen, die XML-Namensräume durchgängig im DOM verfügbar machen und berücksichtigen. Dabei wird davon ausgegangen, daß das name-Attribut in der Node-Schnittstelle den qualifizierten Namen eines Elements oder Attributs enthält, also den eigentlichen Namen mit vorangestelltem Namensraum-Präfix. Zusätzlich sind über die Attribute *namespaceURI*, *localName* und *prefix* der Node-Schnittstelle die einzelnen Bestandteile des qualifizierten Namens sowie der beim Erzeugen des Knotens gültige Namensraum-URI erreichbar. Funktionen, die in Level 1 nur den Namen eines Knotens berücksichtigen (z.B. um Attribut-Werte oder Elemente abzufragen), werden um Funktionen erweitert, die daneben auch den Namensraum-URI berücksichtigen.

9 Diskussion

Web-basierte Informationssysteme decken ein weites Gebiet von Aufgaben ab, dazu zählen insbesondere die Verteilung von Information und deren Vernetzung. Die Mechanismen, mit denen diese Aufgaben erfüllt werden, sind hinreichend einfach, um eine weite Verbreitung solcher Systeme zu fördern. Zu nennen sind in diesem Zusammenhang vornehmlich HTML zur Strukturierung und Steuerung der Darstellung, HTTP zur Distribution und URIs zur Vernetzung von Information.

Allerdings zeigen sich bei näherer Betrachtung eine Reihe von Nachteilen herkömmlicher Web-basierter Systeme. Wird in einer Anwendung verstärkt Wert auf die möglichst durchgängige Verwendbarkeit von Daten gelegt, so erscheint HTML als Auszeichnungs- und Strukturierungssprache für diese Information nicht flexibel genug. Es ist bei der weiteren Verarbeitung von mit HTML annotierten Daten auf jeden Fall mit einem Bruch zu rechnen, der Konvertierungsmechanismen notwendig macht. Auch ist die (u.a. durch HTTP implizierte) Client/Server-Architektur Web-basierter Anwendungen inflexibel, was potentielle Erweiterungen oder Änderungen einer Systemstruktur angeht.

Diese Nachteile sollen im weiteren Verlauf der Arbeit durch die Entwicklung eines adäquaten Modells für die Beschreibung und Erstellung Web-basierter Informationssysteme aufgewogen werden. Die in Unterkapitel 3.2 identifizierten grundlegenden Aufgabengebiete und weitere noch zu identifizierende Anforderungen sollen möglichst orthogonal durch entsprechende Techniken abgedeckt werden, ohne daß sich dabei Konflikte oder Überschneidungen ergeben. Durch eine möglichst eindeutige Zuordnung eines Mechanismus zu genau einer grundlegenden Anforderung soll die optimale Erfüllung der einzelnen Anforderungen erreicht werden.

Die in diesem Abschnitt vorgestellten Grundlagentechniken stellen eine erste Auswahl von Mechanismen dar, die im zu entwickelnden Modell eine Rolle spielen können. Teilweise werden sie bereits in Web-basierten Anwendungen eingesetzt, so daß sie auch aus Gründen der Kompatibilität berücksichtigt werden müssen. Eine erste Einschätzung der Leistungsfähigkeit der Techniken ergibt, daß sie, einzeln genommen, nur gewisse Einzelaufgaben erfüllen, eine Kombination mehrerer Techniken aber geeignet erscheint, die identifizierten Aufgabenbereiche abzudecken. Diese Zuordnung soll noch einmal anhand der wichtigsten Techniken in Abbildung 22 dargestellt werden.

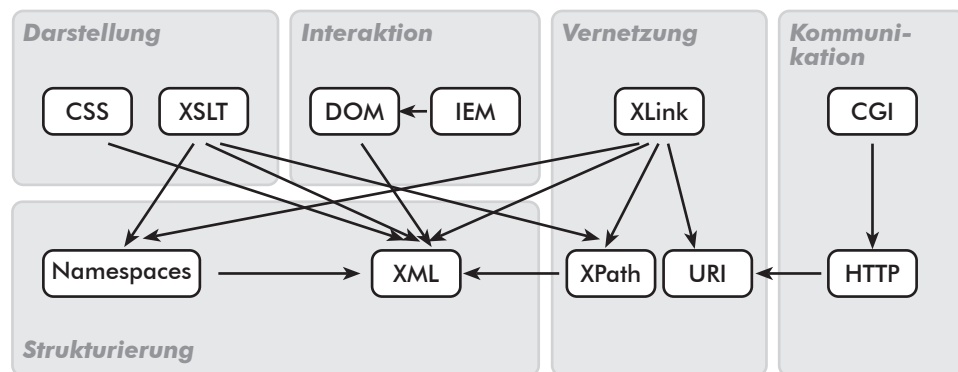


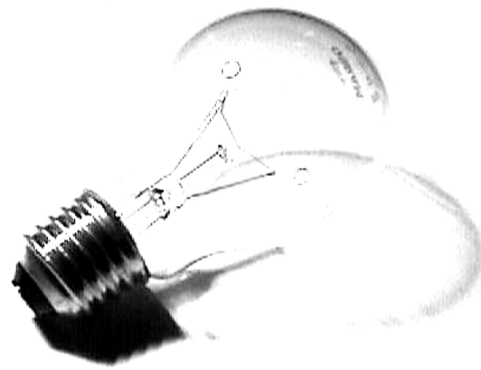
Abb. 22: Nutzungsbeziehungen zwischen grundlegenden Web-Techniken

In der Abbildung sind zusätzlich noch Benutzt-Beziehungen zwischen den Techniken aufgezeigt. So wird deutlich, daß gerade XML eine zentrale Rolle für die meisten anderen Standards spielt. Der Grund hierfür liegt unter anderem in der Notwendigkeit, die in Techniken wie XLink oder XSLT vorhandenen verwandten Informationen zu strukturieren, eine Aufgabe, für die sich XML besonders eignet. Daneben kommen den XML-Namensräumen für eine differenzierte Informationsstrukturierung, den Uniform Resource Identifiers für die Adressierung und dem Hypertext Transport Protocol für die Kommunikation besondere Bedeutung zu. Darauf aufbauend werden die Aufgabengebiete der Interaktion und Darstellung abgedeckt.

Die Anforderungsanalyse, die Auswahl von Basistechniken und das Erstellen eines Modells wird Aufgabe des nächsten Abschnitts sein.

Abschnitt B

Modell



10 Aufgabe des Modells

Nachdem die Grundlagen der Arbeit erläutert wurden, soll in diesem Abschnitt ausgehend von der in Kapitel 2 dargelegten Motivation ein grundlegendes Architekturmodell für Web-basierte Informationssysteme definiert werden. Das Modell beinhaltet dabei die nötigen Definitionen, um als Basis für eine Entwurfsmethode sowie für Anwendungs- und Entwicklungswerkzeuge zu dienen. Es wird aus einer Anforderungsanalyse und einer Analyse verwandter Bereiche und bestehender Modelle heraus gewonnen und soll die benötigten Eigenschaften so eindeutig und vollständig wie möglich beschreiben.

11 Anforderungen an das Modell

Die Anforderungsanalyse hat zum Ziel, die für das Einsatzgebiet des Modells relevanten Anforderungen zu erkennen, zu bewerten und hinsichtlich ihrer Erfüllbarkeit und eventueller Widersprüche zu untersuchen. Dementsprechend wird die Anforderungsanalyse in zwei Schritten durchgeführt: Identifikation von Anforderungen und Analyse der identifizierten Anforderungen.

11.1 Identifikation

Die vier wichtigsten Quellen für die Anforderungen an das zu entwickelnde Modell zeigt Abbildung 23 im schematischen Überblick. Zu beachten ist an dieser Stelle, daß sich die Anforderungen zum einen direkt auf das Modell beziehen, zum anderen aber auf Eigenschaften von Systemen, die nach dem Modell implementiert werden. Das Modell muß es also erlauben, Systeme zu entwickeln, die diesen Anforderungen gerecht werden.

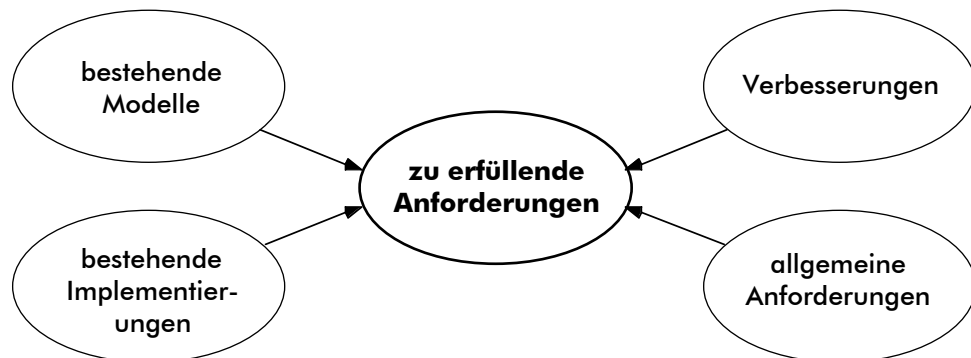


Abb. 23: Quellen für die Anforderungen an das Modell

Anforderungen aus bestehenden Ansätzen

Ein großer Teil der Anforderungen an das Modell läßt sich aus der Motivation dieser Arbeit ableiten: Entwicklung eines Architekturmodells, welches die Vorteile bisheriger Ansätze (gemeint sind damit sowohl Architekturmodelle als auch Implementierungen) vereint und spezifische Nachteile entweder vermeidet oder Mechanismen zur Umgehung bzw. Abmilderung bereitstellt.

Die wichtigsten Anforderungen lassen sich dabei direkt aus dem Aufgaben- und Einsatzgebiet Web-basierter Informationssysteme (vgl. dazu Unterkapitel 3.2) ableiten. Sie seien an dieser Stelle noch einmal kurz aufgezählt:

- Informationsstrukturierung
- Informationsvernetzung
- Kommunikation
- Informationsdarstellung
- Interaktion

Weitere Anforderungen ergeben sich aus den Eigenarten von Client- und Server-basierten Systemen und werden in der Folge behandelt.

Flexibilität

Ein Vorteil von Client-basierten Systemen besteht in der Möglichkeit, Änderungen der Funktionalität durch den Endbenutzer zu erleichtern und die Anpassung an nicht voraussehbare Einsatzgebiete zu ermöglichen. Generell betrachtet bedeutet die Forderung nach Flexibilität, daß die Architektur von Anwendungssystemen in größtmöglicher Weise variiert werden kann, ohne daß Funktionalität verloren geht. Eine wichtige Verbesserung gegenüber rein Client-basierten Systemen ist hier die flexible Verlagerung von Systembestandteilen von Client zu Server und umgekehrt. Dies impliziert eine möglichst geringe Kopplung zwischen den Systembestandteilen, angelehnt an das Ziel „Separation of Concerns“ [66], d.h. die Trennung von Systembestandteilen entsprechend ihrem Einsatzzweck.

Verständlichkeit und Wartbarkeit

Server-seitige Systeme sind durch ihre einfache Struktur (Ablauflogik nur im Server vorhanden) weniger komplex. Die Vorteile, die sich dadurch ergeben, wirken sich bei der Modifikation des Systems aus: die Implementierung des Systems ist leichter verständlich und nachvollziehbar, die Wartbarkeit ist erhöht. Die resultierende Forderung besagt, daß auch Systeme, die nach dem vorgestellten Modell entwickelt werden, diese Eigenschaften besitzen sollen.

Skalierbarkeit

Eng verwandt mit der Flexibilität eines Systems ist die Skalierbarkeit. Skalierbarkeit ist dann gegeben, wenn das System an veränderte Gegebenheiten (meist erhöhte Anforderungen an Leistungsfähigkeit oder Funktionalität) angepaßt werden kann. Darunter würde z.B. die Vergrößerung der Zahl der unterschiedlichen Dokumente im System oder die Erhöhung der Zahl der Client-Systeme fallen. Die Komplexität des Systems soll sich bei solchen Änderungen nicht überproportional vergrößern.

Neben der Skalierbarkeit von Systemen ist auch die Skalierbarkeit des Modells wichtig. Darunter soll die Möglichkeit verstanden werden, daß auch das Modell bei veränderten Anforderungen angepaßt werden kann, ohne daß die Komplexität übermäßig ansteigt. Vielmehr soll im Modell die Möglichkeit zukünftiger Erweiterungen vorgesehen werden.

Verbesserung bestehender Ansätze

Ein weiterer wichtiger Bereich von Anforderungen resultiert aus der Forderung nach der Verbesserung bestehender Ansätze. Dazu zählen Verbesserungen des Formali-

tätsgrades, Abstimmung unterschiedlicher oder widersprüchlicher Eigenschaften und neue Anforderungen.

Beschreibung von Verhalten

Da das Modell außer den Aspekten Client-basierter Systeme auch Server-basierte Systeme abdecken soll, muß die Forderung nach Interaktion erweitert werden, da diese üblicherweise nur vom Endbenutzer ausgelöst wird. Dementsprechend soll es das Modell erlauben, Aktivität oder Verhalten in Web-basierten Systemen zu beschreiben, unabhängig vom Ort der Implementierung und der Art und Weise der Aktivierung.

Zustandsbeschreibungen

Die Beschreibung des Systemverhaltens wird unterstützt durch die Beschreibung des Zustands der Systembestandteile. In klassischen Web-basierten Systemen sind die Zustände der Bestandteile durch die benutzten Techniken bestimmt, allerdings nur in begrenzter Form: entweder sind die möglichen Systemzustände sehr begrenzt (beispielsweise beim HTTP, welches Zustandsinformationen nur innerhalb eines Requests nutzen kann), oder die Beschreibung von Zuständen ist nur informell und implizit möglich (z.B. durch Verwendung von Kommentaren innerhalb von HTML-Dokumenten, Cookies, Einbettung von Zusatzinformationen in URLs oder versteckte Eingabefelder). Das Modell soll hier explizite Beschreibungsmittel einführen.

Verbindung von Zustand, Verhalten und Präsentation

Die oben genannten Beschreibungsmittel für Verhalten und Zustand sowie die unterschiedlichen Präsentationsformen müssen durch das Modell sinnvoll verbunden werden. Sehr deutlich wird diese Forderung bei der Verbindung von Verhalten und Zustand, da das Systemverhalten immer vom aktuellen Systemzustand abhängt. Es muß also möglich sein, auf den Zustand des Systems oder seiner Bestandteile in definierter Art und Weise zuzugreifen und diese Zustände auch zu verändern. Im weiteren Sinne umfaßt dabei die Zustandsbeschreibung auch die Art der Informationsdarstellung.

Wichtig ist eine sinnvolle Verbindung der Bestandteile unter anderem auch deshalb, weil die Forderung nach Flexibilität eine geringe Kopplung der Komponenten impliziert, die in geeigneter Weise konfiguriert eine Anwendung realisieren. Dafür sind wiederum Mechanismen notwendig, die diese Konfiguration oder das Zusammenfügen von Komponenten erlauben.

Definition von Semantiken

Der Einsatz des Modells soll durch eine weitere Forderung erleichtert werden, welche über die Anforderungen an die Basistechniken Web-basierter Systeme hinausgeht: Typische Anwendungsfälle des Modells sollen in Form von Semantiken identifiziert und formal definiert werden. Somit ergibt sich innerhalb des Modells eine Schichtenarchitektur, bei der auf der unteren Ebene einfache Beschreibungsmittel definiert werden, auf denen dann mit zunehmendem Abstraktionsgrad Semantiken aufbauen.

Allgemeine Anforderungen

Eine letzte Gruppe von Anforderungen deckt allgemeine Anforderungen, insbesondere Qualitätskriterien ab. Sie orientieren sich an Qualitätskriterien, wie sie auch für andere Software-Systeme gelten (siehe dazu etwa [66]).

Interoperabilität

Die wichtigste allgemeine Anforderung, die an das zu entwickelnde Modell gestellt wird, besteht in der Forderung nach Interoperabilität. Die Anforderung resultiert aus der Entwicklung und Struktur des Internets, insbesondere des World Wide Webs. Das Internet ist gekennzeichnet durch starke Heterogenität der daran beteiligten Systeme, der verwendeten Techniken und der eingesetzten Werkzeuge. Dies reicht von den eingesetzten Betriebssystemen über die genutzten Übertragungsprotokolle bis hin zur Einbindung von Legacy-Anwendungen. Um hier Interoperabilität zu erreichen, sollen einerseits offene Standards genutzt werden, andererseits ist diese Anforderung auch bei den zu erstellenden Systemen zu berücksichtigen.

Bei Standards, welche Internet-Techniken betreffen, muß dabei zwischen Herkunft und Ausprägung differenziert werden:

De-facto-Standards: Zu den De-facto-Standards gehören Techniken und Methoden, welche nicht durch Standardisierungsgremien definiert werden, jedoch weit verbreitete Anwendung finden und in ausreichender Weise stabil sind.

IETF-Standards: Die Internet Engineering Task Force (IETF) als Zusammenschluß von Herstellern, Organisationen und Einzelpersonen erarbeitet und verabschiedet einen Großteil der Standards im Internet-Bereich, darunter z.B. Protokollstandards wie TCP/IP oder HTTP. Die Entwicklung von IETF-Standards orientiert sich hauptsächlich an der Existenz von Implementierungen für bestimmte Entwürfe von Standards.

Empfehlungen des W3C: Das W3C ist ähnlich organisiert wie die IETF und erarbeitet mit ihr zusammen Empfehlungen, die das WWW betreffen, wie z.B. HTML oder XML. Wichtig ist hier die Bezeichnung *Empfehlung*, die aussagt, daß die Spezifikationen und Beschreibungen nicht bindend sind.

ISO-Standards: Am weitesten reichen die Bereiche, welche von der *International Standards Organization* (ISO) abgedeckt werden. Darunter befinden sich unter anderem der Standard zu SGML [84].

Die Berücksichtigung von Standards soll bei der Entwicklung des Modells so erfolgen, daß möglichst wenige, dafür aber um so stabilere Standards als Basis verwendet werden. Das schränkt die Verwendung von de-facto-Standards zwar ein, trotzdem ist durch die Standardisierungsprozesse des W3C und der IETF aber nicht ausgeschlossen, daß solche Standards innerhalb dieser Organisationen auch definiert werden. Ausgeschlossen als Basis für das Modell sind aber in jedem Fall proprietäre Erweiterungen von Standards. Sie dienen meist nur der Verbesserung bestimmter, begrenzter Teilaspekte und erschweren dadurch die universale Einsatzfähigkeit von Standards.

Auf- und Abwärtskompatibilität

Die Berücksichtigung von zukünftigen Erweiterungen ist gerade im Umfeld des Internets und des WWW besonders wichtig, wenn man die kurzen Entwicklungszyklen der benutzten Techniken betrachtet. Das zu erstellende Modell soll in dieser Hinsicht möglichst offen gestaltet sein, so daß neue Entwicklungen integriert werden können. Dies kann z.B. durch Festlegung von Erweiterungsschnittstellen erfolgen.

Andererseits soll gewährleistet sein, daß das Modell auch in solchen Umgebungen funktionsfähig ist, in denen spezielle Mechanismen und Beschreibungsmittel nicht

zur Verfügung stehen. Diese Forderung nach Interoperabilität soll im vorgestellten Modell besonderes Gewicht erhalten.

Implementierungsunabhängigkeit

Eine sehr wichtige Forderung, welche sich an die Forderung nach Aufwärtskompatibilität anschließt, ist die Implementierungsunabhängigkeit des Modells. Implementierungen sind immer stärker vom Wandel betroffen als Standards. Gründe dafür können Fehlerbereinigungen, Verbesserungen oder Erweiterungen sein. Deshalb soll das Modell zum einen nicht auf eine bestimmte Implementierung hin entwickelt werden, zum anderen soll es sich auch nicht auf bestehende Implementierungen stützen. Diese Aspekte werden in den Abschnitten *Werkzeuge* und *Anwendungen* behandelt.

Orthogonalität und Durchgängigkeit

Auch das Modell selbst soll eine möglichst geringe Komplexität (bei gleichzeitig großer Ausdrucksstärke) aufweisen. Die Reduktion der Komplexität soll durch zwei Mittel erreicht werden:

Orthogonalität: Beschreibungsmechanismen innerhalb des Modells sollen möglichst immer nur einem speziellen Zweck dienen und voneinander unabhängig verwendbar sein. Dadurch sollen Mehrdeutigkeiten ausgeschlossen und die Mechanismen speziell auf ihren Einsatzzweck hin zugeschnitten werden.

Konsistenz: Sind Mechanismen in mehreren Bereichen des Modells einsetzbar, so sollen sie möglichst gleichartig und eindeutig verwendet werden.

11.2 Analyse

Innerhalb der Analyse der Anforderungen soll eine Untersuchung der Zusammenhänge zwischen den einzelnen Anforderungen sowie eine Gegenüberstellung und Abstufung der Anforderungen erfolgen.

Konflikte zwischen Anforderungen

Zuerst werden die miteinander in Konflikt stehenden Anforderungen identifiziert und Lösungsansätze vorgeschlagen. Folgende Konflikte sind zu erkennen:

Flexibilität und Verständlichkeit : Die Flexibilität eines Systems beeinträchtigt die Verständlichkeit. Flexibilität bedeutet, daß die Systembestandteile möglichst abgeschlossen und unabhängig voneinander sind und auf viele verschiedene Arten kombiniert werden können. Damit steigt aber die Komplexität durch die hohe Anzahl von Systembestandteilen und Verbindungen zwischen ihnen. Komplexität hingegen ist hinderlich für die Verständlichkeit.

Innerhalb des Modells soll hier keine Präferenz festgelegt werden. Dies bedeutet, daß es das Modell ermöglichen soll, Systeme mit vielen, unabhängigen Komponenten genauso zu beschreiben wie Systeme mit wenigen Bestandteilen. Anders gesagt wird im Modell keine Aussage darüber getroffen, ob ein System eher Server- oder eher Client-basiert implementiert werden soll. Wichtig ist nur, daß beide Möglichkeiten gleichermaßen genutzt werden können.

Orthogonalität und Konsistenz: Orthogonalität bedeutet die Beschränkung des Einsatzgebietes von Mechanismen auf kleine, abgegrenzte Bereiche, während konsistente Verwendung von Mechanismen besagt, daß diese möglichst universell einsetzbar sein sollen.

An dieser Stelle soll im Modell der Schwerpunkt auf der Orthogonalität von Mechanismen liegen. Erst danach soll versucht werden, Mechanismen zu entwickeln, die durchgängig verwendbar sind, ohne jedoch die Orthogonalität des Modells zu beeinträchtigen.

Abhängige Anforderungen

Bei der Untersuchung der Anforderung auf Abhängigkeiten ergibt sich, daß die Forderung nach Mechanismen zur Informationsstrukturierung Grundlage einer Reihe weiterer Anforderungen ist:

Informationsvernetzung: Im einfachsten Fall muß bei der Vernetzung von Information kenntlich gemacht werden, welche Daten mit welchen anderen Daten verbunden werden. Es ist also mindestens die Unterteilung in Nutzdaten und Verknüpfungsdaten notwendig. Im Bereich der Web-Techniken sind aber meist feinere Abstufungen notwendig, so können beispielsweise die Verknüpfungsinformationen auch Nutzdaten sein. Zur Kennzeichnung dieser unterschiedlichen Informationstypen sollen die Mechanismen zur Informationsstrukturierung dienen.

Beschreibung von Verhalten und Zustand: Ähnlich wie die Kennzeichnung von Verknüpfungsinformationen baut auch die Beschreibung von Verhalten und Zustand auf Strukturierungsmechanismen auf. Verhalten und Zustand stellen, wie Verknüpfungsinformationen, eigenständige Typen von Information dar, welche gekennzeichnet werden müssen.

Eine ebenso wichtige Rolle wie die Informationsstrukturierung spielt die Kommunikation zusammen mit der Informationsvernetzung (die Distribution von Information über Kommunikationskanäle ist dabei unabhängig von der Vernetzung der Information). Darauf bauen folgende Anforderungen auf:

Flexibilität: Die Flexibilität eines Web-basierten Informationssystems wächst mit der Anzahl der Möglichkeiten zur Verteilung von Systembestandteilen. Die verteilte Information wird durch Mechanismen zur Vernetzung zusammengehalten.

Skalierbarkeit: Auch die Skalierbarkeit von Web-basierten Informationssystemen baut auf Mechanismen zur Verteilung von Information auf, insbesondere durch die Verteilung der Ablauflogik.

Zuletzt ist noch zu sagen, daß die Forderung nach Mitteln zur Verhaltensbeschreibung die Forderung nach Mitteln zur Beschreibung von Interaktion abdeckt, weswegen die Interaktion als gesonderte Anforderung nicht weiter betrachtet wird.

Bewertung der Anforderungen

Den höchsten Stellenwert besitzt die Gruppe von Anforderungen, die aus dem Aufgabenbereich Web-basierter Informationssysteme resultiert. Daraus ist insbesondere die Forderung nach Mechanismen zur Informationsstrukturierung und Kommunikation zu nennen, da sie die Basis für weitere Anforderungen bilden. Diese Anforderungen sind auf jeden Fall zu erfüllen.

Einen ähnlichen Stellenwert besitzen die allgemeinen Anforderungen, welche den Charakter von Qualitätsmerkmalen haben. Eine Ausnahme sind nur die Orthogonalität und die Durchgängigkeit der Mechanismen im Modell. Diese Anforderungen sind zueinander konkurrierend und deshalb nicht vordringlich zu erfüllen.

Die letzte wichtige Gruppe von Anforderungen trägt vornehmlich zur Einsatztauglichkeit und Ausdrucksfähigkeit des Modells bei. Sie umfaßt alle Anforderungen, die aus der Verbesserung bestehender Ansätze resultieren. Sie sind aber erst erfüllbar, wenn die grundlegenden Anforderungen realisiert sind. Trotzdem kommt ihnen im Rahmen dieser Arbeit eine besondere Rolle zu, da sie sich direkt aus der Motivation ergeben.

Die Anforderungen sollen für den weiteren Verlauf der Arbeit in funktionale und Qualitätsanforderungen eingeteilt werden, zusätzlich wird eine grobe Einteilung in Kern- und Randanforderungen vorgenommen. Abbildung 24 zeigt diese Einteilung.


Funktionale Anforderungen	Qualitätsanforderungen	
Strukturierung	Implementierungs- unabhängigkeit	Kern- bereich  Rand- bereich
Kommunikation	Interoperabilität	
Vernetzung	Auf-/Abwärtskompatibilität	
Zustandsbeschreibung	Flexibilität	
Verhaltensbeschreibung	Verständlichkeit/ Wartbarkeit	
Präsentation	Skalierbarkeit	
Verbindung von Verhalten/Zustand/Präsentation	Orthogonalität	
Semantiken	Konsistenz	

Abb. 24: Einteilung der Anforderungen

12 Verwandte Bereiche

Nachdem die Anforderungen an ein Architekturmodell gemäß der Aufgabenstellung der Arbeit analysiert worden sind, müssen vor dem eigentlichen Entwurf des Modells zuerst verwandte Bereiche untersucht werden. Darunter fallen Ansätze, die einzelne Teile der Implementierung umfassen, Verbesserung der Infrastruktur zum Ziel haben, die Information selbst betrachten oder umfassende Lösungen darstellen (eine Einteilung der verwandten Bereiche kann natürlich auch auf andere Art und Weise erfolgen).

Die verwandten Bereiche sind insofern wichtig, als sie entweder, ähnlich wie die Basistechniken, Konzepte beinhalten, die in den vorgestellten Ansatz übernommen werden sollten, oder aber Bereiche darstellen, zu denen der Ansatz abgegrenzt werden muß. Eine Abgrenzung soll dabei hinsichtlich der aufgestellten Anforderungen erfolgen.

12.1 Implementierungsebene

Auf der Implementierungsebene existieren vornehmlich solche Ansätze, die entweder nur die Client- oder nur die Server-Seite eines Web-basierten Informationssystems umfassen. Die jeweils andere Seite des Systems wird nicht betrachtet und bleibt im Vergleich zur einfachsten Form eines WWW-Systems (bestehend aus Web-Server und Web-Browser) unmodifiziert.

Server-seitige Systeme

Die wohl wichtigste und weitverbreitetste Technik, um Web-basierte Anwendungen zu implementieren, ist die CGI-Schnittstelle. Sie ermöglicht die Bearbeitung von einzelnen Anfragen an einen Web-Server durch Subprozesse. Dabei kann durch die Verwendung von HTTP als Transportmechanismus grundsätzlich keine Information über die realisierte Anwendung im Web-Server selbst gehalten werden. Alle anspruchsvolleren, CGI-basierten Anwendungen müssen demzufolge die Zustandsinformation über eigene Mechanismen ablegen, beispielsweise über Cookies, innerhalb von Web-Dokumenten oder innerhalb von URLs.

Der Großteil der Server-seitigen Erweiterungen stützt sich auf die CGI-Schnittstelle. Variationen bestehen meist aus den unterschiedlichen Möglichkeiten, auf der Server-Seite Applikationen zu implementieren, während auf der Client-Seite nur einfache Web-Browser eingesetzt werden. Zu den einfachen Erweiterungen zählen beispielsweise *PHP* [9], *W3Msql* [79], *ColdFusion* [2], *Java Server Pages* (JSP) [172] oder *Active Server Pages* (ASP) [190]. Dabei handelt es sich um Systeme, bei denen innerhalb bestehender HTML-Dokumente Programmskripte eingebettet sind. Diese werden einmalig ausgeführt, wenn das umgebende HTML-Dokument von einem Client angefordert wird. Sie stellen meist keinen Mechanismus zur Zustandssteuerung der damit implementierten Applikation zur Verfügung.

Eine Alternative zu den CGI-basierten Erweiterungen sind Mechanismen, die eine stärkere Integration in den Web-Server erlauben. Dadurch wird beispielsweise vermieden, für jeden Aufruf einen neuen Betriebssystemprozeß zu starten. Zu diesen Mechanismen zählen *Servlets* [171] (siehe Seite 63 für eine Einordnung in ein umfangreiches Rahmenwerk), *Apache Module* [4] oder das *Netscape Server API* (NS-API) [125].

Es existieren aber auch umfangreichere Systeme, die zur Implementierung von Applikationsservern dienen können. Ihnen ist gemeinsam, daß sie Objekte oder Entitäten eines Anwendungsbereichs über HTTP zur Verfügung stellen, d.h. eine Anwendung Web-fähig machen. Einige dieser Systeme sollen hier vorgestellt werden.

WebObjects

Einer der ersten verfügbaren Web-Applikationsserver ist das WebObjects-System [6] der Firma Apple (vormals NeXT). WebObjects stellt ein Rahmenwerk für Web-basierte Anwendungen zur Verfügung, bei denen drei Bereiche modelliert werden können: Anteile der Applikation im Server, Anteile der Applikation im Client und die zugrundeliegende Datenhaltung. Die Teile der Applikation, die in der Client-Software realisiert sind, sind optional, so daß eine WebObjects-Anwendung auch als rein Server-basiertes System realisiert werden kann. Die allgemeine Systemarchitektur ist in Abbildung 25 dargestellt.

Web-Objects-Anwendungen bestehen aus mehreren Bestandteilen. Dazu zählen die Komponenten, die die Anwendungslogik realisieren (in der Regel Klassen der Programmiersprachen Java, Objective-C oder der WebObjects-eigenen Programmiersprache WebScript) und die WebComponents, welche wiederverwendbare Darstellungselemente für die HTML-Ausgabe kapseln. Das WebObjects-Framework stellt eine Applikations- und Session-Klasse zur Verfügung, die die fehlende Funktionalität zur Zustandsbeschreibung in HTML nachbildet.

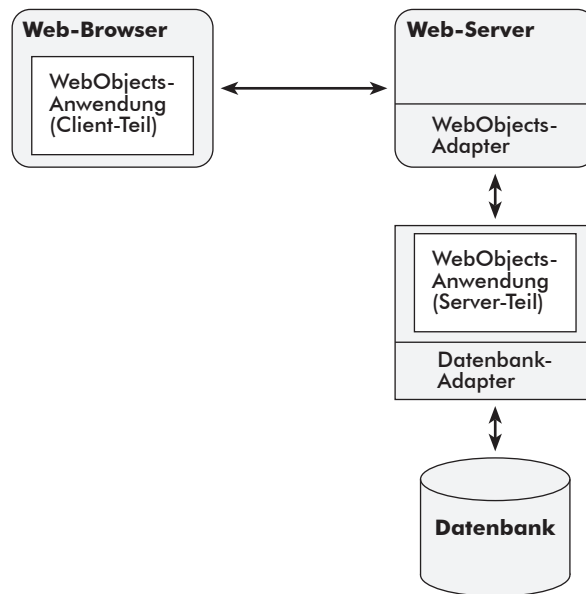


Abb. 25: Architektur einer WebObjects-Anwendung

Untersucht man das WebObjects-System hinsichtlich der im vorigen Kapitel aufgestellten Anforderungen für Web-basierte Informationssysteme, so fallen einige Nachteile auf. Zwar sind viele der funktionalen und der Qualitätsanforderungen erfüllt, allerdings sind bei der Informationsstrukturierung Schwachpunkte zu entdecken. Die Basis für WebObjects-Anwendungen stellen die in Klassen formulierte Applikationslogik, die Datenhaltung in Form von Datenbanken und die Präsentation als HTML-Dokumente dar. Dies bedeutet aber Brüche bei der Informationsverarbeitung an zwei Stellen: am Übergang zwischen Datenhaltung und Applikationslogik und am Übergang zwischen Applikationslogik und Darstellungsschicht bzw. der Client-Software. Information kann also nicht durchgängig unter den gleichen Gesichtspunkten strukturiert und verarbeitet werden, sondern ist Modifikationen unterzogen. Ein weiterer wesentlicher Kritikpunkt ist die fehlende Offenheit und Implementierungsunabhängigkeit des Systems, da es nur für wenige Betriebssystem-/Hardware-Plattformen verfügbar ist (Microsofts Windows NT, Apples MacOS X, Sun Microsystems Solaris). Die Offenheit läßt sich in nur geringen Maßen durch die Verteilbarkeit einzelner Komponenten wie der Datenbank oder der Client-seitigen Teile der Anwendung erhöhen. Der Kern der Anwendungen, die Applikationslogik, ist jedoch nicht portabel.

Zope

Zope [51] ist in erster Linie ein *Object Publishing System*, d.h. ein System, um Objekte im Web zu publizieren. Es basiert auf der objektorientierten Programmiersprache Python [14] und erlaubt die Erstellung von Python-Anwendungen, die mit dem Endbenutzer über ein HTML-Interface kommunizieren. Die Objekte werden im Regelfall persistent in einer Objekt-Datenbank angelegt und sind ähnlich wie ein Dateisystem hierarchisch in Ordnern (welche wiederum Objekte sind) angeordnet. Die Publikation der Objekte geschieht über den Zope-Publisher, welcher vom Web-Server z.B. über die CGI-Schnittstelle angesprochen wird. Der Publisher enthält einen

Object Request Broker, der Operationen auf den Informationsobjekten koordiniert. So ergibt sich eine Architektur, wie sie in Abbildung 26 zu sehen ist.

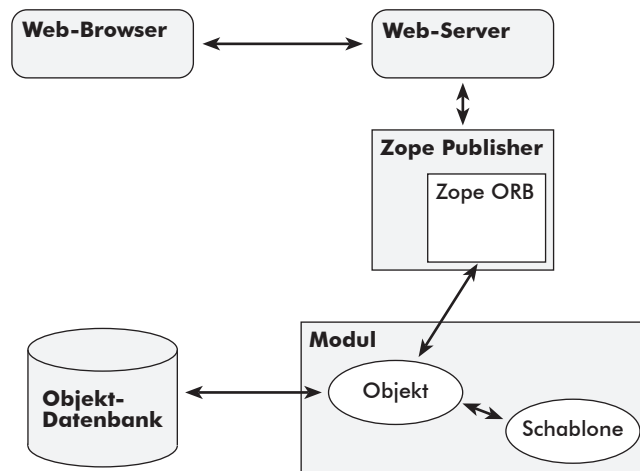


Abb. 26: Architektur von Zope

Zope zeichnet sich durch eine vorgefertigte Systemstruktur für vereinfachtes Anlegen, Modifizieren und Verwalten von Informationsobjekten aus. So ist es über ein HTML-Interface möglich, Informationen dynamisch zu publizieren, ohne daß Anwendungscode erstellt werden muß. Dazu sind vordefinierte Klassen für unterschiedliche Informationsobjekte verfügbar.

Erweiterte Möglichkeiten bieten zum einen die Bildung von Subklassen der vordefinierten Klassen, so daß neue Informationsobjekte für spätere Anwendungen ohne zusätzliche Programmierung zur Verfügung stehen. Zum anderen können auch generische Python-Klassen in das System integriert werden, da Zope fast durchgängig in Python realisiert ist.

Zope eignet sich gut für das dynamische Publizieren von Informationen, allerdings sind diese Möglichkeiten an das zugrundeliegende Objektmodell und die oben dargestellte Systemarchitektur gebunden. So entstehen, ähnlich wie beim WebObjects-System, Brüche im Informationsfluß. Allerdings ist Zope vollständig im Quelltext verfügbar und wird als Open-Source-Projekt entwickelt, so daß Anpassungen möglich sind. So sind z.B. neben HTTP weitere Zugangsprotokolle wie FTP [144] oder WebDAV [68] bereits realisiert.

ActiWeb

Eine interessante Weiterentwicklung der Idee des Applikationsservers und des Object Publishing Systems ist das ActiWeb-System [196]. Es erweitert solche Systeme um die Möglichkeit, Systembestandteile auch als mobile Komponenten zu realisieren. Da hierbei nicht nur Informationen als mobil erachtet werden, sondern auch Programmbestandteile, handelt es sich technisch gesehen um ein *Mobile Code System* (MCS), mit dem beispielsweise Agentensysteme [62] realisiert werden können. Die Architektur, die in Abbildung 27 dargestellt ist, soll im folgenden kurz beschrieben werden.

Die Basis für ActiWeb ist die Programmiersprache XOTcl [130], eine objektbasierte Skriptsprache, die auf Tcl [135] basiert. Darauf aufbauend dienen unterschiedliche Dienste zur Realisierung der Basisfunktionalitäten. Zu nennen sind insbesondere der

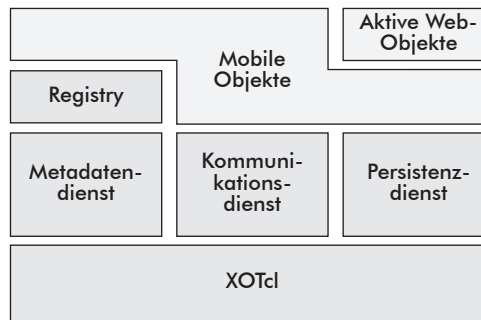


Abb. 27: Architektur des ActiWeb-Systems

Kommunikationsdienst für die Unterstützung mobiler Komponenten und externer Zugriffe, der Persistenzdienst für die dauerhafte Speicherung von Informationen und der Metadatendienst mit der Registry für die Koordination der mobilen Bestandteile. Systeme werden in ActiWeb aus XOTcl-Objekten zusammengestellt, welche ähnlich wie in Zope ein Web-Interface besitzen können. Mit den so entstehenden aktiven Web-Objekten ist eine Interaktion des Benutzers mit dem System über einen Web-Browser möglich. Daneben können die Systemkomponenten aber auch weitere Schnittstellen besitzen, etwa in Form von XML-Dokumenten. ActiWeb ermöglicht damit die Implementierung multivalenter Dokumente wie sie etwa in [141] eingesetzt werden, um unterschiedliche Sichten auf komplexe Objekte zu realisieren.

Wie auch Zope und WebObjects ist ActiWeb objektzentriert, d.h. es stellt ein objektorientiertes System mit einem Web-Interface dar. Die Information ist im System innerhalb der Objekte verankert. Die Objekte können auf unterschiedliche Weise repräsentiert werden, wobei die externe Repräsentation dynamisch erzeugt wird. Es ist bei der Verwendung dieser Repräsentation darauf zu achten, daß sie eng an den Objektzustand gebunden ist und sich verändern kann. Aus diesem Grunde sollte sie nicht losgelöst vom Objekt verwendet werden, da ansonsten Inkonsistenzen entstehen können.

Zur Zeit ist das ActiWeb-System an eine konkrete Implementierung gebunden, wodurch die Offenheit der Systemarchitektur für beliebige Erweiterungen noch reduziert ist. Dieser Nachteil implementierungsgebundener Systeme ließe sich z.B. durch eine durchgängige offene Schnittstellenbeschreibung vermeiden, welche auch fremde Komponenten innerhalb des Systems integrieren kann.

Client-seitige Systeme

Neben den Implementierungen, die auf der Server-Seite eines Web-basierten Informationssystems realisiert sind, existieren Techniken und Werkzeuge, die die Anwendungslogik in der Client-Software umsetzen.

JavaScript

Eine weit verbreitete Technik, um Web-Anwendungen Client-seitig um Applikationslogik anzureichern, ist JavaScript [127], wovon mittlerweile der Sprachkern als ECMAScript [59] standardisiert worden ist. JavaScript ist objektbasiert und stellt eine Reihe von Objekten und Objektklassen zur Verfügung, auf die innerhalb eines Web-Browsers zugegriffen werden kann (diese Objekte dienen teilweise als Ausgangsbasis für die Interfaces in der DOM-Spezifikation, siehe Unterkapitel 8.2). Ja-

vaScript ist nicht objektorientiert, da es nicht erlaubt, neue Abstraktionen in Form von Klassen zu erstellen bzw. von den bestehenden Objektklassen zu erben. Es existiert somit nur eine flache Typhierarchie ohne Vererbungsbeziehungen.

Die nutzbaren Objektklassen können in allgemein nutzbare Klassen wie Datums-, Zeichenketten- oder einfache Datenstruktur-Klassen und speziell auf die Bearbeitung eines Dokuments innerhalb des Browsers ausgerichtete Klassen eingeteilt werden. Klassen besitzen Eigenschaften und Methoden, zusätzlich wird für Klassen, die für die Dokumentverarbeitung genutzt werden können, in einem Ereignismodell festgelegt, auf welche Benutzerinteraktionen sie reagieren können (das Ereignismodell ist in HTML 4.0 mit dem IEM genauer spezifiziert, siehe Unterkapitel 8).

Wichtige Objekte für den Zugriff auf das Dokument im Web-Browser und bestimmte Teile des Web-Browsers sind *document*, *screen*, *window* und *navigator*. So bietet das Objekt *document* Funktionen, um auf alle HTML-Anchor-Elemente, alle FORM-Elemente, die aktuelle URL des Dokuments und weitere Eigenschaften zuzugreifen. Auch die Manipulation ist z.B. über die Funktion *write* möglich, mit der auf einfache Art und Weise der Quelltext des Dokuments verändert werden kann. Auch eine direkte Bearbeitung der Elemente kann über Klassen wie *Anchor*, *Button*, *Form* oder *Image* implementiert werden.

JavaScript ist eng mit dem Dokumentenmodell von HTML sowie einem Web-Browser als Client-Software verbunden. Zwar existiert eine Spezifikation, die JavaScript im Web-Server nutzbar machen soll [128], dennoch ist die Flexibilität und Interoperabilität durch diese starken Bindungen eingeschränkt. Zudem wird in JavaScript keine Trennung von extern nutzbarer Funktionalität und Programmiersprache aufrecht erhalten. Mit ECMAScript wird zwar der Sprachkern beschrieben, es fehlt allerdings eine formale Spezifikation der in Web-Anwendungen benötigten Funktionalitäten.

Plug-In-Komponenten

Die einfachste Form, eine Web-Anwendung in der Client-Software zu realisieren, stellen in Verbindung mit HTML die sog. *Plug-Ins* dar (ein Beispiel sind Netscape-Plug-Ins, beschrieben in [126]). Sie sind sehr eng an eine konkrete Client-Implementierung und -Plattform gebunden und sind damit einerseits nicht portabel, andererseits haben Plug-Ins weitgehende Zugriffsmöglichkeiten auf den Web-Client und die Betriebssystemumgebung. Die Realisierung der Anwendungslogik kann in einem Plug-In in unterschiedlichen Programmiersprachen erfolgen, fixiert ist nur die Zugriffsschnittstelle auf die Client-Software.

Plug-Ins werden als eingebettete Komponenten in einer Web-Seite genutzt. Sie stehen im Regelfall nicht mit dem umgebenden Dokument in Verbindung, es gibt keine Funktionen, um aus dem Plug-In heraus das Dokument strukturiert zu modifizieren oder aus dem Web-Browser heraus das Plug-In zu steuern. So werden Dokumente für Plug-Ins nur als Container für die Einbettung in den Client-Browser genutzt.

Es existieren unterschiedliche Ausprägungen von Plug-In-Realisierungen, die sich beispielsweise in der Breite der Zugriffsschnittstelle auf das ausführende Programm oder das umgebende Dokument unterscheiden. Einige davon werden weiter unten näher beschrieben.

Applets

Eng verwandt mit dem Plug-In-Konzept ist die Idee der *Applets*, welche kleine Anwendungen oder Anwendungsfragmente in eine HTML-Seite einbetten. Die wohl bekannteste Applet-Technik sind *Java-Applets* [170], welche portablen Programmcode entsprechend der Java-Spezifikation [107] enthalten. Andere Techniken sind beispielsweise *Oplets* [36] oder *Tclets* (siehe [106] und [160]). Eine Spezialisierung sind beispielsweise *Displets* (beschrieben in [39] und [181]), die die flexible Darstellung neuer Informationsstrukturen (d.h. XML-Elemente oder neuer HTML-Elemente) erlauben.

Applets erweitern das Konzept des Plug-Ins um eine Spezifikation für die Konstruktion der Anwendungsfunktionalität, etwa durch die Festlegung einer Programmiersprache, verwendbarer Programmierschnittstellen oder Programmbibliotheken. Der wichtigste Punkt ist aber die Plattformunabhängigkeit von Applets. Sie wird durch die Entkopplung der plattformspezifischen Ausführungsumgebung vom eigentlichen Anwendungscode im Applet selbst erreicht. Daraus folgt, daß für unterschiedliche Plattformen und Client-Programme nur die Ausführungsumgebungen angepaßt werden müssen, während die darin ablaufenden Applets portabel sind. Im Fall der *Tclets* (und teilweise auch bei *Java-Applets*) wird diese Ausführungsumgebung tatsächlich als Plug-In realisiert. Eine weitere Laufzeitumgebung ist *Omniware* [108], welches die darin ablaufenden Programme nicht an eine Programmiersprache bindet, sondern stattdessen auf der Übersetzung der Programme von der Ursprungssprache (C, C++, Tcl usw.) in die Zielsprache der Laufzeitumgebung beruht.

Die so entstehende Architektur, die der Verwendung von Applets zugrundeliegt, hat so zwar das Problem der Plattformunabhängigkeit gelöst, andererseits fehlen immer noch weitergehende Zugriffsmöglichkeiten auf die Client-Software und das umgebende Dokument.

Komponenten-basierte, erweiterbare Client-Software

Ein Weg, um auf der Client-Seite Web-basierter Anwendungen mehr Flexibilität zu erreichen, besteht in der Öffnung der Architektur der dort eingesetzten Programme. Dies kann durch eine Zerlegung der Client-Software in Komponenten geschehen, welche definierte Schnittstellen haben und über externe Programme angesprochen werden können. Wird Anwendungslogik ähnlich wie bei Applets in HTML-Dokumente eingebunden, so können die Komponenten aus den Dokumenten heraus angesprochen werden (siehe dazu z.B. Abbildung 28).

Beispielhaft für diesen Ansatz sollen hier drei Systeme vorgestellt werden, die unterschiedliche Realisierungen der Architektur demonstrieren.

Der Web-Browser *Plume* (vormals *SurfIt!*), beschrieben in [10] und [11], ist durchgängig in *Tcl* [135] unter Benutzung von *Tk* [136] für die Benutzeroberfläche, implementiert. Zentraler Bestandteil ist die *Document Handling Package* (DHP), die den Netzwerk- und Dateizugriff sowie die Steuerung der Weiterverarbeitung der HTML-Dokumente umsetzt. Weiterhin existieren Komponenten für das Parsen und die Darstellung der Dokumente. Sie sind im *WWW-Megawidget* eingebettet. Das Megawidget ist auch in der Lage, in Dokumente eingebetteten Tcl-Code auszuführen. Da der gesamte Browser in Tcl implementiert ist, können so alle Komponenten inkl. DHP und Megawidget verwendet werden. Eine Anwendung ist die Steuerung von HTML-FORM-Elementen, z.B. zum Überprüfen der Korrektheit von Benutzereingaben [177]. *Plume* ist auf die Erstellung eigenständiger Anwendung aufbauend auf einem

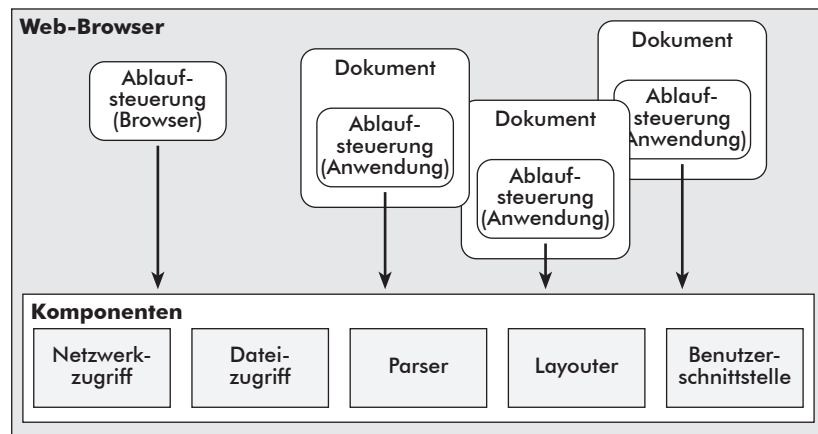


Abb. 28: Aufteilung eines Web-Browsers in Komponenten

Web-Browser ausgerichtet, da die ursprüngliche Anwendung beliebig erweitert werden kann.

Hush [55] ist eine vergleichbare Implementierung mit dem Unterschied, daß Hush als Programmbibliothek in der Programmiersprache C++ realisiert und in Tcl/Tk-Programme eingebettet ist. Es ist aber auch möglich, Hush in C- oder C++-Programmen einzusetzen. Die Web-Funktionalität (Netzwerkzugriff, HTML-Verarbeitung und -Anzeige) wird in Hush mittels des *web-Widgets* umgesetzt.

Eine Erweiterung des Funktionsumfanges der Client-Software wird durch Systeme wie das *Widget (Athena) Front End (Wafe)* [129] erreicht, die einen generellen Mechanismus zur Einbindung von Komponenten in Anwendungsprogramme bieten. Wafe ist eine Tcl-basierte Umgebung, in die bestehende Funktionsbibliotheken wie etwa das *Kino-Widget* [97] zur Anzeige von HTML-Dokumenten eingebunden werden und mittels Tcl die Anwendungslogik realisiert wird. Eine Anwendung ist der *Cineast-Web-Browser* [98], der u.a. Bibliotheken für die sichere Datenübertragung oder die Umwandlung von Bildformaten nutzt.

Generell nähert man sich bei der Aufteilung der Client-Software in Komponenten der Entwicklung eigenständiger Anwendungen mit spezieller Funktionalität an. In solchen Anwendungen werden Standards wie HTML und HTTP nur noch für den Transport und die Darstellung von Informationen genutzt, die Information, welche aus dem jeweiligen Anwendungsbereich stammt, ist hauptsächlich innerhalb der Anwendungen verborgen. Auch hier ergeben sich also Brüche im Informationsfluß, wenn eine externe Anwendung auf diese internen Informationen zugreifen will. Dadurch ist die Interoperabilität eingeschränkt.

Aktive Tioga-Dokumente

Einen sehr interessanten Bereich stellen Weiterentwicklungen einfacher Dokumente zu aktiven Dokumenten dar, die ein allgemeineres Modell der Plug-In- und Applet-Architektur beschreiben. Auf diesem Gebiet existieren einige wichtige Werkzeuge und Ansätze, die im folgenden diskutiert werden sollen. Sie entstanden zum großen Teil vor dem Erfolg des World Wide Web (sprich HTML und HTTP) und der Verbreitung von Standards im XML-Umfeld, weshalb teilweise grundlegende Bestandteile mit spezifiziert werden mußten. Dennoch sind zumindest die theoretischen Erkenntnisse der Ansätze nicht zu vernachlässigen.

Ein erster Ansatz wurde mit dem *Tioga*-Editor [176] realisiert. Hier liegen die Schwerpunkte bei der dynamischen Transformation und den durch Modifikation eines Dokuments ausgelösten Aktivitäten. Das Dokumentenmodell umfaßt dabei hierarchisch angeordnete Knoten, die beliebige Eigenschaften haben können. Dazu zählen der textuelle Inhalt und dessen Formatierung sowie eine Verbindung zu externen Beschreibungen von Aktivität (d.h. Programmskripte). Das Ereignismodell sieht vor, daß Transformationsvorschriften mit jedem einzelnen Knoten verbunden werden können, während auf Modifikation des Dokuments nur über den Wurzelknoten reagiert werden kann.

Eine Erweiterung [21] des *Tioga*-Editors nutzt dessen Fähigkeiten, um neben der Modifikation eines Dokuments auch andere Benutzerinteraktionen zuzulassen, indem Teile eines Dokuments als Bereiche (*Buttons*) markiert werden, die der Benutzer per Mausklick aktivieren kann. Hinter den Buttons können sich beliebige Aktionen verbergen.

Eine weiteres interessantes Projekt im Umfeld des *Tioga*-Systems sind *Scripted Documents* [197]. Hier wird versucht, durch vordefinierte Skripte den Benutzer bei der Navigation durch eine Hypertextstruktur zu unterstützen. Durch die Skripte sollen dem Benutzer vom Autor programmierte Pfade vorgeschlagen werden oder zusätzliche Information übertragen werden. Skripte werden separat vom Dokument gespeichert und können beliebige Aktionen durchführen (z.B. Tondokumente abspielen, Systemkommandos ausführen oder Funktionen des *Tioga*-Editors aufrufen). Sie werden an Knoten innerhalb eines Dokuments gebunden und können vom Benutzer im Editor aktiviert werden. Eine Anwendung dieses Ansatzes für die Navigation durch Multimedia-Dokumente ist in [37] beschrieben.

Andrew Toolkit

Umfangreichere Rahmenwerke stellen komponentenbasierte Systeme dar. Sie bauen auf einem Dokumentenmodell auf, in dem Software-Komponenten verschachtelt werden können, die jeweils unterschiedliche Inhalte (Text, Grafiken, Tabellen usw.) enthalten. Die Komponenten müssen dazu bestimmten Programmierschnittstellen entsprechen, erlauben damit aber eine anwendungsunabhängige Einbettung. Dies steht im Gegensatz zu den weiter oben angesprochenen komponentenbasierten Client-Programmen, die nur über einen begrenzten Satz von Komponenten verfügen, der zudem nicht in dem Dokumenten selbst sondern nur im Client-Programm eingebettet ist.

Neben dem jüngeren *Component Object Mode* (COM) [116] von Microsoft und dem nicht mehr weiterentwickelten *OpenDoc* [123] von Apple und IBM ist das *Andrew Toolkit* (ATK) [138] eines der ersten Rahmenwerke für Komponenten. Aufbauend auf dem ATK wurde mit *Ness* [76] ein Ansatz implementiert, der die Verbindung der Komponenten in einem Dokument mit Programmskripten ermöglicht. Programmskripte werden innerhalb eines Dokuments nur an einer Stelle abgelegt, die Verbindung mit den einzelnen Komponenten erfolgt über einen eindeutigen Komponentennamen. Wie auch bei den oben dargestellten Erweiterungen werden die Programmskripte als Reaktion auf Benutzerinteraktionen oder von andere Komponenten generierte Ereignisse ausgeführt. Die Aktionen, die in den Programmskripten ausgeführt werden können, sollen zumindest die Operationen eines Benutzers nachbilden können (d.h. Bearbeitung von Text, Auswahl von Menüpunkten usw.), die Komponenten können aber auch weitere Funktionalitäten exportieren. Die Doku-

mente, die durch verschachtelte Komponenten gebildet werden, können in einer textuellen Repräsentation abgespeichert werden, die allerdings nur eingeschränkt von Endbenutzern modifizierbar sein soll.

Das ATK und auch Ness sind stark an eine Implementierung gebunden, die zwar eine gewisse Portabilität aufweist, dennoch ist Implementierungsunabhängigkeit nur beschränkt vorhanden. Dies wäre zwar durch eine stärkere Betonung der externen Repräsentation der komponentenbasierten Dokumente behebbar, allerdings scheint das ATK zur Zeit nicht weiterentwickelt zu werden. Auch fehlen Mechanismen, die einen Transport von Informationen und deren systemübergreifende Vernetzung erlauben.

Interleaf

Eine Umsetzung aktiver Dokumente befindet sich in der *Interleaf*-Dokumentenmanagement-Software. Die Architektur aktiver Interleaf-Dokumente ist in [58] beschrieben. Dort werden auch die Anforderungen an Systeme aufgeführt, welche die Bearbeitung aktiver Dokumente erfüllen sollten:

- Es muß ein zugrundeliegendes Dokumentenmodell vorhanden sein, welches die Informationen strukturiert und möglichst vielfältige Elemente enthält. Diese Elemente sollen sich an den üblicherweise in Dokumenten vorkommenden Bestandteilen (Überschriften, Absätze, Fußnoten usw.) orientieren.
- Es muß eine Programmierschnittstelle für den Zugriff auf alle Funktionen des Editors geben, die für die Bearbeitung eines Dokuments benötigt werden.
- Das System sollte objektorientiert aufgebaut sein, um die Wiederverwendung von Komponenten zu fördern.
- Es sollte Objekte geben, die als Agenten des Benutzers Dokumente modifizieren können.
- Das System sollte die Erstellung aktiver Dokumente durch eine Entwicklungsumgebung unterstützen.

Es ist ersichtlich, daß einige dieser Anforderungen sich mit den Anforderungen, die in Kapitel 11 identifiziert worden sind, decken. Allerdings fehlen wichtige Anforderungen wie die Kommunikation und Vernetzung, die bei der Realisierung Web-basierter Informationssysteme essentiell sind.

Grif, Thot und Amaya

Die letzte Gruppe von Techniken, die hier beispielhaft für die Implementierung von Applikationslogik auf der Client-Seite Web-basierter Informationssysteme vorgestellt wird, umfaßt den Editor *Grif* [147] und die *Thot*-Bibliothek [154] sowie in deren Umfeld entstandene Arbeiten.

Wie auch Tioga, ATK und Interleaf zielen Grif und die daraus hervorgegangene Thot-Bibliothek auf die Bearbeitung strukturierter Dokumente. Durch den Bibliothekscharakter von Thot können so gewisse Teile einer komponentenbasierten Client-Software realisiert werden. Die damit verfügbare Funktionalität zur Modifikation von Dokumenten kann auch in Programmskripten genutzt werden, die aufgrund von Benutzerinteraktionen aktiviert werden. Daneben existiert mit der *External Call Facility* (ECF) ein Mechanismus, auch Funktionen im umgebenden Programm aufzurufen.

Als Anwendungen werden in [147] u.a. die Transformation von Dokumentstrukturen (genauer beschrieben in [23]), kooperatives Erstellen von Dokumenten oder die syn-

taxgesteuerte Bearbeitung genannt. Eine Anwendung der Thot-Bibliothek ist der Web-Browser Amaya [74].

Ein interessantes Konzept innerhalb des Grif-Editors sind die sog. *Generic Structure Extensions* (GSE). Sie ermöglichen die Erweiterung einer Dokumentenstruktur um applikationsabhängige Komponenten, ähnlich der Einbettung neuer Elemente in XML-Dokumenten über XML-Namensräume. Dieser Mechanismus fördert die Interoperabilität in solchen Fällen, in denen nicht alle Funktionalitäten, die innerhalb eines Dokuments genutzt werden, vorhanden sind, der Inhalt des Dokuments aber trotzdem verarbeitet werden soll.

Web-Programmiersprachen

Ein wichtiger Teilbereich Web-basierter Informationssysteme sind die Programmiersprachen, mit denen das Systemverhalten festgelegt wird. Neben universell einsetzbaren Sprachen wie Perl oder Tcl, welche durch Bibliotheken oder andere Erweiterungen an den Einsatz im Web angepaßt werden, existieren auch Programmiersprachen, die gezielt für Web-basierte Informationssysteme entwickelt wurden. Hier sollen drei Programmiersprachen beschrieben werden, die unterschiedliche Aspekte von Web-Systemen abdecken und damit zusätzliche Bereiche aufzeigen, aus denen Anforderungen an Werkzeuge und Techniken resultieren.

MAWL

Die Programmiersprache MAWL [103] dient zur Implementierung von Anwendungslogik auf der Server-Seite. MAWL zielt dabei auf Fehlervermeidung, Wiederverwendung sowie die Überwindung der Zustandslosigkeit von HTTP ab. Beim Entwurf der Programmiersprache wurde davon abgesehen, eine existierende Programmiersprache um spezielle Eigenschaften zu erweitern oder eine spezielle, angepaßte Programmiersprache zu entwerfen, die dann um allgemeinere Konstrukte zu erweitern ist. Stattdessen wurden nur die zum Erreichen der angesprochenen Ziele benötigten Konstrukte in MAWL umgesetzt, während allgemeinere Aufgaben durch eine Wirtssprache realisiert werden.

Anwendungen werden als *MAWL-Services* implementiert, wobei ein MAWL-Service aus einem MAWL-Programm und entsprechend erweiterten HTML-Dokumenten besteht. Ein Service kann aus mehreren *Sessions* aufgebaut sein, die ähnlich wie Prozeduren einer funktionalen Zerlegung eines Programms dienen, etwa um Abläufe in einer Anwendung zu gliedern. Zwischen den Sessions können Informationen mittels Variablen ausgetauscht werden, es wird zwischen globalen (d.h. für alle Sessions in einem Service sichtbaren) und statischen (mit persistentem Inhalt) Variablen unterschieden.

Fehlervermeidung soll in MAWL durch die Übersetzung der Services erreicht werden, d.h. es wird auf eine Trennung zwischen Laufzeit- und Übersetzungszeit Wert gelegt, um Fehler vorzeitig zu entdecken. Auch sollen dadurch Optimierungen ermöglicht werden.

WebL

Im Gegensatz zur MAWL versucht die Programmiersprache *WebL* [95], die Implementierung von Web-basierten Client-Programmen zu erleichtern. WebL ist daneben eine universelle Programmiersprache, die vornehmlich für die prototypgestützte Entwicklung von Programmen benutzt werden kann. Das Verarbeitungs-

modell, welches WebL zugrunde liegt, geht von einer sequentiellen Verarbeitung von Web-Ressourcen aus, wie es Abbildung 29 zeigt.

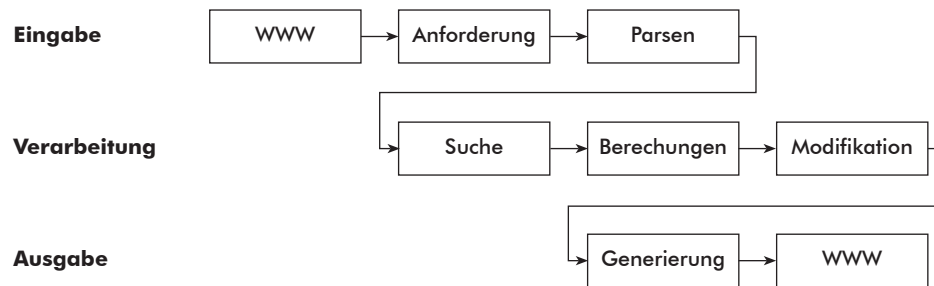


Abb. 29: Verarbeitungsmodell von WebL

In der Eingabephase wird die Anforderung einer Web-Ressource durch *Service Combinators* vereinfacht. Sie bilden bestimmte Web-spezifische Aktionen nach (sog. *Web Reflexes*), etwa die erneute Anforderung einer Ressource bei einer abgebrochenen Verbindung, die parallele und konkurrierende Anforderung von Ressourcen oder den Abbruch einer Verbindung bei Zeitüberschreitung.

In der Verarbeitungsphase kommt eine Auszeichnungsalgebra zum Einsatz, die die Adressierung einzelner Bestandteile und Mengen von Bestandteilen einer HTML-Ressource, ähnlich XPointer, direkt im Programmtext erlaubt.

Betrachtet man die in Kapitel 11 aufgestellten Anforderungen, so wird deutlich, daß durch WebL die Kommunikation, -strukturierung und -vernetzung erleichtert werden soll.

Java-Plattform als übergreifender Ansatz

Neben den Ansätzen, die nur Teilaspekte Web-basierter Informationssysteme betrachten, existieren auch umfassende Konzepte zur Beschreibung und Implementierung solcher verteilter Systeme. Stellvertretend soll hier die Java-Plattform beschrieben werden, welche ein umfangreiches Rahmenwerk für die Implementierung verteilter Systeme unter Nutzung von Standards wie etwa XML, HTML, HTTP und CORBA darstellt.

Insbesondere die Plattformunabhängigkeit in Verbindung mit dem großen Funktionsumfang bestehender Bibliotheken haben zu einer weiten Verbreitung von Java für die Implementierung verteilter Systeme geführt. Der Begriff Java steht dabei im Grunde für drei verschiedene Konzepte:

- die virtuelle Maschine, die eine plattformunabhängige Abstraktion einer Ablaufumgebung für Programme im Java-Bytecode realisiert,
- die Programmiersprache Java, die in den Java-Bytecode übersetzt wird und
- Bibliotheken, die benötigte Funktionalitäten wiederverwendbar implementieren.

Diese Bestandteile existieren in unterschiedlichen Ausprägungen und Konstellationen. Die an dieser Stelle wichtigste Zusammenstellung ist die *Java 2 Platform Enterprise Edition* (J2EE) [173]. Daneben existieren noch die *Standard Edition* für herkömmliche Anwendungen und die *Micro Edition* für die Nutzung von Java auf Kleincomputern. Die Programmiersprache Java ist in Anwendungssystemen in-

terssanterweise nicht zwingend erforderlich, da auch Übersetzer für andere Programmiersprachen existieren, die Java-Bytecode erzeugen können.

In J2EE kommen die weiter oben bereits angesprochenen Mechanismen Applets, Servlets und Java Server Pages zum Einsatz. Dazu werden zwei weitere Techniken eingesetzt: *Java Beans* [168] und *Enterprise Java Beans* [169]. Sie bilden eine Basis für komponentenbasierte Anwendungen, in denen wiederverwendbare Elemente nicht als Klassen oder Klassenhierarchien zur Verfügung gestellt werden, sondern in Komponenten (*Beans*) mit einer definierten Schnittstelle zusammengefaßt werden. Beans können durch diese definierte Schnittstelle in graphischen Entwicklungswerkzeugen direkt ohne Modifikation der Werkzeuge eingesetzt werden.

Die Anwendungsstruktur, die durch J2EE impliziert wird, zeigt Abbildung 30. Sie entspricht einer dreistufigen (*three tier*) Architektur, wie sie auch durch andere Applikationsserver propagiert wird. Ein System wird dementsprechend in die Bereiche Präsentation, Anwendungslogik und Datenhaltung unterteilt, die dann auf die Systemkomponenten verteilt werden.

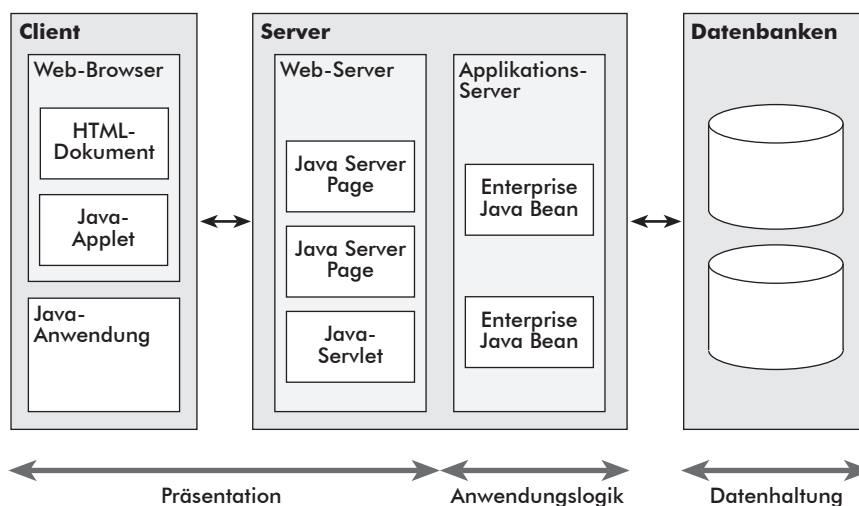


Abb. 30: Systemarchitektur gemäß der Java 2 Enterprise Edition

Die Kommunikation zwischen den Komponenten wird, zwischen Client und Server, mit Javas *Remote Method Invocation* (RMI), HTTP und HTML durchgeführt. Geplant ist auch der Einsatz von XML als offenes Datenaustauschformat zwischen den Komponenten im Server und der Datenhaltung sowie die Öffnung für CORBA-Systeme (siehe unten). Die für die Anbindung von Web-Browsern wichtige Funktionalität für die Realisierung zustandsgesteuerter Abläufe über HTTP wird im Web-Server realisiert, so daß auf der Client-Seite keine Erweiterungen der Anwendungssoftware nötig sind.

Unterstützt wird die Anwendungsentwicklung durch eine Reihe von Diensten innerhalb von J2EE, beispielsweise Namens- und Verzeichnisdienste, Transaktionsdienste, Funktionalität zur Verarbeitung von E-Mail, Verschlüsselungsdienste oder Messaging-Unterstützung.

Die Java-Plattform stellt ein sehr umfangreiches Instrument zur Realisierung verteilter, mehrschichtiger Anwendungen dar. Allerdings ist auch hier zu beachten, daß der Schwerpunkt auf der Entwicklung verteilter Programmsysteme liegt und nicht auf

der Entwicklung einer offenen Informationsarchitektur mit möglichst wenigen Brüchen im Informationsfluß. Dennoch ist die Kombination von Java und XML sehr interessant, da beide Ansätze Plattformunabhängigkeit propagieren und sich gegenseitig ergänzen.

12.2 Infrastrukturebene

Das Zusammenspiel der Systembestandteile (Client und Server) stützt sich auf eine relativ einfache Infrastruktur, welche durch die Protokolle TCP/IP und HTTP gebildet wird. Auch auf dieser Ebene existieren Ansätze, die versuchen, die Infrastruktur zu verbessern oder zu erweitern.

XML-RPC

Eine einfache aber effektive Erweiterung des HTTP-Protokolls ist *XML-RPC* [180]. Wie bereits bei der Betrachtung von HTTP und CGI erwähnt, kann die HTTP-POST-Methode genutzt werden, um auf einem entfernten Rechner Aktionen auszulösen. In XML-RPC wird ein Rahmenwerk definiert, welches die Parameterübergabe bei der HTTP-POST-Methode für die auszuführende Aktion näher spezifiziert. Die Motivation liegt dabei in der weiten Verbreitung von HTTP-fähiger Software, der Vereinfachung des Datenaustauschs und der Schaffung eines einfach zu implementierenden Mechanismus für entfernte Methodenaufrufe.

XML-RPC benutzt eine einfache DTD, um die Parameter für einen Methodenaufruf sowie das Ergebnis zu strukturieren (siehe Abbildung 31).

```
<ELEMENT methodCall (methodName, params?)>
<ELEMENT methodResponse (params | fault)>
<ELEMENT params (param*)>
<ELEMENT fault (value)>
<ELEMENT param (name, value)>
<ELEMENT value (i4 | int | boolean | string |
                double | dateTime.iso8601 | base64 |
                struct | array)>
<ELEMENT struct (member*)>
<ELEMENT member (name, value)>
<ELEMENT array (data)>
<ELEMENT data (value*)>
```

Abb. 31: Document Type Definition für XML-RPC

Ein XML-RPC-Aufruf wird immer über die HTTP-POST-Methode realisiert, die Parameter und die Antwort werden im Rumpf der Nachricht übertragen. Das Ziel der Nachricht (d.h. die übergebene URI) bezeichnet die Instanz, die den Aufruf bearbeiten soll. Die Aufrufsemantik ähnelt so einem Funktionsaufruf in anderen Programmiersprachen, insbesondere die Synchronität der Aufrufe ist von Bedeutung.

XML-RPC stellt einen sehr pragmatischen Mechanismus dar, um Informationsaustausch und Methodenaufrufe über Systemgrenzen hinweg zu realisieren, so daß in unterschiedlichen Programmiersprachen und Systemen bereits XML-RPC-Implementierung existieren, u.a. auch in Zope. Es ist offensichtlich, daß hier ein pragmatischer Ansatz einer vollständigen Problemlösung vorgezogen wurde. Gerade die unterschiedlichen Datentypen der Parameter könnten etwa auch über Metadaten näher spezifiziert werden, die getroffene Auswahl wirkt hier willkürlich.

SOAP

Das *Simple Object Access Protocol* (SOAP) [189] stellt eine Weiterentwicklung von XML-RPC um objektorientierte Techniken dar. SOAP basiert wie XML-RPC auf HTTP, kann aber auch mit anderen Protokollen eingesetzt werden. Wird HTTP genutzt, so wird eine Nachricht per POST-Request an eine verarbeitende Instanz übermittelt. Die Aktion, die die adressierte Instanz durchzuführen hat, wird in einem HTTP-Headerfeld näher spezifiziert. Dies entspricht dem Aufruf einer Methode eines Objekts in objektorientierten Programmiersprachen.

Der Nachrichteninhalt wird in einem *SOAP Envelope* versandt, der als XML-Dokument kodiert ist. In diesem Umschlag befinden sich Meta-Informationen in Form von *SOAP-Header*-Feldern und die eigentlichen Nutzdaten im *SOAP-Body*. Die Nutzdaten müssen gemäß SOAP in XML kodiert werden. Dazu stehen einfache Datentypen, zusammengesetzte Strukturen und Arrays zur Verfügung. Die Kodierung der einfachen Datentypen nutzt vornehmlich die XML-Schema-Spezifikation (insbesondere die Datentypspezifikation [22]).

Eine übermittelte Nachricht kann von der verarbeitenden Instanz mit einer Rückantwort versehen werden. Die Antwort ist in der gleichen Weise strukturiert wie die Nachricht, d.h. es wird wiederum ein Umschlag genutzt, der Meta-Informationen und den eigentlichen Rückgabewert enthält. Mögliche Fehler bei der Ausführung können in einem gesonderten Fehlerelement im SOAP-Body genauer spezifiziert werden.

SOAP bietet gegenüber XML-RPC hauptsächlich exaktere Mechanismen zur Kodierung von Datentypen an. Allerdings hängt SOAP dabei stark von den noch nicht endgültigen XML-Schema-Spezifikationen ab. Der Einsatz von SOAP ist insofern nur unter Vorbehalt zu empfehlen, zumal SOAP selbst noch kein festgelegter Standard ist.

CORBA

Einen wesentlich ambitionierteren Versuch zur Bereitstellung einer Infrastruktur für verteilte Systeme stellt die *Common Object Request Broker Architecture* (CORBA) [150] dar. CORBA basiert auf einer objektorientierten Architektur, in der Anwendungslogik von verteilten Objekten realisiert wird, die über einen gemeinsamen Informationsbus kommunizieren. Der Bus wird durch einen *Object Request Broker* (ORB) realisiert, unterschiedliche Implementierungen von ORBs kommunizieren über das *Internet Inter-ORB-Protocol* (IIOP). Unterstützt werden Anwendungen durch *Common Object Services* (Namensdienste, Transaktionen, Abfragen usw.) und *Common Facilities* (Informations- und Systemmanagement, verteilte Dokumente usw.), dargestellt in Abbildung 32.

Wichtig für die Realisierung verteilter Anwendungen sind die Schnittstellen, über die die Anwendungsobjekte kommunizieren. Dazu stellt jedes Objekt seine Funktionalität über Schnittstellenspezifikationen in der *Interface Definition Language* (IDL) zur Verfügung. Die Schnittstellen sind implementierungsunabhängig, sie werden von einem Übersetzer in Adaptercode für die jeweilige Zielsprache übersetzt.

Anwendungen, die entfernte Objekte nutzen möchten, rufen diese entweder direkt entsprechend der Schnittstellenspezifikationen auf oder erfragen die Schnittstellen beim zugehörigen ORB. Dadurch sind auch dynamische Systeme realisierbar, in denen z.B. Objekte, die keine IDL-Beschreibung besitzen oder deren Interface sich än-

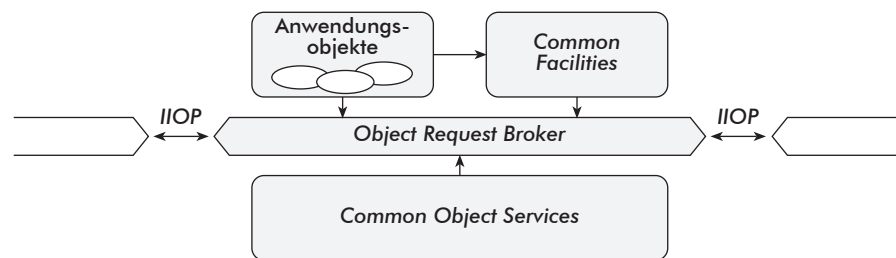


Abb. 32: Architektur eines CORBA-basierten Systems

dert, integrierbar sind. Die Techniken *Dynamic Invocation Interface* (DII) und *Dynamic Skeleton Interface* (DSI) setzen dieses dynamische Verhalten auf Server- bzw. Client-Seite um.

Jede Anwendung, die mit einem CORBA-System kommuniziert, muß einen ORB beinhalten. Dies bedeutet eine erhöhte Komplexität und geringere Flexibilität, da zum einen eine CORBA-Implementierung einen gewissen Umfang mit sich bringt, andererseits viele Implementierungen nicht so modular sind, daß die Common Object Services gegen bereits integrierte Dienste austauschbar sind. Ein weiterer Kritikpunkt ist die starke Bindung von CORBA an Binärformate, die einen Informationsaustausch mit nicht-CORBA-konformen Systemen über offene Textformate erschweren.

Kombination von CORBA und Web-Techniken

Die beiden Welten der CORBA- und Web-basierten Informationssysteme können auf unterschiedlichen Wegen integriert werden, wodurch sich Vorteile insbesondere auf dem Gebiet der Interoperabilität bestehender Systeme und der Verringerung der Komplexität bei einer Client-seitigen CORBA-Anbindung ergeben. Diese Integration ist Bestandteil unterschiedlicher Ansätze. Im einfachsten Fall wird der Zugriff auf ein CORBA-System über ein CGI-Programm im Web-Server hergestellt. Diese Lösung wird in [115] vorgestellt. Dort wird die Skriptspache *CorbaScript* benutzt, um in CGI-Skripten Web-Browsern den Zugriff auf CORBA-Objekte zu ermöglichen.

Ein ähnlicher Ansatz ist in [150] beschrieben, wobei der Zugriff nicht über CGI-Skripte im Web-Server sondern über Proxy-Server erfolgt. Ein Proxy-Server kann auf der einen Seite HTTP-Anfragen verarbeiten und wandelt sie auf der anderen Seite in IIOP-Nachrichten um. Zudem ist noch eine Abbildung von HTTP auf CORBA in Client und Server möglich, so daß Client und Server über CORBA-Protokolle kommunizieren.

Mit WebBroker [179] wird die CORBA-Architektur noch weiter für externe Systeme genutzt. Dazu werden XML-Dokumente an zwei Stellen eingesetzt:

- Die Beschreibung der Objekt-Schnittstellen wird in XML-Dokumenten abgelegt, was den Vorteil hat, bei einem dynamischen Abfragen der Schnittstelle nur einen Zugriff auf die Schnittstellen-Bibliothek zu erfordern. Auch können so unterschiedlichste Systeme, die mit einem XML-Parser ausgestattet sind, die Schnittstelleninformation verarbeiten und gegebenenfalls selbst erzeugen.
- Der Datenaustausch wird wie bei XML-RPC über HTTP unter Einsatz von XML im Rumpf der Nachrichten durchgeführt. Die Daten werden entsprechend einer Typvereinbarung markiert, so daß unterschiedliche Datentypen zur Verfü-

gung stehen. Es wird eine neue Methode mit dem Namen INVOKE eingeführt, die zur Unterscheidung von gewöhnlichen HTTP-Aufrufen dient.

Die konkrete Anbindung an CORBA-Systeme kann dann entweder über eine Erweiterung des ORBs erfolgen, der neben CORBA-Anfragen auch HTTP beherrscht, oder es wird ein Proxy-Server benutzt, der HTTP nach IIOP umsetzt.

Insbesondere durch den WebBroker-Ansatz zeigt sich, daß sich XML und HTTP hervorragend für den Datenaustausch eignen. Beide bieten genügend Möglichkeiten, Informationen zu strukturieren und zu verteilen. Auch sind für unterschiedlichste Datenstrukturen nur jeweils ein XML-Parser zur Verarbeitung und ein HTTP-Client bzw. -Server für den Austausch notwendig.

12.3 Informationsebene

Ein sehr interessanter und wichtiger Bereich wird durch Ansätze abgedeckt, die die eigentliche Information innerhalb des Systems weiter strukturieren und mit zusätzlicher Bedeutung versehen.

Verbindung von Information und Verhalten

Mit *Action Sheets* [7] und *HTML Components* [113] existieren Vorschläge, wie man HTML-Dokumente effektiver mit Verhaltensbeschreibungen wie Programmskripten verbinden kann. Diese Ansätze sind Grundlage für die Erweiterung von CSS [8]. Ihnen liegt das Konzept der HTML-Komponente zugrunde, die mit Elementen in einem HTML-Dokument verbunden werden kann, definierbare Eigenschaften hat und auf bestimmte Ereignisse reagieren kann. Eigenschaften, Ereignisse, Methoden und die Verknüpfung von Ereignissen und Methoden werden in einer HTML-Komponenten über vordefinierte XML-Elemente festgelegt. Eine Verbindung von Komponenten und HTML-Elementen kann über die *behavior*-Eigenschaft in einer CSS-Layoutvorlage ähnlich einer CSS-Layouteigenschaft zugewiesen werden. HTML-Komponenten sind dabei unabhängig von den assoziierten Dokumenten abgelegt.

Die zukünftige Verwendung von HTML-Komponenten in CSS ist nicht genau klar, da sich die Spezifikation der CSS-Erweiterung um Verhaltensbeschreibungen noch in der Diskussionsphase befindet. Es ist jedoch zu bemängeln, daß die Konzepte stark auf HTML aufbauen (etwa für die Implementierung von Komponenten), statt einen allgemeineren Mechanismus wie XML zu nutzen. Zudem ist das Zusammenspiel mit DOM Level 2 nicht geklärt, welches ähnliche Ziele auf der Ebene des programmatischen Zugriffs auf XML-Dokumente über ein eigenes Ereignismodell verfolgt. Zudem wurde in der Spezifikation nicht auf bereits beschriebene Mechanismen wie etwa [99] eingegangen.

Eine ähnliche Verbindung von Information und Verhalten ist auf dem Gebiet von E-Mail-Systemen in [24] oder [70] beschrieben. Allerdings ist die Anwendung dort nur auf versandte Nachrichten beschränkt, d.h. es werden keine allgemein verwendbaren Dokumente mit Verhaltensbeschreibungen verbunden.

Literate Programming

Eine andere Möglichkeit, die Informationsräume für Verhaltensbeschreibung und Nutzdaten zu kombinieren, ist das *Literate Programming* [96]. Literate Programming zielt auf die Kombination der Beschreibung, *was* ein Programm macht, mit der Beschreibung, *wie* es dies erreicht, also eine starke Verbindung von intensi-

ver Dokumentation mit Programmcode. Ein Text, der gemäß Literate Programming verfaßt wurde, kann demnach auf zwei Arten verarbeitet werden. Zum einen kann er als ausführliche Dokumentation darin eingebetteter Programmfragmente betrachtet werden. Zum anderen lassen sich die Programmfragmente zu einem vollständigen Programm zusammenfügen.

Literate-Programming-Quelltexte haben einen Hypertext-ähnlichen Charakter und unterstützen automatisch generierte Indizes, Inhaltsverzeichnisse und Querverweise. Allerdings sind keine Möglichkeiten zur Introspektion gegeben, so daß das beschriebene Programm bei der Ausführung losgelöst vom ursprünglichen Quelltext ist, genau wie der Programmcode bei der Betrachtung der Dokumentationsinformation nicht verarbeitet wird.

Trellis

Eine mögliche formale Definition von Hypertextsystemen mit eingebettetem Verhalten wird mit dem *Trellis*-Modell [65] geliefert. Trellis ist ein Modell, in dem die Knoten eines Hypertext-Systems als Stellen eines Petri-Netzes [151] und die Navigation eines Benutzers in diesem Hypertext als Transitionen aufgefaßt werden. In Trellis werden die Knoteninhalte nicht näher spezifiziert, anders als es etwa HTML- oder XML-Dokumente erlauben. Die Transitionen lösen beim Feuern (d.h. beim Navigieren von einem Knoten zum nächsten Knoten) anwendungsspezifische Aktionen aus.

Die Navigation eines Benutzers resultiert so im Ablauf der mit den Verknüpfungen assoziierten Aktionen, was insgesamt den Ablauf des zugrundeliegenden Anwendungssystems bestimmt.

XML Schema

Die Beschreibung der Struktur von XML-Dokumenten mittels DTDs hat zwei Nachteile: Erstens unterscheidet sich die Syntax einer DTD von der Syntax des restlichen XML-Dokuments, was die Implementierung von Parsern erschwert, und zweitens ist die Ausdrucksmöglichkeit einer DTD beschränkt, so fehlen etwa Mechanismen, um Kardinalitäten im Inhaltsmodell von Elementen zu definieren. Um diese Nachteile auszugleichen, wird am W3C die XML Schema Spezifikation erarbeitet, die aus zwei Teilen besteht: Der Definition der *XML Schema Syntax* [178] und der Definition der *XML Schema Datentypen* [22]. XML Schemata sollen die Ausdrucksfähigkeit von DTDs erreichen und darüber hinaus weitere Möglichkeiten wie die Definition einfacher Datentypen (Datum, Zeit, Zahlen) und komplexer Inhaltsmodelle (Kardinalitäten, Wertebereiche) für Elemente bieten.

Resource Description Framework

Betrachtet man Schemata für Dokumente auf einer abstrakteren Ebene, lassen sie sich als Ausprägungen von Metadaten auffassen, die bestimmte Aussagen über ein Dokument und die darin enthaltenen Informationselemente enthalten. Die Verbindung von solchen Metadaten mit Dokumenten und die Definition von Metadaten-Beschreibungssprachen wird im Zusammenhang mit XML durch das *Ressource Description Framework* (RDF) geleistet. RDF ist in zwei Teilen beschrieben: Die *RDF Model and Syntax Specification* definiert [104], wie Metadaten strukturiert werden, während die *RDF Schema Specification* [34] ein Typsystem für Schemata definiert. Die Model and Syntax Specification ist bereits als Empfehlung verabschie-

det worden und kann in Anwendungen genutzt werden, die Verabschiedung der Schema Specification ist noch nicht erfolgt.

Das Ziel von RDF ist es, Daten nicht nur maschinenlesbar, sondern auch von Maschinen verstehbar zu machen. Anwendungen hierfür sind vielfältig, beispielsweise können so Suchmaschinen effizienter abgefragt, Inhalte von Web-Angeboten je nach Zielgruppe klassifiziert oder Geschäftsprozesse weiter automatisiert werden. RDF eignet sich zur maschinenunterstützten Wissensrepräsentation und –abfrage.

RDF erlaubt es, Ressourcen nach einem einfachen Modell in Aussagen der Form *Subjekt – Prädikat – Objekt* zu beschreiben. Die Beschreibungen werden dabei üblicherweise in XML verfaßt, was zu einer vereinfachten Verarbeitung führt, falls im Anwendungssystem bereits ein XML-Parser zur Verfügung steht.

Die Syntax einer RDF-Beschreibung ist sehr kompakt. Eine RDF-Beschreibung wird mit dem *RDF*-Element eingeleitet. Es enthält ein oder mehrere *Description*-Elemente, welche die zu beschreibende Ressource über das *about*-Attribut referenzieren. Im *Description*-Element werden die Eigenschaften der Ressource festgelegt. Dies geschieht über *Property*-Elemente, welche aus beliebigen Namensräumen stammen können, während *RDF*- und *Description*-Element sowie das *about*-Attribut aus dem *RDF*-Namensraum stammen (siehe Abbildung 33 für ein Beispiel).

```
<rdf:RDF xmlns:rdf=
    "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <rdf:Description about="http://nestroy.../Koeppen/">
    <Author>Eckhart Köppen</Author>
  </rdf:Description>
</rdf:RDF>
```

Abb. 33: Beispiel für eine RDF-Beschreibung

Ein interessanter Aspekt bei der Nutzung von RDF ist die Möglichkeit zur Annotation nicht-modifizierbarer Ressourcen, da eine RDF-Beschreibung sie per *about*-Attribut referenzieren kann, ohne daß die Ressource tatsächlich vorhanden sein bzw. modifiziert werden muß.

Der Einsatz von RDF-Schemata ist im Vergleich zur Nutzung einfacher RDF-Beschreibungen aufwendiger, da die RDF Schema Specification ein Typsystem, ähnlich wie in objektorientierten Sprachen vorhanden, definiert. Dadurch sind zwar die Ausdrucksmöglichkeiten größer, allerdings ist die Erstellung konformer Schemata auch aufwendiger. Auch ist die endgültige Rolle im Vergleich zu anderen Schema-Spezifikationen wie etwa XML-Schema noch nicht vollständig geklärt.

12.4 Zusammenfassung verwandter Bereiche

Eine Analyse der verwandten Bereiche und Arbeiten zeigt, daß in vielen Fällen der eigentlich wichtigste Systembestandteil, nämlich die Information, entweder nur eine kleine Rolle spielt, oder nicht adäquat spezifiziert wird. So stehen im WebObjects-System, bei Zope und bei ActiWeb die zugrundeliegenden Applikationsobjekte im Vordergrund, während die Schnittstellen zu diesen Objekten zumindest bei den beiden erstgenannten Systemen nur in Form von HTML existieren, d.h. die tatsächliche in den Applikationsobjekten vorhandene Information nur modifiziert und unter Verlust zusätzlicher Semantik wiedergeben. So ist es z.B. aus einer HTML-Seite nur unter Aufwand möglich, die Informationen des zugrundeliegenden Zope-Objekts zu

extrahieren. Zumindest ist aber bei diesen Systemen und auch bei anderen, implementierungslastigen Ansätzen wie Thot oder Interleaf eine Nutzung der Information nur bei Vorhandensein einer entsprechenden Umgebung wie etwa dem WebObject-Server oder dem Zope Publisher möglich, da nur diese eine Ablaufumgebung für Applikationsobjekte bereitstellen. Eine Nutzung ohne eine solche Umgebung ist nicht möglich bzw. vorgesehen. Durch das Fehlen wirklich offener (d.h. universell verwendbarer und austauschbarer) Datenformate in solchen Systemen besteht die Gefahr des *data kidnap*, bei dem der weitere Verwendungszweck der Informationen in anderen Kontexten durch ungünstige Strukturierung und Auszeichnung eingeschränkt ist und als Folge zukünftige Anwendungen auf diese Strukturen ausgerichtet werden müssen [89]. Dadurch geht die Flexibilität neuer Anwendungen verloren. Daneben kann aus einer Bindung an ein proprietäres Datenformat auch die Bindung an daran anknüpfende Abläufe wie etwa Geschäfts- oder Workflow-Prozesse resultieren.

Ein solches data kidnapping ist auch zu beobachten, wenn noch nicht verabschiedete Standards wie etwa XSL bereits zu Implementierungszwecken eingesetzt werden, und durch eine weite Verbreitung die Nutzung des endgültigen Standards nicht attraktiv erscheint. Ein solches Vorgehen erscheint insbesondere dann gefährlich, wenn Standards wie im Fall des W3C durch mehrere Kooperationspartner entwickelt werden und gerade in frühen Stadien des Standardisierungsprozesses noch kein Konsens zwischen den beteiligten Instanzen gefunden ist. Eng verwandt mit diesem Vorgehen ist die Taktik des *embrace and extend* [149], bei dem durch Erweiterung eines bestehenden Standards um proprietäre Bestandteile einer genereller Einsatz verhindert wird. Dieses Vorgehen war z.B. in den frühen Entwicklungsphasen von HTML zu beobachten, bei dem der Standard durch Software-Hersteller nach Belieben um eigene Elemente ergänzt wurde.

Viele Nachteile der o.g. Ansätze sind sicherlich auf das Fehlen geeigneter Beschreibungsmittel zurückzuführen, da z.B. HTML als Basis nicht ausdrucksstark genug ist. Mit der Spezifikation von XML ist hier allerdings eine neue Grundlage gegeben. Dabei ist aber zu beachten, daß einige der oben aufgeführten Ansätze durch die Verfügbarkeit von XML als neue Basis an Relevanz verlieren. So ist der Einsatz von HTML Components in einem XML-Umfeld nicht sinnvoll, da XML bereits genügend Möglichkeiten bietet, Informationsräume wie Nutzdaten und Verhaltensbeschreibungen zu mischen.

Eine adäquate Betrachtung und Spezifikation der im System verwendeten Information könnte auch die Probleme der Implementierungs- und Plattformunabhängigkeit und, daraus folgend, mangelnde Flexibilität und Offenheit beseitigen. Diese Probleme treten sicherlich bei den Ansätzen auf, die auf der Implementierungsebene Web-basierter Informationssysteme ansetzen, d.h. solche Lösungen, die vornehmlich an eine konkrete Implementierung gebunden sind und kein offenes Informationsmodell als Grundlage haben.

Liegt Information in einem offenen Format wie XML vor, können darauf unterschiedliche Anwendungen aufbauen. Gerade mit den Möglichkeiten von Namensräumen lassen sich auch Informationen kombinieren, was die Interoperabilität erhöht. Ein erster Schritt in diese Richtung ist der bei einigen Ansätzen wie etwa der Java-Plattform oder XML-RPC erkennbare Weg, XML zumindest als Datenaustauschformat einzusetzen. Eine in diesem Zusammenhang wichtige Entwicklung ist sicherlich der Bereich der Metadaten-Spezifikation, da die Bedeutung der auszutau-

schenden Daten nicht nur implizit aus den Implementierungen, sondern auch explizit über maschinenlesbare Beschreibungen ermittelbar sein muß.

13 Entwurf

Der Entwurf eines Architekturmodells für Web-basierte Informationssysteme erfolgt unter Berücksichtigung der aufgestellten Anforderungen und der gegebenen Basistechniken. Ausgangspunkt ist das grundlegende Architekturmodell für Client-Server-Systeme, welches in Abbildung 3 auf Seite 9 zu sehen ist.

13.1 Design-Entscheidungen

Beim Entwurf des Modelles sind eine Reihe von Design-Entscheidungen zu treffen, welche im folgenden erläutert werden.

Information als integraler Bestandteil

Die Kernanforderungen wurden in Kapitel 11 identifiziert, als wichtigste Punkte sind die Informationsstrukturierung, -vernetzung und Kommunikation zu nennen. Aus der Bedeutung dieser Punkte läßt sich ableiten, daß die Information ein integraler Bestandteil Web-basierter Informationssysteme ist. Auch im hier entwickelten Modell soll die Information die Basis bilden, genauer gesagt werden die Systembestandteile (Datenhaltung, Ablauflogik, Präsentation) durch strukturierte Dokumente gebildet. Neben dem Erfolg Web-basierter Informationssysteme, welche grundsätzlich auch stark informationszentrisch aufgebaut sein können (u.a. durch die einfache Verwendbarkeit von HTML), spielt auch der wachsende Bedarf nach einfachem Informationsaustausch eine große Rolle, beispielsweise in Anwendungen wie der elektronischen Geschäftsabwicklung. Dies ist auch eine Folgerung aus der Betrachtung der verwandten Bereiche. So sollen die dort teilweise vorhandenen Brüche im Informationsfluß vermieden werden. Viele weitere Vorteile einer derartigen Systemarchitektur sind im übrigen in [93] aufgeführt, wo die Bedeutung der Information und deren möglichst robuste Spezifikation herausgehoben wird. Dort wird u.a. argumentiert, daß Information und darin benutzte Vokabulare der Schlüssel für Kollaboration zwischen Unternehmen ist. Dokumente bestimmen so z.B. durch ihren Aufbau und Inhalt den Ablauf von Geschäftsprozessen und stellen dadurch gewissermaßen eine Schnittstelle zu Geschäftsabläufen dar. Sie dienen zur Kapselung und Übertragung der Zustände der Prozesse. Eine besondere Bedeutung kommt dabei der richtigen Strukturierung der Information zu, um eine Verwendung in möglichst vielen Bereichen zu erlauben und das oben angesprochene data kidnapping zu vermeiden.

Auflösung der Trennung zwischen Client und Server

Durch die starke Hervorhebung der Bedeutung der Information tritt die Unterscheidung in Server- und Client-basierte Systembestandteile in den Hintergrund. Im vorgestellten Modell sollen Client und Server gleiche Fähigkeiten besitzen.

Die Auflösung der Trennung zwischen Client und Server besteht jedoch nur konzeptionell, d.h. durch das Modell ist es möglich, Systeme mit und ohne eine Unterteilung in Client und Server zu beschreiben. Gleichwohl kann es aus Implementierungsgründen sinnvoll sein, bestimmte Eigenschaften beider Seiten beizubehalten, um z.B.

durch eine Spezialisierung auf der Serverseite erhöhte Leistungsfähigkeit und auf der Clientseite erhöhte Benutzerfreundlichkeit zu erhalten.

Verbindung von Zustand und Verhalten

Um die Unterteilung zwischen Client und Server aufzulösen, wird die Ablauflogik für die Anwendungen in die zu verteilenden Dokumente integriert. Dadurch wird die nötige Funktionalität sowohl im Client als auch im Server durch die Dokumente geliefert.

Durch diese Architektur wird zugleich die Verbindung von Zustandsbeschreibung und Verhaltensbeschreibung festgelegt. Im Gegensatz zu herkömmlichen verteilten Systemen, bei denen die Programmlogik in den einzelnen Knoten (Rechnern) des Systems implementiert ist und nur die Daten wandern, sind hier Verhalten und Zustand in einer Instanz (dem Dokument) vereinigt. Dadurch soll eine größere Autonomie der Systembestandteile erreicht werden.

Gerade bei vernetzten, kooperativen Systemen besteht zudem die Problematik oft in der Abgrenzung der durch solche Systeme realisierten Anwendungen. Ein Applikationsserver auf ActiWeb- oder J2EE-Basis kann so viele Anwendungen realisieren, bei denen auch Daten ausgetauscht werden. Eine klare Trennung zwischen Daten, die nur zwischen vernetzten Subsystemen ausgetauscht werden, und solchen Daten, die innerhalb von Applikationsobjekten gehalten werden, ist allerdings oft nicht sinnvoll vorzunehmen. Aus diesem Grund erscheint die Auflösung dieser Trennung und die durchgängige Verwendung von Informationen sowohl zwischen als auch innerhalb von verteilten Komponenten von Vorteil.

13.2 Aktive Hypertextdokumente

Den oben genannten Design-Entscheidungen folgend ist eine erste, vorläufige Definition möglich. Die Dokumente, die die Zustandsbeschreibungen, Verhaltensbeschreibungen und die Information transportieren, werden in der Folge als *aktive Hypertextdokumente* (AHD) bezeichnet. Das Modell, welches hierfür entwickelt wird, soll demzufolge als Modell für aktive Hypertextdokumente oder kurz AHDM bezeichnet werden. Aktive Hypertextdokumente haben u.a. folgende Charakteristika:

Struktur: Ein AHD ist ein XML-Dokument, welches mittels bestimmter Auszeichnungselemente strukturiert ist.

Information: Ein AHD transportiert in bestimmter Art strukturierte Information. Die Information ist dabei entweder wesentlich für das eigentliche Dokument oder fällt in die drei nachfolgenden und durch die bei der folgenden Beschreibung des AHDM näher definierten Kategorien:

Zustand: In engem Zusammenhang mit der Struktureigenschaft von AHDs steht die Zuordnung eines bestimmten Zustands zu einem AHD. Die explizite Beschreibung eines Zustands geschieht unter Verwendung bestimmter designierter Strukturelemente, deren Inhalt den Zustand eines AHDs näher festlegt.

Verhalten: Ähnlich wie der Zustand eines AHDs ist auch das Verhalten mittels vorgegebener Strukturelemente definiert. Sie beinhalten Beschreibungen möglichen Verhaltens beispielsweise in Form von Programmskripten. Das Verhalten ist dabei in einem AHD nötigenfalls von der vorgenannten Zustandsinformation

abhängig. Es kann aber auch von der allgemeinen im Dokument enthaltenen Information beeinflusst werden.

Präsentation: Ein AHD kann optional auch Angaben über die Präsentation der enthaltenen Information enthalten. Dies ist aber nur dann notwendig, wenn das AHD von Endbenutzern genutzt wird.

Vernetzung: Da ein aktives Hypertextdokument ein XML-Dokument ist, stehen in solchen Dokumenten die Vernetzungsmöglichkeiten XML-basierter Standards wie XLink zur Verfügung. Dementsprechend kann ein AHD logisch aus mehreren vernetzten physikalischen Komponenten bestehen.

Kommunikation und Mobilität: Aktive Hypertextdokumente sind nicht ortsgebunden, d.h. sie sind im Internet verteilt und mobil.

Die genauen Eigenschaften eines AHDs werden im nächsten Unterkapitel definiert und diskutiert. In Anlehnung an die Architektur von Client-Server-Systemen wie sie in Abbildung 2 zu sehen ist, zeigt Abbildung 34 eine ähnliche, durch das AHDM implizierte Systemstruktur.

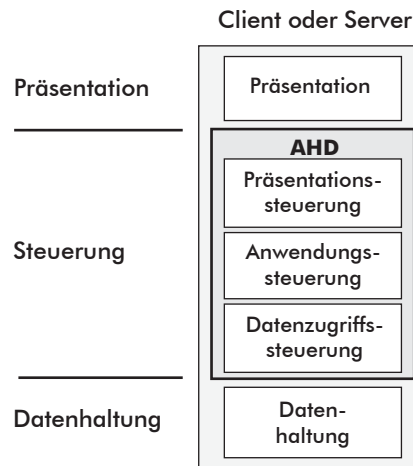


Abb. 34: Modifikation der Client-Server-Architektur im AHDM

Dies ist jedoch nur eine vereinfachte Sicht, da hier Verhalten, Zustand, Vernetzung und Kommunikation nicht berücksichtigt sind. Statt dessen liegt der Schwerpunkt auf funktionalen Aspekten in der Steuerungsschicht von herkömmlichen Client-Server-Systemen. Unter Einbeziehung der besonderen Eigenschaften aktiver Hypertextdokumente ist eine Architektur, wie sie in Abbildung 35 vereinfacht dargestellt ist, genauer. Gerade die Vernetzungs- und Mobilitätsaspekte stellen eine Erweiterung typischer Client-/Server-Systeme dar.

Zu beachten ist dabei, daß die klassische Einteilung in Präsentationsschicht, Steuerungsschicht und Datenhaltungsschicht aufgelöst wurde, da die Dokumente selbst Teile dieser Funktionalität abdecken. Stattdessen existiert in jedem Netzwerknoten eine Laufzeitumgebung für aktive Hypertextdokumente. Die Laufzeitumgebung unterstützt zum einen die Präsentation, die Ablaufsteuerung und die Datenhaltung, zum anderen liefert sie die für das AHDM zusätzlich benötigte Funktionalität. Dieser Bereich wird im Unterkapitel 13.6 vertieft.

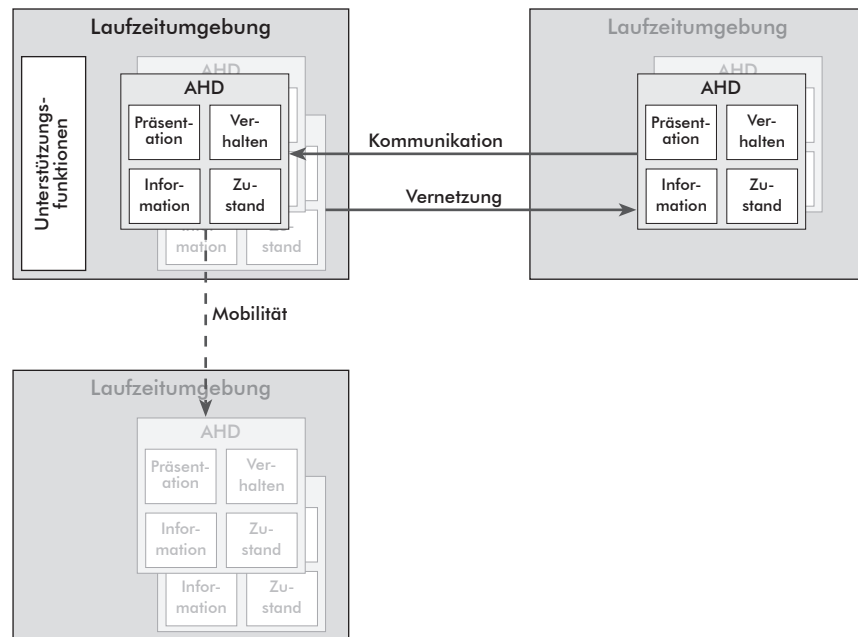


Abb. 35: Vereinfachtes allgemeines AHD-Architekturmodell

13.3 Allgemeine Eigenschaften aktiver Hypertextdokumente

Zu den allgemeinen Eigenschaften zählen die durch die grundlegenden Anforderungen an Web-basierte Informationssysteme notwendig gewordenen Charakteristika. Sie werden im folgenden definiert.

Struktur

Ein aktives Hypertextdokument gemäß dem AHDM ist ein XML-Dokument. Die Wahl für die Strukturierung der Information in einem aktiven Hypertextdokument fiel auf XML, weil dadurch in erster Linie ein Mechanismus zur Verfügung steht, der *general markup*, d.h. die Auszeichnung und Strukturierung von Daten unabhängig vom Einsatzzweck, ermöglicht. Gerade bei Daten, die nicht nur in einer einzelnen, stationären Anwendung eingesetzt werden, sondern Bestandteil verteilter Systeme sind und zudem sowohl innerhalb von Subsystemen als auch zum Informationsaustausch zwischen Komponenten verwendet werden, ist diese Anwendungsunabhängigkeit ein vordringliches Ziel. Zudem soll neben dem Einsatz in unterschiedlichen bestehenden Systemen auch der Einsatz in zukünftigen Systemen ermöglicht werden. Die Verwendung von SGML als Alternative scheidet aufgrund der hohen Komplexität von SGML und der daraus resultierenden Anforderungen an die Verarbeitungswerkzeuge aus. In Zusammenhang mit der Verknüpfung unterschiedlicher Anwendungen erscheint der Einsatz von XML als universelles und offenes Datenaustauschformat sinnvoll, eine Eigenschaft, die gerade im heterogenen Internet gefordert ist.

Um AHDs im Kontext von Web-basierten Informationssystemen von anderen Dokumentarten zu unterscheiden, sollen sie über den MIME-Typen [25] „text/x-ahd“ gekennzeichnet werden. Es besteht demnach logisch aus hierarchisch angeordneten Elementen und physikalisch aus Entities. Die Struktur eines AHDs wird durch die Elemente gebildet, sie kann mit einer DTD, einem Schema oder anderen Mechanismen definiert werden. Eine Strukturbeschreibung ist aber nicht zwingend vorgesehen, d.h. ein AHD ist im Sinne von XML nur wohlgeformt, aber nicht unbedingt valide.

Um bestimmte strukturelle Eigenschaften von aktiven Hypertextdokumenten im Verlauf dieser Arbeit genauer spezifizieren zu können, soll hier eine einfache formale Definition eines strukturierten XML-Dokuments beschrieben werden. Zwar existiert mit dem Entwurf zum XML Information Set [46] eine Beschreibung der Datentypen, wie sie in XML-Dokumenten auftritt, es fehlt allerdings eine formale Definition der Zusammenhänge.

Ein XML-Dokument ist, vereinfacht gesagt, ein *Wurzelbaum* D beschrieben durch:

- eine Menge von Knoten V , welche die Elemente repräsentieren,
- die Menge der in XML-Dokumenten erlaubten Zeichen Σ ,
- die Funktion $p: V \rightarrow V$ zur Zuordnung von Elementen zu ihren Vorgängerelementen,
- die Funktion $attr: V \times \Sigma^+ \rightarrow \Sigma^*$ zur Zuordnung von Attributwerten zu Attributnamen und Knoten und
- die Funktion $gi: V \rightarrow \Sigma^+$ zur Zuordnung von Elementen zu Elementnamen.

Diese Betrachtungsweise vereinfacht zwei Sachverhalte: einerseits wird die allgemeine Beschreibung eines Graphen (beschrieben etwa in [132]) so modifiziert, daß nur die relevanten Bestandteile der Beschreibung enthalten sind, andererseits wird insbesondere die Reihenfolgebeziehung zwischen den direkten Nachfolgern eines Elements nicht weiter beschrieben. Eine Einbeziehung der Reihenfolge ist im Verlauf der Arbeit für die Formulierung von Algorithmen und Strukturbeziehungen nicht notwendig, ließe sich aber durch Hinzufügen von beispielsweise neuen Kantentypen oder der Definition einer Algebra für XML-Dokumente erreichen, ähnlich wie in [75] für strukturierte Dokumente definiert.

Zur vereinfachten Beschreibung an späterer Stelle werden noch zwei Mengen definiert:

- Die Nachfolgermenge $S(v)$ ist die Menge aller direkten und indirekten Nachfolger des Knotens v , also alle Knoten, die von v aus erreicht werden können.
- Die Vorgängermenge $P(v)$ umfaßt alle Knoten, von denen der Knoten v aus erreicht werden kann.

Referenzierung

Ein AHD ist eindeutig referenzierbar. Diese Referenz kann auch als Speicherort begriffen werden. Sie wird durch eine *URI* [18] gebildet, wobei typischerweise eine URL benutzt wird. Durch die Referenz ist ein AHD von anderen Instanzen aus zugreifbar, sie kann aber nicht als Unterscheidungsmerkmal oder eindeutigen Identifizierer für AHDs verwendet werden, da eine Referenz auf dasselbe AHD wechseln kann.

Strukturzugriff

Ergänzend zur Referenzierung ist die Definition des Strukturzugriffs nötig, da die Referenzierung nur ein gesamtes Dokument betrifft. Der Strukturzugriff erlaubt es, einzelne Elemente innerhalb eines AHDs gezielt zu adressieren. Der Zugriff wird im AHD-Modell über einen XPointer oder die Parametrisierung des letzten Segments in URL-Pfadangaben realisiert. Diese Parametrisierung wird durch Anhängen des Werts des Namensattributs an die URL des Dokuments durchgeführt. So wird ein Element, dessen Namensattribut den Wert *print* hat und welches sich in einem Dokument mit dem Namen *order.ahd* befindet, über *order.ahd;print* adressiert. Dies ist vergleichbar mit dem Einsatz eines Fragment-Identifizierers zur Adressierung einzelner Anchor-Elemente in HTML. Allerdings ist die Parametrisierung der URL im Gegensatz zur Nutzung von Fragment-Identifizierern fester Bestandteil einer URL. Eine andere Schreibweise für diese Parameterisierung stellt der XPath-Ausdruck *//*[attribute::name=value]* dar, wobei *value* für den gesuchten Wert des Namensattributs steht. Das oben genannte Beispiel ließe sich unter Nutzung einer XPointer-URL auch als *order.ahd#xpointer(//*[attribute::name=print])* schreiben. Befinden sich mehrere Elemente mit dem gleichen Wert für das Namensattribut in einem XML-Dokument, so soll im AHDM davon ausgegangen werden, daß das erste in einer Tiefensuche gefundene Element adressiert wird. Sind solche Mehrdeutigkeiten auszuschließen, so sind explizite XPath-Ausdrücke zu verwenden.

Zusätzlich aber soll ein weiterer, einfacher Mechanismus für den Strukturzugriff eingesetzt werden, welcher mit XML-Abfragesprachen verwandt ist und im folgenden kurz erläutert wird. Der Mechanismus soll hier mit dem Namen *Struktur-Matching* bezeichnet werden. Die zugrundeliegende Idee ist die, daß eine bestehende Struktur auf strukturelle Übereinstimmung mit einer weiteren Musterstruktur untersucht wird. Um den Mechanismus einfach zu halten, wird als einzige strukturelle Eigenschaft die Vorgänger-/Nachfolgerbeziehung geprüft und auf die Betrachtung anderer Beziehungen wie z.B. die Reihenfolge von untergeordneten Elementen verzichtet. Als einfache Regel gilt, daß eine gegebene Elementhierarchie mit einer Musterhierarchie gemäß dem Struktur-Matching übereinstimmt, wenn in der gegebenen Hierarchie die gleichen Vorgänger-/Nachfolger-Beziehungen vorhanden sind wie in der Musterhierarchie. Für das oberste Musterelement muß gelten, daß im gegebenen Elementbaum ein passendes Element existiert, welches den gleichen Element-Namen trägt und daß auch für alle direkten Unterelemente des obersten Musterelements eine Entsprechung unter den Unterelementen des passenden Elements aus dem gegebenen Elementbaum existiert. Diese Bedingung muß für alle weiteren untergeordneten Musterelemente zutreffen.

Formal beschrieben lautet die Bedingung für die strukturelle Ähnlichkeit eines Musterdokuments D_1 und einem Dokument D_2 mit den Knotenmengen V_1 und V_2 wie folgt:

$$\begin{aligned} \forall v_1, v_1' (v_1, v_1' \in V_1 \wedge p(v_1') = v_1 \rightarrow \\ \exists v_2, v_2' (v_2, v_2' \in V_2 \wedge \\ v_2' \in S(v_2) \wedge \\ gi(v_1) = gi(v_2) \wedge \\ gi(v_1') = gi(v_2')))) \end{aligned}$$

Abbildung 36 zeigt ein Beispiel für eine solche Übereinstimmung, bei dem die vorgegebene Hierarchie mit der Musterhierarchie auf der rechten Seite übereinstimmt.

<pre> <message> <printformat> <style> HighQuality </style> <defaultprinter> LaserWriter8 </defaultprinter> </printformat> <voiceformat> ... </voiceformat> ... </message> </pre>	<pre> <message> <printformat> <defaultprinter> </defaultprinter> </printformat> </message> </pre>
---	--

Abb. 36: Beispiel für zwei strukturell übereinstimmende Hierarchien

Vernetzung

Durch die Definition von Struktur und Referenzierung ist die Markierung von vernetzten Elementen innerhalb eines AHDs möglich. Hierzu wird XLink benutzt. Elemente, die mittels XLink als zu vernetzende Elemente gekennzeichnet werden, referenzieren über eine URI externe Dokumente oder Teile hiervon. Sie sollen durch ihre größere Ausdrucksfähigkeit der Vernetzung mittels Entities vorgezogen werden, da XLinks es beispielsweise ermöglichen, die Art der Verbindung hinsichtlich der Verarbeitung des Zielelements, der Rolle der Verbindung oder der Aktivierung der Verbindung genauer zu spezifizieren.

Kommunikation und Mobilität

Der Hauptmechanismus zur Kommunikation von Inhalten und zur Übertragung aktiver Hypertextdokumente soll HTTP sein, da eine entsprechende Infrastruktur (bestehend aus Web-Servern und Web-Browsern) weitverbreitet existiert und gerade auf Hypertextdokumente zugeschnitten ist. Allerdings sind auch andere Distributionsmechanismen denkbar, beispielsweise die Übertragung eines AHDs über die Mail-Protokolle SMTP [143] und POP [121] oder über LDAP [182]. Diese und andere Übertragungsmechanismen sollen als Alternativen zu HTTP ebenso verwendbar sein. Die Distribution der Information ist eng verwandt mit dem weiter unten beschriebenen Kommunikationsmodell. Wichtig ist an dieser Stelle, daß die Inhalte, die im AHDM ausgetauscht werden, einerseits herkömmliche Informationsstrukturen wie XML-Dokumente oder Texte sein können, zum anderen aber auch aktive Hypertextdokumente selbst mobil sein können.

13.4 Informationsmodell

Durch die Forderung nach der Verbindung von Zustands-, Verhaltens-, Präsentationsbeschreibung und Nutzdaten wird im AHDM ein Informationsmodell für aktive

Hypertextdokumente definiert. Es regelt die Kombination der folgenden Informationsarten:

Information

Ein AHD transportiert im Normalfall immer Information, die nicht unmittelbar mit dem aktiven Charakter des Dokuments zusammenhängt. Diese Information macht die eigentlichen Nutzdaten eines AHDs aus.

Zustand

Als erstes, nur für das AHDM designiertes Konstrukt wird die Zustandsbeschreibung innerhalb eines AHDs eingeführt. Die Zustandsinformation ist insofern von der allgemeinen Information innerhalb eines AHDs zu unterscheiden, als daß sie nur dazu dient, das Verhalten eines zu AHDs beeinflussen. Wird die Zustandsinformation entfernt, so muß andererseits der eigentliche Informationsgehalt des AHDs erhalten bleiben.

Die Realisierung der Zustandsbeschreibung kann wie auf Seite 14 (*Attribute versus Elemente*) diskutiert, entweder über die Definition eines eigenständigen Elements oder über die Definition eines Attributs erfolgen. Konkret stellt sich die Frage, in welcher Form ein Element als Zustandsbeschreibung gekennzeichnet werden soll. Hier ergeben sich drei Möglichkeiten:

Zustand im Attribut: Spielt die Validität des AHDs und ihre Überprüfbarkeit eine große Rolle, so haben Attribute den Vorteil, daß sie jedem vorhandenen Element durch eine ergänzende Attributlisten-Deklaration einfach hinzugefügt werden können. Wird aber die Zustandsinformation innerhalb des Attributwertes gehalten, so ergeben sich durch den erlaubten Wertebereich und die fehlende Möglichkeit, einem Element mehrere gleichnamige Attribute zuzuordnen, Nachteile.

Zustand im Element (durch Attribut markiert): Wird die Zustandsinformation nicht im Attributwert sondern als Inhalt eines Elements abgelegt, so kann die Kennzeichnung des Elements als Zustandselement über ein gesondertes Attribut erfolgen (vergleichbar mit der Kennzeichnung eines Links bei XLink). Ähnlich wie bei der vorigen Alternative kann dieses Attribut für bestehende Elemente deklariert werden. Der Vorteil dieser Lösung ist eine mögliche Mehrfachnutzung des Element-Inhalts, zum einen als Zustandsbeschreibung, zum anderen als Nutzdatum.

Der Nachteil dieser und der vorigen Lösung besteht aber darin, daß Attribute nicht zum Inhalt eines XML-Dokumentes gehören (der Inhaltsbegriff ist hier weiter zu fassen als bei der Trennung von allgemeiner Information und Zustandsinformation). Im AHDM ist aber die Markierung der Zustandsinformation und die Zustandsinformation selbst Bestandteil des Inhalts.

Zustand im eigenständigen Element: Als letzte Möglichkeit bietet sich an, für die Zustandsinformation einen eigenen Elementtyp zu definieren. Der Nachteil hiervon ist aber, daß das Content Model der die Zustandselemente beinhalten den Elemente jeweils geändert werden müßte, wozu eine Modifikation der zugrundliegenden DTD nötig ist. Deshalb kommt dieser Mechanismus nur dann in Frage, wenn die Validität eines AHDs nachrangig ist. Eine Lösung für dieses Problem stellen allerdings Namensräume dar, welche explizit für die Kombination verschiedener Strukturbeschreibungen geeignet sind.

Beispiele für alle drei Möglichkeiten zeigt Abbildung 37, der Präfix *ahd* bezeichnet dabei Elemente und Attribute aus dem AHD-Namensraum.

Zustand im Attribut:

```
<title ahd:sect_header="Introduction">...</title>
```

Zustand im Element durch Attribut markiert:

```
<title ahd:name="sect_header">Introduction</title>
```

Zustand im eigenständigen Element:

```
<ahd:var name="sect_header">Introduction</ahd:var>
```

Abb. 37: Möglichkeiten für das Einbinden von AHD-Zustandsinformation

Im AHDM soll die letzte Möglichkeit zur Realisierung von Zustandsbeschreibungen genutzt werden. Hierfür gibt es hauptsächlich drei Gründe: die Zustandsbeschreibungen sind fester Bestandteil des Inhalts eines AHDs, die Validität eines AHDs ist keine vordringliche Anforderung, und schließlich läßt sich die Validität über Namensräume wiederherstellen.

Das Element, welches im AHDM Zustandsinformation aufnimmt, ist das *AHD-Variablen-Element*, kurz *var-Element*. Abbildung 38 zeigt die XML-Deklaration.

```
<!ELEMENT var (#PCDATA)>
<!ATTLIST var
    name          CDATA          #REQUIRED>
```

Abb. 38: Deklaration des var-Elements

Das Content-Model des var-Elements läßt beliebige PCDATA zu, aber keine anderen Elemente. Das var-Element besitzt ein obligatorisches Attribut mit dem Namen *name*, welches zur Identifikation dient. Die Deklaration des name-Attributes als ID wäre grundsätzlich empfehlenswert, um beispielsweise Referenzen auf ein solchermäßen gekennzeichnetes Element per IDREF zu bilden. Allerdings ist in einem AHD nicht festgelegt, daß die Werte der name-Attribute aller AHD-Elemente unterschiedlich sind, es kann durchaus Elemente mit gleichem name-Wert geben, die aber durch die unterschiedliche hierarchische Position im Dokument zu unterscheiden sind.

Verhalten

Ähnlich wie der Zustand eines AHDs ist auch das Verhalten mittels vorgegebener Strukturelemente definiert. Sie beinhalten Beschreibungen möglichen Verhaltens, beispielsweise in Form von Programmskripten. Die Entscheidung, in welcher Form die Verhaltensbeschreibung in einem AHD erfolgt, ist hier eindeutig zugunsten eines eigenständigen Elements zu fällen. Einerseits gehört auch die Verhaltensbeschreibung zum Inhalt eines AHDs (weshalb die Verwendung eines Attributs zur Kennzeichnung ausscheidet), andererseits ist die Unterscheidung zwischen Nutzdaten und Verhaltensbeschreibung klarer als bei der Zustandsbeschreibung.

Die Beschreibung von Verhalten erfolgt mittels des *AHD-Funktions-Elements*, kurz *func-Element*. Die Deklaration ist aus Abbildung 39 ersichtlich.

```
<!ELEMENT func (#PCDATA) >
<!ATTLIST func
    name          CDATA          #REQUIRED
    type          CDATA          #REQUIRED
    returns       CDATA          "text/plain">
```

Abb. 39: Deklaration des func-Element

Wie das *var-Element* hat auch das *func-Element* ein obligatorisches *name*-Attribut. Daneben ist ein Attribut mit dem Namen *type* deklariert. Es beinhaltet den MIME-Typen der im Element benutzten Programmiersprache und ist, wie das *Names-Attribut*, obligatorisch. Als drittes Attribut kann schließlich der MIME-Typ der von der Funktion zurückgegebenen Daten angegeben werden. Vorgegeben ist hier „text/plain“.

Präsentation

Die Nutzdaten (und gegebenenfalls auch die Zustandsdaten) in einem AHD sind bei der Verwendung durch Endbenutzer in geeigneter Form zu präsentieren. Dazu gehört u.a. die Bildschirmdarstellung, die Druckausgabe oder die Sprachausgabe. Im AHDM werden zu diesem Zweck *Cascading Stylesheets* (CSS) verwendet. Sie erlauben eine einfache Zuordnung von Inhalt zu Repräsentation. Grundsätzlich wären dafür auch andere Techniken wie DSSSL oder XSL nutzbar, allerdings werden dadurch vermehrt aktive Aspekte während der Informationsaufbereitung abgedeckt. Solche Aufgaben sollen aber im AHDM durch die dort eingeführten Mittel (Zustands- und Verhaltensbeschreibungen) bearbeitet werden.

Nutzung von Namensräumen

Um eine möglichst weitreichende Separation der oben aufgeführten Informationselemente und damit eine möglichst hohe Kombinierbarkeit zu erreichen, sollen im AHDM Namensräume nach dem XML-Namespaces-Standard [32] benutzt werden.

Namensräume im AHDM werden an zwei Stellen benutzt: einerseits, um die oben angeführten Strukturelemente zu kennzeichnen, andererseits um die angesprochenen Mittel zur Verhaltensbeschreibung abzugrenzen. Es wird eine Verwendung, wie sie Abbildung 40 in einem einfachen Beispiel zeigt, angestrebt.

Der Prefix „ahd“ soll auch in weiteren Beispielen durchgängig verwendet werden, obgleich er auch anders lauten kann.

Die Trennung der unterschiedlichen Bestandteile des Beispiel-AHDs kann mit Namensräumen einfach durchgeführt werden. So läßt sich die Nutzinformation durch Entfernen aller Elemente mit dem Prefix „ahd“ gewinnen, wie Abbildung 41 zeigt.

Mit dieser Separierung ist eine einfache Form der Validierung möglich. Die einzelnen separierten Informationsmengen sind jeweils durch einen gemeinsamen Prefix gekennzeichnet. Diesem Prefix ist durch die Namensraum-Deklaration eine Schema-Definition zugeordnet, anhand derer sie validiert werden können. In Abbildung 41 kann so das Content-Model für das Element „order“ unverändert bleiben, da die AHD-Elemente bei der separierten Validierung ignoriert werden.

```

<order
  xmlns:ahd = "http://www.w3.org/NS/ahdm.dtd"
  xmlns      = "http://www.e-commerce.org/order.dtd">

  <ahd:func  name = "new_entry" type = "text/tcl">
    ...
  </ahd:func>
  <ahd:func  name = "calculate_total" type = "text/tcl">
    ...
  </ahd:func>
  <ahd:var   name = "transaction_number"> ... </ahd:var>

  <recipient>
    <name> ... </name>
    <address> ... </address>
  </recipient>
  ...
</order>

```

Abb. 40: Beispielhafte Verwendung von XML-Namensräumen

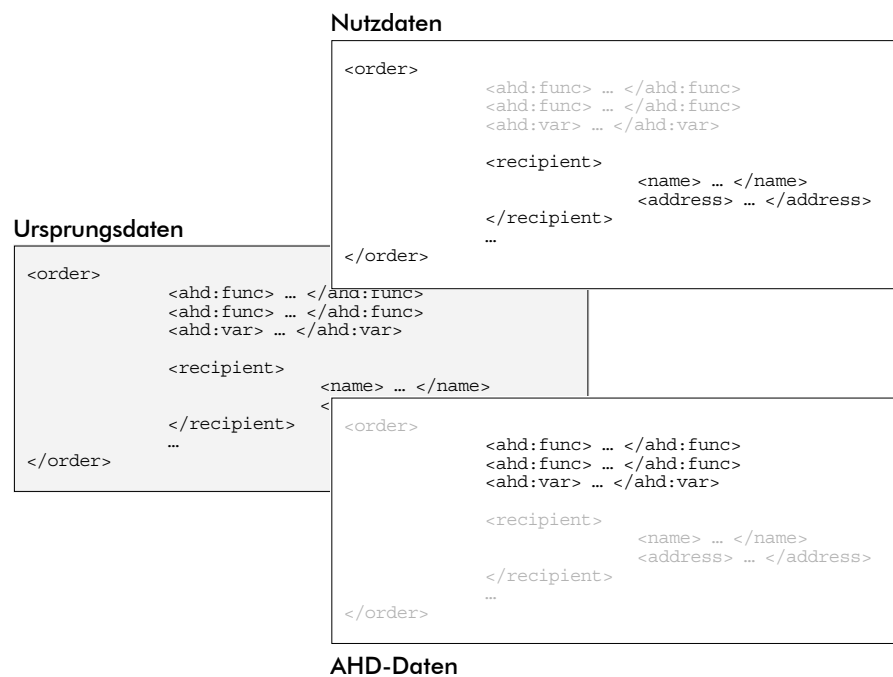


Abb. 41: Separierung von AHD-Daten und Nutzdaten

Diese einfache Art der Validierung soll im Folgenden als *unscharfe* Validierung bezeichnet werden, da sie eine vollständige Validierung des Dokumentes nicht ermöglicht.

13.5 Objektmodell

Durch das Informationsmodell können aktive Hypertextdokumente designierte Elemente zur Beschreibung des Verhaltens enthalten. Diese Verhaltensbeschreibung erfolgt im Regelfall durch Programmteile von Skript-Sprachen wie beispielsweise Tcl [135], XOTcl [195], Lua [83], Guile [63] oder Perl [184]. Die in diesem Zusammenhang wichtigen Aspekte werden nachfolgend behandelt.

Die einzelnen Elemente zur Verhaltensbeschreibung (und auch zur Zustandsbeschreibung) sind dabei nicht unabhängig voneinander. Durch die Formulierung von Sichtbarkeitsregeln soll eine objektähnliche Struktur erzielt werden. Die Sichtbarkeitsregeln definieren Sichtbarkeitsbereiche, die zusammengehörige Einheiten wie etwa Objekte umfassen, vergleichbar mit den Scoping-Regeln von Programmiersprachen. Die func- und var-Elemente sind vergleichbar mit Instanzvariablen und -methoden ihres Vatelements. Dieses kann somit als Objekt betrachtet werden. Aus einer Methode heraus sind alle objekt-lokalen Methoden und Variablen eines Objekts direkt ansprechbar, ähnlich sind aus einem func-Element alle lokalen func- und var-Elemente (also diejenigen Elemente mit dem gleichen Vatelement) direkt ansprechbar. Die Referenzierung der Variablen und Methoden über Sichtbarkeitsregeln ist insbesondere beim Aufruf der Methoden oder beim Modifizieren der Variablen aus anderen aktiven Bestandteilen eines AHDs wichtig, die Referenzierung wird dabei durch die Methoden der in Unterkapitel 13.6 beschriebenen Laufzeitumgebung durchgeführt.

Bezogen auf das Beispiel aus Abbildung 40 bedeutet dies, daß dem Element *order* die Funktionen *new_entry* und *calculate_total* sowie die Variable *transaction_number* zugeordnet sind und einen Sichtbarkeitsbereich bilden.

Die Zustands- und Verhaltenselemente sind in die allgemeine Struktur eines AHDs eingebettet, d.h. sie können auf jeder Hierarchie-Ebene des Dokuments auftreten, sofern die Content Models der Elemente im Dokument dies zulassen, das Dokument nicht validiert wird oder Namensräume verwendet werden. Durch das Informationsmodell und die Nutzung von Namensräumen wird eine flexible Zuordnung der einzelnen Elementtypen zueinander erreicht, so daß sich ein dynamisches Objektmodell ergibt. Dadurch kann das Verhalten eines aktiven Hypertextdokuments durch strukturelle Änderungen modifiziert werden (siehe dazu auch Unterkapitel 13.8).

Im AHDM werden nun folgende Sichtbarkeitsregeln *direkter Zugriff*, *Parent Delegation* und *Remote Delegation* definiert.

Direkter Zugriff

Ausgehend vom Wurzel-Element eines Dokuments sind alle untergeordneten Elemente über XPath-Mechanismen sichtbar. Dazu zählt insbesondere der Zugriff über das name-Attribut und entsprechende parametrisierte URLs bzw. die Nutzung von XPointer-Ausdrücken.

Parent Delegation

Innerhalb eines Verhaltenselements sind alle func- und var-Elemente, welche sich auf der gleichen Hierarchie-Ebene befinden, direkt über das name-Attribut ansprechbar, d.h. es existiert eine relative Adressierung im Gegensatz zur absoluten Adressierung von der Wurzel des Dokuments aus. Weiterhin sind auch alle umschließenden Elemente ebenso ansprechbar, vom direkt umschließenden Element bis hin zum

Wurzel-Element. Dazu kommen noch die direkten Kinder dieser Elemente, sofern es sich um func- oder var-Elemente handelt.

Die parent delegation bewirkt die Fortsetzung der Semantik der Zuordnung von Funktionen und Variablen zu direkt übergeordneten Elementen auf die Kette der Väterelemente. Wird demnach ein func- oder var-Element nicht im aktuellen Element gefunden, so wird bei der weitergehenden Suche das Väterelement zum aktuellen Element. Wichtig ist an dieser Stelle aber, daß diese Ähnlichkeit zu Objekten nicht integraler Bestandteil des AHDM ist, da hier vornehmlich Wert auf lose Kopplung zwischen den Informationsräumen und damit auch zwischen Nutzdaten und AHD-Elementen gelegt wird. Im Vergleich dazu ist die Beziehung zwischen Objekten, Instanzfunktionen und -Variablen in den meisten statisch typisierten objektorientierten Programmiersprachen dauerhafter ausgelegt, etwa über die Nutzung von Klassen.

Die in Frage kommenden Elemente für die parent delegation ausgehend von einem Element v bezüglich eines gesuchten Elementtyps t (z.B. func oder var) und des Werts des name-Attributs n sind über folgende Definition definiert:

$$\begin{aligned} scope(v, t, n) = \{v' \in V \mid \\ p(v') \in P(v) \wedge \\ gi(v') = t \wedge \\ attr(v', name) = n\} \end{aligned}$$

In dieser Menge wird das Element ausgewählt, welches die geringste Distanz zum Ausgangselement innerhalb des Wurzelbaums hat. Existieren mehrere Elemente mit gleicher Distanz so wird das Element ausgewählt, welches bei einem Links-Rechts-Tiefendurchlauf des Baumes als erstes gefunden wird. In Abbildung 42 ist ein Beispiel für die sichtbaren Elemente ausgehend von einem Ursprungselement dargestellt. Unter der Annahme, daß von Element v ausgehend Funktions- oder Variablenelemente referenziert werden, wird entlang des Pfades bis zum Wurzelement r nach direkt untergeordneten Elementen mit den gewünschten Eigenschaften (gegebenes Attribut/Wert-Paar und Elementname) gesucht. Im gezeigten Beispiel liegen von v aus alle mit v' bezeichneten Elemente im gleichen Sichtbarkeitsbereich und können über parent delegation direkt angesprochen werden.

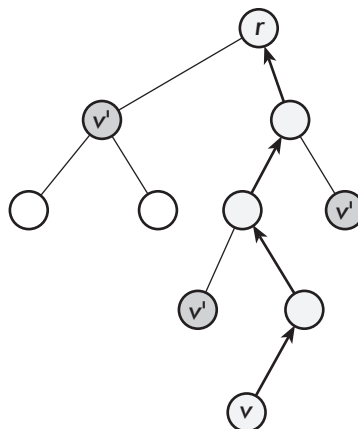


Abb. 42: Beispiel für sichtbare Elemente bei der parent delegation

Remote Delegation

Wird bei der Auflösung einer Referenz ein Element nicht gefunden, so ist als dritte Möglichkeit noch die Referenz von Elementen in externen AHDs möglich. Dazu wird bei der parent delegation beim Erreichen des Wurzelements in einem externen Dokument weitergesucht. Die Referenz wird als delegate-Element bezeichnet und gemäß Abbildung 43 deklariert. Das *href*-Attribut enthält die Referenz auf das Dokument, bei dem die Suche nach dem angesprochenen Element fortgeführt wird.

```
<!ELEMENT delegate EMPTY>
<!ATTLIST delegate
    href          CDATA          #REQUIRED>
```

Abb. 43: Delegate-Element

13.6 Laufzeitumgebung

Die Aufgaben, welche sich aus der Architektur von Client-Server-Systemen ergeben, sollen durch die Definition einer *Laufzeitumgebung* abgedeckt werden. Die Laufzeitumgebung stellt die Funktionen zur Verfügung, die für den Zugriff und zur Manipulation von aktiven Hypertextdokumenten sowie zur Ausführung der aktiven Bestandteile benötigt werden. Neben diesen für die Implementierung der aktiven Komponenten eines AHDs wichtigen Funktionen realisiert die Laufzeitumgebung auch die Kommunikation zwischen Netzknoten und die Persistenz aktiver Dokumente. Dabei kann sie sich auf bereits vorhandene Komponenten eines eventuell vorhandenen Wirtsprogrammes wie einen Web-Server oder Web-Browser stützen, d.h. die Laufzeitumgebung wird im Regelfall in ein umgebendes Programm eingebettet und implementiert hauptsächlich die für das AHDM relevanten Funktionalitäten. Ein Beispiel ist die Erweiterung eines Web-Servers um AHDM-Funktionen mittels eines Servlets wie es in Kapitel 27 beschrieben ist.

Modifikation aktiver Hypertextdokumente

Die Verhaltensbeschreibungen mittels Programmskripten setzt sinnvollerweise voraus, daß Mittel existieren, um die in einem AHD vorhandene, strukturierte Information zugreifbar und modifizierbar zu machen. Mit dem *Document Object Model* (DOM) ist ein solcher Mechanismus gegeben. Es beschreibt, in welcher Form auf die Elemente und Attribute in einem XML-Dokument zugegriffen werden kann. Wichtig für das AHDM ist die Eigenschaft, daß dadurch ein strukturierter Zugriff über unterschiedliche Datentypen (z.B. Elemente, Attribute, Listen) möglich ist, im Gegensatz zum unstrukturierten, rein textbasierten Zugriff, bei dem nicht zwischen den Bestandteilen eines XML-Dokuments unterschieden wird. Ein weiterer Vorteil ist die Programmiersprachenunabhängigkeit des DOM, so daß in einem AHD aus mehreren Programmiersprachen heraus auf die Struktur zugegriffen werden kann (vorausgesetzt, die DOM-Schnittstelle ist in der Sprache auch implementiert). Die erste Aufgabe der Laufzeitumgebung ist somit die Bereitstellung der DOM-Funktionalität für die Skript-Bestandteile aktiver Hypertextdokumente.

Zugriff auf Zustandsinformation

Obwohl die Modifikation und der Zugriff über das DOM weitreichende Möglichkeiten bieten, sind dadurch die oben angesprochenen Sichtbarkeitsregeln für AHD-Elemente noch nicht fest im AHDM verankert. Deswegen werden Funktionen definiert, welche neben der DOM-Schnittstelle für Zugriffe auf AHD-Elemente gemäß der Sichtbarkeitsregeln genutzt werden sollen. Diese sind *get* und *put*. Die Funktionen sind ähnlich wie die DOM-Funktionen programmiersprachenunabhängig, d.h. sie werden von der Laufzeitumgebung in verschiedenen Programmiersprachen zur Verfügung gestellt.

Die Funktion *get* liefert den Wert eines var-Elements zurück. Der Wert ist eine Zeichenkette, die alle Zeichen zwischen Start- und Ende-Tag des Variablen-Elements enthält. Befinden sich innerhalb des Variablen-Elements weitere Elemente (beispielsweise aus anderen Namensräumen), so werden diese ignoriert. Das var-Element wird über das name-Attribut referenziert, wobei die Sichtbarkeitsregeln der parent und der remote delegation gelten. Die Referenz kann neben dem Wert des name-Attributs auch ein XPath-Ausdruck sein, um einen direkten Zugriff zu erlauben. Im Vorgriff auf die Beschreibung des Kommunikationsmodells, welches die Basis für die Methoden der Laufzeitumgebung definiert und auf HTTP aufsetzt, soll hier bereits auf eine weitere Möglichkeit eingegangen werden, die die get-Methode bietet. Da die Referenzierung abgesehen von der verkürzten Schreibweise mittels parametrisierter URLs oder der Angabe des Namensattributs grundsätzlich über einen XPath-Ausdruck unter Verwendung eines XPointers und einer URL erfolgt, können somit auch größere Dokumentbestandteile oder auch ganze Dokumente adressiert werden. Die get-Funktion liefert bei der Adressierung dann eine textuelle Repräsentation des Dokuments oder des Dokumentfragments zurück.

Die *put*-Funktion setzt den Wert einer AHD-Variablen, welcher wie bei der get-Funktion eine Zeichenkette ist. Analog zur get-Methode wird auch das Variablen-Element bei der put-Methode über das Namens-Attribut oder einen XPath referenziert, um direkten Zugriff oder Zugriff über die Delegationsmechanismen zu erlauben. Wird bei der put-Methode als Ziel ein Dokument oder ein Dokumentfragment angegeben, so wird das Ziel durch den übergebenen Wert ersetzt.

Die Funktionen *get* und *put* sind technisch gesehen teilweise durch DOM-Funktionen realisierbar. Dennoch ist eine explizite Bereitstellung der Funktionalität für den Zugriff auf Elementinhalte und deren Modifikation über beide Funktionen aus zwei Gründen sinnvoll. Erstens werden durch die Funktionen die Delegationsmechanismen umgesetzt, so daß ein uniformes Verhalten bei der Nutzung der Funktionen in unterschiedlichen Kontexten gegeben ist. Die Delegationsmechanismen müßten ohne das Vorhandensein der get- und put-Funktion mittels DOM-Funktionen nachgebildet werden. Im AHDM hingegen stehen diese Funktionen als abstraktere Konzepte bereits zur Verfügung. Der zweite Punkt betrifft die Kommunikationsebene: Durch die beiden Funktionen wird die Kommunikation zwischen Dokumenten und einzelner Dokumentbestandteile definiert, was durch DOM-Funktionen alleine nicht erreicht werden kann. Im DOM fehlt die dazu notwendige Netzwerkfunktionalität vollständig.

Ausführung der Verhaltensbeschreibungen

Im Zusammenhang mit dem AHDM ergibt sich als wichtigste Funktion der Laufzeitumgebung die Bereitstellung eines Mechanismus zur Ausführung der Verhaltensbeschreibungen in einem AHD. Im Regelfall enthält die Laufzeitumgebung für diese Zwecke einen oder mehrere Interpreter für die nutzbaren Programmiersprachen. Die Interpreter müssen um die Schnittstellen zum DOM und zu den AHD-Funktionen ergänzt werden. Die in diesem Zusammenhang auftretenden Fragen nach der System-sicherheit sind zwar nicht Kernbestandteil des AHDM, sollen aber in der abschließenden Betrachtung im Ausblick weiter untersucht werden.

Die Verhaltensbeschreibungen in einem AHD können in den Skript-Bestandteilen eines solchen Dokuments über die *call*-Funktion aktiviert werden, die durch die Laufzeitumgebung implementiert ist. Auch hier erfolgt die Referenzierung über das Namens-Attribut oder einen XPath-Ausdruck und über direkten oder delegationsgesteuerten Zugriff. Parameter werden immer benannt übergeben, im Unterschied zu positionalen Parametern bei vielen Programmiersprachen. Bei den Parameterwerten und dem Rückgabewert handelt es sich nur um einfache Zeichenketten und nicht um komplexe Objekte, d.h. das AHDM stützt sich hier auf eine *call-by-value*-Semantik.

Ähnlich wie die Realisierung der get- und put-Methode können bei der call-Funktion neben dem Zugriff auf dokumentlokale Funktionen auch Funktionen in entfernten Dokumenten aufgerufen werden, so daß hier von einem *Remote Procedure Call* (RPC) gesprochen werden kann. Dazu werden URLs mit XPointer-Fragment-Identifizierern oder parametrisierte URLs verwendet. Solche Funktionsaufrufe sind blockierend, d.h. die Ausführung wird erst dann fortgesetzt, wenn die entfernte Funktion beendet wurde.

Zur Integration aktiver Hypertextdokumente in bestehende Systeme ist es aber auch möglich, Funktionen anonym aufzurufen, d.h. Funktionen nicht über das name-Attribut anzusprechen, sondern einen Funktionsaufruf an das gesamte Dokument zu richten. In diesem Fall werden alle Funktionen mit dem Namen *onpost* aktiviert, eine Erweiterung des Ereignismodells, wie es im nächsten Kapitel erläutert wird. Im Gegenzug kann über die call-Methode auch ein ganzes Dokument adressiert werden, ohne daß explizit eine aufzurufende Funktion angegeben wird. Hierbei handelt es sich um das Äquivalent eines CGI-Aufrufs. Das AHDM bietet somit die Möglichkeit, Kommunikation einerseits auf Dokumentbasis und andererseits auf Elementbasis zu nutzen. Der genaue Ablauf der Kommunikation ist im Kommunikationsmodell (siehe dazu Unterkapitel 13.7) geregelt.

Im Detail werden an jeden aufgerufenen Programmteil die in Tabelle 3 aufgeführten Parameter übergeben, hier in der Notation und mit den Datentypen, wie sie auch das DOM zur Interface-Beschreibung verwendet. Wichtig ist insbesondere das über *ahd_ahd* bereitgestellte Objekt, welches einen Zugriff auf die AHD-Laufzeitumgebung erlaubt und dessen exportierte Methoden (so z.B. die put- und get-Methode) in Abbildung 44 definiert sind.

Parameter	Beschreibung
Element ahd_caller	Aufrufendes Skriptelement bzw. das Skriptelement, welches den Methodenaufruf verursacht hat
Element ahd_element	Element, welches den Programmcode des auszuführenden Skripts enthält

Parameter	Beschreibung
Document ahd_document	Dokument, in welchem sich das aufgerufene Element befindet
DOMString ahd_params	Parameter in URL-Form, welche vom aufrufenden Skript übergeben wurden
int ahd_reason	Grund für den Funktionsaufruf (Event-Code gemäß dem IEM)
AHDRuntime ahd_ahd	Objekt, welches die Laufzeitumgebung implementiert und alle AHDM-Funktionen bereitstellt

Tabelle 3: Parameter beim AHD-Funktionsaufruf

Der Rückgabewert eines Programmteils ist in der Variable *ahd_result* abzulegen, welche vom Typ DOMString ist. Das Voranstellen des Präfixes „ahd_“ vor die Parameternamen soll eine Überschneidung mit vorhandenen Funktions- oder Objektnamen verhindern, ähnlich der Funktionalität von Namensräumen.

Nebenläufigkeit

Im AHDM wird volle Nebenläufigkeit bei mehreren externen Aktivierungen eines func-Elements eingesetzt. Dies ist bei der Umsetzung von Algorithmen flexibler, es treten dadurch aber auch verstärkt Synchronisationsprobleme auf. Die Synchronisation erfolgt durch zusätzliche Mechanismen: ein aktives Hypertextdokument kann für gleichzeitigen Zugriff gesperrt werden. Dazu dienen die Funktionen *lock* und *unlock*, welche exklusives Sperren eines Dokuments erlauben. Ein Aufruf der Funktion *lock* sperrt ein Dokument. Ist ein Dokument bereits gesperrt, so blockiert der Funktionsaufruf solange, bis die bestehende Sperre durch *unlock* aufgehoben wurde. Kritische Bereiche in Programmen sind dementsprechend zwischen ein *lock*- und *unlock*-Funktionsaufruf zu stellen.

Struktur-Matching

Das Struktur-Matching soll über die Funktion *match* innerhalb der Laufzeitumgebung zur Verfügung stehen. Die Funktion *match* vergleicht eine gegebene Elementhierarchie mit einer Musterhierarchie. Das Ergebnis ist die Liste aller übereinstimmenden Knoten aus der gegebenen Hierarchie.

Persistenz

Die Daten und Dokumente, welche in einer Laufzeitumgebung gehalten werden und teilweise auch zur Ausführung kommen, können entweder durch das Anfordern externer Ressourcen oder durch das aktivieren lokaler AHDs in diese Laufzeitumgebung gelangen. Für die Aktivierung lokaler Dokumente muß die Laufzeitumgebung den Zugriff auf ein Dateisystem oder ähnliche Mechanismen zur Datenhaltung besitzen. Auch die lokale Speicherung (etwa bei der *put*-Funktion) ist ein solcher Mechanismus notwendig. Die Laufzeitumgebung unterstützt damit die persistente Speicherung aktiver Dokumente. Wichtige unterstützende Aufgaben erbringen hierbei die Funktionen *fromText* und *toText*, die die Serialisierung und Deserialisierung aktiver Dokumente von und zu einer XML-Repräsentation erlauben. Die genaue Vorgehensweise bei der Serialisierung wird in Unterkapitel 13.8 beschrieben.

Kommunikation

Ähnlich wie in Client-Server-Systemen muß die Laufzeitumgebung auch die Übertragung von Informationen ermöglichen. Durch den Wegfall der Trennung von Client und Server muß im AHDM jeder Netzknoten die gleiche Funktionalität bieten, d.h. fähig sein, Daten sowohl zu empfangen als auch zu versenden. Zusätzlich soll die Möglichkeit der Datenübertragung auch aus einem AHD heraus nutzbar sein, was durch die AHD-Funktionen (get, put und call) gewährleistet wird. Insgesamt gesehen resultiert aus der angestrebten Peer-to-Peer-Netzwerkarchitektur eine Kopplung von Web-Client und Web-Server in einem einzigen Netzknoten, im AHDM durch die Laufzeitumgebung definiert. Es ist zwar auch möglich, auf die Komponente, die eingehende HTTP-Requests empfängt, zu verzichten, damit entfällt allerdings die Möglichkeit, Dokumente in einem solchen Netzknoten von außen anzusprechen.

Im Detail wird die Datenübertragung mit dem Kommunikationsmodell, welches im nächsten Unterkapitel definiert wird, beschrieben.

Präsentation

Da die Präsentation eines AHDs über CSS gesteuert wird, muß die Laufzeitumgebung in der Lage sein, CSS-Beschreibungen auszuwerten und die Elemente in einem AHD gemäß dieser Beschreibungen anzuzeigen. Ein weiterer Bereich ist die Benutzerschnittstelle bei Netzknoten, die von Endbenutzern genutzt werden sollen, also das Erzeugen von IEM-Ereignissen und die Umsetzung von Eingabeelementen (Textfelder, Schaltflächen, Auswahllisten usw.)

Schnittstelle

Die Funktionalität der Laufzeitumgebung wird, wie das Document Object Model, über eine Schnittstellenbeschreibung mittels der IDL definiert. Dabei handelt es sich um die Funktionalität, welche innerhalb eines AHDs genutzt werden kann, d.h. die Laufzeitumgebung exportiert die gezeigten Funktionen zur Verwendung in den aktiven Bestandteilen eines AHDs. Sie stehen damit genauso wie die DOM-Funktionen in Programmskripten zur Verfügung.

Innerhalb der Beschreibung in Abbildung 44 wird die Schnittstelle unter Verwendung der in der DOM-Spezifikation definierten Schnittstellen beschrieben. Die dort gezeigte Schnittstelle enthält vier Gruppen von Funktionen, welche jede Implementierung einer AHD-Laufzeitumgebung zur Verfügung stellt. Die Kernfunktionen umfassen den Zugriff auf AHD-Variablen und Ausführung von Funktionselementen. Dieser Gruppe werden noch die Hilfsfunktionen *here* und *encode* hinzugefügt. Die Funktion *encode* realisiert die URL-Kodierung von Zeichenketten, die Funktion *here* liefert die URL der aktiven Laufzeitumgebung zurück. Die nächste Gruppe beinhaltet die *match*-Funktion, gefolgt von den Funktionen zur Unterstützung von Persistenz und der Gruppe der Funktionen zur Steuerung nebenläufiger Programme.

Der Aufruf der Methoden der Laufzeitumgebung erfolgt innerhalb der Programmskripte eines AHDs über das übergebene Objekt mit dem Namen *ahd_ahd*, wie es in Tabelle 3 bereits erwähnt wurde. Die Typen der Parameter und der Rückgabewerte der dargestellten Funktionen wie etwa Document, DOMString oder NodeList sind dem DOM entnommen.

```

interface AHDRuntime {

```

Kernfunktionen	void	put (in Element in DOMString in DOMString	current, varName, varValue);
	DOMString	get (in Element in DOMString	current, varName);
	DOMString	call (in Element in DOMString in DOMString	current, funcName, paramString);
	DOMString	encode (in DOMString	text);
	DOMString	here ());
Nebenläufigkeit	void	lock (in Document	document);
	void	unlock (in Document	document);
Persistenz	Document	fromText (in DOMString	text);
	DOMString	toText (in Document	document);
Strukturmatching	NodeList	match (in Element in Element	source, pattern);

```

};

```

Abb. 44: IDL-Interface der Laufzeitumgebung

Der erste Parameter der Funktionen get, put und call steht für das Element, von dem die Suche nach dem jeweils referenzierten Element begonnen werden soll, d.h. der Parameter bezeichnet den Ausgangspunkt für die parent und remote delegation. Im Regelfall wird hierbei das Element übergeben, welches das gerade aktive Programmfragment enthält, d.h. das im Parameter ahd_element übergebene Element wie es auch aus Abbildung 45 ersichtlich ist. Als zweiter Parameter wird bei allen drei Funktionen der Wert des Namensattributes des angesprochenen Elements übergeben. Bei den Funktionen put und call wird als dritter Parameter entweder per codierte Parameterstring für den AHD-Funktionsaufruf oder der neue Wert der referenzierten AHD-Variable übergeben.

Ein Beispiel für einen Aufruf der get-Methode aus einem func-Element in den Programmiersprachen JavaScript und Tcl zeigt Abbildung 45. Die konkrete Implementierung wird im Abschnitt „Werkzeuge“ vertieft.

Die Nutzung der AHDM-Funktionen außerhalb der Programmskripte eines AHDs soll ebenso über die hier beschriebene Schnittstelle erfolgen. Allein die innerhalb dieser Programmskripte zur Verfügung stehenden Objekte wie etwa ahd_ahd oder ahd_current sind außerhalb der Skripte nicht unbedingt unter diesen Namen vorhanden.

Die Funktionalität zur Präsentation, zur Umsetzung der DOM-Funktionen und zum Empfangen von eingehenden Requests ist nicht Bestandteil des oben definierten Interfaces. Die DOM-Funktionen werden über die im DOM-Standard definierten Interfaces wie Document oder Node bereitgestellt.

AHD-Methodenaufruf in JavaScript:

```
<func name="get_total" type="text/javascript"
      returns="text/plain"><![CDATA[
      ahd_result = ahd_ahd.get (ahd_current, "total");
    ]]></func>
```

AHD-Methodenaufruf in Tcl:

```
<func name="get_total" type="text/tcl"
      returns="text/plain"><![CDATA[
      set ahd_result [
        AHDRuntimeGet $ahd_ahd $ahd_current "total"
      ]
    ]]></func>
```

Abb. 45: Beispiele für Aufrufe von AHDM-Funktionen

13.7 Kommunikationsmodell

Die Kommunikationsflüsse im AHDM sollen gesondert betrachtet werden, da sich durch die Wahl der zugrunde liegenden Techniken und Strukturen einige wichtige Implikationen ergeben. Insbesondere HTTP spielt hier eine besondere Rolle, da es die Basis der Kommunikation im AHDM bildet, einerseits auf der technischen Ebene zwischen Netzknoten, andererseits auf der semantischen Ebene durch die Beschreibung des Kommunikationsverhaltens bei Funktionsaufrufen und Variablenzugriffen von AHDM-Elementen.

Kommunikationspartner

Grundsätzlich können an den Kommunikationsprozessen im AHDM drei Gruppen beteiligt sein. Zum ersten die aktiven Dokumente selbst, weiterhin die Laufzeitumgebung, die eine verbindende Rolle zwischen den aktiven Dokumenten und der Systemumgebung einnimmt und schließlich als letzte Gruppe die sonstigen Ressourcen wie beispielsweise andere Web-Server. Als externe Gruppe existiert zusätzlich noch die Gruppe der Benutzer, die über Anwendungsprogramme und die darin eingebettete AHD-Laufzeitumgebung mit den aktiven Dokumenten interagieren.

Bei der Betrachtung der Rolle der aktiven Dokumente ist zwischen der Referenzierung eines Dokuments als Ganzes oder der Referenzierung nur von Teilen eines Dokuments zu unterscheiden. Wichtige Teile sind unter anderem die Elemente aus dem AHDM-Namensraum, also Funktionen und Variablen. Als Folge ergibt sich nicht nur Kommunikation zwischen aktiven Dokumenten, sondern auch innerhalb von Dokumenten. In Abbildung 46 sind die möglichen Kommunikationswege skizziert, wobei mit Ausnahme der Interaktion des Benutzers jeglicher Datenaustausch über HTTP bzw. mit HTTP-Semantik realisiert wird. Diese Kommunikationswege sollen im folgenden näher untersucht werden.

Kommunikationsinhalte

Durch die Wahl von XML und HTTP als Grundlagentechniken, welche vornehmlich einen textbasierten Informationsaustausch implizieren, soll die Unabhängigkeit von

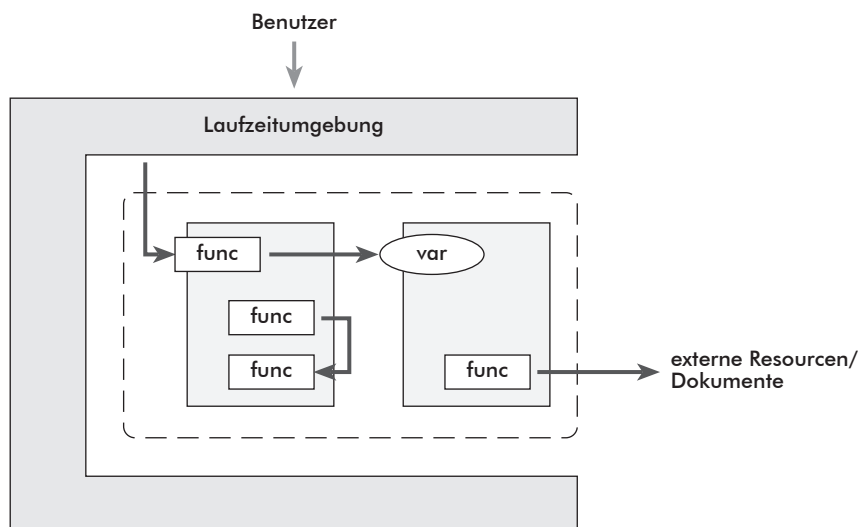


Abb. 46: Kommunikationspfade im AHDM

Implementierungen oder Anwendungsgebieten unterstützt werden. Für die Inhalte der ausgetauschten Information im AHDM bedeutet dies, daß auch hier vornehmlich eine textuelle Repräsentation bevorzugt wird. Das schließt ausdrücklich auch XML- oder AHD-Fragmente als Werte für Funktionsparameter ein. Dennoch wird kein besonderes Format für die Kommunikation zwischen aktiven Dokumenten propagiert, wie es beispielsweise bei SOAP oder XML-RPC der Fall ist. Dadurch soll die Interoperabilität mit nicht-AHD-fähigen Systemen erhöht werden.

Werden nicht nur Dokumentfragmente oder einzelne Zeichenketten sondern vollständige AHDs übertragen, so ist zu beachten, daß daraus keine eigentliche Mobilität aktiver Hypertextdokumente resultiert. Vielmehr werden in solchen Fällen nur Kopien eines AHDs in Form einer textuellen Repräsentation übermittelt. Die weiteren Auswirkungen werden in Unterkapitel 13.8 genauer dargestellt.

Bei den Inhalten, welche über die in Abbildung 46 dargestellten Wege übermittelt werden, sind grundsätzlich keine Einschränkungen gegeben. Allerdings ist zu beachten, daß Daten wie Pointer nur in einer Ausführungsumgebung genutzt werden können und nicht über die Grenzen eines func-Elements hinaus weitergereicht werden dürfen. Es handelt sich hierbei insbesondere um alle Variablenwerte, welche innerhalb der Programmteile eines aktiven Dokuments genutzt werden, etwa lokale Variable, oder auch die Objekte, welche von den DOM-Funktionen zurückgeliefert werden. Sie können zum einen programmiersprachenabhängig sein, zumindest sind sie aber nur sinnvoll innerhalb des Adressraums des aktuellen Prozesses nutzbar, in diesem Fall der Laufzeitumgebung. Da die Kommunikation allerdings implizit auch die Grenzen der Laufzeitumgebung überschreiten kann (z.B. bei der remote delegation), soll im AHDM grundsätzlich darauf verzichtet werden, solche implementierungsabhängigen Daten auszutauschen. Dies stimmt im übrigen mit den Regeln für die persistente Speicherung von aktiven Dokumenten überein, welche bei der Vorstellung AHD-spezifischer Semantiken in Unterkapitel 13.8 erläutert werden. Für die Referenzierung von Dokumentbestandteilen heißt dies, daß die Adressierung über eindeutige Strukturmerkmale, Attribute (wie etwa das id- oder name-Attribut) oder Elementnamen zu erfolgen hat.

Da bestimmte Daten aber unverzichtbar sind für die Reaktion auf bestimmte Ereignisse, werden diese durch die Laufzeitumgebung der aufgerufenen Funktion in geeigneter Weise zur Verfügung gestellt, wie in Abbildung 47 dargestellt ist. Die Laufzeitumgebung ist im AHDM bei jeder Aktivierung einer aktiven Komponente innerhalb eines Dokuments als vermittelnde Instanz beteiligt.

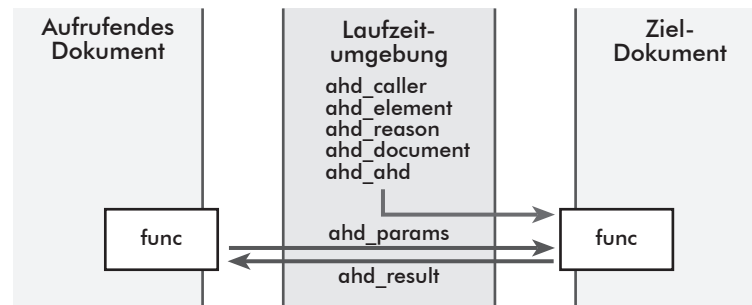


Abb. 47: Rolle der Laufzeitumgebung beim Funktionsaufruf

Ereignisse

Kommunikation wird in vielen Fällen durch die Interaktion des Benutzers mit dem System initiiert. Die Interaktion wird mittels externer Ereignisse beschrieben, die von der Laufzeitumgebung generiert und an ein AHD herangetragen werden. Als Standard für solche Ereignisse bietet sich das *Intrinsic Event Model* (IEM) an. Dadurch ist definiert, welche externen Ereignisse welche Aktionen in einem AHD auslösen. Alternative Möglichkeiten böten z.B. Action Sheets (in [7] beschrieben).

Eine Erweiterung des IEM durch das AHDM besteht darin, daß nicht nur Funktionen, welche durch Attribute an Elemente gebunden sind, aktiviert werden können, sondern auch AHD-Funktionen, die einem Element zugeordnet sind. So kann ein onclick-Ereignis, welches von einem Element empfangen wird, entweder durch das im onclick-Attribut angegebene Skript oder durch eine dem Element zugeordnete AHD-Funktion mit dem Namen onclick behandelt werden.

Das IEM ist für die Behandlung von benutzerinitiierten Ereignissen gut geeignet, trotzdem soll es im AHDM um ein weiteres Ereignis ergänzt werden. Wie später bei der Erläuterung des Kommunikationsmodells noch weiter vertieft wird, sind auch solche Ereignisse interessant, die nicht allein durch den Benutzer, sondern auch durch die Laufzeitumgebung oder andere aktive Dokumente verursacht werden. An dieser Stelle wird deshalb das *onmatch*-Ereignis hinzugefügt, welches beim strukturellen Matching zweier Elementhierarchien ausgelöst wird. Die Behandlung erfolgt dabei in den Elementen der Musterhierarchie, welche die entsprechenden Ereignisse entgegennehmen. Als Wert des Parameters `ahd_caller` wird eine Referenz auf das passende Element in der vorgegebenen Hierarchie übergeben. Ein Beispiel (angelehnt an Abbildung 36) ist in Abbildung 48 dargestellt, bei dem die Elementhierarchie auf der linken Seite beim Matching mit der Vorgabehierarchie auf der rechten Seite das dort enthaltene `func`-Element aktiviert.

Ein weiteres Ereignis, welches neu eingeführt wird, ist das bereits oben angesprochene *onpost*-Ereignis. Es wird immer dann erzeugt, wenn ein Dokument Ziel eines Funktionsaufrufs ist, bei dem die aufzurufende Funktion nicht genannt wird. Wie alle

<pre><message> <printformat> <style> HighQuality </style> </printformat> ... </message></pre>	<pre><message> <printformat> <func name="onmatch"> ... </func> </printformat> </message></pre>
--	--

Abb. 48: Behandlung des onmatch-Ereignisses

anderen externen Ereignisse wird auch dieses Ereignis dann über entsprechend benannte Funktionen verarbeitet.

Adressierung

Wie in Unterkapitel 13.3 bereits erwähnt, wird als Adressierungsmechanismus der URI benutzt, wobei in den meisten Fällen ein URL mit zusätzlichem XPath-Ausdruck eingesetzt wird. Damit lassen sich alle Elemente innerhalb eines Dokuments ansprechen. Für die Kommunikation zwischen Dokumenten sind hier insbesondere die func- und var-Elemente von Bedeutung.

Eine weitere Adressierungsmethode kommt bei der Reaktion auf IEM-Ereignisse zum Tragen. Dort lassen sich zwei Fälle unterscheiden. Einerseits existieren im IEM Ereignisse, die eine Umsetzung von benutzerinitiierten Ereignissen wie Mausklicks oder Tastendrücken auf die handelnden Elemente erfordern. Andererseits existiert auch eine Gruppe von Ereignissen, bei der die auslösende Instanz nicht explizit festlegt, welches Element die Ereignisse behandeln soll. Dazu zählen die Ereignisse onload, onunload, onpost und onmatch. Sie basieren auf einer impliziten Adressierung des handelnden Elements und stellen somit eine einfache Multicast-Kommunikation zur Verfügung.

Die explizite und implizite Adressierung wird für direkte und indirekte Kommunikation genutzt, welche im folgenden auf technischer Ebene erläutert werden sollen.

Direkte Kommunikation

Unter direkter Kommunikation soll hier jegliche Kommunikation verstanden werden, bei der der Adressat explizit genannt wird. Sie wird bei der Behandlung des Großteils der IEM-Ereignisse und bei der Nutzung der AHD-Zugriffsfunktionen eingesetzt.

get

Technisch wird der Zugriff auf var-Elemente in externen Dokumenten über einen HTTP-GET-Request umgesetzt, bei dem die Referenzierung einer Variablen über einen XPath oder eine parametrisierte URL im entfernten Netzknoten durchgeführt und nur der Variablenwert zurückgeliefert wird. Ebenso können auch entfernte Dokumente angefordert werden. Beide Möglichkeiten wurden bei der Beschreibung des Strukturzugriffs in Unterkapitel 13.3 bereits erläutert. Die Nutzung von HTTP ist allerdings bei Zugriffen innerhalb einer Laufzeitumgebung nicht zwingend erforder-

lich, hier bietet sich aus Optimierungsgründen ein direkter Zugriff an. Allerdings dabei ist die Beibehaltung der Semantik der HTTP-GET-Methode (z.B der blockierende Aufruf, nötigenfalls die Kodierung der URL) wichtig.

put

Die technische Umsetzung erfolgt beim Setzen von Variablen in externen Dokumenten über einen HTTP-PUT-Request. Neben dem Setzen von Variablenwerten kann sich der PUT-Request wie bei der get-Funktion aber auch auf ein gesamtes Dokument beziehen.

call

Analog zur get- und put-Funktion erfolgt die technische Realisierung beim Ansprechen externer AHD-Funktionen durch ein HTTP-Request, hier wird der POST-Request benutzt. Daraus resultiert im einfachsten Fall eine Parameterübergabe, welche dem CGI-Mechanismus entspricht: Die Parameter werden dort in einer Umgebungsvariable codiert (entsprechend den MIME-Typen *multipart/form* für die Übermittlung von Dateiinhalten oder *application/x-www-form-urlencoded* für einfache zeichenkettenbasierte Parameter, siehe [112] und [148]) abgelegt, das Ergebnis der Funktion wird über die Standard-Ausgabe zurückübermittelt. Dies ist aber nur die einfachste Form, im AHDM werden durch die Implementierung der Laufzeitumgebung die Variablen *ahd_params* und *ahd_result* genutzt, um die Parameter und einen Rückgabewert zu übermitteln. Zudem sollen die Parameter noch in entsprechend den Parameternamen benannten Variablen übergeben werden.

Indirekte Kommunikation

Neben der expliziten Adressierung der Kommunikationspartner können diese auch indirekt miteinander kommunizieren. Dazu kommen die Delegations- und Struktur-Matching-Mechanismen zum Einsatz.

Parent Delegation

Die erste Realisierung indirekter Kommunikation ist per parent delegation umgesetzt. Gemäß den Sichtbarkeitsregeln beim Elementzugriff über AHD-Funktionen werden Funktions- und Variablenelemente in unterschiedlichen, hierarchisch angeordneten Sichtbarkeitsbereichen gefunden, auch wenn sie nicht direkt adressiert worden sind.

Remote Delegation

Eine Erweiterung der Menge der adressierbaren AHD-Elemente erfolgt über die remote delegation. Durch diese Delegationsform werden entfernte Dokumente in den Sichtbarkeitsbereich mit aufgenommen. Diese Referenzierung braucht aber nicht beim Funktionsaufruf oder Variablenzugriff zu erfolgen, sie wird stattdessen durch das delegate-Element innerhalb der Elementhierarchie realisiert.

Multicast über Struktur-Matching

Während die Delegationsmechanismen im AHDM die implizite Adressierung einzelner Funktions- und Variablenelemente erlauben, können über das Struktur-Matching ganze Elementhierarchien oder Dokumente gleichzeitig adressiert werden, wobei bestimmte Elemente in den Dokumenten über das Matching gezielt ausgewählt werden. So kann ein aufrufendes Dokument über eine Menge aufzurufender

Dokumente iterieren und ein Matching zwischen einer Vorgabestruktur und den einzelnen Dokumenten vornehmen. Die dabei erzeugten onmatch-Ereignisse identifizieren die jeweils passenden Elemente. Die Ereignisse aktivieren eventuell vorhandene func-Elemente, die den Namen onmatch besitzen (siehe dazu auch Abbildung 48).

Multicast über onload, onunload und onpost

Neben den speziell aus dem AHDM stammenden Multicastmechanismen können auch die im IEM definierten Ereignisse beim Laden und Deaktivieren eines Dokuments genutzt werden. Dazu kommt noch der Sonderfall, in dem ein Dokument per POST-Methode aktiviert wird, aber keine Funktion explizit aufgerufen wird. Dies entspricht der normalen Aktivierung eines CGI-Skripts.

In diesen Fällen werden alle Funktionen, welche den Namen des entsprechenden Ereignisses tragen, in einem Multicast-Aufruf aktiviert, z.B. werden beim Laden eines Dokuments alle func-Elemente, deren Namensattribut den Wert onload besitzt, aufgerufen. Gleiches gilt für die Ereignissen onunload und onpost.

13.8 Semantiken

Die vorangegangenen Ausführungen beschreiben die Eigenschaften aktiver Hypertextdokumente und der Laufzeitumgebung aus einer statischen Sicht. Werden die Eigenschaften genutzt und verändert, so ergibt sich eine dynamische Sicht. Die daraus resultierenden Semantiken werden in diesem Unterkapitel beschrieben.

Aktivität

Ein AHD kann zwei Grundzustände annehmen: Entweder ist ein AHD *aktiv*, d.h. innerhalb der Laufzeitumgebung können die Verhaltensbeschreibungen ausgeführt werden, oder es ist *inaktiv*. Wie in der Beschreibung der Laufzeitumgebung dargelegt, werden aktive AHDs innerhalb der Laufzeitumgebung gehalten und inaktive AHDs in einem Dateisystem oder einer Datenbank abgelegt.

Der Wechsel vom aktiven zum inaktiven Zustand und umgekehrt wird nicht durch das AHD selbst durchgeführt. Vielmehr wird ein AHD aktiv, wenn es in die Laufzeitumgebung gelangt, beispielsweise durch das Laden über das Netz oder aus einer Datei. Umgekehrt deaktiviert die Laufzeitumgebung ein AHD, z.B. durch explizites Abspeichern, durch die Beendigung des Programms, das die Laufzeitumgebung enthält, oder durch Schließen eines Dokumentenfensters.

Der Zustandswechsel wird an ein AHD durch die IEM-Ereignisse „onload“ und „onunload“ signalisiert.

Persistenz

Um Systemzustände dauerhaft und über Systemgrenzen hinweg festzuhalten, sind Mechanismen zur Speicherung und Wiederherstellung dieser Zustände notwendig. Im AHDM sollen diese Mechanismen auf Dokumentenebene bereitgestellt werden, darüber hinausgehende Techniken wie z.B. die Zusicherung von Konsistenz bei verteilten Anwendungen sind nicht Bestandteil des Modells, sondern müssen auf Anwendungsebene realisiert werden. Dazu stehen u.a. auch die in Web-basierten Informationssystemen eingesetzten Techniken wie etwa das Cookie-basierte Session-Management zur Verfügung.

Die Persistenz wird im AHDM sowohl über die Nutzdaten eines AHDs (der Inhalt des Dokuments) als auch über die Zustandsbeschreibungen (die var-Elemente) hergestellt. Hiervon abzugrenzen sind aber die Zustände, die innerhalb der Verhaltensbeschreibungen, beispielsweise durch lokale Variable, geöffnete Dateien oder hergestellte Netzwerkverbindungen, definiert sind. Diese sind im AHDM nicht persistent, sondern liegen im Bereich der Laufzeitumgebung.

Die persistente Speicherung eines AHDs stützt sich im wesentlichen auf die Funktionen `toText` und `put`, wobei erstere eine textuelle XML-Repräsentation eines aktiven Hypertextdokuments erzeugt, die mittels der `put`-Methode unter einer gegebenen URI abgespeichert werden kann. Die textuelle Repräsentation umfaßt alle Bestandteile des Dokuments, die durch das DOM adressiert werden können, ausgenommen sind also die Werte der o.g. programmiersprachenspezifischen Variablen innerhalb der Programmskripte in einem AHD. Sollen diese Variable ebenso persistent gespeichert werden, so müssen ihre Werte in AHD-Variable im Dokument abgelegt werden. Die persistente Speicherung erfolgt im AHDM nicht automatisch, sondern nur einen expliziten Aufruf der entsprechenden Funktionen, wie im folgenden in Zusammenhang mit der Frage nach der Identität aktiver Hypertextdokumente näher erläutert wird.

Identität

Aus der Referenzierbarkeit, der Übertragung mittels HTTP, der Aktivität und der persistenten Speicherung aktiver Hypertextdokumente sowie des Aufbaus der Laufzeitumgebung entsteht die Frage nach der Identität von AHDs. Dazu zählt beispielsweise die Integrität von mehrfachen Kopien ein und desselben AHDs, die Unterscheidung zwischen persistent gespeicherter Repräsentation und in der Laufzeitumgebung aktiver Instanz eines AHDs oder die Bedeutung der URI bei der Referenzierung eines AHDs.

Grundlegend für den Begriff der Identität ist die Tatsache, daß ein AHD, welches über eine URI angesprochen wird, nur eine einzige aktive Instanz in einer einzigen Laufzeitumgebung besitzt. Die Laufzeitumgebung, die diese aktive Instanz beinhaltet, ist dabei durch die URI direkt oder indirekt eindeutig identifiziert. Wird also ein AHD von außerhalb der Laufzeitumgebung über eine URI angesprochen (etwa als Ziel der `get`-Methode), so wird bei unterschiedlichen externen Referenzierungen immer dieselbe aktive Instanz benutzt. Demnach sind zwei AHDs, die über die gleiche URI identifiziert werden, identisch.

Aktive Instanzen werden im Regelfall durch das Laden einer textuellen Repräsentation eines AHDs (z.B. einer Datei oder eines Datensatzes einer Datenbank) in die Laufzeitumgebung erzeugt. Modifikationen der aktiven Instanz eines AHDs wirken sich allerdings nicht auf diese gespeicherte Form aus, d.h. um die Änderungen an einem AHD dauerhaft festzuhalten, muß die aktive Instanz unter einer gegebenen URI per `put`-Methode gespeichert werden. Die aktive Instanz und die textuellen Repräsentationen eines AHDs sind demnach nicht identisch. Dies gilt auch, wenn ein AHD über die `get`-Methode angefordert wird. Hierbei wird lediglich eine Kopie der aktiven Instanz zurückgeliefert. Aus dieser Kopie kann eine zweite aktive Instanz mittels der `fromText`-Methode generiert werden, die aber unabhängig von der ersten Instanz ist.

Aus den o.g. Ausführungen ergeben sich mehrere Konsequenzen für die Nutzung von AHDs. So sind AHDs nicht im eigentlichen Sinne mobil, stattdessen werden le-

diglich Kopien von AHDs über unterschiedliche Transportwege übermittelt. Die Kopien und daraus generierte aktive Instanzen sind jeweils unabhängig voneinander, solange sie nicht über die gleiche URI referenziert werden können. Auch muß der Abgleich von aktiver Instanz und textueller, persistenter Repräsentation manuell durchgeführt werden. Zwar stellen diese Mechanismen eine Einschränkung des Systementwicklers dar, allerdings vereinfachen sie die Implementierung der Laufzeitumgebung erheblich und erleichtern den Einsatz von AHDM-Techniken in unterschiedlichen Systemen.

Modifikation

Im AHDM sind die Funktionen, die das DOM anbieten, Grundlage für Modifikationen. Denkbar ist aber auch der Einsatz deklarativer Ansätze, wie beispielsweise XSLT. Im AHDM werden drei unterschiedliche Arten der Modifikation unterschieden: erstens die Modifikation der Struktur eines Dokuments beispielsweise durch Entfernen oder Hinzufügen von Elementen, zweitens die Modifikation des textuellen Inhalts von Elementen und drittens die Modifikation von Attributwerten. Alle drei Modifikationsarten können zudem eingesetzt werden, um die Vernetzung in und zwischen Dokumenten zu verändern. Außer den strukturellen Änderungen werden sowohl die inhaltliche als auch die Änderung der Attributwerte innerhalb des AHDM genutzt. Die Modifikation des textuellen Inhalts dient im AHDM zur Veränderung der AHD-Variablen bei der *put*-Methode, die Veränderung der Vernetzung von AHDs wird bei der weiter unten angesprochenen Verhaltensänderung eingesetzt.

Referenz und Speicherungsart

Die Referenzierung eines Dokuments bedeutet die Bezugnahme auf einen physikalischen Speicherungsart. Dies resultiert aus der überwiegenden Nutzung von URLs als Referenzierungs- oder Adressierungsmechanismus. Ein URL beschreibt eine *location*, d.h. einen Ort, während allgemeinere Mechanismen wie URI oder URN nur einen *identifier* oder *name* beinhalten. Andererseits kann der Wert einer Referenz auch andere Semantiken haben, so kann er Versionsinformation beinhalten, unterschiedliche Datenformate eines sonst gleichen Inhalts unterscheiden oder andere Meta-Informationen transportieren. Im Rahmen des AHDM stellt sich die Frage, inwieweit die Werte einer Referenz sowie deren Veränderung relevant für das Modell an sich sind.

Angesichts der komplexen Ausdrucksmöglichkeiten, die die Mechanismen URI, URN und URL bieten und die absehbar auch ausgeweitet werden, wird an dieser Stelle darauf verzichtet, dem Wert einer Referenz eine Bedeutung zuzuteilen.

Es bleiben als offene Fragen, was durch die Veränderung einer Referenz ausgedrückt wird, welche Auswirkungen sie hat und wie sie im AHDM, der Entwicklungsmethode oder in einer Implementierung behandelt wird:

Bedeutung: Allgemein formuliert bedeutet eine Veränderung der Referenz eine Veränderung der mit der Referenz verbundenen Meta-Information. Die wichtigsten Fälle sind der Wechsel des Speicherungsortes, der Wechsel des Transport- oder Speicherungsmediums, der Wechsel der Repräsentationsform oder die Kennzeichnung einer inhaltlichen Veränderung, etwa durch Versionierung. Abbildung 49 zeigt einfache Beispiele für die Änderung von Referenzen.

Änderung des Speicherorts:

`http://mohegan/Mitarbeiter.html`
⇒ `http://nestroy/Mitarbeiter.html`

Änderung des Transportmediums:

`http://nestroy/kino-2.5.0.tgz`
⇒ `ftp://nestroy/kino-2.5.0.tgz`

Änderung der Repräsentationsform:

`http://nestroy/Mitarbeiter.html`
⇒ `http://nestroy/Mitarbeiter.wml`

Änderung von Versionsinformation:

`http://nestroy/kino-2.5.0.tgz`
⇒ `http://nestroy/kino-3.0.0.tgz`

Abb. 49: Beispiele für Referenzänderungen

Auswirkungen: Durch die Nutzung von XLink und XPath als Mechanismus zur Vernetzung werden Referenzen als Links innerhalb eines Dokuments genutzt. Es existiert im Regelfall also kein zentraler Punkt, an dem die Referenzen verwaltet werden. Dies bedeutet jedoch, daß Änderung des Orts eines Dokuments oder die Modifikation von dessen Struktur in allen referenzierenden Dokumenten nachgezogen werden müssen.

Behandlung: Die möglichen Alternativen für die Behandlung von Referenzänderungen sind vielfältig. Generell läßt sich zwischen zwei Ansätzen unterscheiden: solchen, die eine zentrale Instanz für die Aufrechterhaltung der Integrität der Referenzen nutzen, und dezentralen Ansätzen, bei denen die referenzierten Dokumente genügend Information zur Integritätssicherung beinhalten. In die erste Gruppe fällt die durchgängige Nutzung von URNs mit einem Namensdienst, der die URNs zu physikalischen Speicherorten auflösen kann. In die zweite Gruppe fallen solche Ansätze, bei denen die Referenzintegrität durch Mittel wie Rückwärtsreferenzen oder Stellvertreterdokumente aufrecht erhalten wird.

Die endgültige Entscheidung für einen zentralen oder einen dezentralen Ansatz soll nicht innerhalb des AHDMs getroffen werden. Vielmehr wird an dieser Stelle nur darauf hingewiesen, daß Ansätze, bei denen eine zentrale Instanz benötigt wird, in der Systemarchitektur aufwendiger sind als dezentrale Lösungen, die zudem den Vorteil der Eigenständigkeit der Dokumente mit sich bringen.

Verhaltensänderung

Die Änderung des Verhaltens eines AHDs kann auf zwei Arten erfolgen: durch Änderung der Verhaltensbeschreibungen und durch Änderung der Dokumentstruktur. Die Änderung der Verhaltensbeschreibungen kann durch Modifikation des textuellen Inhalts der entsprechenden func-Elemente erfolgen. Zu beachten ist dabei allerdings, daß die Nutzung von übersetztem Code (z.B. Java-Bytecode) statt textbasierter Programmskripte in diesen Programmfragmenten eine nachträgliche

Änderung problematisch macht, da hierfür der entsprechende Übersetzer zur Verfügung stehen muß.

Die zweite Möglichkeit der Verhaltensänderung besteht in der Modifikation der Dokumentstruktur. Da im AHDM auf die Funktions- und Variablenelemente über parent delegation zugegriffen wird, spielt die Struktur eines AHDs eine große Rolle bei der Ausführung von Programmcode. Durch Verlagern eines Funktionselements bekommt dieses einen neuen Kontext (d.h. neue Väterelemente) und somit Zugriff auf andere übergeordnete AHD-Elemente. Andererseits kann auch durch den Austausch der übergeordneten AHD-Elemente eine Verhaltensänderung bewirkt werden, ohne daß die untergeordneten Beschreibungen geändert werden müssen. Diese interessante Eigenschaft von AHDs geht in die im nächsten Abschnitt erläuterte Entwurfsmethode ein. Weiterhin ist es, wie bereits oben angedeutet, möglich, über die Modifikation von Hyperlinks das Verhalten bei remote delegation zu ändern, indem auf andere, externe Dokumente zur Ausführung von delegiertem Verhalten verwiesen wird.

Jegliche Modifikation eines Dokuments während der Ausführung der eingebetteten Programmskripte ist eine kritische Operation, die über die Methoden lock und unlock abgesichert werden muß, damit Probleme bei nebenläufiger Programmausführung vermieden werden. Dies gilt insbesondere für die Manipulation zur Änderung des Verhaltens. Da das AHDM hierbei keine expliziten Schutzmechanismen vorsieht, muß bei der Modifikation eines AHDs darauf geachtet werden, daß das dabei aktive Programmfragment durch die Modifikation weiterhin korrekt arbeitet. So sind Modifikationen zu vermeiden, die die parent delegation ausgehend vom aktiven func-Element beeinflussen.

14 Zusammenfassung

Das vorgestellte Modell zur Beschreibung aktiver Hypertextdokumente wurde in einem mehrstufigen Prozeß entwickelt. Die Motivation und die identifizierten Anforderungen stellen die Basis für das Modell dar, die Betrachtung verwandter Bereiche liefert einerseits eine Abgrenzung, andererseits auch nützliche Impulse. Der eigentliche Entwurf des Modells orientiert sich an den vorstehenden Punkten und erfordert Design-Entscheidungen, die wesentlich zu den Charakteristika des Modells beitragen. Schließlich werden die einzelnen funktionalen Anforderungen durch entsprechende Mechanismen abgedeckt.

Bei der Diskussion des Modells in diesem Kapitel soll die Erfüllung der aufgestellten Anforderungen untersucht werden, insbesondere auch die Erfüllung nicht-funktionaler Anforderungen. Schließlich weisen das AHDM und die damit beschreibbaren aktiven Hypertextdokumente eine Reihe von charakteristischen Eigenschaften auf, die noch einmal klar herausgestellt werden sollen.

14.1 Erfüllung der Anforderungen

Die Anforderungen wurden bei der Analyse in funktionale und nicht-funktionale Anforderungen aufgeteilt. Während die funktionalen Anforderungen häufig einfach erfüllt werden können und meist direkt in einem darauf abzielenden Beschreibungsmittel resultieren, sind nicht-funktionale Anforderungen schwerer zu bewerten und zu erfüllen.

Funktionale Anforderungen

Eine kurze Betrachtung der funktionalen Anforderungen liefert folgende Zuordnung von Anforderungen zu Beschreibungsmitteln:

Informationsstrukturierung: Ein aktives Hypertextdokument ist immer ein XML-Dokument.

Kommunikation: Die Kommunikation und Übertragung von AHDs erfolgt bevorzugt über HTTP, wobei ein AHD über einen URI referenziert werden kann.

Informationsvernetzung: Zur Vernetzung werden XLink und XPath verwendet.

Informationsdarstellung: Die Darstellung der in einem AHD enthaltenen Information erfolgt über CSS, andere Mittel wie DSSSL oder XSL sind aber ebenso nutzbar.

Interaktion: Die Koordination der Interaktion zwischen Benutzer, AHD und Laufzeitumgebung wird durch das IEM abgedeckt.

Beschreibung von Verhalten: Verhaltensbeschreibungen werden durch das `func`-Element in ein AHD eingebettet, sie enthalten typischerweise Skripte in Interpreter-Sprachen wie Tcl/OTcl, Perl, Lua, Scheme usw.

Zustandsbeschreibungen: Der Zustand eines AHDs wird in `var`-Elementen gehalten.

Verbindung von Zustand, Verhalten und Präsentation: Bei der Verbindung von Zustand, Verhalten und Präsentation kommen XML und XML-Namespaces als Beschreibungsmittel zum Einsatz, was zum Informationsmodell im AHDM führt.

Definition von Semantiken: Einerseits liegen Eigenschaften wie Aktivität und Persistenz im Kernbereich des AHDMs, andererseits aber kommen andere wichtige Semantiken erst in einer Entwurfsmethode oder einer Implementierung zum Tragen.

Nicht-funktionale Anforderungen

Neben den funktionalen sind es gerade die folgenden nicht-funktionalen Anforderungen Qualitätsmerkmale, an denen das AHDM gemessen werden soll:

Interoperabilität und Ausnutzung bestehender Standards: Die Forderung nach Interoperabilität impliziert im Einsatzgebiet AHD-basierter Systeme vornehmlich die Ausnutzung bestehender Standards. Alle im AHDM verwendeten Beschreibungsmittel sind offene Standards. Ein kritischer Bereich ist hier allerdings die Nutzung von HTTP als Mittel zur Referenzierung und Aktivierung von AHD-Elementen, wodurch HTTP-Requests eine erweiterte Bedeutung erlangen. Die Nutzung von HTTP ist aber nur eine Empfehlung, durch die die Nutzung eines weiteren Mechanismus wie beispielsweise RPC vermieden wird.

Ferner ist bei der Realisierung AHD-basierter Systeme auf eine offene Architektur zu achten, die die Interaktion mit anderen Systemen erhöht. Es ist zu erwarten, daß dies durch den Einsatz offener Standards erleichtert wird.

Auf- und Abwärtskompatibilität: Dokumente, die nach dem AHDM erstellt wurden, können entsprechend dem Informationsmodell in ihre einzelnen Komponenten (Verhalten, Zustand, Präsentation und Nutzdaten) zerlegt werden, so

daß, je nach Systemumgebung, die Teile ausgeblendet werden können, die sich auf nicht verfügbare Techniken stützen, wie z.B. Verhaltensbeschreibungen.

Die Aufwärtskompatibilität wurde insofern versucht zu berücksichtigen, als daß solche offenen Standards als Basis für das Modell gewählt wurden, die ein möglichst großes Einsatzgebiet haben. Dadurch sollen zukünftige Anforderungen leichter erfüllbar sein.

Flexibilität: Der Vorteil bei der Verwendung von XML und XLink als Grundlage für die Strukturierung und Vernetzung von Information liegt in der hohen Flexibilität der resultierenden Systeme. Die kleinste logische Einheit eines XML-Dokuments sind die Elemente, und mit XLink ist es möglich, einzelne Elemente miteinander zu verbinden. Elemente können aber auch in anderen Konstellationen zu anderen Dokumentstrukturen zusammengefügt werden, wobei insbesondere XML-Namensräume helfen, diese Elemente nötigenfalls zu separieren. Insbesondere die Parent Delegation erlaubt es zudem, Verhaltensbeschreibungen so zu entwerfen, daß keine expliziten Annahmen über die Dokumentstrukturen nötig sind.

Verständlichkeit und Wartbarkeit: Verständlichkeit und Wartbarkeit der Dokumente, die mit dem AHDM erstellt werden, sind Anforderungen, die nicht direkt mit dem AHDM erfüllt werden können. Hier ist eher die Entwurfsmethode gefordert, die Empfehlungen hinsichtlich des Aufbaus vom AHD-basierten Systemen aussprechen soll. Das AHDM erlaubt durch flexible Gestaltungsmöglichkeiten das Befolgen der Empfehlungen.

Skalierbarkeit: Auch die Skalierbarkeit ist direkt von der Flexibilität im AHDM abhängig. So sind keine Restriktionen vorhanden, die eine bestimmte Verteilung der Systembestandteile bevorzugen, vielmehr können Systeme bei geänderten Anforderungen entsprechend umstrukturiert werden.

Implementierungsunabhängigkeit: Das AHDM wurde so gestaltet, daß offene Standards als Basis dienen und kein Vorgriff auf eventuelle Implementierungen erfolgt. Dies schränkt zwar den Beschreibungsumfang des AHDM ein – so sind Sicherheitsaspekte oder Vereinfachungsfunktionen nicht Kernbestandteil – allerdings sollen solche weitergehenden Aspekte in den folgenden Abschnitten beschrieben werden.

Orthogonalität und Konsistenz: Die Orthogonalität drückt sich insbesondere durch die Nutzung eigener Beschreibungsmittel für Zustand und Verhalten in einem AHD aus. Dadurch wurde eine klare Trennung im Informationsmodell erreicht, die z.B. der Abwärtskompatibilität zugute kommt. Die Konsistenz zeigt sich in der Nutzung von HTTP als Transportmechanismus einerseits und als Aktivierungs- bzw. Referenzierungsmechanismus für AHD-Elemente andererseits. Auch die alleinige Nutzung von XML zur Strukturierung trägt zur Durchgängigkeit bei.

14.2 Charakteristika des Modells

Das vorgestellte Modell weist einige besondere Charakteristika auf, dazu zählen folgende Punkte:

- Das Modell selbst besteht nur aus wenigen Beschreibungsmitteln, es weist eine geringe Komplexität auf, erlaubt aber durch die Möglichkeiten der Basistechniken vielfältige Anwendungen.
- Bei der Wahl der Basistechniken wurden nur offene Standards verwendet.
- Der Mechanismus der parent delegation führt zu autonomen Elementen, da die expliziten Referenzen auf die Umgebung der Elemente minimiert wird. Solche autonomen Elemente können einfacher neu zusammengesetzt werden.
- Das AHDM beinhaltet ein Informationsmodell, welches die einzelnen Bestandteile aktiver Hypertextdokumente sinnvoll voneinander abgrenzt. Zusätzlich bietet sich mit unscharfer Validierung die Möglichkeit, die einzelnen Bestandteile gegen eine Strukturbeschreibung zu prüfen.
- Da das Modell implementierungsunabhängig ist, werden bestimmte Teile und Semantiken erst in einer Entwicklungsmethode oder einer Implementierung beschrieben. So ergibt sich eine Hierarchie, bei der das AHDM die Basis darstellt, darauf bauen die Entwicklungsmethode und schließlich die möglichen Implementierungen auf.

Als eine wichtige Erkenntnis zeigt sich die Mächtigkeit von XML. Ohne XML wäre das AHDM nicht sinnvoll zu formulieren, die gestellten Anforderungen könnten nicht im gewünschten Maße erfüllt werden. Dies wird auch offensichtlich, wenn man die verwandten Bereiche betrachtet. Auch die XML-Namensräume spielen eine entscheidende Rolle, um das dem AHDM zugrundeliegende Informationsmodell zu realisieren. Offen bleibt die Frage, inwieweit unterschiedliche Informationsstrukturen mit ähnlicher zugrundeliegender Bedeutung ausgetauscht werden können, etwa bei der Verknüpfung von Geschäftsprozessen unterschiedlicher Unternehmen. Es ist jedoch zu erwarten, daß XML in diesem Fall eine stabile Basis für den Informationsaustausch bietet und Techniken wie XSLT und RDF die notwendige Anpassung auf semantischer Ebene ermöglichen.

14.3 Charakteristika aktiver Hypertextdokumente

Abschließend sollen noch die besonderen Eigenschaften aktiver Hypertextdokumente für die Implementierung von Informationssystemen im Vergleich zu anderen Techniken herausgestellt werden.

AHD-Komponenten

Ein Aspekt aktiver Hypertextdokumente ist die Verknüpfung von Programmcode mit Daten. Dabei werden Funktionslemente und Variablenelemente einem übergeordneten Element zugeordnet. Dieses Konzept liegt den objektorientierten und -basierten Programmiersprachen zugrunde, bei dem das übergeordnete Element als Objekt bezeichnet wird. Im AHDM sollen diese Zuordnungen flexibel änderbar sein. Durch das Informationsmodell und die Nutzung von XML-Namensräumen wird eine technische Möglichkeit zur expliziten Trennung dieser Elemente geschaffen.

Ein weiterer Punkt, in dem sich das AHDM von statisch typisierten objektorientierten Programmiersprachen unterscheidet, ist der Verzicht auf ein Typ- und Klassensystem. Aus diesem Grund ist die Vererbung von Eigenschaften wie sie in vielen objektorientierten Programmiersprachen zentraler Bestandteil ist, nicht möglich. Durch das fehlende Typsystem können zudem keine statischen Überprüfungen der Systemintegrität wie z.B. die Gültigkeit von Parametertypen oder erlaubten Funkti-

onsaufrufen durchgeführt werden. Der Verzicht auf solche Mechanismen erlaubt aber eine wesentlich flexiblere Nutzung von Systembestandteilen. Gerade die Heterogenität des WWW und des Internet erfordert die möglichst vielseitige Nutzbarkeit von Systembestandteilen. Hierbei liefert XML eine strukturierende Basis für diese Bestandteile, weitere Eigenschaften werden mit dem AHDM definiert. Darüber hinaus aber sollen aktive Hypertextdokumente im Einsatz nicht weiter festgelegt werden.

Der Verzicht auf Mittel wie Vererbung oder komplexe Typsysteme läßt sich auch in der komponentenbasierten Software-Entwicklung beobachten, insbesondere bei der Entwicklung verteilter Systeme [185]. Solche Systeme können zur Laufzeit Veränderungen unterworfen sein, die bei der Implementierung des Systems noch nicht vorzusehen waren, was die vorausschauende Implementierung etwa durch Definition möglichst stabiler Interfaces zwischen den Systemkomponenten erschwert. Zwar ist der Begriff der Software-Komponente nicht eindeutig definiert (für einen Überblick über den Stand der Komponenten-basierten Software-Entwicklung siehe etwa [35]), dennoch läßt sich sagen, daß Komponenten durch eindeutige Schnittstellen definiert sind und möglichst variabel eingesetzt und konfiguriert werden können. Häufig besitzen Komponenten auch eine gröbere Granularität als beispielsweise Klassen in objektorientierten Programmen.

Die oben aufgezählten Eigenschaften von Software-Komponenten lassen sich zu einem gewissen Grade auf aktive Hypertextdokumente übertragen. Aktive Hypertextdokumente besitzen mit dem Wurzelement und den direkt untergeordneten Funktions- und Variablenelementen eine nach außen sichtbare Schnittstelle (siehe auch das Muster *Document Facade*, beschrieben ab Seite 123). Sie sind durch diese Schnittstelle und die Laufzeitumgebung überall dort verwendbar, wo HTTP unterstützt wird. Die Granularität ist im Regelfall grob, da AHDs als Ganzes von außen über die Schnittstelle genutzt werden. Allerdings ist es ebenso möglich, auf die interne Struktur eines AHDs zuzugreifen. Die Anpassung eines AHDs an die Systemumgebung kann unter anderem durch den Einsatz von remote delegation erfolgen. Werden einzelne Elemente oder Elementhierarchien als Komponenten genutzt, wird die Anpassung durch die indirekte Adressierung von Funktions- und Variablenelementen über parent delegation erleichtert.

Das Zusammenspiel von Komponenten wird oft durch Verbindungsmechanismen wie etwa Konfigurations- oder Skriptsprachen erleichtert [137], so daß bestehende Komponenten etwa mittels eines *Object System Layer* [69] angesteuert werden können. Im AHDM werden Verbindungstechniken an zwei Stellen genutzt: Innerhalb von aktiven Dokumenten realisieren die func-Elemente die Verbindung zwischen den Informationselementen. Die Dokumente selbst werden über HTTP verbunden, welches die Basis für andere Techniken wie etwa IIOP über HTTP oder XML-RPC sein kann. Zudem lassen sich auch bereits bestehende Komponenten (z.B. HTTP-basierte Applikationsserver) in AHD-basierte Anwendungen einbinden, was zusätzlich den Vorteil der Schaffung vernetzter Systeme schafft.

Software-Agenten

Zwar sind aktive Hypertextdokumente nicht im eigentlichen Sinne mobil, wie es in Unterkapitel 13.8 bei der Diskussion der Identität von AHDs bereits beschrieben worden ist, dennoch können AHDs zur Implementierung von agentenbasierten Systemen genutzt werden. Aktive Hypertextdokumente können zur Ausführung an ent-

fernte Orte transportiert werden und kommen dadurch dem *Remote Programming* [187] nahe, daß Software-Agenten zugrunde liegt. Sie unterstützen die Verlagerung von applikationsspezifischen Bestandteilen in mobile, eigenständige Einheiten (den Dokumenten) statt der Erstellung spezifischer Applikationsserver. Aktive Hypertextdokumente unterscheiden sich allerdings von den für eine Aufgabe angepaßten Agenten durch die über das Informationsmodell des AHDM gegebene Möglichkeit, nicht nur aufgabenspezifische Informationen zu beinhalten, sondern beliebige Nutzdaten zu transportieren.

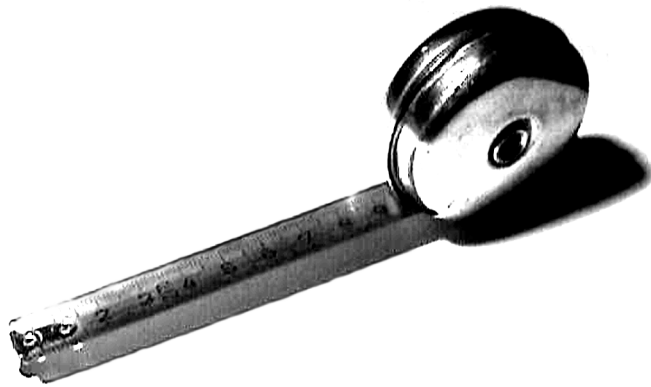
Allerdings stellt das AHDM nur eine sehr grundlegende Basis für Agentensysteme zur Verfügung. Wichtige Aspekte von Agentensystemen wie die Zusicherung von Informationssicherheit (diskutiert etwa in [111]) sind nicht im AHDM festgelegt. Auch können die Mechanismen des Funktionsaufrufes, Variablenzugriffs oder der Kommunikation nur Grundlage für Konzepte wie Treffen, Ort, Berechtigungen (vgl. dazu wiederum [187]) oder abstraktere Aufgaben wie Informationsaustausch oder Kooperation zwischen Agenten oder Vereinbarungen über zu erbringende Dienste sein.

Weiterentwicklung des Dokumentenbegriffs

Die wichtigste Eigenschaft aktiver Hypertextdokumente ergibt sich aus der Weiterentwicklung des Dokumentenbegriffs, welche in drei Schritten erfolgt: Ausgehend vom unstrukturierten Dokument wird im ersten Schritt die Schaffung von Informationsstrukturen innerhalb des Dokuments mittels XML ermöglicht, womit eine maschinelle Verarbeitung der ursprünglich nur durch Menschen verstehbaren Dokumente erleichtert wird. Im zweiten Schritt werden Dokumente nicht als eigenständige Komponenten betrachtet, sondern über Hypertextmechanismen wie etwa XLink miteinander vernetzt. Das AHDM steuert dann im dritten Schritt die Ausführbarkeit der Dokumente bei und erweitert somit das Einsatzspektrum um die Realisierung von Web-basierten Informationssystemen. Dies ist in etwa vergleichbar mit der in [175] vorgeschlagenen Erweiterung von Netzwerkarchitekturen um aktive Komponenten, mit dem Unterschied, daß im AHDM die Information und deren Vernetzung im Mittelpunkt stehen. Der nächste Schritt wäre das maschinelle Verstehen von Dokumenten, der etwa durch RDF realisiert werden kann, aber nicht in dieser Arbeit weiter diskutiert werden soll.

Abschnitt C

Methode



15 Aufgabe der Methode

Die Beschreibung eines Modells reicht zur Erstellung eines Systems in den wenigsten Fällen aus. Sehr oft sind im Modell Freiheitsgrade enthalten (beispielsweise um Flexibilität bei der Systementwicklung zu erreichen), die weitere, vertiefende Richtlinien erfordern. Diese Richtlinien sollen ein Bestandteil einer Entwicklungsmethode sein, wie sie in diesem Abschnitt vorgestellt wird. Weiter soll die Methode aber auch Konstrukte zur Verfügung stellen und beschreiben, die bei der Entwicklung AHD-basierter Systeme wiederholt eingesetzt werden können. Dies resultiert unter anderem auch direkt in zu implementierenden Konstrukten und Mechanismen.

16 Anforderungen

Die Anforderungen an die Methode lassen sich in zwei Gruppen einteilen. Einerseits existieren Anforderungen, die die Methode direkt erfüllen muß, andererseits sind Anforderungen an die Systembestandteile (also die AHDs) und ihre Verbindung zu erfüllen, welches durch die Methode unterstützt werden soll.

16.1 Anforderungen an die Methode

Generell soll die Methode dazu dienen, aktive Hilfestellung beim Entwurf und bei der Realisierung von AHD-basierten Systemen zu leisten. Dies bedeutet, daß neben der reinen Dokumentation einer Systemarchitektur auch die Entwicklung derselben in möglichst klar definierten Schritten unterstützt werden soll. Es sind also sowohl eine Notation als auch ein Vorgehensmodell gefragt.

Anknüpfend an den vorangehenden Abschnitt sollen zusätzlich die dort identifizierten Semantiken als grundlegend verwendbare Konstrukte weiter formalisiert werden. An dieser Stelle sei bereits darauf hingewiesen, daß die Entwicklungsmethode stark auf die Nutzung solcher wiederverwendbarer Konstrukte ausgerichtet ist, äquivalent zur Verwendung von Patterns und Frameworks bei der objektorientierten Software-Entwicklung wie in [38] vorgestellt.

16.2 Anforderungen an die Systembestandteile

Die Bestandteile eines zu realisierenden Systems haben idealerweise einer Reihe allgemeiner Charakteristika zu folgen, die sich aus generellen Qualitätsanforderungen bei der Software-Entwicklung ableiten. Die Bestandteile umfassen dabei primär die aktiven Dokumente selbst, allerdings spielen durch die Ausrichtung des AHDM auf verteilte, vernetzte Systeme auch die Verbindungen zwischen den Dokumenten eine wichtige Rolle. Auch ist zu beachten, daß die Dokumente durch die Strukturierung mittels XML selbst wieder in kleinere Einheiten zerlegt werden können und daß auch die Laufzeitumgebungen als Netzwerkknoten in die Systemarchitektur eingehen.

Eine der wichtigsten Anforderungen ist die Frage der Wiederverwendbarkeit der Bestandteile bis auf Elementebene hinab. Sie ergibt sich aus dem gewünschten Einsatz von bewährten Konstrukten während der Entwicklung. Abgeleitet davon ist die Forderung nach möglichst loser Kopplung der Bestandteile untereinander sowie eine hohe Kohäsion innerhalb der Bestandteile, um die Verwendung nicht auf eine einzige Konfiguration zu beschränken.

Hilfreich für das erleichterte Verstehen eines Systems und für eine initiale Strukturierung des Systems erweisen sich weiterhin die Kapselung zusammengehöriger Bestandteile hinter einer gemeinsamen Schnittstelle und die logische Trennung der Informationsarten des dem AHDM zugrundeliegenden Informationsmodells.

17 Beschreibung der Methode

Bei der Entwicklung der Methode ist es hilfreich, die sich durch die Delegationsmechanismen ergebende Struktur eines aktiven Dokuments zu nutzen. So sind Elemente, die Unterelemente aus dem AHDM-Schema enthalten (also Zustandsvariablen und Funktionen), als Objekte, wie sie in objektbasierten Programmiersprachen benutzt werden, zu betrachten. Diese Eigenart kann die Verwendung objektorientierter Techniken sinnvoll machen. Bei den Beschreibungsmethoden wäre beispielsweise der Einsatz der *Unified Modelling Language* (UML) [157] denkbar. Allerdings ist die UML wesentlich umfangreicher als zur Modellierung AHD-basierter Systeme notwendig, andererseits fehlen Beschreibungsmittel, um die hierarchische Struktur aktiver Dokumente zu dokumentieren. Auch ist neben der objektorientierten Natur aktiver Hypertextdokumente deren Anlehnung an Software-Komponenten mindestens genau so wichtig.

Hier soll die *Business Object Notation* (BON) [183] als Anregung für die Entwicklung einer eigenen Methode genutzt werden. Sie unterstützt eine Reihe interessanter Aspekte, so z.B. die Minimierung von Beschreibungsmitteln, hierarchische Strukturen oder die Erleichterung der Überführung von Implementierungsprodukten (hier also AHDs) zurück zu Analyseprodukten (beispielsweise Übersichtsdiagramme). Auch bietet die BON ein Vorgehensmodell, welches hier adaptiert genutzt werden soll. Anzumerken ist an dieser Stelle weiterhin, daß sich das vorgeschlagene Vorgehen und die Charakteristika der Systembestandteile gegenseitig beeinflussen.

17.1 Notation

Die Notation benutzt vier Symbole und drei Beziehungstypen zwischen den Elementen, dargestellt in den Abbildungen 50 bis 52. Die Symbole repräsentieren aktive Hypertextdokumente, die XML-Elemente innerhalb der Dokumente und schließlich noch die Attribute dieser Elemente. Dokumente können zudem in *Clustern* zusammengefaßt werden, welche wiederum hierarchisch angeordnet werden können. Sie entsprechen typischerweise Verzeichnissen in einem Dateisystem. Durch die vorgestellte Notation soll nur die statische Struktur von AHD-basierten Systemen festgehalten werden. Die dynamischen Aspekte eines solchen Systems lassen sich nur schwer graphisch in ihrem vollen Umfang darstellen, weshalb hier darauf verzichtet werden soll.

Die Notation umfaßt damit zwei Hauptsichtweisen: Einerseits die Betrachtung der aktiven Dokumente und ihrer internen Struktur, andererseits die (hierarchische) Gruppierung von Dokumenten in Netzwerkknoten. Abbildung 50 zeigt ein Beispiel für die Cluster-Ansicht. Cluster werden als Rechtecke mit abgerundeten Ecken dargestellt, Dokumente als einfache Rechtecke.

Die interne Struktur eines AHDs wird rekursiv auf Elementebene dargestellt. Neben den Elementen, die die Kerninformation eines aktiven Dokuments enthalten, werden die Elemente aus dem AHDM-Schema getrennt dargestellt. Abbildung 51 enthält die

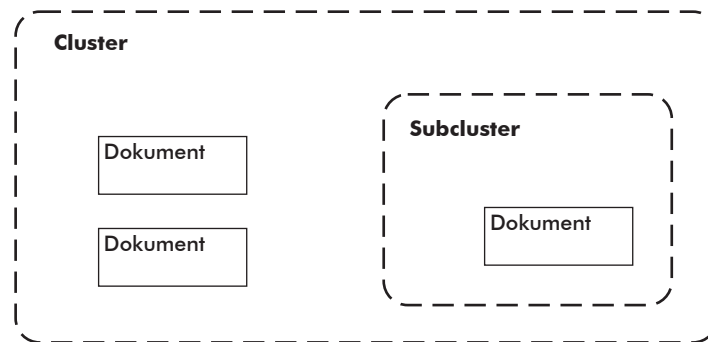


Abb. 50: Cluster-Ansicht

Sicht auf die oberste Hierarchieebene eines Dokuments, d.h. das Wurzelement eines Dokuments und alle direkten Unterelemente. Die Unterelemente sind aufgeteilt in AHDM-Elemente (graphisch am Rand des Dokuments positioniert, da sie die Schnittstelle für das Dokument darstellen) und sonstige Elemente. Diese Sichtweise kann rekursiv auf die Unterelemente im Dokument angewandt werden. In jedem Element wird der General Identifier (d.h. der Tag-Name) in der linken oberen Ecke eingetragen, zusätzlich wird bei AHDM-Elementen der Wert des name-Attributes zentriert notiert. Die Abbildung zeigt weiter noch die Definition eines Attributs.

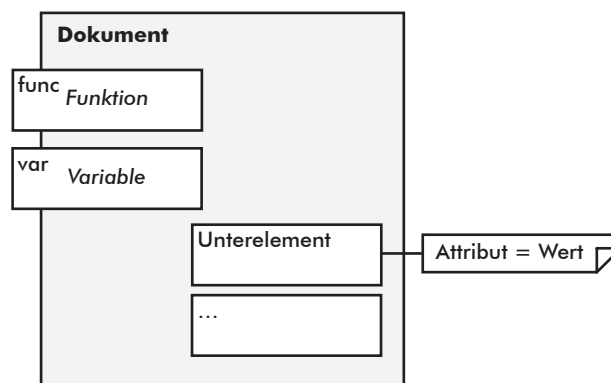


Abb. 51: Struktursicht auf ein Dokument

Die Beziehungen zwischen den Elementen fallen in zwei Kategorien: Verknüpfungen über XLink und Aufruf-Beziehungen durch die call-Funktion der AHDM-Laufzeitumgebung. Bei den XLink-Verbindungen soll weiter zwischen reinen Verweisen und Inklusions-Beziehungen unterschieden werden. Zu den Inklusions-Verbindungen gehören solche XLinks, bei denen das Attribut show den Wert parsed hat. Dazu kommen Verweise auf Stylesheets und externe Entities. XLink-Verbindungen beginnen immer an einem bestimmten Element und können auf ein anderes Element, ein anderes Entity oder eine Gruppe von Elementen verweisen, während AHDM-Aufrufe nur zwischen zwei Elementen existieren können. In Abbildung 52 sind von links nach rechts der einfache XLink, ein Inklusions-XLink und eine Aufrufbeziehung über eine der AHD-Methoden (get, put und call) dargestellt. Ein AHD-Aufruf kann mit zusätzlichen Informationen annotiert werden, welche bei einem HTTP-Request benötigt werden: der HTTP-Methode, zusätzlichen Headern und einem Request-Rumpf.

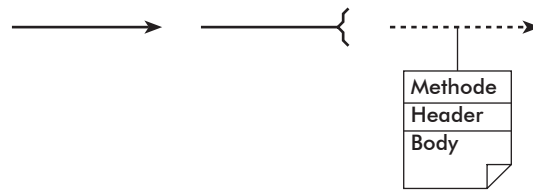


Abb. 52: Einfacher XLink, Inklusions-XLink und AHD-Aufruf

Die getrennte Betrachtung von Inklusions-XLinks dient der Hervorhebung virtueller Dokumentenstrukturen, die sich aus dem Zusammenschluß von physikalischen Entitäten per XLink ergeben. Dieser Mechanismus wird weiter unten näher erläutert.

17.2 Vorgehen bei der Systementwicklung

Da die Information in einem Hypertextsystem im Vordergrund steht, orientiert sich auch das Vorgehen bei der Entwicklung eines AHD-basierten Systems an der im System vorhandenen Information, d.h. an ihrer Semantik, Struktur und Verteilung. Unterschieden wird dabei nach den im Informationsmodell des AHDM aufgeführten Kategorien: aktive Bestandteile mit Zustandsinformation, Präsentationsinformation und schließlich Nutzdaten.

Ein wichtiges Ziel bei der Systementwicklung ist es, unabhängige Informationseinheiten und Elemente zu gewinnen, die möglichst flexibel miteinander kombinierbar sind. Der Vorteil bei der Verwendung solcher Elemente zeigt sich bei der vereinfachten Rekombination der Elemente bei geänderten Anforderungen oder als Mittel, um z.B. das Systemverhalten schnell zu modifizieren. Dementsprechend steht in den ersten Schritten der Vorgehensweise die Definition der Systemelemente im Vordergrund, während in den späteren Schritten ihre Verbindungen untereinander und die Verteilungsstrategie wichtig sind. Die Schritte stellen sich wie folgt dar:

1. Strukturierung der Information
2. Gruppierung zusammengehöriger Komponenten
3. Definition der Verteilung der Komponenten
4. Definition der Kommunikationsstrategie
5. Realisierung der Komponenten

Diese übergeordneten Aktivitäten sollen im folgenden weiter präzisiert werden.

Strukturierung der Information

Der erste Schritt beim hier vorgestellten empfohlenen Vorgehen ist an die Entwicklungsmethode der BON angelehnt. Die initiale Strukturierung des Informationsraums beginnt mit der Festlegung der Systemgrenze. Die Systemgrenze umschließt die Komponenten, die in die tatsächliche Implementierung eingehen, ausgenommen sind also solche Komponenten, die zwar Bestandteil der Anforderungsspezifikation sind, aber nicht in das zu realisierende System aufgenommen werden. So liegen z.B. existierende Datenbanken außerhalb der Systemgrenze in der Systemumgebung, ebenso bereits vorhandene Applikationen. In solchen Fällen wird meist eine Schnittstellenkomponente innerhalb der Systemgrenze benötigt, die die Interaktion mit den externen, vorhandenen Bestandteilen einer Applikation ermöglichen.

Im nächsten Schritt sind erste Komponenten zu identifizieren, die in der Implementierung als XML-Elemente oder Dokumente realisiert werden. Eine wichtige Quelle

für solche Komponenten stellt dabei das grundlegende Vokabular des Anwendungsgebiets dar. Die dort identifizierten Konzepte stellen potentielle XML-Elementtypen dar. Hinzu kommen wiederverwendbare Konstrukte, die im Sinne eines *pattern*- oder musterorientierten Vorgehens eingesetzt werden sollen. Dieser Punkt wird an späterer Stelle weiter vertieft.

Die gefundenen Komponenten (sprich einzelne XML-Elemente oder Dokumente) sind daraufhin soweit zu strukturieren, daß für jedes Dokument eine entsprechende Strukturbeschreibung in Form eines Schemas oder einer DTD festgelegt ist. Dies kann durch Kombination von XML-Elementen oder durch Zerlegung von Dokumenten erfolgen. Beide Vorgehensweisen (d.h. Top-Down- und Bottom-Up-Analyse) sind hierbei kombiniert anwendbar.

Zur Strukturierung der Information gehört auch die Definition der funktionalen Schnittstellen der einzelnen Elemente, also die Beschreibung der exportierten AHD-Funktionen und -Variablen.

Gruppierung zusammengehöriger Komponenten

Im nächsten Schritt sind die gefundenen Komponenten in Clustern zu gruppieren. Die Gruppierung von Komponenten charakterisiert dabei eine grundlegende Systemarchitektur, bei der Subsysteme als zusammengehörige Komponenten realisiert werden können. Bei der Einteilung eines Systems in Cluster sollte eine durchgängige Systemsicht gewählt werden, so z.B. die Unterteilung in unterschiedliche Funktionsbereiche wie Kommunikation, Datenhaltung oder die Benutzerschnittstelle oder die Unterteilung in vorhandene, wiederverwendete Komponenten und applikationsspezifische Komponenten (für die Kriterien für die Bildung von Clustern siehe auch [183]).

Wie auch im vorigen Schritt ist es an dieser Stelle sinnvoll, auf wiederverwendbare Konstrukte zurückzugreifen, welche sich in ähnlichen Systemen als Lösungsansatz bewährt haben. Solche wiederverwendbaren Konstrukte auf Systemebene lassen eine erste Kategorisierung des Systems zu. So ist eine Einteilung in einfache, nicht-verteilte Systeme, Systeme aus kooperierenden Komponenten und Systeme aus mobilen Komponenten möglich.

Bestimmung der Verteilung der Komponenten

Bei der Bestimmung der Verteilung der Systemkomponenten sind die identifizierten Cluster auf entsprechende Netzwerkknoten zu verteilen. Hier kann die gefundene Systemkategorie erste Hinweise liefern, zusätzlich kommen Kriterien wie Skalierbarkeit, Flexibilität oder auch Ausfallsicherheit des Gesamtsystems in Frage.

Definition der Kommunikationsstrategie

Der nächste Punkt ist die Festlegung der Kommunikationsstrategie zwischen den aktiven Komponenten. Der im AHDm definierte Methodenaufruf über Dokumentengrenzen hinweg ist eine synchrone, direkt adressierte Kommunikation zwischen zwei Komponenten. Dies ist jedoch nicht in allen Anwendungsfällen ausreichend. Vielmehr sollen auch asynchrone Verbindungen, Multicast-Mechanismen oder die Verwendung von Nachrichten bzw. migrierenden Dokumenten ermöglicht werden. Sie sind über die Mittel zu realisieren, die das AHDm zur Verfügung stellt und als wiederverwendbare Kommunikationsmuster im nächsten Kapitel beschrieben.

Realisierung der Einzelkomponenten

Die in den vorigen Schritten identifizierten Komponenten sind zum Abschluß zu implementieren, d.h. je nach Informationskategorie sind XML-Dokumente zu erstellen, aktiver Code innerhalb der Dokumente zu implementieren oder Style Sheets zu definieren.

17.3 Richtlinien

Bestimmte erwünschte Eigenschaften aktiver Hypertextdokumente sind weniger durch ein schrittweises Vorgehen als durch begleitende Richtlinien erreichbar. Zu diesen Eigenschaften zählen hauptsächlich die Wiederverwendbarkeit, flexible Nutzbarkeit und Skalierbarkeit. Um sie zu erreichen, sollten folgende Richtlinien bei der Systementwicklung beachtet werden:

- Komponenten sollten so entworfen werden, daß ein externes Interface in gesonderten Funktionen und Variablen die tatsächliche Realisierung kapselt. Konkret bedeutet dies beispielsweise die Nutzung des Wurzelements eines Dokuments als Interface-Element, welches eine Reihe von AHD-Funktionen und -Variablen aufweist.
- Innerhalb der aktiven Teile eines AHDs sind direkte Bezüge zu anderen Bestandteilen des Dokuments oder zu anderen Systemkomponenten zu vermeiden, da sie die Wiederverwendung in anderen Kontexten erschweren. Vielmehr sollte auf die Sichtbarkeitsregeln gemäß der parent und remote delegation zurückgegriffen werden, die eine implizite Adressierung ermöglichen.

17.4 Musterorientiertes Vorgehen

Wie bei der Vorstellung des Vorgehensmodells bereits angesprochen ist die Entwicklung wiederverwendbarer Konstrukte ein wichtiges Ziel. Die Entwicklung und Nutzung wiederverwendbarer Konstrukte wird im Software-Engineering im Bereich der *Patterns* (Entwurfsmuster) untersucht (siehe dazu etwa [145] und [38]). Interessant ist dabei, daß Patterns auch in vielen anderen Bereichen eine wichtige Rolle spielen. Dies resultiert aus ihrer einfachen Definition: ein Pattern oder Muster beschreibt die bewährte Lösung eines wiederkehrenden Problems innerhalb eines bestimmten Problemumfelds. Eine solche generische Umschreibung kann in vielen Bereichen genutzt werden, so stammt beispielsweise die Idee für die Verwendung von Patterns ursprünglich aus der Architektur, beschrieben in [1]. Neben dem Einsatz von Patterns für die Erstellung einer Systemarchitektur können Muster auch für Abläufe wie z.B. Entwurfsprozesse oder Entscheidungsfindungsprozesse eingesetzt werden.

18 Konstrukte und Muster

Bei der Nutzung von Mustern stehen zwei Aufgaben im Vordergrund: Einerseits die Identifikation geeigneter Muster, andererseits die Anleitung zur Verwendung der Muster. Für das hier vorgestellte Vorgehensmodell sollen die in den nächsten Abschnitten identifizierten Muster und Konstrukte grundsätzlich nach ihrem Einsatz innerhalb der vorgeschlagenen Entwurfsschritte kategorisiert werden. Damit soll die Auswahl geeigneter Muster vereinfacht werden. Sinnvoll erscheint in diesem Zusammenhang die Einteilung, wie sie in [38] vorgeschlagen wird. Dort wird zwischen

- Architekturmustern,
- Entwurfsmustern und
- Idiomen

unterschieden. Die Bedeutung der einzelnen Kategorien wird in den nächsten Abschnitten genauer definiert, entsprechend der Orientierung am Entwurfsprozeß kommen Architekturmuster vorwiegend beim initialen Systementwurf zum Einsatz, Entwurfsmuster decken Problembereiche unterhalb der Systemebene ab, während Idiome die besonderen Semantiken des AHDMs weiter vertiefen.

Die Definition der Muster erfolgt über drei charakteristische Punkte: der Kontext des Musters, das darin auftretende Problem und die vorgeschlagene Lösung. Bei der Vorstellung der Muster soll insbesondere auch untersucht werden, inwieweit bereits bekannte Muster mit den Mitteln des AHDM realisiert werden können oder ob sich sogar vereinfachte Umsetzungen ergeben. Falls die vorgestellten Muster bereits in anderen Kontexten beschrieben und eingesetzt werden, so soll darauf hingewiesen werden, da gerade die Nutzung von Muster-Katalogen die konsistente Verwendung von Mustern erleichtert. Die Patterns sollen mit englischsprachlichen Namen versehen werden, um die Verwendung in der Dokumentation und im Quelltext eines Programmes zu vereinfachen.

In dieser Arbeit sollen vornehmlich AHDM-typische Muster vorgestellt werden. Es kann zwar kein vollständiger Pattern-Katalog für AHD-basierte Anwendungen erstellt werden, dennoch sollen die typischsten Muster in solchen Anwendungen beschrieben werden. Daneben sollen die in [38] und [145] beschriebenen Muster zum Vergleich herangezogen werden. Ist das Muster noch nicht in anderen Quellen aufgeführt oder weist es AHDM-spezifische Charakteristika auf, so kommt zur Beschreibung von eine Übersicht über die Struktur der Musters und eine Beschreibung des Systemverhaltens.

18.1 Architekturmuster

Zu den Architekturmustern werden solche Konstrukte gezählt, die die grundlegende Systemarchitektur der Anwendung bestimmen. Demzufolge werden sie vornehmlich bei der Auswahl einer Systemarchitektur angewandt, bzw. eine gefundene Systemarchitektur wird mit ihrer Hilfe beschrieben. Zu beachten ist, daß mit der Wahl einer zugrunde liegenden Architektur oft auch die Kommunikations- und die Verteilungsstrategie innerhalb eines System bestimmt wird, somit ein Vorgriff auf spätere Entwurfsphasen existiert.

Standalone Active Document

Eine der einfachsten Anwendungen für aktive Dokumente sind eigenständige Applikationen, bestehend aus einem einzigen, nicht weiter vernetzten Dokument. Der Nutzen dieses Musters liegt weniger in der konkreten Anleitung zur Lösung eines gegebenen Problems als vielmehr in der Dokumentation einer Systemkonfiguration.

Kontext

Die Nutzung eines eigenständigen Dokuments als Anwendung leitet sich häufig aus einem einfachen Anforderungskatalog ab, in dem komplexere, verteilte Anwendungen keinen Sinn machen. Weiter ist es auch denkbar, daß es nicht möglich ist, Netz-

werkverbindungen aufzubauen, und damit weder entfernte Methodenaufrufe noch das Einbinden entfernter Bestandteile genutzt werden können.

Problem

Neben geringen Anforderungen an die zu erstellende Applikation soll dieses Muster auch die fehlende Möglichkeit zur Vernetzung dokumentieren.

Lösung

Bei diesem sehr einfachen Muster stellen sich sowohl Struktur als auch Verhalten als wenig komplex dar. Durch die fehlende Vernetzungsmöglichkeit sind die unterschiedlichen Informationsarten in einem Dokument zu vereinen.

Struktur

Abbildung 53 zeigt die daraus resultierende Struktur des *Standalone Active Document*. Eine solche eigenständige Anwendung enthält als Informationsarten sowohl Präsentationsinformation (üblicherweise in Form eines STYLE-Elements), Zustandsdaten (var-Element aus dem AHDM), Verhaltensbeschreibungen (func-Elemente) und Nutzdaten. Es exportiert nur solche Funktionen, die auch durch die Laufzeitumgebung durch Ereignisse aus dem IEM aktiviert werden können.

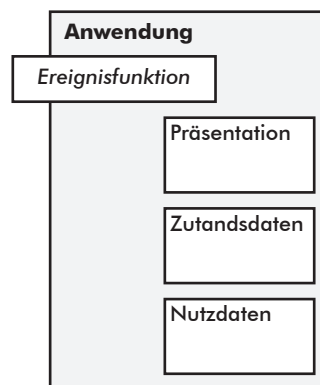


Abb. 53: Eigenständige Anwendung realisiert durch ein Standalone Active Document

Verhalten

Das Verhalten einer eigenständigen Anwendung ist vollkommen durch die im Dokument eingebetteten func-Elemente bestimmt. Aktivität wird durch Aufrufen der exportierten Funktionen etwa bei der Reaktion auf Benutzerereignisse ausgelöst.

Networked Active Documents

Weitaus häufiger als eigenständige Anwendungen sind Systeme bestehend aus vernetzten Komponenten. Durch die Vernetzung von Komponenten können mehrere Ziele verfolgt werden: Trennung der Informationsarten, Mehrfachverwendung von Komponenten oder auch Nutzung externer, bereits vorhandener fremder Ressourcen.

Kontext

Der Kontext für den Einsatz kooperativer Anwendungen ist sehr weit gefaßt. Grundsätzlich fallen alle Anwendungen darunter, die die Verknüpfungsmechanismen per XLink nutzen. Damit sind auch solche Verknüpfungen eingeschlossen, die entfernte Methodenaufrufe nutzen oder bei denen Teile der Anwendung von externen Kompo-

nenten über Methodenaufrufe genutzt werden können, vergleichbar mit dem Konzept eines Applikationsservers, welcher Anwendungsfunktionalität netzbasiert zur Verfügung stellt.

Problem

Die Problemstellungen, die den Einsatz kooperativer Anwendungen erfordern, umfassen neben einfachen Vernetzungsaufgaben (z.B. um externe DTDs einzubinden) den gesamten Bereich verteilter Programmierung und Informationsdistribution. So können einerseits Informationen auf unterschiedlichen Netzknoten zur Verfügung stehen, etwa in Form von bestehenden Web-Angeboten, oder es ist andererseits wünschenswert, auch Rechenleistung zu verteilen. Dabei sind die unterschiedlichen Komponenten einer kooperativen Anwendung als ortsfest anzusehen.

Lösung

Wie vorher bereits erwähnt, ist die Systemarchitektur einer kooperativen Anwendung in weiten Grenzen realisierbar. Wesentlich für die Architektur sind nur zwei Charakteristika: die Komponenten sind ortsfest, und die Verknüpfung zwischen diesen Komponenten dient zum Austausch reiner Information oder zum Aufruf entfernter Methoden. Im hier vorgestellten Muster der *Networked Active Documents* werden die Komponenten durch aktive Hypertextdokumente realisiert.

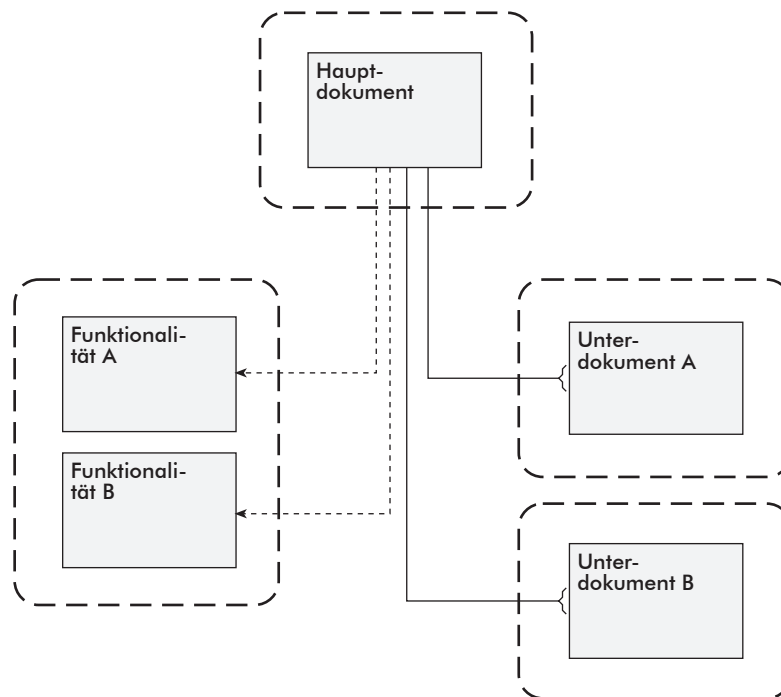


Abb. 54: Kooperative Anwendung aus vernetzten Dokumenten

Struktur

Die Struktur einer kooperativen Anwendung basierend auf vernetzten AHDs besteht aus einem Hauptdokument und einer Anzahl referenzierter Unterdokumente oder Datenquellen, die per XLink eingebunden werden oder über die AHDM-Methoden get, put, load und store angesprochen werden (siehe Abbildung 54).

Verhalten

Das Verhalten ist durch die Programmfragmente in den einzelnen Dokumenten oder externen Komponenten bestimmt, die Kommunikation erfolgt dabei entsprechend der AHDM-Beschreibung synchron.

Mobile Active Documents

Eine Variante der vernetzten Systeme stellen die agentenähnliche Systeme dar. Dort werden nicht nur reine Nutzdaten zwischen den Systemkomponenten ausgetauscht, sondern es werden migrierende, aktive Dokumente als eigenständige Agenten eingesetzt. Beispiele dafür sind u.a. Applikationen zum Sammeln von Informationen, zur temporären Kooperation zur Lösung einer Aufgabe an einem bestimmten Ort oder die ortsunabhängige Nutzung von Komponenten.

Kontext

Mobile AHD-basierte Systeme sind in solchen Konstellationen zu finden, in denen kooperative Anwendungen eingesetzt werden können, bei denen aber die einfachen, synchronen Kommunikationsmechanismen im AHDM nicht ausreichen.

Problem

Bei der Realisierung einer kooperativen, vernetzten Anwendung mit mehreren verteilten Komponenten kann es nötig sein, auch asynchrone Kommunikation einzusetzen, etwa wenn die Netzwerkverbindungen zwischen einzelnen Komponenten nicht immer besteht. Auch ist es denkbar, daß die Fixierung eines physikalischen Standorts für eine Komponente nicht möglich oder sinnvoll ist.

Lösung

Um asynchrone Kommunikation zu ermöglichen oder mobile Komponenten zu implementieren, muß eine Anwendung zusätzliche Applikationslogik gegenüber einfachen kooperativen Anwendung besitzen, um die ausgetauschten Daten zu verarbeiten. Die Applikationslogik soll hier nach dem vorgestellten Muster der *Mobile Active Documents* in die übertragenen Dokumente eingebunden werden, die so zu eigenständigen Agenten werden. Hierbei handelt es sich u.a. um Anwendungen des *Remote Programming*, d.h. der Übermittlung aktiver Komponenten an einen entfernten Ort, um sie dort zur Ausführung zu bringen.

Struktur

Ein System, welches migrierende aktive Dokumente benutzt, stützt sich im wesentlichen auf stationäre und mobile Bestandteile. Die mobilen Bestandteile sind dabei als eigenständige Applikationen entsprechend dem vorgenannten Architekturmuster realisiert. Sie bestehen zumindest aus Nutzdaten, aktiven Bestandteilen und Zustandsinformation. Neben den diesen mobilen Agenten existieren stationäre Anwendungen, zwischen denen die Agenten migrieren.

Verhalten

Hinzu kommen Mechanismen zur Kommunikation zwischen mobilen und stationären Bestandteilen und erweiterte Netzwerkfunktionalität für die Mobilität. In Abbildung 55 bewegt sich beispielsweise ein Agent aus einer Anwendung heraus in eine entfernte Laufzeitumgebung, kontaktiert dort eine zweite Anwendung und führt Operationen durch und migriert schließlich zurück zur Ausgangsanwendung.

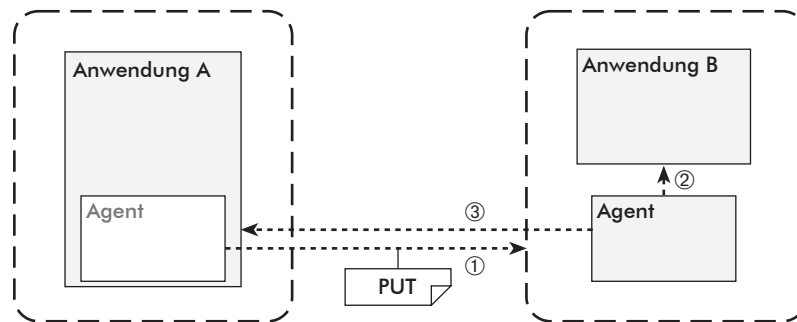


Abb. 55: Mobile Aktive Dokumente

Layered Active Documents

Eng verwandt mit den kooperativen Anwendungen sind die Mehrschichtensysteme. Dabei geht es vornehmlich um die Aufteilung einer einzelnen Anwendung in Schichten gleicher Funktionalität und die Verteilung der einzelnen Schichten auf unterschiedliche Netzknoten. Das Architekturmuster der Mehrschichtensysteme ist u.a. in [38] unter dem Namen *Layers* beschrieben.

Kontext

Mehrschichtensysteme können überall dort eingesetzt werden, wo Netzknoten unterschiedliche Rollen innerhalb einer Applikation spielen sollen. Eine typische Aufteilung trennt beispielsweise die Darstellungskomponente, die Applikationslogik und die Datenhaltung voneinander.

Problem

Mehrschichtensysteme sollen unter anderem helfen, ein System skalierbar zu halten, so daß ressourcenintensive Schichten wie z.B. die Applikationslogik oder die Datenhaltung in eigenen Netzknoten realisiert werden und bei Bedarf unabhängig vom Rest des Systems modifiziert oder optimiert werden können. Auch spielt die Wiederverwendbarkeit einzelner Schichten eine große Rolle, so daß beispielsweise ein Knoten dezidiert die Datenhaltung für mehrere Applikationen realisiert.

Lösung, Struktur und Verhalten

Mehrschichtensysteme in Form von *Layered Active Documents* werden durch die statische Verknüpfung der einzelnen Schichten realisiert, d.h. die Verbindungen zwischen den Schichten sind permanenter Natur und durch XLinks realisiert. Eine verbreitete Aufteilung unterteilt ein Mehrschichtensystem in die Darstellungskomponente, die Anbindung der Darstellungskomponente an die Applikationslogik, die Applikationslogik, den Datenhaltungszugriff, die Datenhaltung und die Netzwerkfunktionalität (vgl. Abbildung 56). Das Systemverhalten ist dort größtenteils durch die mittlere Schicht bestimmt.

18.2 Entwurfsmuster

Neben den Architekturmustern, die die grundlegende Struktur eines Systems erfassen, stehen mit Entwurfsmustern Konstrukte zur Verfügung, die innerhalb der Systemarchitektur einzelne Komponenten realisieren können.

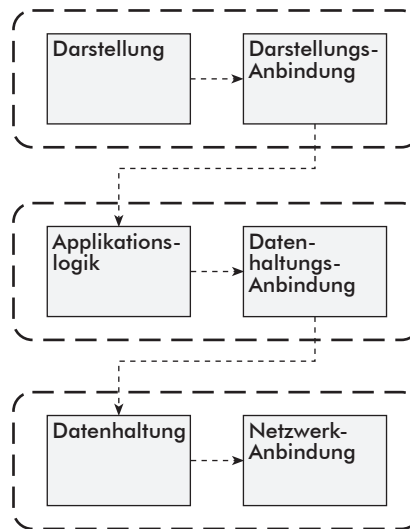


Abb. 56: Layered Active Documents

Die Zahl nutzbarer Entwurfsmuster ist praktisch nicht begrenzt, Beispiele sind innerhalb von Muster-Katalogen in großer Anzahl zu finden. An dieser Stelle sollen deshalb nur wenige Muster vorgestellt werden, die aber die Charakteristika des AHDMs unterstreichen, so etwa die Nutzung bestimmter Eigenschaften aktiver Hypertextdokumente, um Muster vereinfacht realisieren zu können.

Template Document

Ein generelles Anliegen bei der Software-Entwicklung ist die Wiederverwendung bestehender Systembestandteile. So existieren mehrere Ansätze, wie etwa Vererbung oder Funktionsbibliotheken. Im AHDM soll das Konzept des *Template Document* (Vorlagendokument) genutzt werden, um existierende Bausteine wiederverwenden zu können. Es vergleichbar mit dem *Prototype*-Muster aus [145].

Kontext

Die Verwendung von Vorlagendokumenten bietet sich insbesondere in solchen Kontexten an, in denen schon Bausteine zur Verfügung stehen, die z.B. aus anderen Systemen stammen, oder bei solchen Systemen, bei denen gemeinsame Strukturen oder gemeinsames Verhalten ausfaktoriert werden soll.

Problem

Ein Problem, welches bei der Wiederverwendung von Systembestandteilen auftreten kann, ist die Lokalisierung dieser Bestandteile. Problematisch ist insbesondere der Fall, wenn die Komponenten zugreifbar sein müssen, um sie wiederzuverwenden, etwa beim dynamischen Verbinden von Programm und wiederverwendeten Funktionsbibliotheken.

Lösung

Bei der Nutzung von Vorlagendokumenten entfällt die Notwendigkeit der Erreichbarkeit der wiederzuverwendenden Komponenten. Vielmehr werden sie zur Verwendung komplett kopiert. Die Nutzung solcher Dokumente kann im AHDM dadurch erfolgen, daß eine Anzahl von Vorlagen an einer zentralen Stelle vorgehal-

ten wird, welche durch die get- oder load-Funktion angefordert werden können. Dadurch ist am Verwendungsort eine Kopie des Prototypdokuments verfügbar. Es sind keine weiteren Mechanismen außer den beiden oben angesprochenen Funktionen notwendig, da durch das AHDM gewährleistet ist, daß eine textuelle Repräsentation, wie sie bei der Übertragung verwendet wird, ein komplettes Abbild der angeforderten Ressource ist.

Struktur

Die Nutzung eines Template Document erfolgt üblicherweise durch Referenzierung eines Dokuments durch eine Anwendung über die AHD-Funktionen get oder load, die eine Kopie eines entfernten Dokuments anfordern (siehe z.B. Abbildung 57). Neue Vorlagen können in ein System dann entweder durch Austausch bestehender Vorlagendokumente an einer bestimmten Adresse oder durch Änderung der Referenzierung eingebracht werden.

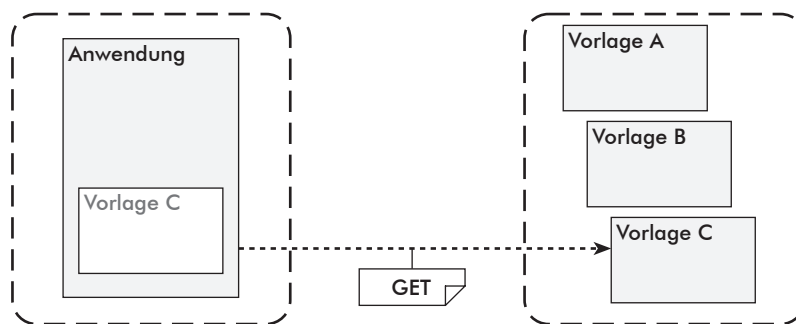


Abb. 57: Struktur des Prototyp-Musters

Verhalten

Die Nutzung eines Vorlagendokuments erfolgt wie erwähnt durch Anforderung mittels get oder load, das weitere Verhalten ist dann von der Implementierung des Dokuments abhängig, z.B. ist eine zusätzliche Initialisierung durch Behandlung des onload-Ereignisses möglich.

Delegation Chain

Die Delegationsmechanismen im AHDM lassen sich direkt für ein Entwurfsmuster nutzen, welches die Modularisierung und lose Kopplung zwischen Software-Komponenten fördert. Das hier mit dem Namen *Delegation Chain* (Delegationsskette) bezeichnete Muster ähnelt der *Chain of Responsibility* aus [145], die Unterschiede werden weiter unten erläutert.

Kontext

Eine Delegationsskette kann dort genutzt werden, wenn der Empfänger bei der Behandlung von Ereignissen oder der Bearbeitung von Funktionsaufrufen in der aufrufenden oder auslösenden Instanz nicht näher benannt werden soll. Zudem kann es angebracht sein, die Behandlung eines Ereignisses oder die Ausführung einer Funktion in mehreren Schritten durchzuführen.

Problem

Die Problematik bei einer Delegationskette liegt in der Bestimmung und Adressierung der behandelnden Instanzen.

Lösung

Im AHDM wird das Konzept der Delegationskette direkt durch die parent und remote delegation unterstützt. So werden einerseits IEM-Ereignisse vom auslösenden Element durch die Elementhierarchie und gegebenenfalls an ein entferntes Dokument zur Behandlung durchgereicht. Andererseits wird dieses Prinzip auch beim Auffinden von Funktionen beim Funktionsaufruf angewandt. Dadurch ergibt sich eine Kette von Instanzen, welche einen Funktionsaufruf oder ein Ereignis entgegennehmen können. Es ist keine direkte Adressierung notwendig, vielmehr werden Aufrufe und Ereignisse automatisch an das übergeordnete Element bzw. an ein entferntes Element weitergeleitet.

Der Unterschied zur Chain of Responsibility (Verantwortlichkeitskette) liegt in zwei Punkten: Die Delegationsmechanismen sind weniger allgemein als die Mechanismen der Verantwortlichkeitskette, d.h. eine Delegationskette läßt sich im Falle der parent delegation nur durch Modifikation der Elementhierarchie im Dokument ändern. Auch sind die behandelten Ereignisse bei der Delegationskette nur Funktionsaufrufe im Sinne des AHDM.

Struktur

Ein Beispiel für die Struktur einer Delegationskette zeigt Abbildung 58. Wird innerhalb von Funktion f1 eine Funktion f4 aufgerufen, so kann diese innerhalb des direkt umschließenden Elements (in der Abbildung mit 1 gekennzeichnet) gefunden und somit ausgeführt werden. Wird sie, wie im Beispiel gezeigt, nicht im direkt umschließenden Element gefunden, so wird die Suche über alle Elemente bis zum Wurzelement und in eventuell per delegate-Link referenzierten Links fortgesetzt. Jedes Element auf diesem Suchpfad kann die gesuchte Funktion realisieren, genauso wie alle per delegate-Link referenzierten Dokumente.

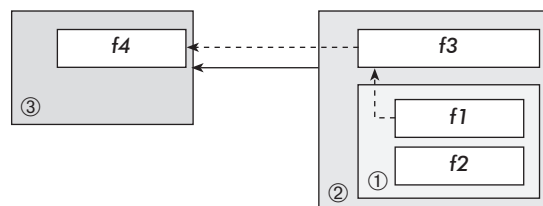


Abb. 58: Beispiel für eine Delegationskette

Verhalten

Die Nutzung der Delegationsmechanismen erfolgt direkt bei der Anwendung der AHD-Funktionen get, put und call. Die AHD-Laufzeitumgebung realisiert das bei der Delegationskette notwendige Weiterleiten der Funktionsaufrufe.

Delegation Strategy

In dynamischen Systemen kann unter anderem auch die Änderung des Systemverhaltens oder des Verhaltens einzelner Komponenten gefordert sein. Auch hierfür lassen sich die Delegationsmechanismen des AHDM nutzen, insbesondere die

Delegationsmechanismen. Das dazu nutzbare Muster ist unter dem Namen *Strategy* bekannt [145] und kann direkt mit Mitteln des AHDM umgesetzt werden.

Kontext

Strategieänderungen sind dann sinnvoll, wenn das Verhalten des Systems oder einer Komponente in Grundzügen spezifiziert werden kann, spezielle Abläufe aber dynamisch ausgewählt werden.

Problem

Innerhalb der Verhaltensbeschreibungen muß es möglich sein, sich implizit auf ausführende Instanzen zu beziehen, weiterhin muß es gewährleistet sein, daß die Adressierung dieser ausführenden Instanzen einfach geändert werden kann.

Lösung, Struktur und Verhalten

Generell wird beim Strategiemuster ein Algorithmus oder eine Funktionalität indirekt aufgerufen und eine Strategieänderung über die Änderung der Aufrufreferenz erreicht. Damit ergeben sich als beteiligte Komponenten die aufrufende Instanz, die Referenz auf eine konkrete behandelnde Instanz und eine Menge von alternativ aufrufbaren Instanzen.

Der Mechanismus der remote delegation erlaubt es auf einfache Art und Weise, die behandelnde Instanz durch Veränderung des delegate-Links auszutauschen. Es ist auch denkbar, die Elementhierarchie zu ändern und das aufrufende Element in einen anderen Aufrufkontext zu plazieren oder neue Elemente in der Hierarchie einzufügen. Zudem können durch die Delegationsmechanismen die Funktionsaufrufe, die spezielles Verhalten realisieren, indirekt adressiert werden.

Document Factory

Eine besondere Anwendung des Prototyp-Musters und des Strategie-Musters ist die *Document Factory* (Fabrik, auch unter dem Namen *Factory* bekannt), welche es zuläßt, kontextabhängig unterschiedliche Instanzen eines Dokuments, einer Elementhierarchie oder anderer XML-Daten zu erzeugen.

Kontext

Eine Fabrik kann immer dann genutzt werden, wenn mehrere Instanzen eines Dokuments nach einem bestimmten Muster erzeugt werden sollen und die Erzeugung konfigurierbar sein soll. In etwa entspricht dies der Erzeugung von Objekten unterschiedlicher Klassen in objektorientierten Programmiersprachen.

Problem

Wie auch beim Strategie-Muster bestehen die Anforderungen in einer einfachen Austauschbarkeit der Funktionen, welche für die Erzeugung von Instanzen benutzt werden und in einer impliziten Adressierung dieser Funktionen, d.h. einer möglichst losen Kopplung zwischen aufrufendem Dokument und behandelndem Element.

Lösung, Struktur und Verhalten

Die Austauschbarkeit und implizite Adressierung erfolgt wie beim Strategie-Muster durch Nutzung der Delegationsmechanismen. Die erzeugenden Funktionen können so beispielsweise in mehreren entfernten Dokumenten liegen, wobei je nach delegate-Link unterschiedliche Funktionen aufgerufen werden und gemäß dem Prototyp-

Muster eine Dokument-Instanz zurückliefern. Die Struktur eines Fabrik-Musters ist äquivalent zur Struktur des Strategie-Musters, d.h. es existiert eine aufrufende Instanz, eine Anzahl von Funktionen, die zur Erzeugung von Komponenten aufgerufen werden können und eine Verbindung dazwischen. Die Auswahl der zu erzeugenden Komponenten geschieht über die Veränderung dieser Verbindung, die sich z.b. über remote delegation realisieren läßt. Analog zum Verhalten im Strategie-Muster stellen sich auch die Abläufe im Fabrik-Muster dar.

Document Facade

Ein wichtiges Muster zur Förderung der Kapselung von aktiven Dokumenten ist die *Document Facade* (Dokumenten-Fassade), eine Spezialisierung des *Facade*-Musters aus [145]. Im AHDM wird sie eingesetzt, um das Interface zu einem Dokument zu realisieren.

Kontext

Eine Fassade für ein aktives Dokument ist insbesondere dann wichtig, wenn es von anderen Dokumenten benutzt werden und weiterhin die tatsächliche Implementierung nicht offengelegt werden soll, da sie sich beispielsweise gemäß des Strategie-Musters oder der Delegationskette ändern kann.

Problem

Um eine sinnvolle Kapselung zu erreichen, muß definiert werden, wie auf die Fassade generell zugegriffen werden kann, ohne daß im aufrufenden Dokument weiter Informationen über den Aufbau des aufgerufenen Dokuments nötig sind.

Lösung

Die naheliegendste Lösung besteht darin, das Wurzelement eines aktiven Dokuments als Fassade für die darunterliegenden Elemente zu nutzen. Somit ergeben sich zwei Vorteile: Einerseits ist von außen immer das Wurzelement als Fassade anzusehen, andererseits können auch die aktiven Elemente innerhalb eines Dokuments die Fassade über parent delegation nutzen. Somit stellt das Wurzelement eine koordinierende Instanz für Funktionsaufrufe dar.

Struktur

Die Struktur einer Dokumenten-Fassade ist beispielhaft in Abbildung 59 dargestellt. Eine Dokumenten-Fassade besteht demnach aus einem Dokument und einer Reihe von Elementen, die direkte Nachfolger des Wurzelements im Dokument sind. Diese Elemente stammen aus dem AHDM-Namensraum, d.h. es handelt sich um func- und var-Elemente.

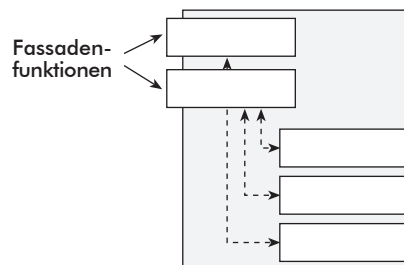


Abb. 59: Fassaden-Muster

Verhalten

Die Fassaden-Elemente dienen einerseits als Schnittstelle für externe Aufrufe, andererseits können über parent delegation auch Verhaltensbeschreibungen innerhalb des Dokuments auf diese Elemente zugreifen. Ein Aufruf über ein Fassaden-Element wird durch das Element an das eigentliche Zielelement weitergeleitet, welches z.B. konkret über einen festen URI adressiert wird.

Proxy Document

Da das hier vorgestellte Modell insbesondere zur Erstellung von vernetzten Systemen eingesetzt werden soll und diese Systeme auch mobile Komponenten haben können, kann ein Proxy-Dokument herangezogen werden, um mobile Komponenten über eine stationäre Instanz anzusprechen. Generell kann eine Proxy eine transparente, indirekte Verbindung zwischen Komponenten herstellen. Das Proxy-Muster hat zahlreiche Anwendungen, weshalb es in vielen Muster-Katalogen enthalten ist, so auch in [38] und [145].

Kontext

Ein Proxy kann immer dann eingesetzt werden, wenn die genaue Adressierung einer Komponenten für die Nutzer nicht bekannt sein soll. Stattdessen wird mit dem Proxy ein Stellvertreter benutzt, welcher die Anfragen und Funktionsaufrufe weiterleitet.

Problem

Um einen Proxy zu realisieren, muß gewährleistet sein, daß sowohl der Proxy als auch die durch den Proxy repräsentierte Komponente das gleiche Interface haben. Zudem muß es möglich sein, die Funktionsaufrufe und Anfragen transparent weiterzuleiten.

Lösung, Struktur und Verhalten

Das Proxy-Muster besteht aus einer Client-Komponente, dem eigentlichen Proxy und einer Originalkomponente (Abbildung 60). Um eine transparente Nutzung zu ermöglichen, müssen Proxy und Original die gleiche Schnittstelle (Fassade) besitzen. In objektorientierten Programmiersprachen kann dies über eine gemeinsame Oberklasse realisiert werden, eine Möglichkeit, die im AHDM nicht vorhanden ist. Deswegen muß bei der Implementierung eines Proxies das Interface manuell angepaßt werden.

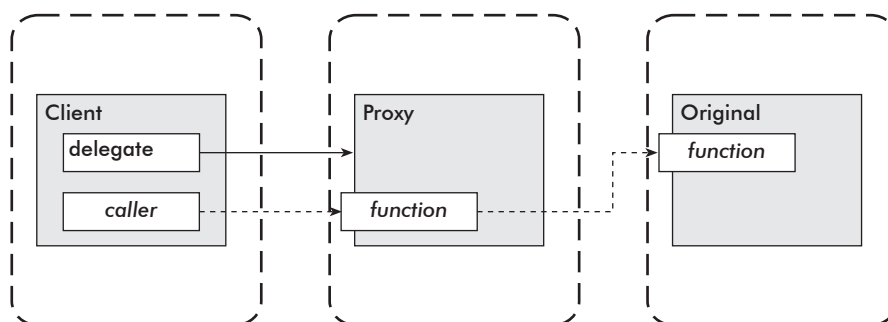


Abb. 60: Proxy-Muster

Ein AHD kann als Proxy fungieren, indem ein delegate-Link im Wurzel-Element eingefügt wird. Dieses referenziert das Dokument, welches die eigentliche Funktio-

nalität realisiert. Werden Funktionen über den Proxy aufgerufen und sind sie nicht innerhalb des Proxies realisiert, so erfolgt eine automatische Weiterleitung über den delegate-Link. Es ist aber auch möglich, Funktionen im Proxy zu realisieren und nur bei Bedarf weiterzuleiten, um eine Filterfunktionalität zu erreichen. Der Client nutzt das Original in dieser Konstellation transparent über den Proxy, d.h. der Client muß nicht modifiziert werden, um den Proxy zu nutzen. Dies ist allerdings keine Voraussetzung des Proxy-Musters.

18.3 Idiome

Auf der niedrigsten Abstraktionsebene der Muster befinden sich Idiome, die nur bei der Realisierung aktiver Hypertextdokumente eine Rolle spielen. Sie definieren hauptsächlich solche Konstrukte, wie sie beim Einsatz des AHDMs nützlich sind. Dazu zählen beispielsweise auch solche Muster, die die im vorigen Abschnitt identifizierten Semantiken in wiederverwendbarer Form beschreiben.

Role Identifier Attribute

Die Semantik der einzelnen Elemente innerhalb eines XML-Dokuments kann auf unterschiedliche Weise verdeutlicht werden. Einerseits kann der Name des Elements eine Bedeutung transportieren, andererseits können Elemente auch durch einen Namensraum identifiziert werden, in dem sie definiert werden. Eine aufwendigere Möglichkeit besteht im Einsatz zusätzlicher Beschreibungsmittel wie RDF. Mit dem Konzept des *Role Identifier Attribute* (kurz Rollen-Attribut) soll hier ein alternativer, einfacher Mechanismus vorgestellt werden.

Kontext

Die nähere Beschreibung der Rollen von XML-Elementen soll insbesondere in aktiven Dokumenten zum Einsatz kommen, in denen die aktiven Bestandteile solche Elemente abhängig von der Rolle abfragen oder modifizieren.

Problem

Grundsätzlich existieren nur wenige, oben bereits angesprochene Mechanismen zur Beschreibung der Semantik (oder auch Rolle) eines Elements. Gerade der Name eines Elements oder auch der Namensraum können nur jeweils eine mögliche Bedeutung beschreiben, während RDF einen gewissen Aufwand zur Beschreibung erfordert. Es ist also ein einfacher Mechanismus zur Annotation von Elementen mit Metadaten gesucht.

Lösung

Die Lösung besteht hier in der Verwendung eines Attributes aus dem AHD-Namensraums, welches wie auch die Attribute aus dem XLink-Namensraum beliebigen Elementen zugefügt werden kann. Dieses Vorgehen wird unter anderem auch im HyTime-Standard (vgl. [85] und [131]) genutzt, um sog. Architectural Forms an ein Element zu binden und die Semantik dieser Elemente in Dokumenten mit unterschiedlichen DTDs näher zu beschreiben.

Struktur

Abbildung 61 zeigt die Definition des Role Identifier Attribute in einem Parameter-Entity und seine Verwendung in einer Attributliste-Deklaration. Es kann analog an anderen Elementen hinzugefügt werden.

```
<!ENTITY % role.att
    "role    CDATA    #IMPLIED">

<!ATTLIST element
    %role.att; >
```

Abb. 61: role-Attribut

Das Attribut kann beliebige Werte annehmen und soll die Bedeutung des zugehörigen Elements für die Ausführung der aktiven Bestandteile eines AHDs festlegen. Ein Beispiel dafür ist das in Kapitel 21 vorgestellte Eingabefeld, bei dem drei Elemente (Titel, Wert und Eingabefeld) in einem Dokument über Attribute verknüpft werden, um den Ablauf beim Editieren durch den Benutzer zu steuern. Die jeweiligen Werte der Attribute wechseln je nach Bearbeitungszustand des Eingabefelds.

Der Vorteil bei der Verwendung von Attributen für die nähere Bestimmung der Rolle eines Elements liegt in der Flexibilität. Attribute können mittels XML-Namensräumen einfach zu Elementen hinzugefügt werden, ohne daß die Validität eines Dokuments dadurch verloren geht, während beim Hinzufügen von neuen Elementen nur noch eine unscharfe Validierung möglich ist. Zudem tragen Attribute nicht zum eigentlichen Inhalt eines Dokuments bei und können so bei Bedarf ignoriert werden.

Verhalten

Das Verhalten eines AHDs kann direkt von Rollen-Attributen abhängig gemacht werden, indem die Werte der Attribute in den Programmfragmenten abgefragt werden. Daneben kann auch eine Verarbeitung durch externe Programme durch die Attributwerte gesteuert werden, z.B. bei der Konvertierung in andere Zielformate.

Reference

Die Verwendung von URLs als spezielle URIs zur Referenzierung von Ressourcen, Dokumenten oder Elementen wurde bereits in Unterkapitel 13.8 angesprochen. An dieser Stelle soll auf mögliche Lösungen für entstehende Probleme eingegangen werden.

Kontext

URL-Referenzen auf andere Ressourcen werden in allen Anwendungen benötigt, die nicht eigenständig laufen, also in verteilten, kooperativen oder agentenbasierten Anwendungen. Weiterhin können bereits bestehende Web-Ressourcen Referenzen in Form von URLs enthalten.

Problem

Die Bestandteile einer URL enthalten vornehmlich Informationen, die Implementierungsmechanismen offenlegen, beispielsweise den Rechnernamen, die Portnummer oder Verzeichnisstrukturen. Eine solche Adressierung ist ortsgebunden, so daß bei einer Verschiebung der Resource die URL in allen referenzierenden Instanzen verändert werden muß.

Lösung und Struktur

Als Lösung bieten sich URNs an, welche eine Resource über einen eindeutigen Namen identifizieren. Mittels eines URN ist es möglich, ein getrenntes Schema zur Identifikation zu nutzen. Im AHDM könnte dies beispielsweise ein AHD-Namensraum sein, welcher aktive Hypertextdokumente über ein gemeinsames Schema-Präfix identifiziert. Abbildung 62 zeigt einen dementsprechenden URN in einem einfachen XLink.

```
<delegate xlink:type="simple" actuate="auto" show="parsed"
  href="urn:AHDS:SWT/utilities.ahd">
```

Abb. 62: Verwendung eines URN

Verhalten

Um solche eindeutigen Namen aufzulösen, ist allerdings ein Verzeichnisdienst notwendig. Es bietet sich z.B. eine LDAP-basierte Lösung an, wenn die AHD-Laufzeitumgebungen den Charakter persönlicher Repositories haben und schon LDAP-Server für die Verwaltung von Personendaten genutzt werden. Die Auflösung der URNs ist Aufgabe der Laufzeitumgebungen, welche die eindeutigen Namen in URLs umsetzen müssen.

Runtime Environment Monitor

Viele Anwendungen erfordern Interaktion zwischen aktiven Dokumenten innerhalb einer Laufzeitumgebung. Um die Interaktion zu vereinfachen, läßt sich in einer Laufzeitumgebung ein *Runtime Environment Monitor* in Form eines Monitordokuments einsetzen, welches die Kommunikation koordiniert. Es ist eine spezielle Anwendung des Musters *Client-Dispatcher-Server* aus [38] oder des *Mediators* aus [145], die die allgemeine Aufgabe des Vermittelns von Funktionsaufrufen zwischen Systemkomponenten realisieren.

Kontext

Ein Monitordokument ist in solchen Anwendungen nutzbar, bei denen Kommunikation zwischen den Dokumenten innerhalb einer Laufzeitumgebung stattfindet.

Problem

Insbesondere die indirekte Adressierung von mehreren Dokumenten bei Zustandswechseln eines Dokuments erfordert eine Koordinierungsinstanz. Zu solchen Zustandswechseln zählen insbesondere die Aktivierung und Deaktivierung eines Dokuments, etwa beim Verlassen einer Laufzeitumgebung per unload oder bei Eintritt per load oder store.

Lösung

Grundsätzlich wäre es möglich, einen uniformen Benachrichtigungsdienst für on-load- und onunload-Ereignisse in die Laufzeitumgebung selbst zu integrieren. Allerdings sind die Anforderung nicht in allen Anwendungsfällen ausreichend homogen, um sie mittels eines einzigen Mechanismus zu erfüllen. So können beispielsweise Filter notwendig sein, um nur bestimmte Dokumente zu adressieren, oder es sollen zusätzliche Parameter bei der Benachrichtigung übergeben werden. Deswegen wird hier eine Basislösung skizziert, welche mit aktiven Dokumenten realisiert ist und

entsprechend den Anforderungen angepaßt werden kann. Dazu wird ein Vermittler oder Monitor zwischen den aktiven Dokumenten eingesetzt.

Struktur

Die zugrundeliegende Struktur zeigt Abbildung 63. Am Muster des Runtime Environment Monitor sind drei Gruppen von Komponenten beteiligt: Die Gruppe allgemeiner Dokumente (in der Abbildung mit *Subject* bezeichnet), deren Aktivierung und Deaktivierung überwacht werden soll, die Gruppe der dabei zu benachrichtigenden Dokumente (*Observers*) und schließlich das Monitor-Dokument (*Monitor*), welches die Kommunikation zwischen den Gruppen realisiert.

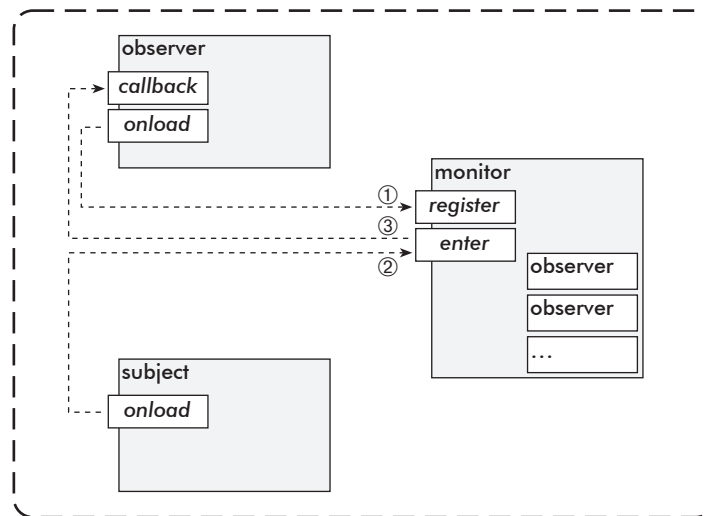


Abb. 63: Laufzeitumgebungs-Monitor

Verhalten

Im ersten Schritt registriert ein Observer-Dokument bei seiner Aktivierung die Funktion *callback* beim Monitor-Dokument über die *register*-Funktion. Wird nun ein Dokument in der Laufzeitumgebung aktiv, so benachrichtigt es das Monitor-Dokument über die *enter*-Funktion. Diese Funktion ruft in allen registrierten Dokumenten die entsprechende Rückruffunktion auf. In ähnlicher Art und Weise kann eine Implementierung für die Behandlung von *unload*-Ereignissen umgesetzt werden.

Bei einer solchen Realisierung ist zu beachten, daß die zu beobachtenden Dokumente sowohl die Adresse des Monitor-Dokuments kennen müssen, als auch einen expliziten Funktionsaufruf bei der Aktivierung durchzuführen haben.

Structure-based Multicast

Wie bei der Beschreibung des Kommunikationsmodells in Unterkapitel 13.7 bereits erwähnt, läßt sich mit Hilfe des Struktur-Matching eine Art Broadcast-Funktionsaufruf implementieren. Dies soll hier im Broadcast-Muster angelehnt an die Tupel-Räume von etwa *Linda* [90] genutzt werden.

Kontext

So wie ein Monitor-Dokument indirekte Kommunikation zwischen mehreren Dokumenten vereinfacht, kann es nötig sein, mehrere Elemente innerhalb einer Hierarchie implizit anzusprechen.

Problem

Das Problem bei einer impliziten Adressierung mehrerer Elemente gleichzeitig ist die Festlegung des Auswahlkriteriums für die Adressierung einerseits und die Durchführung von Aktionen auf den selektierten Elementen andererseits.

Lösung

Das Struktur-Matching stellt einen einfachen Mechanismus dar, der die oben angesprochenen Probleme zu lösen hilft. Als Auswahlkriterium für die Adressierung mehrerer Elemente gleichzeitig wird die strukturelle Ähnlichkeit zweier Hierarchien genutzt. Passen zwei Elementhierarchien zusammen, so werden über das onmatch-Ereignis die zu adressierenden Elemente angesprochen.

Struktur

An einem *Structure-based Multicast* (strukturbasierten Multicast) sind zwei Elementhierarchien beteiligt, einerseits die Musterhierarchie, die zum strukturellen Vergleich benötigt wird, und andererseits die zu vergleichenden Hierarchien, die adressiert werden sollen.

In Abbildung 64 ist ein Dokument inklusive Programmcode dargestellt, welches einen Broadcast-Aufruf eines anderen Dokuments durchführt.

Verhalten

Beim strukturbasierten Multicast enthält die Musterhierarchie oder deren Vorgängerelemente die Programmteile, die bei einer strukturellen Ähnlichkeit zu einer anderen Elementstruktur ausgeführt werden sollen.

Im obigen Beispiel wird die onmatch-Funktion in der Musterhierarchie vom gegebenen Dokument aus durch die match-Funktion der Laufzeitumgebung aufgerufen. Solche Ereignisbehandlungsfunktionen können an beliebiger Stelle innerhalb der Musterhierarchie eingefügt werden, so daß bei einem Struktur-Matching mehrere Funktionen indirekt aufgerufen werden. Dadurch ergibt sich die gewünschte Multicast-Semantik.

Request/Service Dispatcher

Die Kombination der Idiome des Monitor-Dokuments und des Broadcast-Funktionsaufrufs über Struktur-Matching kann eingesetzt werden, um eine Vermittlungsstelle zwischen Aufträgen und Diensten zu implementieren. Hierbei handelt es sich im Vergleich zum Laufzeitumgebungsmonitor eine direkte Umsetzung des Musters *Client-Dispatcher-Server* aus [38] mit Techniken des AHDM.

Kontext

Sobald Dokumente mit Auftragscharakter wie z.B. Druckaufträge oder Benachrichtigungen innerhalb eines Systems verschickt werden, kann es nützlich sein, diese Aufträge möglichst generisch zu verfassen, ohne einer späteren Verarbeitung vorzugreifen, wobei aber trotzdem ein einfaches Bearbeiten der Aufträge ermöglicht wer-

Dokument, welches den Broadcast-Aufruf durchführt:

```
<broadcaster>
  <func name="do_broadcast" type="text/tcl"><![CDATA[
    set list [DocumentGetElementsByTagname \
      $ahd_document request]
    set pattern [NodeListItem $list 0]
    NodeListDelete $list
    set match [AHDRuntimeLoad \
      $ahd_ahd urn:AHDS:request.ahd]
    AHDRuntimeMatch $ahd_ahd $match $pattern
  ]]></func>

  <request>
    <func name="onmatch" type="text/tcl"><![CDATA[
      puts "[ElementGetTagName $ahd_caller] matched"
    ]]></func>
    <type/>
    <title/>
    <body/>
  </request>
</broadcaster>
```

Dokument an der Adresse urn:AHDS:request.ahd:

```
<request>
  <title>Memo</title>
  <type>print_request</type>
  <body>This is a printable memo.</body>
</request>
```

Abb. 64: Beispiel für einen Broadcast-Aufruf über Struktur-Matching

den soll. Insbesondere in Kontexten, in denen die Art und Weise der späteren Verarbeitung noch nicht klar ist, ist ein solches Vorgehen angebracht.

Problem

Eine generische Architektur zur Vermittlung zwischen Aufträgen und Diensten setzt voraus, daß möglichst wenig Annahmen bei der Festlegung der Auftragsstruktur bezüglich der Verarbeitung gemacht werden und daß es feste Kriterien gibt, nach denen Aufträge den angebotenen Diensten zugeordnet werden können. Zudem muß eine Kommunikationsmöglichkeit zwischen Aufträgen und Diensten zur Verfügung stehen, die eine lose Kopplung beider Seiten erlaubt.

Lösung, Struktur und Verhalten

Die Basis für das Idiom *Request/Server Dispatcher* (Auftrag/Dienst-Vermittler) bildet der Laufzeit-Monitor, welcher die Dokumente, welche Dienste anbieten, beim Eintreffen eines Auftrags-Dokuments in einer Laufzeitumgebung benachrichtigt. Die Rolle des Vermittlers in diesem Muster wird so durch den Laufzeitumgebungs-Monitor erfüllt, die eintreffenden Aufträge sind die Subject-Dokumente und die Dienste, die die Bearbeitung der Aufträge durchführen, stellen die Object-Dokumente aus dem genannten Muster dar.

Die Zuordnung zwischen Aufträgen und Diensten wird durch strukturelle Übereinstimmung von Auftrag und Teilen der Dienstanbieter-Dokumente hergestellt. Dazu werden beim Eintreffen eines neuen Auftrags durch den Laufzeitmonitor alle bereitgestellten Dienstdokumente mit dem Auftrag strukturell verglichen. Bei einer Übereinstimmung werden durch die im Strukturmatching beschriebenen Mechanismen die passenden Programmteile im Dienstdokument ausgeführt. Durch eine solche Konstellation muß weder im Auftrags- noch im Dienstdokument eine explizite Verknüpfung vorhanden sein, es reicht hier die strukturelle Ähnlichkeit. Natürlich ist bei dieser Anwendung des Strukturmatching eine geeignete Auswahl der Musterhierarchie in den Dienstdokumenten notwendig, um eine fälschliche Bearbeitung von Aufträgen zu vermeiden. Auch muß eine Spezifikation der erbrachten Dienste im Vorhinein erfolgen.

Eine typische Konfiguration eines Auftrag/Dienst-Vermittlers ist in Abbildung 65 zu sehen.

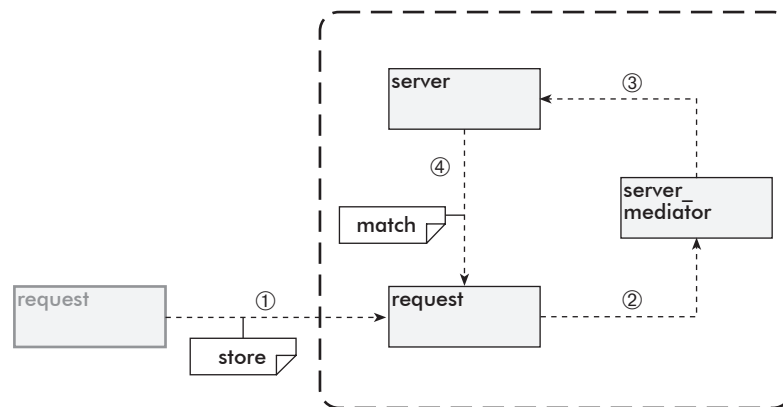


Abb. 65: Konfiguration eines Auftrag/Dienst-Vermittlers

Virtual Document

Eine charakteristische Eigenschaft von Hypertextdokumenten ist die Vernetzungsmöglichkeit. Diese soll genutzt werden, um *Virtual Documents* (logische Dokumente) aus unterschiedlichen Komponenten transparent zusammenzusetzen.

Kontext

Virtuelle Dokumente können in jedem Kontext genutzt werden, in dem ein Dokument aus mehreren Komponenten über Hyperlinks zusammengesetzt werden kann.

Problem

Die Forderung nach Wiederverwendung kann es erforderlich machen, Dokumente in Einzelkomponenten zu zerlegen, welche in andere Dokumente eingefügt werden können. Es müssen also Mechanismen zur Dekomposition und Komposition von Dokumenten vorhanden sein.

Lösung

Durch die Nutzung von XML und XLink sind die grundlegenden Mechanismen für die Realisierung von virtuellen Dokumenten bereits vorhanden. Als physikalische Komponenten kommen hierbei XML-Entities zum Einsatz, welche über Entity-Re-

ferenzen oder XLinks transparent in ein Dokument eingebunden werden. Die Transparenz ist allerdings erst nach der Verarbeitung des gesamten Dokuments durch einen XML-Prozessor gegeben, welcher die Vernetzungen auflöst. So erhält eine Instanz in Abbildung 66, welche das Hauptdokument per GET-request anfordert, ein virtuelles Gesamtdokument, nachdem das Hauptdokument und alle Komponenten geladen sind.

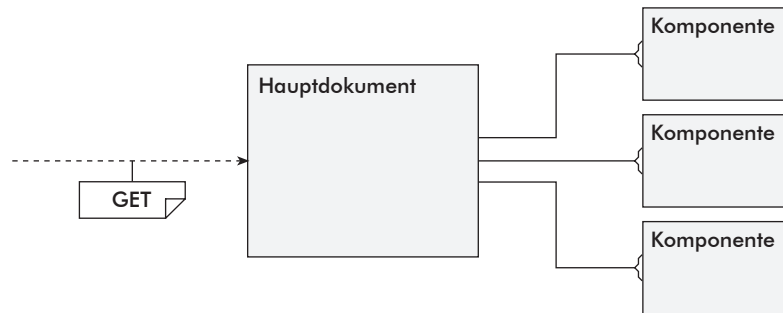


Abb. 66: Virtuelles Dokument bestehend aus Hauptdokument und Komponenten

Struktur

Ein virtuelles Dokument besteht immer aus einer Hauptkomponente, die mittels XLink- oder Entity-Referenzen Subkomponenten derart referenziert, daß sie in das Hauptdokument eingebunden werden.

Verhalten

Das Verbinden von Haupt- und Subkomponenten erfolgt automatisch durch den XML-Parser beim Auflösen der Referenzen, so daß keine weitere Applikationslogik notwendig ist.

19 Diskussion

Die Methode, welche in diesem Abschnitt vorgestellt wurde, soll die Konstruktion von AHD-basierten Systemen erleichtern. Dazu zählt auch die Identifikation von zusätzlichen Konstrukten, die im Modell nicht beschrieben wurden. Die zu Beginn des Abschnitts formulierten Anforderungen erstrecken sich dabei auf Anforderungen an die Methode selbst sowie Anforderungen an die mittels dieser Methode erstellten Systembestandteile. Neben der Methode wird auch eine Notation für die graphische Darstellung der Architektur von AHD- und Web-basierten Systemen vorgestellt.

Wichtiger Bestandteil der Methode ist die Verwendung von Entwurfsmustern, welche zwei Zielen dienen. Sie erleichtern die Identifikation von wiederkehrenden Problemstellungen und sie stellen einen Grundstock von wiederverwendbaren Bausteinen zur Verfügung. Beim Einsatz von Entwurfsmustern ist aber zu beachten, daß sie nicht dazu verwendet werden können, um neue Lösungen zu entwickeln. Vielmehr dokumentieren Entwurfsmuster per Definition nur bewährte Lösungen. Neuartige Probleme können so nicht bewältigt werden [140]. Es muß zudem auch vermieden werden, gegebene Probleme und Aufgabenstellungen so anzupassen, daß der Einsatz bestimmter Entwurfsmuster möglich ist. Stattdessen muß sich der Einsatz von Entwurfsmustern immer nach den vorhandenen Anforderungen richten.

Über den hier vorgeschlagenen Einsatz von Entwurfsmustern hinaus lassen sich diese auch für die Lösung von Problemen, welche typisch für Hypertext-Systeme sind, einsetzen, z.B. für eine konsistente Navigationsführung oder eine einheitliche Darstellung. An dieser Stelle sei für eine weitere Betrachtung auf die Ausführungen in [156] verwiesen.

19.1 Erfüllung der Anforderungen

Trotz der genannten Einschränkungen unterstützen Entwurfsmuster den Entwicklungsprozeß durch die Vereinfachung der Strukturierung mittels bereits verfügbarer Komponenten. Dies dient der Erfüllung der Anforderungen aus Unterkapitel 16.1, da so ein inkrementelles Vorgehen entlang der Systemebenen durch den Einsatz entsprechender Architektur- und Entwurfsmuster sowie Idiome gefördert wird. Demnach können z.B. Architekturmuster in den initialen Phasen der Entwicklung eingesetzt werden, während Idiome bei der konkreten Umsetzung eines Designs zum Tragen kommen.

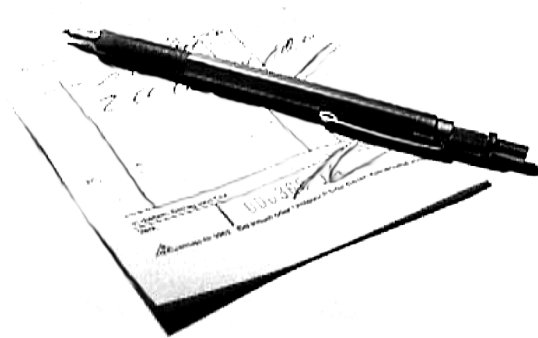
Die Wiederverwendung von Komponenten wird durch Entwurfsmuster ebenfalls gefördert, allerdings ist hierbei zu bemerken, daß durch die vorgeschlagene Methode keine wiederverwendbaren Komponenten direkt erstellt werden. Dies ist allerdings ein generelles Problem der Software-Entwicklung, da sich die Eignung von Komponenten zur Wiederverwendung schwer bei der Erstellung eines einzelnen Systems beurteilen läßt. Vielmehr steht die Erfüllung der Systemanforderungen im Vordergrund.

19.2 Besondere Charakteristika

Die in diesem Abschnitt beschriebenen Muster dienen neben der Unterstützung des Entwicklungsprozesses auch zur Demonstration von in Zusammenhang mit aktiven Hypertextdokumente typischen Konzepten wie Aktivität, Persistenz oder Rollen. Es sei an dieser Stelle auch auf andere Methoden verwiesen, die zur Entwicklung Web-basierter Systeme benutzt werden können, so etwa OODHM [155] oder RMM [88]. Der Vorteil der hier vorgestellten Methode besteht jedoch in der Einbeziehung auch von aktiven Komponenten, die zudem in nicht-aktive Komponenten eingebettet sein können.

Abschnitt E

Anwendungen



20 Auswahl von Anwendungen

Das vorgestellte Modell und die dazugehörige Entwicklungsmethode für aktive Hypertextdokumente soll durch das Erstellen von darauf aufbauenden Anwendungen verifiziert werden. Dazu gehört insbesondere die Betrachtung der Anforderungen, welche an das Modell und die Methode gestellt wurden. Die Erfüllung der Anforderungen einerseits als auch die Analyse weiterer Charakteristika aktiver Hypertextdokumente andererseits soll zum Abschluß dieses Abschnitts erfolgen.

Grundsätzlich soll sich die Auswahl der Anwendungen am Einsatzgebiet aktiver Hypertextdokumente orientieren, dazu zählt beispielsweise auch die Realisierung bereits bestehender Anwendungen mit den Mitteln des AHDM. Weiterhin soll versucht werden, Anwendungen zu realisieren, die möglichst alle grundlegenden Mechanismen des AHDM sowie die darauf aufbauenden Idiome, Muster und Semantiken demonstrieren. Eine erste Kategorisierung von Anwendungen läßt sich mittels der im Zusammenhang mit der Entwicklungsmethode vorgestellten Architekturmuster vornehmen. Im folgenden soll noch einmal kurz zusammengefaßt werden, welche Charakteristika ausgewählte Architekturmuster verdeutlichen können.

20.1 Eigenständige Anwendungen

Die einfachste Form eines AHD-Anwendung ist die eigenständige Anwendung. Sie besteht aus einem stationären Dokument, welches keine externen Ressourcen benötigt. Anhand einer solchen Anwendungen können folgende, durchgängig zu den grundlegenden AHDM-Mechanismen gehörenden Punkte verdeutlicht werden:

- Wechsel zwischen Aktivität und Inaktivität
- Persistenz von Dokumenten
- Modifikation des Inhaltes und der Struktur
- Ausführung von Funktionen
- Interaktion mit dem Benutzer

20.2 Kooperative Anwendungen

Kooperative Anwendungen sind im Gegensatz zu einfachen eigenständigen Anwendungen offen für andere Anwendungen und nutzen externe Ressourcen über das Netz. Auch ist eine verteilte Realisierung der Anwendung charakteristisch. Sie zeigen die Nutzung einer Reihe weiterer Punkte (siehe dazu auch [93]):

- Referenzierung von Netzressourcen
- Aktive Referenzierung durch Delegationsmechanismen
- Verhaltensänderung
- Wiederverwendung bestehender Ressourcen
- Bereitstellung neuer Komponenten zur Wiederverwendung

20.3 Agentensysteme

Die nächste Kategorie möglicher Systemarchitekturen umfaßt solche Systeme, bei denen aktive Komponenten selbst verschickt werden, d.h. stationäre und mobile Komponenten zusammen eingesetzt werden, um die Systemfunktionalität zu realisieren. An dieser Stelle sollen Agentensysteme eine Reihe wichtiger Charakteristika demonstrieren:

- Implizite Kommunikation über Delegations- und Matching-Mechanismen
- Koordination mobiler aktiver Dokumente
- Nutzung von AHD-basierten Komponenten in unterschiedlichsten Umgebungen zur Untersuchung der Interoperabilität

Gerade die Gruppe der Agentensysteme und der kooperativen Anwendungen soll in diesem Abschnitt die Möglichkeiten demonstrieren, wie aktive Hypertextdokumente auch in solchen Systemen genutzt werden können, in denen keine AHD-Unterstützungskomponenten wie die Laufzeitumgebung zur Verfügung stehen. Dies steht in engem Zusammenhang zu der Zielsetzung beim Einsatz von XML im allgemeinen, welcher gerade auf die Verbindung heterogener Systeme abzielt.

21 Kontaktmanager

Die erste Anwendung repräsentiert die Gruppe der eigenständigen Anwendungen. Es soll ein einfacher Kontaktmanager realisiert werden, welcher die Kundenkontakte in einer Aquisitionsphase einer Direktmarketing-Kampagne verwaltet. Die Umsetzung erfolgt nach dem in Unterkapitel 17.2 beschriebenen Schema. Zuerst jedoch sei die Aufgabenstellung weiter präzisiert.

21.1 Aufgabenstellung

Bei der Aquisition neuer Kunden im Rahmen einer Marketing-Kampagne werden unterschiedliche Instrumente eingesetzt: Anrufe, Briefe und E-Mails. Für jeden potentiellen Kunden sollen der Einsatz dieser Instrumente aufgezeichnet werden, um einerseits eine Fortschrittskontrolle zu erhalten und andererseits spätere Kampagnen aufgrund der Datenbasis planen zu können. Die Aufzeichnung ist mit einer Zustandsbeschreibung verbunden, welche aussagt, ob ein Kunde kontaktiert und erfolgreich akquiriert wurde oder die Kampagne nicht erfolgreich war.

Zu den Informationen, die verwaltet werden sollen, zählen die Kundendaten wie etwa Ansprechpartner, Adresse oder zusätzliche Notizen, weiterhin die einzelnen Kontakte in Form von Anrufen, Telefaxen, Briefen oder E-Mails sowie die Zustandsbeschreibung. Jegliche Materialien sollen über den Kontaktmanager zugreifbar sein, insbesondere eingegangene und versandte Faxe, Briefe und E-Mails.

Weiterhin soll der Kontaktmanager es ermöglichen, alle relevanten Daten neu anzulegen und zu manipulieren. Dies gilt insbesondere für die Kontaktdaten.

21.2 Strukturierung der Information

Aus der Aufgabestellung ergeben sich direkt die folgenden Informationselemente:

- Kundendaten (Ansprechpartner, Adresse, Zustand der Aquisition usw.)
- Kontaktdaten (Datum, Notizen usw.)
- Ressourcen zu den Kontakten (eingegangene und versandte Faxe, Briefe und E-Mails)

Durch die geringe Komplexität sowohl der Funktionalität als auch der zugrunde liegenden Information kann ein Verzeichnis der Kontakte zu einem Kunden zusammen mit den Informationen über den Kunden in einem aktiven Hypertextdokument abgelegt werden. Davon getrennt werden sollen allerdings die Daten und Ressourcen, welche mit einem Einzelkontakt verbunden sind, also die eigentlichen Briefe und E-

Mails. Sie sollen nur referenziert werden. Eine Top-Down-Analyse der Daten ergibt auf der obersten Ebene das Kundenkontakt-Dokument, wie es in Tabelle 4 dargestellt ist.

Element	Beschreibung
customer	Enthält die Daten zu einem Kunden
state	Beschreibt den Zustand der Aquisition, gültige Werte sind „initial“, „kontaktiert“, „akquiriert“ und „nicht akquiriert“
log	Enthält eine Liste der einzelnen Kontakte

Tabelle 4: Elemente im Kundenkontakt-Dokument

Demnach ergibt sich als einzige aktive Anwendungskomponente das Kundenkontakt-Dokument. Denkbar wäre allerdings auch noch ein weiteres Dokument, das alle Kundenkontakt-Dokumente referenziert und das Anlegen eines neuen Dokuments erlaubt. Diese Erweiterung soll in der Abschlußbetrachtung diskutiert werden, da damit der Übergang zu der Gruppe der kooperativen Anwendungen erreicht wird.

Führt man mit der Top-Down-Analyse fort, so sind zunächst die Kundendaten (also die Unterelemente des customer-Elements) zu präzisieren (siehe dazu Tabelle 5).

Element	Beschreibung
person	Name der Kontaktperson
address	Postanschrift (nicht weiter aufgeteilt in Ort o.ä.)
email	E-Mail-Adresse
phone	Telefonnummer
fax	Telefaxnummer
notes	freie Notizen

Tabelle 5: Unterelemente des customer-Elements

Die Liste der erfolgten Kontakte wird innerhalb des Kundenkontakt-Dokuments im log-Element abgelegt, welches in Tabelle 6 näher beschrieben ist

Element	Beschreibung
call	Daten zu einem Anruf (es wird nicht zwischen ein- und ausgehenden Anrufen unterschieden)
letter_out	Daten zu einem versandten Brief, Fax oder einer E-Mail
letter_in	Daten zu einem eingegangenen Brief, Fax oder einer E-Mail

Tabelle 6: Elemente innerhalb des log-Elements

Die Daten, welche für einen erfolgten Kontakt festgehalten sind, sind für Anrufe, Briefe, Faxe und E-Mails sehr ähnlich. Neben dem Datum und zusätzlicher Notizen

werden für Briefe, Faxe und E-Mails noch Verweise auf die Originaldokumente (falls in elektronischer Form vorhanden) abgelegt.

Element	Beschreibung
date	Datum des Kontakts
notes	freie Notizen
uri	Referenz auf das Originaldokument, sofern es elektronisch vorliegt und referenzierbar ist (wird nicht beim Anruf verwendet)

Tabelle 7: Unterelemente für das call-, letter_in und letter_out Element

Die daraus resultierende XML-DTD ist in Abbildung 67 zu sehen. Das Wurzelement wurde dabei mit contactlog benannt.

```

<!ELEMENT contactlog (customer, state, log)>
<!ELEMENT customer (person, address,
                    email, phone, fax, notes)>
<!ELEMENT person (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT log (call | letter_in | letter_out)*>
<!ELEMENT call (date, notes)>
<!ELEMENT letter_in (date, notes, uri)>
<!ELEMENT letter_out (date, notes, uri)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT url (#PCDATA)>

```

Abb. 67: DTD für das Kundenkontakt-Dokument

Bei der Verwendung der oben dargestellten DTD ist zu berücksichtigen, daß sie im Grunde nur den Namensraum für die eigentlichen Kunden- und Kontaktinformationen beschreibt. Weitere Namensräume wie beispielsweise der AHDM-Namensraum sind nicht enthalten und können auch nicht auf einfache Art und Weise mit dieser DTD kombiniert werden. Dies ist eine generelle Problematik bei der Verwendung von XML-Namespaces, die hier wie bereits in Unterkapitel 13.4 beschrieben durch die Nutzung der unscharfen Validierung abgemildert werden soll.

Da die Anwendung als eigenständige Komponenten realisiert werden soll, ergeben sich auf der Ebene des Kundenkontakt-Dokuments keine externen Schnittstellen in Form von AHD-Funktionen oder -Variablen. Dies ist erst auf der Unterelementebene notwendig. Dort sind insbesondere solche Funktionen zu definieren, die eine Benutzerinteraktion ermöglichen. Sie werden in Unterkapitel 21.6 implementiert, da sie keine eigentlichen Schnittstellen zwischen aktiven Komponenten innerhalb des Systems darstellen.

21.3 Gruppierung zusammengehöriger Komponenten

Der Übersichtlichkeit halber werden die Bestandteile des Systems in vier Cluster unterteilt: ein Cluster für die Speicherung der Kundenkontakt-Dokumente sowie drei

Cluster für die Original-Ressourcen (Briefe, Faxe und E-Mails). Diese Einteilung ist allerdings nicht bindend, da die Komponenten über generische URIs referenziert werden und somit auch andere Gruppierungen möglich sind. Werden allerdings URLs genutzt, so sind alle Referenzen bei einer Veränderung der Struktur nachzuziehen. Die hier gewählte Aufteilung ist in Abbildung 68 dargestellt.

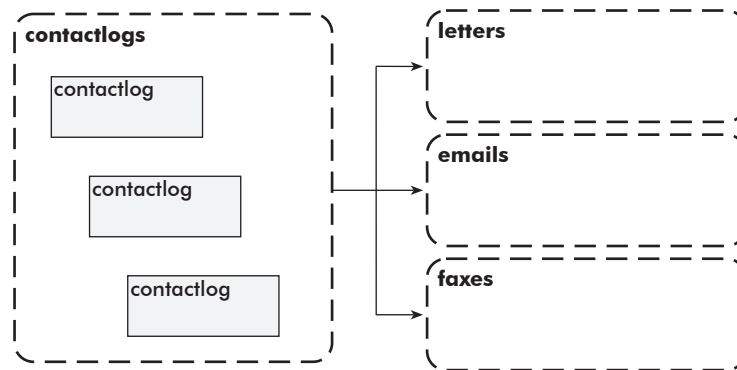


Abb. 68: Aufteilung der Kontaktmanager-Anwendung in Cluster

21.4 Definition der Verteilung der Komponenten

Auch bei der Verteilungsstrategie wird die einfachste Lösung gewählt, d.h. alle Dokumente sind innerhalb eines Netzwerkknotens abgelegt. Auch hier ist aber diese Vereinfachung nicht zwingend, sofern generische URIs zur Verknüpfung genutzt werden.

21.5 Definition der Kommunikationsstrategie

Die Kommunikation zwischen den Komponenten in der Anwendung umfaßt initial nur die Kommunikation zwischen Benutzer und Anwendung. Demnach ist die Funktionalität der Modifikation der Daten innerhalb der Anwendung über IEM-Ereignisse anzubinden. Durch die Einfachheit der Anwendung entfallen alle weiteren Kommunikationswege wie externe Funktionsaufrufe oder mobile Komponenten.

21.6 Realisierung der Komponenten

Da die Kontaktmanager-Anwendung eine eigenständige Anwendung ist, müssen sowohl die Präsentationsschicht als auch die erforderliche Funktionalität zur Modifikation implementiert werden. Dabei ist das Kundenkontakt-Dokument die einzige zu realisierende Komponente.

Präsentationsschicht

Die Darstellung des Kundenkontakt-Dokuments baut direkt auf den Nutzdaten auf, welche in der o.g. DTD definiert wurde. Wie dort zu sehen ist, fehlen allerdings Elemente, welche die Darstellung erst sinnvoll ermöglichen, etwa Überschriften oder nähere Beschreibungen von Datenelementen. Zwar können die Nutzdaten über entsprechende CSS-Eigenschaften visualisiert werden, dennoch muß eine geeignete

Technik gefunden werden, die zusätzlich benötigten Elemente in das Kundenkontakt-Dokument einzubinden.

Eine dedizierte Methode für solche Zwecke sind die Pseudo-Elemente der CSS2-Spezifikation, die es erlauben, für ein Element bestimmte Texte zusätzlich darstellen zu lassen, etwa vor oder nach den eigentlichen Elementen. Da diese Methode nur zu Darstellungszwecken genutzt werden kann und an die CSS2-Spezifikation gebunden ist, soll an dieser Stelle über XML-Namensräume eine allgemeiner nutzbare Technik verwandt werden.

Dazu werden die Elemente, die nur zu Präsentationszwecken benötigt werden, in einem gesonderten Namensraum definiert. Sie können dann bei Bedarf in einem XML-Dokument über die Mechanismen der XML-Namespacespezifikation eingebunden werden. In solchen Kontexten, in denen sie nicht relevant sind, können sie anhand der Namensraum-Herkunft ausgefiltert werden. Ein besonderer Vorteil ergibt sich bei diesem Vorgehen durch die Wiederverwendbarkeit der Präsentationselemente in unterschiedlichen Dokumenten. Ein Beispiel für die Nutzung ist in Abbildung 69 zu sehen. Dort sind die Elemente aus dem im contactlog-Element referenzierten und deklarierten Namensraum für Präsentationselemente hervorgehoben. Alle anderen Elemente stammen aus der oben definierten DTD für den Kontaktmanager.

```
<contactlog xmlns:p="urn:AHDS:General/presentation.dtd">
  <customer>
    <p:heading>Kontakt-Log</p:heading>
    <p:field>
      <p:label>Kontaktperson: </p:label>
      <person>Eckhart Köppen</person>
    </p:field>
    ...
  </customer>
  ...
</contactlog>
```

Abb. 69: Beispiel für die Verwendung von Präsentationselementen

Ein Teil der Elemente, welche hier für Darstellungszwecke genutzt werden, sind in Tabelle 8 erläutert. Sie sind in einer DTD definiert, welche im Anhang zu finden ist.

Element	Beschreibung
heading	Überschriftenelement
field	Gruppierung eines Datenfeldes mit einem zugehörigen Beschreibungsfeld
label	Beschreibungsfeld
separator	Trennlinie zwischen Dokumentteilen

Tabelle 8: Präsentationselemente

Zusätzlich sind noch Stylesheets notwendig, um die CSS-Eigenschaften der Elemente im Dokument festzulegen. Diese werden über eine processing instruction entsprechend dem in [42] beschriebenen Mechanismus mit dem Dokument verbunden. Auch wird zur Erhöhung der Wiederverwendbarkeit ein Stylesheet definiert, welches auch in anderen Anwendungen eingesetzt werden kann. Dies wird dadurch erreicht, daß die CSS-Eigenschaften nicht direkt über den Elementnamen vergeben werden,

sondern stattdessen das class-Attribut genutzt wird. Auch dieser Stylesheet ist im Anhang zu finden. Er enthält unter anderem vordefinierte Styles für Überschriften, Absätze, Inline-Elemente oder Beschreibungselemente, wie aus Tabelle 9 ersichtlich.

Style-Regel	Beschreibung
heading1-5	Überschriften
para	Absätze (Wert der Display-Eigenschaft ist block)
inline	Inline-Elemente (Wert der Display-Eigenschaft ist inline)
label	Beschreibungsfelder (fette Schriftart, inline)
document	Enthält Vorgabewerte für Ränder, Schriftart und -größe
func, var	Nicht anzuzeigende AHD-Elemente
list	Liste mit einem breiteren linken Rand, schachtelbar (dadurch ergeben sich eingerückte Unterlisten)

Tabelle 9: Vordefinierte Styles

Abbildung 70 zeigt schließlich die Anwendung unter Benutzung des Stylesheets.

Kontakt-Log

Kontaktperson: Eckhart Köppen
Adresse: Fachbereich 5, Universitätsstraße 9, 45151 Essen
E-Mail: koeppen@wi-inf.uni-essen.de
Telefon: +49 (201) 183 4075
Telefax: +49 (201) 183 4073
Notizen:
Aktueller Zustand: initial

Kontakte

Abb. 70: Ausschnitt aus dem Kundenkontakt-Dokument

Funktionalität

Bis hierher ist das Kundenkontakt-Dokument noch rein statisch bzw. nur durch externe Programme manipulierbar. Es wird jedoch zu einem aktiven Hypertextdokument, sobald die im AHDM definierten Mechanismen und Techniken eingesetzt werden.

Wie bereits angesprochen, soll das Dokument durch Benutzerinteraktion manipuliert werden. Es wird dabei angenommen, daß ein leeres Dokument bereits Elemente entsprechend der DTD enthält, also alle Kundendaten, die Zustandsbeschreibung und eine leere Liste der Kontakte. Auch sollen alle Präsentationselemente vorhanden sein. Folgende Manipulationen werden nun realisiert:

- Veränderung der Inhalte aller Datenelemente aus der Kundenkontakt-DTD

- Erzeugen neuer Einträge für Anrufe sowie eingegangene und versandte E-Mails, Briefe und Faxe

Diese beiden Funktionalitäten werden getrennt behandelt.

Modifikation von Elementinhalten

Grundsätzlich lassen sich strukturierte Dokumente, wie sie auch AHDs sind, auf mehrere Arten durch Benutzerinteraktion modifizieren. Eine Methode besteht in der Bereitstellung eines Editors, welcher über DTDs ein strukturbasiertes Editieren eines Dokuments ermöglicht. Dies ist entweder in einer Struktursicht oder unter Berücksichtigung von Style-Angaben ähnlich einer Textverarbeitung möglich. Allerdings ist die Komplexität solcher Editoren recht hoch, zudem ist eine Validierung und ein strukturgestütztes Editieren nur dann möglich, wenn nicht mehr als ein Namensraum für die Elemente genutzt wird. Die zweite Methode bietet sich gerade bei AHD-Anwendungen an. So bietet das AHDM ausreichende Mittel, um ein Dokument auch ohne Editor strukturgestützt zu modifizieren. Dazu wird das Aufrechterhalten der definierten Struktur in der Funktionalität zur Manipulation des Dokuments implementiert.

In der hier zu realisierenden Anwendung wird ein einfacher Mechanismus zur Modifikation von textuellen Elementinhalten eingeführt, welcher sich auf die im DOM definierten HTML-Form-Elemente (hauptsächlich Eingabefelder) stützt. In Zukunft erscheint auch der Einsatz der im Moment noch in der Spezifikationphase befindlichen Standards zu XML-Formularen (siehe etwa [56] oder [29]) sinnvoll. Andererseits wäre es generell auch möglich, die gesamte Modifikation über IEM-Tastendruckereignisse zu implementieren, davon wurde aber an dieser Stelle aus Performance-Gründen Abstand genommen. Vielmehr wird die Modifikation als zweiteiliger Prozeß durchgeführt:

- Der Benutzer signalisiert durch das Klicken auf das Bezeichnungsfeld eines Elements, daß er den Inhalt des Elements ändern möchte, worauf ein Texteingabefeld an der entsprechenden Stelle im Dokument eingefügt wird.
- Hat der Benutzer den neuen Inhalt eingegeben, so signalisiert er das Ende der Modifikation wiederum durch Klicken auf das Beschreibungsfeld, worauf der Wert des Eingabefelds als neuer Inhalt des Elements eingefügt wird.

In Abbildung 71 ist das Editieren des Ansprechpartner-Elements über ein Eingabefeld zu sehen, nachdem auf das Beschreibungsfeld links neben dem Element geklickt wurde.

Kontaktperson:
Adresse: Fachbereich 5, Universitätsstraße 9, 45151 Essen
E-Mail: koeppen@wi-inf.uni-essen.de

Abb. 71: Editieren eines Elementinhalts

Die Struktur eines solchen Eingabefeldes besteht aus drei Bestandteilen: dem Datenelement, dem Beschreibungselement und einem umschließenden Element, welches die Koordination der Abläufe vornimmt. Sie ist in Abbildung 72 dargestellt. Für das Zusammenspiel dieser Komponenten werden ähnlich wie bei den Stylesheets die drei Elemente nicht über ihren Elementnamen, sondern über ein gesondertes Attribut referenziert. Dabei handelt es sich um das in Unterkapitel 18.3 angesprochene *role-Attribut (Role Identifier Attribute)*.

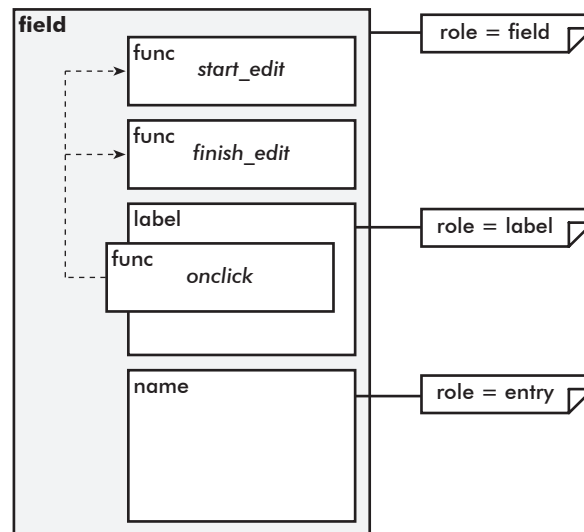


Abb. 72: Struktur eines Eingabefelds

Wie aus der Übersichtsdarstellung ersichtlich, werden die beiden Schritte zum Editieren in den Funktionen `start_edit` und `finish_edit` (dem übergeordneten Element zugeordnet) realisiert und von der IEM-Behandlungsfunktion `onclick` angestoßen. Der Programmcode ist in Abbildung 73 dargestellt.

Zur Vereinfachung der Dokumentenstruktur ist der Programmteil, welcher die beiden oben aufgeführten Funktionen anstößt, im `onclick`-Attribut des Beschreibungselements implementiert. Der Wert des Attributs wird abhängig vom nächsten durchzuführenden Schritt gesetzt.

Beide Funktionen nutzen den Mechanismus der parent delegation aus, um unabhängig von der konkreten Struktur und den Elementnamen die verwendeten Elemente zu referenzieren. Die Delegation erfolgt dabei ausgehend vom aufrufenden Element, in diesem Fall dem Beschreibungselement, welches das `onclick`-Ereignis bearbeitet. Tatsächlich können durch diese Struktur die beiden Funktionen an beliebiger Stelle innerhalb der Kette der übergeordneten Elemente, ausgehend von der Ereignisbehandlungsfunktion, eingefügt werden. Demzufolge reicht es aus, wenn die Funktionen beispielsweise direkt dem Wurzelement untergeordnet sind. Dies bedeutet aber auch, daß sie auf diese Weise von allen so strukturierten Eingabefeldern innerhalb eines Dokuments wiederverwendet werden können.

Erzeugung neuer Elemente

Die zweite benötigte Funktionalität neben der Modifikation der bestehenden Elemente ist das Hinzufügen neuer Elemente für Anrufe und Briefe (wozu auch E-Mails und Faxe gezählt werden). Dazu sollen wiederum HTML-Form-Elemente benutzt werden. Das Einfügen neuer Elemente soll durch Button-Elemente möglich sein, welche unterhalb der Kontaktliste im Dokument angeordnet sind. Wie auch bei den Beschreibungsfeldern der Eingabefelder werden die Programmskripte über `onclick`-Ereignisse ausgelöst. Die eigentlichen Funktionen zur Erzeugung neuer Anruf- oder Briefelemente sind dem Wurzelement untergeordnet und werden von den Button-Elementen über parent delegation aufgerufen. Eine vereinfachte Darstellung zeigt Abbildung 74.


```
<ahd:func name="start_edit" type="text/tcl"><![CDATA[
    set value [NodeParentLookup $ahd_caller role value]
    set label [NodeParentLookup $ahd_caller role label]
    set field [NodeParentLookup $ahd_caller role field]

    ElementSetCSSProperty $value display none
    set entry [HTMLInputElementNew $ahd_document]
    NodeAppendChild $field $entry
    ElementSetAttribute $entry role entry
    ElementSetAttribute $entry value \
        [ElementGetContents $value]
    ElementSetAttribute $label onclick \
        {AHDRuntimeCall $ahd_ahd $ahd_element finish_edit {}}
]]></ahd:func>

<ahd:func name="finish_edit" type="text/tcl"><![CDATA[
    set value [NodeParentLookup $ahd_caller role value]
    set label [NodeParentLookup $ahd_caller role label]
    set field [NodeParentLookup $ahd_caller role field]
    set entry [NodeParentLookup $ahd_caller role entry]

    ElementSetContents $value \
        [ElementGetAttribute $entry value]
    ElementSetCSSProperty $value display inline
    NodeDelete $entry
    ElementSetAttribute $label onclick \
        {AHDRuntimeCall $ahd_ahd $ahd_element start_edit {}}
]]></ahd:func>
```

Abb. 73: Funktionen start_edit und finish_edit im Eingabefeld

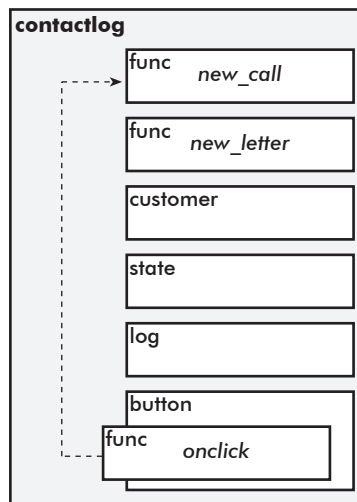


Abb. 74: Erweiterung des Dokuments für das Erzeugen neuer Elemente

Anders als bei der Implementierung der Eingabefelder ist diese Funktionalität weniger universell zur Wiederverwendung geeignet, da die erzeugten Elemente stark von der Anwendung abhängen. Deshalb werden innerhalb der beiden Funktionen

new_call und new_letter andere Elemente expliziter adressiert, als es bei den Eingabefeldern der Fall ist. So zeigt der Anfang der Funktion new_call in Abbildung 74, daß der Einfügapunkt für ein neues Anruf-Element durch eine einfache Suche über den Elementnamen erfolgt. Dabei wird auch angenommen, daß das log-Element, in das die neuen Elemente eingefügt werden sollen, genau einmal im Dokument vorhanden ist.

```
<ahd:func name="new_call" type="text/tcl"><![CDATA[
    set l [DocumentGetElementsByTagName $ahd_document log]
    set log [NodeListGetItem $l 0]
    NodeListDelete $l

    set call [ElementNew]
    NodeAppendChild $log $call
    ElementSetTagName $call "call"
    ...
]]></ahd:func>
```

Abb. 75: Beginn der Funktion zur Erzeugung eines Anruf-Elements

21.7 Auswertung

Eine Auswertung der Implementierung hinsichtlich der Verwendung des AHDM zeigt einige interessante Punkte. Zwar besitzt die Anwendung nur eine geringe Komplexität, trotzdem können einige grundlegende Techniken des AHDM eingesetzt werden. So sind Modifikation des Inhaltes und der Struktur über die DOM-Funktionen, das Ausführung von Funktionen über die Funktionen der AHD-Laufzeitumgebung oder die Interaktion mit dem Benutzer über IEM-Ereignisse innerhalb der Anwendung realisiert.

Die Praktikabilität des Einsatzes aktiver Hypertextdokumente ist allerdings auch noch von anderen Faktoren abhängig, die sich auch aus den Anforderungen aus dem Bereich der Software-Entwicklung herleiten lassen. Die vorgestellte Anwendung zeigt zwei Eigenschaften, welche in diesem Zusammenhang von Bedeutung sind: Wiederverwendbarkeit und Modularität.

Modularität

Die Kontaktmanager-Anwendung läßt sich in mehrere Module oder Bestandteile einteilen. Dazu zählen die Elemente, welche in der DTD für diese Anwendung definiert sind, die Elemente, welche in der DTD für die Präsentation deklariert sind, der Stylesheet und die genutzten AHD-Funktionen. Sie sind nur lose (beispielsweise über die XML-Namespace-Mechanismen oder vereinbarte Attribut-Werte) gekoppelt und sind unabhängig voneinander spezifiziert worden. Demzufolge würden sie im Rahmen von Entwicklungsprojekten auch unabhängig voneinander realisiert werden.

Wiederverwendbarkeit

Die Wiederverwendbarkeit von Komponenten hängt eng mit der Modularität einer Anwendung zusammen, denn die einzelnen Module stellen in der Regel die wiederverwendbaren Einheiten dar. In der vorliegenden Anwendung sind die Präsentations-

elemente, der Stylesheet und die Funktionen für die Implementierung der Eingabefelder auf einfache Art und Weise wiederverwendbar:

- Die Präsentationselemente sind in einer DTD definiert, welche die Bedeutung, das Aussehen und die Struktur der Elemente dokumentieren soll. Ein Dokument kann diese Elemente durch Referenzierung der DTD beispielsweise als XML-Namespace nutzen.
- Der Stylesheet kann ähnlich wiederverwendet werden. Er läßt sich aus einem Dokument heraus referenzieren, so daß bei einer Änderung des Stylesheets alle referenzierenden Dokumente entsprechend der Änderung anders dargestellt werden. Eine weitere Möglichkeit wäre die Nutzung eines speziellen Style-Elementes, welches der HTML-Standard vorsieht, so daß ein Dokument eine Kopie des Stylesheets enthält, welche unabhängig von Änderungen der Vorlage ist.
- Die Funktionalität für die Realisierung von Eingabefeldern kann sowohl innerhalb eines Dokuments von mehreren Eingabefeldern gleichzeitig genutzt, als auch in unterschiedlichen Dokumenten eingesetzt werden. Wie in den beiden vorangegangenen Fällen kann dies durch die Referenzierung per XLink oder Entity-Referenz geschehen. Eine gültige Anwendung zeigt beispielsweise Abbildung 76.

```
...
<include xml:link="simple" show="embed" actuate="auto"
        href="urn:AHDS:General/inputfield.ahd" />

<katalogeintrag role="field">
  <text role="label" type="text/tcl"
        onclick="AHDRuntimeCall \
                $ahd_ahd $ahd_element start_edit {}">
    Artikelbeschreibung:
  </text>
  <artikel role="entry">
    PowerBook G3
  </artikel>
</katalogeintrag>
...
```

Abb. 76: Fragment zur Nutzung der Eingabefeld-Funktionalität

Die oben aufgezählten Möglichkeiten der Wiederverwendung sind möglich, da die Komponenten nur minimale Abhängigkeiten vom späteren Verwendungszweck haben. So sind bei den Stylesheets und den Eingabefunktionen nur bestimmte Attribute zu einem Element hinzuzufügen, um die Komponenten nutzen zu können: Im Falle des Eingabefelds sind die beteiligten Elemente durch das role-Attribut gekennzeichnet, für die Nutzung der Style-Definitionen wird das class-Attribut genutzt. Die Werte dieser Attribute können zur weiteren Vereinfachung in der DTD gesetzt werden.

Wegen der universellen Einsetzbarkeit sollen deshalb die Struktur und die Funktionen des Eingabefelds in die Gruppe der Idiome aus Unterkapitel 18.3 aufgenommen werden.

Übergang zur verteilten Anwendung

Es bleibt als Erweiterungsmöglichkeit noch zu klären, wie die einzelnen Kundenkontakt-Dokumente verwaltet werden. Dazu gehört beispielsweise der Zugriff auf

die Dokumente, das Anlegen neuer Dokumente oder das Löschen von Dokumenten. Diese Funktionen wurden im obigen Beispiel ausgespart, weil sich damit ein Übergang zu verteilten Anwendungen ergibt, welche im nächsten Kapitel behandelt werden sollen.

Trotzdem sei an dieser Stelle eine Lösung skizziert, die diese Funktionalitäten erbringt. Dazu wird die Anwendung um ein Zentraldokument erweitert, welches die Referenzen auf die einzelnen Kundenkontakt-Dokumente enthält und diese verwaltet. Ähnlich wie bei der Erzeugung neuer Anruf- oder Brief-Elemente in der Kontaktliste könnte dieses Zentraldokument neue Elemente für Kundenkontakt-Dokumente erzeugen und auch löschen. Interessant ist dabei der Punkt, an dem nicht nur eine neue Referenz in das Zentraldokument eingefügt, sondern tatsächlich auch ein neues Dokument erstellt wird. Hier bietet sich das Entwurfsmuster des *Prototyps* an. Dabei wird ein leeres Kundenkontakt-Dokument als Vorlage für alle neuen Dokumente genutzt und bei der Erzeugung eines neuen Dokuments kopiert. Dieser Vorgang kann vollständig mit den Mitteln des AHDM implementiert werden.

Weiterhin interessant bei der Betrachtung des Zentraldokuments ist die Tatsache, daß darüber nicht nur Dokumente verwaltet werden können, die entsprechend der oben vorgestellten Anwendung realisiert werden. Vielmehr ließen sich auch Referenzen auf andere Dokumente mit dem Zentraldokument verwalten, welches damit einen uniformen Zugriff auf unterschiedliche Arten von Kundenkontakt-Dokumenten erlaubt.

Eine vereinfachte Darstellung der Anwendung mit Zentraldokument, Prototypdokument und Referenzen auf einzelne Kundenkontakt-Dokumente ist in Abbildung 77 zu sehen.

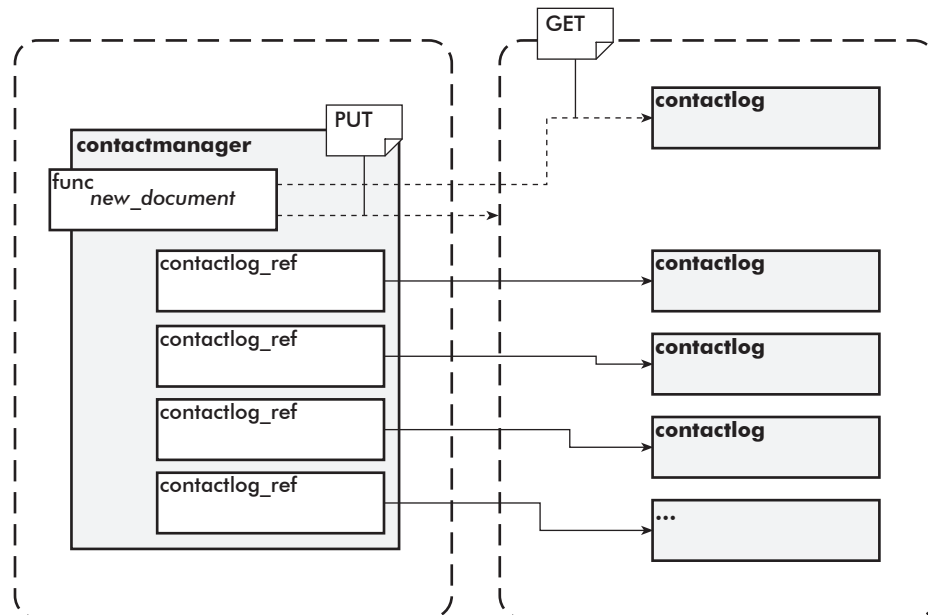


Abb. 77: Zentraldokument für die Verwaltung der Kundenkontakt-Dokumente

Verwendete Muster

In der vorgestellten Anwendung wurden drei Entwurfsmuster eingesetzt. Den Kern der Anwendung bildet dabei die eigenständige Anwendung (*Standalone Active Document*). Bei der Realisierung der Funktionalität für Benutzereingaben wurde das Rollenattribut (*Role Identifier Attribute*) eingesetzt, um die Aufgabenverteilung innerhalb des Eingabefelds zwischen dem Bezeichnungselement, dem Eingabefeld und dem Wert des Feldes festzulegen. Das Eingabefeld selbst ist durch seine einfache Integrierbarkeit in bestehende Dokumente ein Idiom, welches in einen Muster-Katalog für aktive Hypertextdokumente aufgenommen werden sollte. Als letztes Muster sei noch die kooperative, vernetzte Anwendung (*Networked Active Documents*) erwähnt, die als Erweiterung der eigenständigen Anwendung zur Realisierung von unterschiedlichen Kategorien von Kontakten vorgeschlagen wurde. Sie nutzt zudem das *Prototype*-Muster, um vordefinierte Kontaktinformationen als Vorlagen für tatsächliche Kontaktinformationen in das Zentraldokument zu laden.

22 Bookmark-Verwaltung

Um neben den grundlegenden Aspekten des AHDM auch die Charakteristika zu demonstrieren, welche bei einem Web-basierten Informationssystem zum Tragen kommen, soll in diesem Kapitel eine Anwendung zur verteilten Verwaltung von WWW-Bookmarks realisiert werden. Sie fällt in die Kategorie der kooperativen Anwendungen aus der Gruppe der Architekturmuster (siehe dazu Unterkapitel 18.1), konkret stellt die Anwendung die geforderte Funktionalität als Applikationsserver zur Verfügung.

Die verwendeten DTDs, Quelltexte und Stylesheets werden in diesem Kapitel nur auszugsweise besprochen, sie sind vollständig im Anhang zu finden.

22.1 Aufgabenbeschreibung

Die Bookmark-Verwaltung dient dazu, Bookmarks an einer zentralen Stelle zu verwalten und über das Internet zur Verfügung zu stellen. Bookmarks werden von der Anwendung in einer Hierarchie von Kategorien verwaltet, welche über Schlüsselworte charakterisiert sind. Die Anwendung soll diese Schlüsselworte nutzen, um die Bookmarks beim Einfügen automatisch einer Kategorie zuzuordnen. Bookmarks sollen aber auch manuell in andere Kategorien eingeordnet wie auch gelöscht werden können. Der Mechanismus der automatischen Einordnung soll nicht fest vorgegeben werden, als eine Beispielimplementierung soll aber Gewichtung eines Dokuments gegenüber einer Kategorie anhand im Dokument enthaltener Schlüsselworte realisiert werden. Das Erstellen neuer Kategorien und die Modifikation der Kategoriestruktur soll nicht Bestandteil der Aufgabe sein, allerdings wird in der Abschlußbetrachtung der Anwendung ein Lösungsansatz vorgestellt, welcher diese Funktionalität enthält.

22.2 Strukturierung der Information

Die Top-Down-Analyse der Information dieser Anwendung ergibt initial folgende Elemente:

- Das Bookmark-Dokument, welches die Bookmarks in einer Liste verwaltet und die notwendigen Funktionen dafür zur Verfügung stellt
- Die Kategorien, in die die Bookmark-Elemente eingeordnet werden
- Die Bookmark-Elemente innerhalb der Kategorien

Weiterhin müssen noch die Bookmarks selbst, die Dokumente, welche dadurch referenziert werden, und die Art und Weise, wie die erforderliche Funktionalität in Komponenten gekapselt wird, näher spezifiziert werden.

Informationselemente

Das Bookmark-Dokument soll zwei Zielen dienen: einerseits soll es die Bookmarks verwalten (d.h. Funktionalität zum Einfügen, Ändern und Löschen anbieten), andererseits soll es auch zum Zugriff auf die referenzierten Dokumente dienen, ähnlich wie es die Bookmark-Verwaltung gängiger Web-Browser ermöglicht. Es ähnelt demnach dem im vorigen Anwendungsbeispiel implementierten Kundenkontakt-Dokument. Allerdings enthält es durch die verteilte Nutzung und die Forderung nach automatischer Kategorisierung der Dokumente neue Funktionen, welche es zu einer kooperativen Anwendung machen. Tabelle 10 beschreibt zunächst die reinen Informationselemente des Bookmark-Dokuments.

Element	Beschreibung
category	Kategorie mit Schlüsselworten, welche Bookmark-Elemente oder wiederum Kategorien enthalten kann

Tabelle 10: Elemente des Bookmark-Dokuments

Die Kategorien für die Bookmark-Elemente können neben den Bookmarks auch wiederum Kategorien enthalten, wie es Tabelle 11 zeigt.

Element	Beschreibung
title	Eindeutiger Titel der Kategorie, wird als Referenz genutzt, wenn Bookmarks manuell eingefügt werden sollen
keywordlist	Liste von Schlüsselworten (einzelne Worte durch Whitespace getrennt)
category	optional ein oder mehrere Unterkategorien
bookmark	Enthält den Verweis auf ein externes Dokument als Bookmark

Tabelle 11: Elemente innerhalb der Bookmark-Kategorien

Die Struktur der Bookmark-Elemente ist in Tabelle 12 näher beschrieben. Wichtig ist dabei, daß die Adresse des referenzierten Dokuments über gängige Internet-Protokolle wie beispielsweise HTTP auflösbar ist und das Dokument auch geladen werden kann. Zudem muß das Dokument zur automatischen Einordnung in der gewählten Implementierung auf Schlüsselworte hin untersucht werden können. Ansonsten müßte ein Mechanismus realisiert werden, der die Schlüsselwörter auf andere Art und Weise ermittelt.

Element	Beschreibung
label	Eindeutiger Titel für das Bookmark, wird wie auch der Kategorietitel als Schlüsselement bei der Modifikation genutzt

Element	Beschreibung
address	Internet-Adresse des referenzierten Dokuments

Tabelle 12: Unterelemente des Bookmark-Elements

Die resultierende DTD für das Bookmark-Dokument zeigt Abbildung 78.

```

<!ELEMENT bookmarks (category*)>
<!ELEMENT category (title, keywordlist, category*, bookmark*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT keywordlist (#PCDATA)>
<!ELEMENT bookmark (label, address)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT label (#PCDATA)>

```

Abb. 78: DTD für das Bookmark-Dokument

Als Alternative zu der hier entworfenen DTD wäre es auch möglich gewesen, die *XML Bookmarks Exchange Language* (XBEL) [52] zu benutzen. Diese nutzt allerdings als entscheidenden Nachteil Attribute statt Elemente, um wichtige Daten zu speichern, etwa für URLs. Es erscheint an dieser Stelle sinnvoller, diese Daten als Elemente zu realisieren, um ihrer Bedeutung für die Anwendung gerecht zu werden (siehe dazu auch die Diskussion ab Seite 14). Eine mögliche Schnittstelle zur XBEL wird allerdings bei den Erweiterungen der Bookmark-Anwendung beschrieben.

Kapselung der Funktionalität

Im Gegensatz zur vorigen Anwendung soll an dieser Stelle die Funktionalität schon bei der Strukturierung der Information betrachtet werden. Der Grund dafür liegt in der Forderung nach flexiblen Kategorisierungsmechanismen für die Dokumente. Dies läßt sich mit dem Strategie-Entwurfsmuster derart realisieren, daß die unterschiedlichen Mechanismen für die Kategorisierung in unterschiedlichen Komponenten mit gleichen Interfaces realisiert werden. Die Verwendung der Komponenten kann dann zur Laufzeit über den Austausch der Referenz konfiguriert werden.

Die Komponenten zur Gewichtung der Dokumente sollen nur über ihre Schnittstelle definiert werden, um größtmögliche Flexibilität zu erreichen. Sie exportieren nur eine Funktion und werden über einen POST-Request aktiviert. Aktive Dokumente, welche mit den Mitteln des AHDM realisiert werden, implementieren dazu Funktionen, welche das bei der Definition der Verhaltensbeschreibung und der Laufzeitumgebung (Unterkapitel 13.4 und 13.6) eingeführte onpost-Ereignis behandeln. Falls die Funktionalität nicht von aktiven Hypertextdokumenten nach dem AHDM implementiert werden soll, so bestehen die Komponenten aus CGI-Skripten, welche per POST-Request aktiviert werden. Die Parameter für den Funktionsaufruf werden bei der Definition der Kommunikationsstrategie näher erläutert.

Im folgenden soll davon ausgegangen werden, daß eine Beispielimplementierung des Gewichtungsdokuments realisiert wird, welches ein Wurzelement mit dem Namen docscaler besitzt. Diesem ist ein AHD-Funktionselement untergeordnet, dessen Namensattribut den Wert scale_doc hat. Des weiteren wird eine zweite Funktion implementiert, welche auf das onpost-Ereignis reagiert und daraufhin die Funktion scale_doc aufruft. So ist das Gewichtungsdokument auch über einen allgemeinen POST-Request ansprechbar.

22.3 Gruppierung zusammengehöriger Komponenten

Falls die Funktionalität zur Gewichtung eines Dokuments noch nicht durch bestehende Komponenten realisiert ist, bietet sich an, die neu zu implementierende Komponente zusammen mit dem Bookmark-Dokument zu gruppieren.

22.4 Definition der Verteilung der Komponenten

Durch die Gruppierung von Bookmark-Dokument und Gewichtungs-Dokument ist festgelegt, daß beide Komponenten in einem Netzknoten gehalten werden. Die referenzierten Dokumente sind netzweit verteilt.

22.5 Definition der Kommunikationsstrategie

Die Kommunikation mit und zwischen den Bestandteilen erstreckt sich einerseits auf das Hinzufügen, Ändern und Löschen von Bookmarks und andererseits auf den Aufruf der Gewichtungsfunktion. Beide werden über HTTP-POST-Requests abgewickelt, wobei beide Komponenten parallel mehrere Requests bearbeiten können müssen. Da im Falle des Bookmark-Dokuments ein Request zur Modifikation des Dokuments führt, müssen Maßnahmen zum Schutz vor gleichzeitigem Zugriff erfolgen. Die Funktionen, welche die beiden Komponenten exportieren, zeigt Tabelle 13.

Funktion	Beschreibung
add_bookmark	Fügt ein Bookmark entweder automatisch oder über die Angabe der Zielkategorie ein. Falls das Bookmark unter dem angegebenen Titel schon im Dokument vorhanden ist, wird es ersetzt und dabei gegebenenfalls in einer neuen Kategorie eingeordnet (von Bookmark-Dokument exportiert)
del_bookmark	Löscht ein Bookmark (von Bookmark-Dokument exportiert)
scale_doc	Gewichtet ein Dokument gegen einen Satz von Schlüsselworten (Funktion wird vom Gewichtungsdokument exportiert)

Tabelle 13: Benötigte Funktionen in der Bookmark-Anwendung

Die ausgetauschten Daten werden dabei über die Funktionsparameter definiert, welche nachfolgend präzisiert werden sollen. Tabelle 14 enthält die Parameter der Funktion add_bookmark, Tabelle 15 definiert die Parameter der Funktion del_bookmark. Beide Funktionen geben eine textuelle Repräsentation des modifizierten Dokuments zurück.

Parameter	Beschreibung
label	Titel, den das Bookmark erhalten soll
address	Internet-Adresse für das Bookmark
category	Titel der Kategorie, falls das Bookmark nicht automatisch eingeordnet werden soll bzw. als bereits bestehendes Bookmark in eine andere Kategorie einzuordnen ist

Tabelle 14: Parameter der Funktion add_bookmark

Parameter	Beschreibung
label	Title des zu löschenden Bookmarks

Tabelle 15: Parameter der Funktion del_bookmark

Die Parameter für die Gewichtungsfunktion scale_doc sind in Tabelle 16 beschrieben.

Parameter	Beschreibung
address	Internet-Adresse des zu gewichtenden Dokuments
keywords	Liste von Schlüsselworten (durch Whitespace getrennt), welche eine Kategorie näher charakterisieren

Tabelle 16: Parameter der Gewichtungsfunktion doc_scale

Der Rückgabewert der Funktion ist ein Wert zwischen 0 und 100, welcher ausdrückt, in welchem Maße das Dokument durch die übergebenen Schlüsselworte charakterisiert ist.

Aufgrund der bis hierher definierten Systemkomponenten und -funktionen kann eine erste Architektur entworfen werden. Sie ist in Abbildung 79 skizziert, wobei die Darstellung nur den groben Aufbau wiedergibt. Die genaue Realisierung der Bestandteile hängt von den eingesetzten DTDs und ihrer Verbindung ab und wird im nächsten Unterkapitel besprochen.

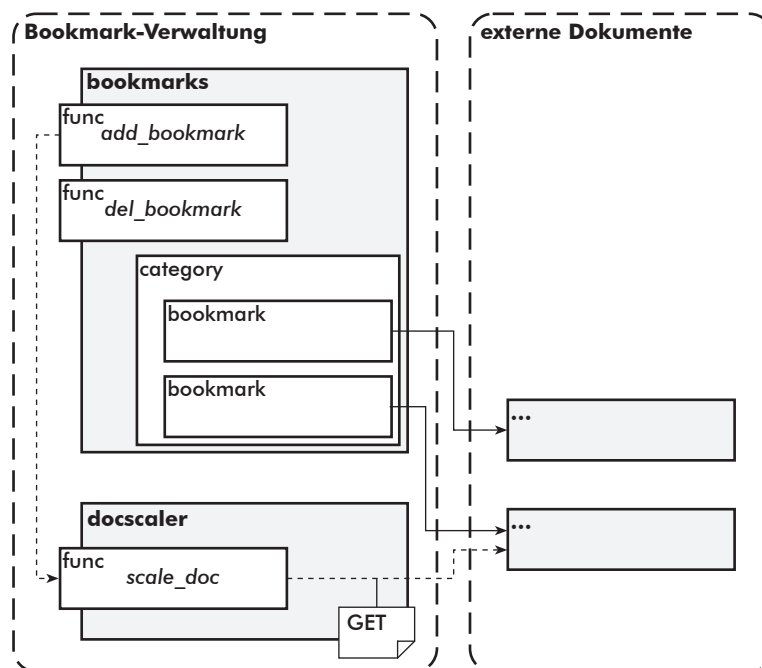


Abb. 79: Überblick über die Architektur der Bookmark-Verwaltung

22.6 Realisierung der Komponenten

Auch die Bookmark-Verwaltung enthält mit dem Bookmark-Dokument eine Komponente, die durch den Endbenutzer direkt verwendet werden kann. Deshalb wird für das Bookmark-Dokument eine Präsentationsschicht realisiert.

Präsentationsschicht

Für die Präsentation des Bookmark-Dokuments soll der Namensraum der Präsentationselemente sowie der zugehörige Stylesheet aus der vorigen Anwendung wiederverwendet werden. An dieser Stelle sind allerdings weniger Präsentationselemente notwendig, da das Bookmark-Dokument eine bereits präsentationsfähige Struktur aufweist. So sind beispielsweise das category- und bookmark-Element durch den vordefinierten Style list darstellbar, so daß Bookmarks und untergeordnete Kategorien eingerückt dargestellt werden. Ähnlich können die title- und label-Elemente mit den Styles label und heading versehen werden. Den Quelltext und das resultierende Aussehen eines Teils des Dokuments zeigt Abbildung 80.

```
...
<p:heading>Bookmarks</p:heading>
<category>
  <title>XML-Parser</title>
  <keywordlist> ... </keywordlist>
  <bookmark>
    <label>Expat</label>
    <address>
      http://www.jclark.com/xml/expat.html
    </address>
  </bookmark>
  <bookmark>
    <label>XML4C</label>
  </bookmark>
...
```

Bookmarks

XML-Parser

Expat
XML 4C

Abb. 80: Teil des angezeigten Bookmark-Dokuments

Funktionalität

Die benötigte Funktionalität wird mit den Funktionen `add_bookmark` und `del_bookmark` im Bookmark-Dokument sowie `scale_doc` im Gewichtungs-Dokument realisiert.

add_document

Die Funktion zum Hinzufügen eines neuen Dokuments wird in drei Schritten durchgeführt: zuerst wird ein eventuell vorhandenes Bookmark mit dem gleichen Titel (also dem gleichen Inhalt des label-Elements) entfernt, danach wird die neue Einfügeposition in der passenden Zielkategorie gesucht und schließlich wird an dieser Position ein neues Bookmark-Element eingefügt.

Da der erste Schritt des Entfernens eines bestehenden Bookmarks in der Funktion `del_document` wiederverwendet werden kann, wird hier auf die Möglichkeit, in der Laufzeitumgebung Definitionen von nativen Funktionen in den einzelnen Skript-Fragmenten in späteren Funktionsaktivierung zu nutzen, Gebrauch gemacht. Dazu werden in dieser Anwendung bei der Reaktion auf das `onload`-Ereignis die entsprechenden nativen Funktionen definiert. Eine weitere Möglichkeit wäre die Implementierung solcher wiederzuverwendenden Funktionen als AHD-Funktionen, doch dagegen sprechen zwei Argumente:

- AHD-Funktionen sollten dann implementiert werden, wenn sie auch von externer Seite her genutzt werden, d.h. ihre Wiederverwendung ist hauptsächlich auf die externe Nutzung ausgerichtet.
- Die Parameter, welche an AHD-Funktionen übergeben werden können, schließen alle Objekte wie Zeiger oder Referenzen aus, die nur innerhalb einer Funktionsausführung Gültigkeit haben (dies ist im Detail in Unterkapitel 13.7 bei der Definition des Kommunikationsmodells erläutert).

Im zweiten Schritt sind u.a. die Schlüsselworte jeder Kategorie zu ermitteln. Da dies eine wiederkehrende Aufgabe ist, wird auch hier eine native Funktion benutzt, so daß sich schließlich eine `onload`-Funktion gemäß Abbildung 81 ergibt.

```
<ahd:func name="onload" type="text/tcl"><![CDATA[
    proc get_cat_keywords {cat} {
        set list [ElementGetElementsByTagName $cat \
            keywordlist]
        set result [ElementGetContents \
            [NodeListItem $list 0]]
        NodeListDelete $list
        return $result
    }

    proc del_bookmark {doc label} {
        set found {}
        set list [DocumentGetElementsByTagName $doc label]
        for {set i 0} {$i < [NodeListGetLength $list]} {incr i} {
            if {$label == [ElementGetContents \
                [NodeListItem $list $i]]} {
                set found [NodeListItem $list $i]
            }
        }
        NodeListDelete $list
        if {$found != {}} {
            NodeDelete [NodeGetParentNode $found]
        }
    }
}]></ahd:func>
```

Abb. 81: `onload`-Funktion im Bookmark-Dokument

Die native Tcl-Funktion `del_document` kann so direkt in den AHD-Funktionen genutzt werden. In der Funktion `add_document` wird sie zu Beginn aufgerufen. Danach wird im zweiten Schritt die Zielkategorie für das neue Bookmark-Element gesucht. In Abbildung 82 ist ein Auszug aus der Funktion dargestellt, die diesen Schritt realisiert.

```
...
set max_score 0
set target_cat {}
set list [DocumentGetElementsByTagName $ahd_document category]
for {set i 0} {$i < [NodeListGetLength $list]} {incr i} {
    set keywords [get_cat_keywords [NodeListGetItem $list $i]]
    set score [AHDRuntimeCall $ahd_ahd $ahd_element \
        scale_doc address=$address&keywords=$keywords]
    if {$score > $max_score} {
        set max_score $score
        set target_cat [NodeListGetItem $list $i]
    }
}
NodeListDelete $list
...
```

Abb. 82: Code-Fragment für das Ermitteln der Zielkategorie

Zu beachten ist bei der Ermittlung des Gewichts eines Dokuments gegenüber einer Kategorie, daß der Aufruf der Funktion `scale_doc` über remote delegation erfolgen soll. Dazu existiert als direktes Kind des Wurzelements ein `delegate`-Element, welches auf das Gewichtung-Dokument verweist. Gemäß dem Strategie-Muster wäre es nun auf einfache Art und Weise möglich, diese Referenz zu modifizieren, so daß eine andere Komponente die Gewichtung durchführen kann.

Das Einfügen der Bookmark-Elemente im dritten Schritt wird ähnlich durchgeführt wie das Einfügen neuer Kontakt-Elemente im vorigen Beispiel, also hauptsächlich über DOM-Funktionen zum Erzeugen und Einfügen von Elementen.

del_document

Um Bookmarks zu löschen, kann die AHD-Funktion `del_document` direkt auf die nativ definierte Funktion gleichen Namens zurückgreifen.

scale_doc

Die Gewichtung eines Dokuments anhand von Schlüsselworten wird in der Funktion `scale_doc` über einen sehr einfachen Algorithmus realisiert, welcher Worthäufigkeiten aufsummiert. Wie aber bereits weiter oben angesprochen, ist es beabsichtigt, hier unterschiedliche Gewichtungssverfahren durch Veränderung der remote delegation einzusetzen.

Die hier eingesetzte Gewichtungsfunktion lädt zuerst das einzuordnende Dokument, extrahiert dann die Text-Elemente daraus und kumuliert schließlich die Häufigkeiten der übergebenen Schlüsselworte. Das Extrahieren der Text-Elemente eines beliebigen Knotens innerhalb eines XML-Dokuments soll dabei über eine nativ definierte Funktion erfolgen. Die Gewichtungsfunktion zeigt Abbildung 83.

22.7 Erweiterungen

Wie im Verlauf des Kapitels schon angeführt, ergeben sich dadurch, daß die Bookmark-Verwaltung als kooperative Anwendung realisiert ist, einige Erweiterungsmöglichkeiten. Diese und andere Erweiterungen sollen im folgenden kurz skizziert werden.

```
<ahd:func name="scale_doc" type="text/tcl"><![CDATA[
    set doc [AHDRuntimeGet $ahd_element $address]
    set text ""
    extractTextElements $doc text
    regsub -all {[^a-zA-ZüöäüÖÄß0-9]} $text { } text
    set text [string tolower $text]

    foreach word $keywords {set count($word) 0}
    set score 0
    foreach word $text {
        if {[info exists count($word)]} {
            if {$count($word) > 0} {set d 1} else {set d 5}
            incr count($word) $d
            incr score $d
        }
    }
    set ahd_result $score
}]></ahd:func>
```

Abb. 83: Gewichtungsfunktion scale_doc

Erweiterung der Benutzerschnittstelle

Ein mögliches Gebiet für Erweiterungen ist die Präsentationsschicht der Bookmark-Verwaltung. In der bisherigen Ausführung dient das Dokument lediglich der Anzeige sowie der Modifikation über externe Funktionsaufrufe. Allerdings ist es auch denkbar, das Bookmark-Dokument wie auch das Kundenkontakt-Dokument direkt per Benutzerinteraktion zu ändern.

Direkte Benutzerinteraktion

Um eine Modifikation des Benutzers über die Präsentationsschicht zu ermöglichen, sind zwei Änderungen denkbar: einerseits die Änderung der Daten innerhalb der Bookmark-Elemente (Titel und Internet-Adresse), andererseits das Hinzufügen und Löschen von Bookmark-Elementen.

Die erste Änderung ließe sich beispielsweise über das Eingabefeld-Muster realisieren, wobei zusätzliche Elemente aus dem Namensraum für Präsentationselemente notwendig wären. Analog zu Abbildung 76 könnte ein Bookmark-Element entsprechend Abbildung 84 editierbar gemacht werden.

Diese Lösung ist allerdings nicht optimal, da dadurch dem Dokument eine große Anzahl redundanter Elemente hinzugefügt wird (hauptsächlich die Beschreibungselemente für das Titel- und Adress-Element). Besser wäre sicherlich eine Struktur, die ein Editieren ohne zusätzliche Felder ermöglicht. Dies könnte etwa durch Initiieren der Modifikation durch Klicken auf den Elementinhalt und nicht auf ein Beschreibungsfeld sowie durch Beenden des Editierens durch Klicken auf ein nur temporär sichtbares Bestätigungsfeld realisiert werden.

Auch die Funktionalität zum Hinzufügen neuer Bookmark-Elemente könnte ähnlich wie bei der Kontaktmanager-Anwendung erfolgen. Denkbar ist etwa, bei jedem Kategorie-Element einen entsprechenden Button hinzuzufügen, der in dieser Kategorie ein neues leeres Bookmark-Element anlegt. Zum Löschen wäre allerdings ein Button zu jedem Bookmark notwendig. Dies zeigt eine Problematik, die die bis jetzt vorgestellten Strukturen bei der Interaktion mit dem Benutzer aufwerfen. Die Strukturen

```
...
<bookmark xmlns:p="urn:AHDS:General/presentation.dtd">
  <p:field role="field">
    <p:label role="label">Titel: </p:label>
    <label role="entry">Expat</label>
  </p:field>
  <p:field role="field">
    <p:label role="label">Adresse: </p:label>
    <address role="entry">
      http://www.jclark.com/xml/expat.html
    </address>
  </p:field>
</bookmark>
...
```

Abb. 84: Nutzung des Eingabefeld-Musters für Bookmark-Elemente

beruhen auf direkter Manipulation, d.h. dem Durchführen von Aktionen in einem Schritt und nicht dem zweistufigen Schema des Auswählens von Elementen, gefolgt vom Auswählen der durchzuführenden Aktion. Es spricht allerdings nichts dagegen, dieses Schema mit AHD-Mitteln zu realisieren und auch in einem wiederverwendbaren Muster festzuhalten.

Verbesserung der Darstellung der Kategorie-Hierarchie

Eine weitere nützliche Verbesserung besteht in einer kollabier- und expandierbaren Darstellung der Kategorien-Hierarchie, so daß beispielsweise Unterkategorien und Bookmarks in einer Kategorie wahlweise angezeigt oder verdeckt werden können.

Eine solche flexible Darstellung ist über das CSS-Modell durch eine dynamische Änderung der display-Eigenschaft von XML-Elementen möglich. Diese Änderung kann durch Klicken auf den Kategorie-Titel ausgelöst werden. Weiterhin sind die untergeordneten Elemente einer Kategorie noch in einem einzigen Element zu kapseln, so daß die Änderung der display-Eigenschaft nur für dieses Element durchgeführt werden muß. Die resultierende Struktur zeigt Abbildung 85.

Kooperative Nutzung

Ein wichtiger Aspekt der Nutzung aktiver Hypertextdokumente nach dem AHDM ist die Interoperabilität, welche dadurch erreicht werden kann. Dieser Aspekt soll an dieser Stelle unter dem Gesichtspunkt einer kooperativen Nutzung untersucht werden, wobei hier die Anwendung so adaptiert wird, daß sie mit anderen Anwendungen kooperieren kann. Grundsätzlich besteht aber auch die Möglichkeit, die anderen Anwendungen so anzupassen, daß eine Kooperation möglich ist. Begünstigt würden solche Anpassungen dadurch, daß XML als Basis des AHDM bereits den vereinfachten Datenaustausch als Ziel hat.

Austausch des Gewichtungsdokuments

Die erste Erweiterung besteht im Austausch des Gewichtungsdokuments durch ein externes Programm. Dazu muß dieses Programm seine Funktionalität über HTTP anbieten, um ein einfaches Einbinden zu ermöglichen.

Als Beispiel wird hier die Einbindung des ActiWeb-Systems (siehe dazu Unterkapitel 12.2) vorgenommen, da es einerseits HTTP als Transportmechanismus unterstützt und andererseits auf einer universell einsetzbaren Programmiersprache basiert.

```

<ahd:func name="onload" type="text/tcl"><![CDATA[
...
proc toggle_display {element} {
    if {[ElementGetCSSProperty $element display] == \
        block} {
        ElementSetCSSProperty $element display none
    } else {
        ElementSetCSSProperty $element display block
    }
}
proc toggle_para {element} {
    if {[ElementGetTagName $element] == category} {
        set list [ElementGetElementsByTagName \
            $element paragraph]
        if {[NodeListGetItem $list 0] != {} {
            toggle_display [NodeListGetItem $list 0]
        }
        NodeListDelete $list
    }
}
...
]]></ahd:func>
...
<category onclick="toggle_para $ahd_element" type="text/tcl">
    <p:paragraph>
        <bookmark>
            ...
        </bookmark>
    </p:paragraph>
</category>
...

```

Abb. 85: Erweiterung für kollabier- und expandierbare Hierarchien

Die Einbindung erfordert eine Anpassung der unterschiedlichen Aufrufkonventionen von AHDM und ActiWeb. So muß der HTTP-POST-Request für die Funktionsausführung im AHDM in einen entsprechenden GET-Request umgewandelt werden. Die Umwandlung soll transparent innerhalb des Bookmark-Dokuments erfolgen. Dafür wird das delegate-Element durch eine Implementierung der scale_doc-Funktion ersetzt, die den Funktionsaufruf an das ActiWeb-Objekt weiterleitet. Durch diese Struktur bleibt die Implementierung aller Funktionen, welche die scale_doc-Funktion über einen Delegations-Mechanismus ansprechen, unberührt. In Abbildung 86 ist diese Struktur und der notwendige Quelltext dargestellt, der Quelltext des ActiWeb-Objektes befindet sich im Anhang.

```

<bookmarks>
    <ahd:func name="scale_doc" type="text/tcl">
        set ahd_result [AHDRuntimeGet $ahd_ahd $ahd_element \
            urn:ActiWeb:Demo/docscaler+scale_doc+$address+\
$keywords]
    </ahd:func>
    ...
</bookmarks>

```

Abb. 86: Einbindung eines ActiWeb-Objekts

Im- und Export von Daten

Obwohl das Bookmark-Dokument durch die Verwendung von XML schon an sich in einem generisch verwendbaren Format vorliegt, kann es dennoch notwendig sein, auch Fremdformate zu unterstützen. Denkbar sind hierfür eine HTML-Repräsentation für die Nutzung des Dokuments in einem HTML-Browser und eine Import-Komponente, die beispielsweise bestehende Bookmark-Dokumente im HTML-Format oder Dokumente und Elemente entsprechend XBEL zu AHD-Bookmark-Dokumenten konvertiert.

Für den Export des Bookmark-Dokuments ins Netscape- oder Internet-Explorer-Format ist ein HTML-Dokument entsprechend der DTD aus Abbildung 87 zu erzeugen (die Dokumente besitzen allerdings fehlerhafterweise kein Wurzelement).

```
<!ELEMENT NETSCAPE-Bookmark-file-1 (TITLE, H1, DL?)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT A (#PCDATA)>
<!ATTLIST A
    HREF CDATA #REQUIRED>
<!ELEMENT H1 (#PCDATA)>
<!ELEMENT H3 (#PCDATA)>
<!ELEMENT DL (p, ((DL, p) | DT)*)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT DT (A | H3)*>
```

Abb. 87: DTD für Netscape- und Internet-Explorer-Bookmarks

Bei der Konvertierung werden category-Elemente als DL-Elemente ausgegeben, welche wiederum DL-Elemente für Kategorien oder DT-Elemente, entsprechend den bookmark-Elementen, enthalten. In den Bookmark-Elementen werden die label-Elemente zu A-Elementen, deren HREF-Attribut den Inhalt des address-Elements als Wert hat.

Für den Import von Daten sind die Eingangsdaten entsprechend der DTD aus Abbildung 78 zu konvertieren.

Beide Konvertierungswege können entweder als Bestandteil eines AHD-Bookmark-Dokuments oder als eigenständige Programme realisiert werden. Die Integration in ein Bookmark-Dokument erleichtert allerdings den Datenaustausch, da so keine zusätzlichen Komponenten genutzt werden müssen.

22.8 Auswertung

Die Bookmark-Anwendung zeigt, wie eine kooperative Anwendung mit den Mitteln des AHDM realisiert werden kann. Dazu wurde insbesondere das Kommunikationsmodell des AHDM genutzt, um einfach strukturierte Daten zwischen unterschiedlichen Komponenten auszutauschen. Diese Komponenten können wie gezeigt entweder aktive Hypertextdokumente oder andere Ressourcen sein. Wichtig bei der Einbindung ist lediglich eine HTTP-Unterstützung auf allen Seiten.

Wiederverwendung

Weiterhin konnten bestimmte Muster und Bestandteile aus der Kontaktmanager-Anwendung wiederverwendet werden. Dabei handelte es sich hauptsächlich um Bestandteile, die die Präsentation von aktiven Dokumenten sowie ihre Modifikation durch den Endbenutzer betrafen. Ein wichtiger Mechanismus hierbei waren die

XML-Namensräume, die ein Zusammenführen der unterschiedlichen Informationsarten in einem AHD ermöglichen. Dazu kam die Adressierung von Elementen nicht über den Elementnamen sondern über Attribute, so daß eine Umstrukturierung eines aktiven Dokuments nicht zwangsläufig eine Modifikation der Programmbestandteile zur Folge hat.

Das Konzept der Delegation in Form von parent und remote delegation schließlich erleichtert die Rekonfiguration von Komponenten durch implizite Adressierung aufrufender Funktionen. So konnte im Bookmark-Dokument die Funktionalität zur Gewichtung von Dokumenten einerseits durch parent delegation so verändert werden, daß ein ActiWeb-Objekt als externe Resource angesprochen wurde. Andererseits kann mit dem delegate-Element auf unterschiedliche externe Komponenten verwiesen werden, soweit sie der Aufrufkonvention des AHDM folgen.

Der Übergang zur nächsten Kategorie von möglichen Systemarchitekturen im AHDM ergibt sich aus der weiteren Öffnung der Systeme für externe Anwendungen, der Nutzung von XML auch für die Nachrichteninhalte und der Nutzung von mobilen aktiven Dokumenten. Ein Beispiel für solche Systeme wird im nächsten Kapitel realisiert.

Verwendete Muster

Die Basis für die Bookmark-Verwaltung liefert die vernetzte, kooperative Anwendung, gebildet aus vernetzten AHDs (*Networked Active Documents*). Dabei wurde eine Aufteilung gemäß unterschiedlicher Funktionalitäten der Dokumente vorgenommen. In diesem Zusammenhang kann das *Strategy*-Muster eingesetzt werden, um zwischen unterschiedlichen Bewertungsfunktionen bei der Analyse eines einzuordnenden Dokuments zu wechseln. Bei der Erweiterung der Bookmark-Verwaltung kann auf das Eingabefeld-Muster zurückgegriffen werden, allerdings erscheint dessen Nutzung durch die hohe Zahl von potentiellen Eingabefeldern in einem Bookmark-Dokument nicht ratsam.

23 Beschaffungsprozeß

Die Gruppe der Systeme, die mobile aktive Hypertextdokumente nutzen, lassen sich auch als agentenbasierte Systeme bezeichnen. Agenten sind dabei eigenständige aktive Dokumente, welche ihren Standort wechseln, mit anderen Systemkomponenten kommunizieren und Informationen zwischen Komponenten transportieren können. Anhand solcher Systeme können in Zusammenhang mit dem AHDM eine Reihe interessanter Punkte gezeigt werden. Darunter ist beispielsweise die Nutzung von Matching-Mechanismen für die implizite Kommunikation, Einsatz von Koordinationsmechanismen für aktive Dokumente oder die Nutzung der Eigenschaften von AHDs zum Datenaustausch in offenen Systemen. Das in diesem Zusammenhang verwendbare Muster ist das Muster für *Mobile Active Documents*.

Die in diesem Kapitel vorgestellte Anwendung von aktiven Hypertextdokumenten für den Ablauf von Beschaffungsprozessen soll diese Aspekte demonstrieren. Dabei wird ein Geschäftsprozeß dezentral über einzelne AHDs realisiert, ohne eine zentrale Instanz zur Workflow-Steuerung zu benötigen. So soll gezeigt werden, wie das AHDM genutzt werden kann, um mehr Flexibilität und Autarkie beim Abwickeln von Geschäftsprozessen zu erreichen.

Auch bei dieser Anwendung werden nur Auszüge präsentiert, der vollständige Quelltext befindet sich im Anhang.

23.1 Aufgabenbeschreibung

Der abzubildende Geschäftsprozeß ist an den Ablauf bei der Beschaffung von Büchern im Universitätsbereich angelehnt, läßt sich aber auf andere Prozesse übertragen. An der Abwicklung des Prozesses sind Personen in vier Rollen beteiligt. Diese Rollen sind der Besteller, der Koordinator, der Entscheider und der Einkäufer. Der Vorgang selbst besteht aus mehreren Schritten (siehe dazu auch Abbildung 88):

1. Ein Besteller (dies ist meist ein Mitarbeiter, es können aber auch andere Personen wie Studenten eine Beschaffung initiieren) stellt einen Beschaffungsvorschlag zusammen und gibt ihn an einen Koordinator weiter.
2. Der Koordinator sammelt Beschaffungsvorschläge einer Gruppe von Bestellern (beispielsweise an einem Lehrstuhl), ergänzt fehlende Informationen und übermittelt einen Block von Beschaffungsanträgen an den Entscheider.
3. Der Entscheider prüft, ob die beantragten Beschaffungen durchgeführt werden können oder ob es Gründe dagegen gibt, wie beispielsweise Doppelbeschaffungen oder Budgetüberschreitungen. Die genehmigten Beschaffungsanträge werden an den Einkäufer weitergeleitet.
4. Der Einkäufer beschafft die beantragten Bücher und überwacht dabei den Lieferstatus. Sobald die Literatur eingetroffen ist, wird eine Benachrichtigung an den Entscheider, den Koordinator und den Besteller weiterleitet.

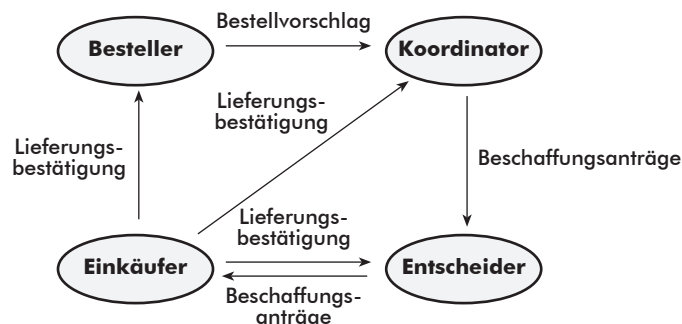


Abb. 88: Ablauf einer Literaturbestellung

Das zu erstellende System soll zudem die Rollen flexibel bestimmten Personen zuordnen und auch Änderungen der Zuordnung unterstützen.

23.2 Strukturierung der Information

Um die Information, welche dieser Anwendung zugrundeliegt, zu strukturieren, ist aufgrund der erhöhten Komplexität des Systems zuerst eine Aufstellung aller potentiellen Informationselemente sinnvoll.

Offensichtlich ist die Notwendigkeit zur Beschreibung der zu bestellenden Bücher. Sie werden innerhalb einer Bestellung genutzt, wobei es zwei unterschiedliche Beschreibungsmöglichkeiten für eine Bestellung gibt: einerseits den Bestellvorschlag, andererseits den formalen Beschaffungsantrag. Beschaffungsanträge selbst sollen in Listen gruppiert werden können. Weiterhin werden Buchinformationen bei der Lieferbestätigung benötigt.

Interessant ist allerdings die Frage, wie die unterschiedlichen Rollen und die beteiligten Personen innerhalb des Systems mit Informationselementen beschrieben werden. Es ergibt sich somit folgende Liste potentieller Informationselemente:

- Rollen (Besteller, Koordinator, Entscheider, Einkäufer)
- Personen, die eine oder mehrere Rollen einnehmen können und denen Informationen übermittelt werden können
- Buchdaten wie Titel, Autor, Verlag usw.
- Bestellvorschlag, enthält Buchdaten, die nicht unbedingt vollständig sein müssen
- Beschaffungsantrag mit Buchdaten
- Liste von Beschaffungsanträgen
- Lieferbestätigung für eingetroffene Bücher
- Funktionskomponenten, welche die Aufgaben von Besteller, Koordinator, Entscheider und Einkäufer unterstützen

Die potentiellen Informationselemente sollen in einer Bottom-Up-Analyse weiter präzisiert werden, wobei mit den Personendaten und Rollen begonnen wird.

Personendaten und Rollen

Wie bereits oben aufgeführt, nehmen die beteiligten Personen bestimmte Rollen ein und sind für Informationsübermittlungen erreichbar. Das Konzept der Rolle soll an dieser Stelle nicht näher vertieft werden, d.h. die Rollen werden einer Person nur implizit durch die Teilnahme am Beschaffungsprozeß zugeordnet. Eine Erweiterungsmöglichkeit bestünde in der Verwaltung und expliziten Zuordnung von Rollen sowie der Steuerung der Informationsweiterleitung über Rollen.

Zudem sollen auch die Personen nur implizit innerhalb der Anwendung mit einem Ablauf verbunden sein. Es reicht hier vorerst aus, nur die Zielpunkte für die Kommunikation näher zu beschreiben. Dazu soll an den entsprechenden Stellen nur ein URI benutzt werden, welcher zu einem URL aufgelöst werden kann und so ein Übertragungsprotokoll und einen Zielort beschreibt.

Buchdaten

Die Buchdaten, welche für einen Bestellvorschlag und einen Beschaffungsantrag benötigt werden, sollen in einem gemeinsamen Informationselement zusammengefaßt werden. Dazu soll auf den Vorschlägen der Dublin-Core-Workshops aufgebaut werden, welche Metadaten zur Beschreibung von Publikationen umfassen [186]. Im einzelnen werden dazu die in Tabelle 17 beschriebenen Elemente benutzt. Die daraus abgeleitete DTD ist im Anhang zu finden.

Element	Beschreibung
title	Titel der Publikation
creator	Ein oder mehrere Autoren
subject	Nähere Beschreibung der Publikation mit Schlüsselworten oder Phrasen
description	Inhaltsbeschreibung
publisher	Veröffentlichende Institution (z.B. Verlag, Firma, Universität)
contributor	Personen, die zur Erstellung der Publikation beigetragen haben

Element	Beschreibung
date	Datum der Publikation
type	Typ der Publikation (Buch, Artikel, usw.)
format	Format (Buch, Zeitschrift, CD-ROM, ...)
identifizier	ISBN- oder ISSN-Nummer, möglicherweise auch andere zur eindeutigen Identifikation nutzbaren Merkmale (Bestellnummern usw.)
source	Angabe einer Quelle, von der diese Publikation abgeleitet ist
language	Sprache, in der die Publikation verfaßt ist
relation	Beziehung zwischen der beschriebenen Publikation und einer im source-Element beschriebenen Originalquelle
coverage	Zeitraum oder räumlicher Bereich, den die Publikation beschreibt
rights	Nähere Beschreibung von Nutzungs- und weiteren Rechten

Tabelle 17: Elemente der Dublin-Core-Metadaten

Zwar können sicherlich nicht alle Elemente im Rahmen einer Literaturbeschaffung sinnvoll eingesetzt werden, allerdings erhöht sich durch die Verwendung eines verbreiteten und akzeptierten Schemas wie Dublin Core die Möglichkeit der Nutzung externen Ressourcen. So ist es denkbar, daß Verlage Dublin-Core-konforme Beschreibungen von Publikationen auf elektronischem Wege zur Verfügung stellen, so daß eine Resource eindeutig beschrieben ist und die Beschaffung vereinfacht wird.

Bestellvorschlag

Der Vorschlag für eine Bestellung enthält zum einen die Elemente für die Buchdaten, zum anderen Informationen über den Besteller. Die Buchdaten entsprechen im Grunde den Dublin-Code-Metadaten, welche in ein zusätzliches Element eingebettet werden. Die Information über den Besteller soll nur die Adresse für die Übermittlung der Lieferbestellung umfassen. Die Elemente sind in Tabelle 18 beschrieben.

Element	Beschreibung
orderproposal	Wurzelement für den Bestellvorschlag, enthält die nachfolgenden Buch- und Bestellerdaten
book	Beschreibung des zu bestellenden Buches, ausgedrückt mit Hilfe der Dublin-Core-Metadaten als Unterelemente
price	Preis des Buches
count	Anzahl der zu bestellenden Exemplare
urgency	Dringlichkeit der Bestellung
comments	Kommentare zum Bestellvorschlag
orderer	Ziel für die Übermittlung der Lieferbenachrichtigung (URI oder URL)

Tabelle 18: Elemente für den Bestellvorschlag

Abbildung 89 zeigt schließlich die verwendete DTD ohne die Dublin-Core-Elemente.

```
<!ELEMENT orderproposal (book, price?, count, urgency?,
                        comments?, orderer?)>
<!ELEMENT book (title, creator, subject?, description?,
                publisher, contributor?, date?, type?, format?,
                identifier, source?, language?, relation?,
                coverage?, rights?)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT count (#PCDATA)>
<!ELEMENT urgency (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT orderer (#PCDATA)>
```

Abb. 89: DTD des Bestellvorschlag-Dokuments (ohne Dublin-Core-Elemente)

Die DTD verdeutlicht ein Manko bei der Verwendung von XML: Es fehlen Mechanismen, um die Inhalte der Elemente formal näher zu beschreiben und einzuschränken. So sind beispielsweise die Elemente für die Preis- oder Mengenangabe nur unzureichend präzisiert, d.h. eine Einschränkung der Elementinhalte auf positive ganze Zahlen oder Währungsbeträge kann nicht formuliert werden. Diese Problematik kann aber durch den Einsatz von entsprechenden Metadaten zusätzlich oder gar anstelle der DTD vermieden werden. Allerdings befinden sich die dazu notwendigen Mechanismen wie der RDF- sowie der XML-Schema-Standard ([34] und [22]) noch in der Spezifizierungsphase.

Beschaffungsantrag

Der Beschaffungsantrag wird vom Koordinator aus einem Bestellvorschlag erzeugt. Dazu wird fehlende oder unvollständige Information ergänzt. Ein Beschaffungsantrag besitzt eine ähnliche Struktur wie ein Bestellvorschlag. Das Wurzelement heißt nun `order` statt `orderproposal`, zusätzlich wird wie auch beim Besteller noch eine URI angegeben, unter der dem Koordinator Informationen übermittelt werden können. Da der Beschaffungsantrag in dieser Form auch vom Einkäufer verarbeitet werden soll, wird noch ein Element benötigt, um dem Entscheider eine Lieferbestätigung zukommen zu lassen. Alle anderen Elemente können aus der DTD aus Abbildung 89 übernommen werden. Die DTD zeigt Abbildung 90.

```
<!ELEMENT order (book, price?, count, urgency?,
                comments?, orderer?, coordinator?, decider?)>
<!ELEMENT coordinator (#PCDATA)>
<!ELEMENT decider (#PCDATA)>
```

Abb. 90: Element-Definitionen für den Beschaffungsantrag

Antragsliste

Auch die Liste der Beschaffungsanträge baut auf bestehenden Elementen auf. In der DTD der Antragsliste umfaßt das Wurzelement mit dem Namen `orderlist` nur eine Anzahl von `order`-Elementen.

Lieferbestätigung

Die Lieferbestätigung beinhaltet Informationen über das eingetroffene Buch und die Anzahl der gelieferten Bücher. Dazu werden die in der Bestellvorschlag-DTD definierten Elemente `book` und `count` in ein `delivery`-Element eingebettet, welches das

Wurzelement der Lieferbestätigung ist. Hinzu kommen Informationen über die an der Bestellung beteiligten Personen ähnlich wie beim Beschaffungsantrag, so daß ein delivery-Element gemäß der DTD aus Abbildung 91 zusammengesetzt ist.

```
<!ELEMENT delivery (book, count,  
                    orderer?, coordinator?, decider?)>
```

Abb. 91: Deklaration des delivery-Elements

Funktionskomponenten

Neben den reinen Informationselementen sollen auch die Komponenten näher beschrieben werden, die die Aufgaben der beteiligten Personen unterstützen. Diese Komponenten haben allerdings nicht die Bedeutung der Informationselemente, da eine Buchbestellung unabhängig von der Implementierung von Einzelabläufen durchgeführt werden können soll.

Um die Funktionskomponenten zu spezifizieren, sind zuerst die Aufgaben innerhalb der Anwendung anhand der Aufgabenbeschreibung näher zu untersuchen. So sind mögliche Bereiche für eine Unterstützung durch das zu implementierende System bei folgenden Punkten möglich:

1. Die Erstellung von Anschaffungsvorschlägen durch den Besteller könnte durch eine entsprechende aktive Komponente mit Präsentationsschicht erleichtert werden, ebenso wie das Versenden von Anschaffungsvorschlägen. Auch eine Benachrichtigungskomponente zur Visualisierung eintreffender Lieferbestätigungen kann hier realisiert werden.
2. Weiterhin automatisch realisieren ließe sich das Bündeln von Bestellvorschlägen und das Aussortieren unvollständiger Vorschläge im Aufgabenbereich des Koordinators, gefolgt vom Konvertieren zu gültigen Beschaffungsanträgen und Versenden zum Entscheider. Zudem sind vom Einkäufer eintreffende Lieferbestätigungen an den ursprünglichen Besteller zu übermitteln.
3. Beim Entscheider sind Beschaffungsanträge nach Prüfung freizugeben und an den Einkäufer weiterzuleiten, wie auch eintreffende Lieferbestätigungen an den Koordinator.
4. Der Einkäufer könnte durch eine Komponente zum Generieren einer Lieferbestätigung aus einem erfolgreich durchgeführten Beschaffungsantrag unterstützt werden.

Aus diesen möglichen Funktionalitäten sollen nun eine Reihe von unterstützenden Komponenten gebildet werden.

Erzeugen einer Bestellung

Das Generieren einer Bestellung kann im Grunde auf unterschiedlichste Arten erfolgen, solange der Bestellungs-vorschlag der dafür definierten DTD entspricht. So sind dedizierte Programme oder generelle XML-Editoren einsetzbar. Hier wird als Beispiel eine einfache Komponente implementiert, welche zusätzlich noch die Übermittlung des Bestellvorschlags an den Koordinator übernimmt. Das Wurzelement wird mit `propgen` benannt, welchem direkt ein `orderproposal`-Element untergeordnet ist (siehe Abbildung 92). Die Funktionalität des Editierens soll über das Eingabefeld-Muster hergestellt werden, welches hier mit den Unterelementen des `orderproposal`-Elements benutzt werden soll.

```
<!ELEMENT propgen (orderproposal)>
```

Abb. 92: Wurzelement des Besteller-Dokuments

Visualisierung von Lieferbestätigungen

Die Anzeige von eingegangenen Lieferbestätigungen wird über eine Komponente realisiert, welche eine Anzahl von delivery-Elementen innerhalb des showdelivery-Wurzelementes enthält (Abbildung 93). Zusätzlich sollen noch Elemente aus dem Präsentations-Namensraum sowie der Stylesheet aus der Kontaktmanager-Anwendung eingesetzt werden.

```
<!ELEMENT showdelivery (delivery*)>
```

Abb. 93: Wurzelement des Dokuments zur Visualisierung von Lieferungen

Bündelung und Konvertierung von Bestellvorschlägen

Die Behandlung eintreffender Bestellvorschläge wird über eine Komponente mit dem Wurzelement collectprops abgewickelt. Dabei sollen Bestellvorschläge auf Vollständigkeit geprüft und gesammelt werden, bis eine bestimmte Zahl erreicht wird. Die gültigen Vorschläge sollen dann zu Beschaffungsanträgen konvertiert und als orderlist-Element an den Entscheider übertragen werden. Zusätzlich sollen in diesem Dokument auch die unvollständigen Bestellvorschläge verwaltet werden, wie es Abbildung 94 zeigt.

```
<!ELEMENT collectprops (validprops, invalidprops)>  
<!ELEMENT validprops (orderproposal*)>  
<!ELEMENT invalidprops (orderproposal*)>
```

Abb. 94: Elemente der Komponente zum Sammeln von Bestellvorschlägen

Prüfung und Weiterleitung von Beschaffungsanträgen

Beschaffungsanträge werden beim Entscheider durch eine Komponente gesammelt, geprüft und an den Einkäufer weitergeleitet. Sie soll das Wurzelement collectororders besitzen, wie es Abbildung 95 zeigt. Die Funktionalität zur Prüfung soll im Rahmen dieser Beispielanwendung nicht weiter vertieft werden, weshalb eine Delegation des Prüfverfahrens an eine externe Komponente genutzt wird.

```
<!ELEMENT collectororders (order*)>
```

Abb. 95: Wurzelement der Komponente zur Sammlung von Bestellungen

Erzeugen einer Lieferbestätigung

Das Erzeugen einer Lieferbestätigung soll durch eine eigene Komponente mit dem Wurzelement deliverygen erfolgen, welche eine Liste der aktuellen Beschaffungsanträge (d.h. eine Anzahl order-Elemente, siehe auch Abbildung 96) enthält und aus diesen per Benutzerinteraktion entsprechende Bestätigungen erzeugt.

```
<!ELEMENT deliverygen (order*)>
```

Abb. 96: Wurzelement des Dokuments zum Erzeugen von Lieferbestätigungen

Sammeln von Lieferbestätigungen

Die Lieferbestätigungen, welche vom Einkäufer an den Entscheider und den Koordinator übermittelt werden, sollen von einer eigenen Komponente gesammelt werden, welche das Wurzelement `collectdelivery` besitzt. Die dort gesammelten Lieferbestätigungen stehen so für eine beliebige Weiterverarbeitung zur Verfügung.

Netzwerkknoten und Cluster

Da diese Anwendung einen verteilten Prozeß abbildet, soll an dieser Stelle bereits eine erste Struktur von Netzwerkknoten erstellt werden. Sie orientiert sich eng an der ursprünglichen Aufgabenstellung und sieht vier Gruppen von Netzwerkknoten vor, die jeweils den Rollen der beteiligten Personen entsprechen. Die Netzwerkknoten sollen im folgenden nach den Rollennamen `Besteller`, `Koordinator`, `Entscheider` und `Einkäufer` benannt werden. Sie sollen jeweils durch einen Cluster realisiert werden, der alle notwendigen Dokumente und Komponenten zur Realisierung der gewünschten Funktionalität enthält.

23.3 Gruppierung zusammengehöriger Komponenten

Die bisher gefundenen Komponenten und ihre Zusammengehörigkeit sind noch einmal in Tabelle 19 beschrieben.

Element	Beschreibung
book	Dient der Beschreibung von Literatur im allgemeinen oder Büchern innerhalb dieser Anwendung
orderproposal	Enthält Daten für einen Bestellvorschlag, u.a. zählt dazu ein book-Element
order	Besteht wie auch der Bestellvorschlag u.a. aus einem book-Element und beschreibt ein zu beschaffendes Buch
orderlist	Wird benutzt, um einen Block von order-Elementen an einen Entscheider und Einkäufer zu übermitteln
delivery	Enthält ein book-Element und eine Mengenangabe für die Benachrichtigung beteiligter Personen bei Liefereingang
propgen	Erzeugt per Benutzerinteraktion einen Bestellvorschlag und leitet diesen an einen Koordinator weiter
showdelivery	Zeigt eine Liste eingetroffener Bestellungen an
collectprops	Sammelt, prüft und versendet Bestellvorschläge
collectorders	Sammelt und versendet Beschaffungsanträge zur Unterstützung des Entscheiders, eine Prüfung der Anträge wird delegiert
deliverygen	Erzeugt aus einer Liste von Beschaffungsanträgen per Benutzerinteraktion Lieferbestätigungen
collectdelivery	Sammelt Lieferbestätigungen

Tabelle 19: Wurzelemente der Buchbestellungs-Anwendung

Von diesen Komponenten werden hauptsächlich das book-Element, das order-Element und das delivery-Element in andere Komponenten eingebettet, die anderen Elemente sind jedoch größtenteils voneinander unabhängig verwendbar.

Ordnet man den identifizierten Clustern die Komponenten aus Tabelle 19 zu, so erhält man das in Abbildung 97 dargestellte Bild.

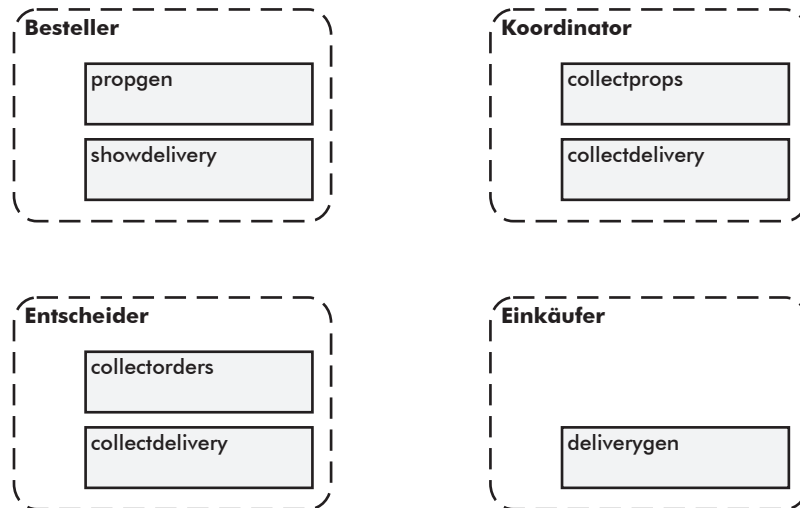


Abb. 97: Verteilung der Komponenten auf die Cluster

Die dort nicht dargestellten Elemente werden im übernächsten Unterkapitel zugeordnet, da sie Bestandteil der Kommunikationsstrategie sind und nicht nur innerhalb eines einzigen Clusters verwendet werden können.

23.4 Definition der Verteilung der Komponenten

Die Cluster sind in der Regel jeweils einem Netzwerkknoten zugeordnet, allerdings kann ein Netzwerkknoten durchaus dazu dienen, die Funktionalitäten zur Unterstützung nicht nur einer Personengruppe zu realisieren und so mehr als einen Cluster enthalten.

Die Zuordnung ist dabei nicht fix, d.h. eine Person kann über einen assoziierten Netzwerkknoten in einer beliebigen Rolle am Beschaffungsprozeß teilnehmen. Eine feste Zuordnung erfolgt nur aufgrund von tatsächlich stattfindender Kommunikation zwischen den Personen.

23.5 Definition der Kommunikationsstrategie

Wie bereits in der Aufgabenbeschreibung und den vorangegangenen Ausführungen erkennbar, werden bei der Abwicklung eines Beschaffungsprozesses unterschiedliche Informationselemente zwischen den beteiligten Personen ausgetauscht. Dazu gehören Bestellvorschläge, Beschaffungsanträge und Lieferbestätigungen.

Entsprechend der bisherigen Analyse sind diese Komponenten passive Informations-träger. Allerdings ist zur Unterstützung der Funktionskomponenten und der automatischen Abwicklung von Teilen des Beschaffungsprozesses eine Ergänzung um aktive Bestandteile notwendig. Insbesondere der Einsatz von Laufzeitumgebungs-Monitoren (beschrieben mit dem *Runtime Environment Monitor* ab Seite 127) bietet

sich an, um die in einem Netzwerkknoten eintreffenden Informationselemente automatisch weiterzuverarbeiten. Dementsprechend sind die mobilen Komponenten um Funktionen zum Kontaktieren eines solchen Monitors in einer Laufzeitumgebung zu ergänzen.

Die Abwicklung der Kommunikation zwischen den Komponenten bei der Bestellung und bei der Benachrichtigung bei Liefereingang zeigt Abbildung 98.

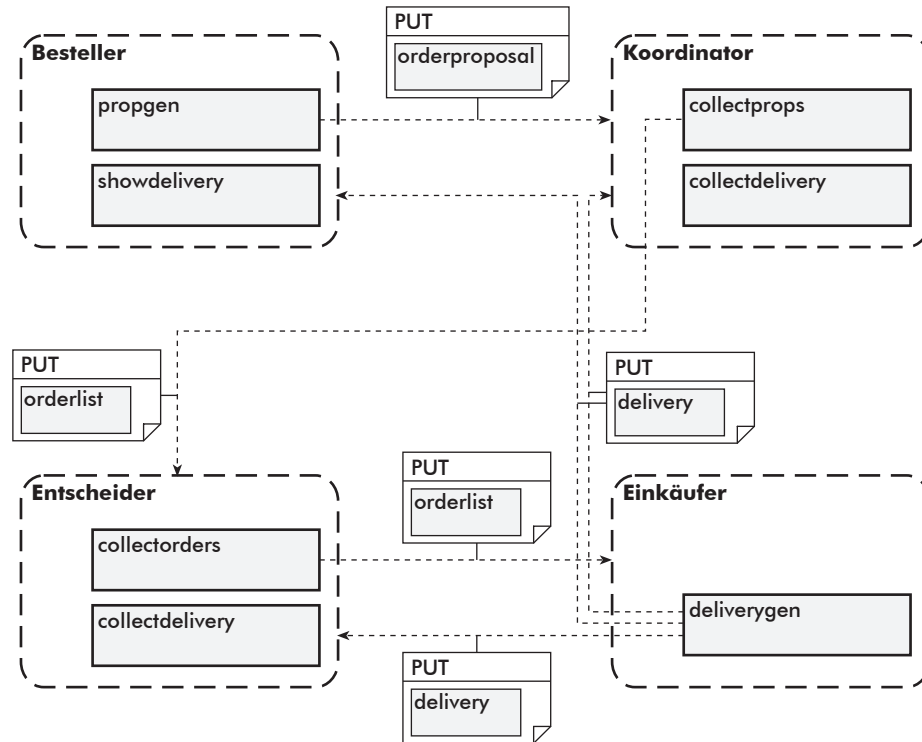


Abb. 98: Ablauf und Inhalt der Kommunikation

Jegliche Kommunikation wird über die PUT-Methode abgewickelt, bei der nur die Zieladresse, aber kein bearbeitender Prozeß oder Parametername wie bei der POST-Methode bekannt sein muß. Dadurch ist es unter anderem möglich, die Verarbeitung von eintreffenden Informationen auch auf andere Art und Weise zu realisieren, ohne daß der Sender davon Kenntnis haben muß.

23.6 Realisierung der Komponenten

Nachdem die Informationselemente strukturiert und auf die Netzwerkknoten verteilt worden sind und die Kommunikationsstruktur definiert wurde, sollen jetzt die noch fehlenden Bestandteile ergänzt werden. Dazu gehören insbesondere alle aktiven Bestandteile.

Netzwerkknoten

Wie bereits weiter oben angesprochen, soll die Kommunikation über Laufzeitumgebungs-Monitore abgewickelt werden. Dazu wird innerhalb jedes Netzwerkknotens ein Monitor-Dokument unter dem Namen `monitor.ahd` gestartet, welches im Anhang aufgelistet ist und eine Struktur gemäß des in Unterkapitel 18.3 vorgestellten Moni-

tor-Entwurfsmusters besitzt. Die Registrierung eines Dokuments, welches später von einem Monitor benachrichtigt werden soll, erfolgt beim Laden eines Dokuments gemäß des Code-Fragments aus Abbildung 99. Dieses Code-Fragment wird als wiederverwendbare Ressource an einer zentralen Stelle abgelegt, so daß es per XLink in ein Dokument eingebettet werden kann.

```
<ahd:func name="onload" type="text/tcl">
<![CDATA[
    AHDRuntimeCall $ahd_ahd $ahd_element \
        [AHDRuntimeHere $ahd_ahd]monitor.ahd\#register \
        document=$ahd_origin&callback=callback
]]>
</ahd:func>
```

Abb. 99: Registrierung eines Dokuments beim Monitor-Dokument

Tritt nun ein Dokument in eine Laufzeitumgebung ein, so muß zuerst die entsprechende Funktion im Monitor-Dokument aufgerufen werden. Dazu kann das Skript aus Abbildung 100 dienen, welches ebenso per XLink automatisch in mobile Dokumente eingebunden werden kann.

```
<ahd:func name="onload" type="text/tcl">
<![CDATA[
    AHDRuntimeCall $ahd_ahd $ahd_element \
        [AHDRuntimeHere $ahd_ahd]monitor.ahd\#enter \
        document=$ahd_origin
]]>
</ahd:func>
```

Abb. 100: Skript für die Benachrichtigung des Monitor-Dokuments

Die bereits identifizierten Komponenten sind im folgenden wie bereits die Monitor-Dokumente konkret in aktiven Hypertextdokumenten innerhalb der Netzwerkknoten anzulegen. Dazu soll zuerst die in Tabelle 20 beschriebene Zuordnung von Clustern zu Verzeichnissen erfolgen.

Cluster	Verzeichnisname
Besteller	orderer
Koordinator	coordinator
Entscheider	decider
Einkäufer	buyer

Tabelle 20: Zuordnung von Cluster-Namen zu Verzeichnissen

Die entsprechend Tabelle 19 definierten Wurzelemente sowie das Monitor-Dokument werden tatsächlichen Dateinamen gemäß Tabelle 21 zugeordnet. Dabei werden solche Dokumente, welche mehrfach im System auftreten (wie etwa Beschaffungsanträge oder Bestellvorschläge), über eine Kombination aus fortlaufender Nummerierung und Personenkennung unterschieden.

Element	Dateiname (mit Pfad)
orderproposal	coordinator/proposal-<Bestellerkennung>-<Nummer>.ahd

Element	Dateiname (mit Pfad)
orderlist	decider/coordinator-<Koordinatorerkennung>-<Nummer>.ahd und buyer/coordinator-<Entscheidererkennung>-<Nummer>.ahd
delivery	orderer/delivery-<Enkäufererkennung>-<Nummer>.ahd, coordinator/delivery-<Enkäufererkennung>-<Nummer>.ahd und decider/delivery-<Enkäufererkennung>-<Nummer>.ahd
propgen	orderer/propgen.ahd
collectprops	coordinator/collectprops.ahd
collectorders	decider/collectorders.ahd
deliverygen	buyer/deliverygen.ahd
collectdelivery	coordinator/collectdelivery.ahd und decider/collectdelivery.ahd
showdelivery	orderer/showdelivery.ahd
monitor	monitor.ahd

Tabelle 21: Zuordnung von Komponenten zu vollständigen Dateinamen

Generierung von Bestellvorschlägen

Das Generieren von Bestellvorschlägen in Form von orderproposal-Elementen wird im propgen-Dokument umgesetzt. Dazu ist dort ein orderproposal-Element als Vorlage inklusive eingebetteter Eingabefelder eingebunden. Das Versenden eines Bestellvorschlags wird per Klick auf ein Link im Dokument ausgelöst, so daß das propgen-Dokument schließlich eine Struktur gemäß Abbildung 101 besitzt.

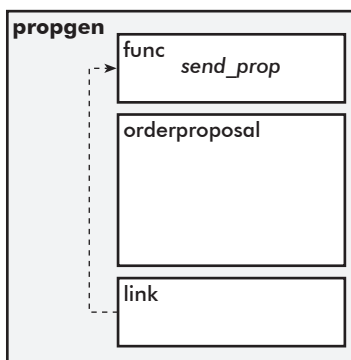


Abb. 101: Struktur des propgen-Dokuments

Sammeln von Bestellvorschlägen

Die Funktionalität zum Sammeln und Verschicken der Bestellvorschläge beim Koordinator wird in drei AHD-Funktionen realisiert. Im ersten Schritt ist das behandelnde collectprops-Dokument beim Monitor-Dokument zu registrieren, was beim Laden des Dokuments erfolgt. Dazu wird das Code-Fragment aus Abbildung 99 eingebunden. Die Annahme von Bestellvorschlägen wird in der dabei registrierten callback-Funktion durchgeführt, wobei die gültigen und ungültigen Vorschläge den entsprechenden Elementen untergeordnet werden. Das Erkennen von eintreffenden Bestellvorschlägen erfolgt anhand eines Strukturmatchings mit einem im Dokument

eingebetteten Vorgabeelement. Dies ist durch ein role-Attribut mit dem Wert `matcher` gekennzeichnet und in einem eigenen Namensraum abgelegt.

Die callback-Funktion ruft nach erfolgter Annahme wiederum die `send_orders`-Funktion auf, welche eine bestimmte Anzahl gültiger Bestellvorschläge in Beschaffungsanträge umwandelt und an den Entscheider weiterleitet.

Das `collectprops`-Dokument erlaubt es weiterhin noch, die ungültigen Vorschläge zu modifizieren, zu löschen oder den gültigen Vorschlägen unterzuordnen. Dazu werden die Elemente der ungültigen Vorschläge um Elemente für die Realisierung des Eingabefeld-Musters erweitert. Dies geschieht beim Eintreffen eines Bestellvorschlags in der callback-Funktion. Da die Funktionalität zum Umwandeln eines XML-Elements in ein Eingabefeld auch in anderen Anwendungen wiederverwendet werden kann, wird sie in einer Funktion gekapselt und ins Eingabefeld-Muster integriert. Die Funktion fügt einem XML-Element ein übergeordnetes Feld-Element und ein passendes Label-Element mit den entsprechenden Rollen-Attributen hinzu, wobei diese hinzugefügten Elemente per XML-Namespace-Attribut inhaltlich vom Rest des Dokuments getrennt werden.

Schließlich werden noch zwei Funktionen zum Freigeben bzw. Löschen eines Vorschlags implementiert, so daß das `collectprops`-Dokument eine Struktur gemäß Abbildung 102 aufweist.

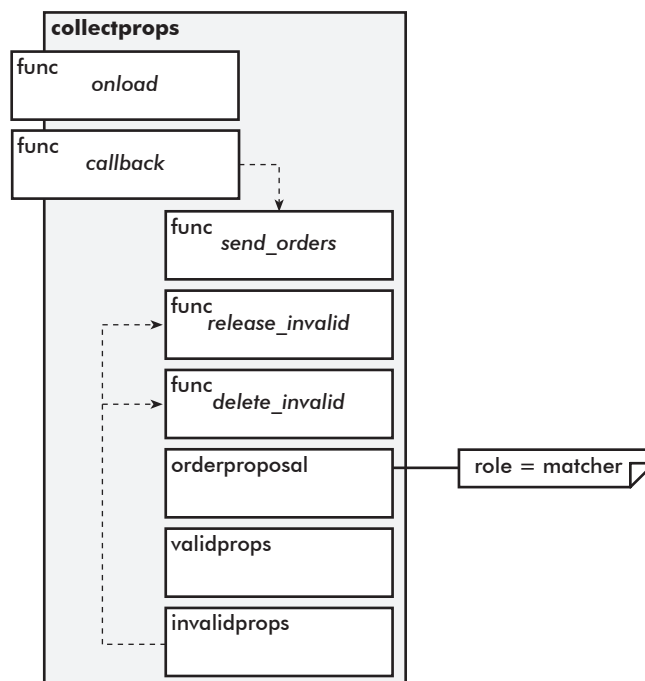


Abb. 102: Funktionen im `collectprops`-Dokument

Unter Verwendung des bereits in den anderen Beispielen verwendeten Stylesheets ergibt sich ein Layout wie beispielsweise in Abbildung 103.

Sammeln von Beschaffungsanträgen

Die gültigen Bestellvorschläge, welche vom Koordinator in Beschaffungsanträge umgewandelt worden sind, werden vom Entscheider geprüft und gegebenenfalls an

Bestellvorschläge

Gültig

Titel: The Psychology of Computer Programming
Autor: Gerald M. Weinberg
Verlag: Van Nostram Reinhold
ISBN/ISSN: 0-442-29264-3
Anzahl: 1

Ungültig

Titel: The Mythical Man-Month
Autor: Frederik P. Brooks
Verlag:
ISBN/ISSN:
Anzahl: 1
[Freigeben](#) [Löschen](#)

Abb. 103: Beispiel für ein collectprops-Dokument

den Einkäufer weitergeleitet. Die Prüfung wird hier nicht weiter spezifiziert, könnte aber beispielsweise unter finanziellen oder sachlichen Gesichtspunkten erfolgen, etwa bei Budgetüberschreitungen oder Doppelbestellungen. Diese Schritte sollen im collectprops-Dokument implementiert werden.

Als zu realisierende Funktionalität sind in der hier gewählten einfachen Variante drei Punkte zu implementieren: das Annehmen von Beschaffungsanträgen, die Prüfung der Anträge und das Weiterleiten an den Beschaffer. Für das Annehmen von Anträgen soll das bereits im vorigen Dokument genutzte Muster des Laufzeitmonitors genutzt werden, welcher registrierte Dokumente bei Eintreffen eines neuen Dokuments benachrichtigt. Auch die Überprüfung, ob es sich bei den eintreffenden Dokumenten um Beschaffungsanträge handelt, soll wie bereits im collectorders-Dokument über einen Strukturvergleich erfolgen, so daß sich ein Aufbau des collectorders-Dokuments wie ihn Abbildung 104 zeigt, ergibt.

Bei der Ankunft neuer Beschaffungsanträge, welche ja als Liste innerhalb eines Dokuments mit einem orderlist-Element als Wurzelement übertragen werden, werden die gültigen Anträge einzeln unter dem collectorders-Dokument eingeordnet und innerhalb der send_orders-Funktion wiederum in einem orderlist-Dokument an den Einkäufer übermittelt.

Generierung von Lieferbestätigungen

Unabhängig vom tatsächlichen Ablauf eines Einkaufs werden die Beschaffungsanträge, die beim Einkäufer eintreffen, zwischengespeichert, um daraus später Lieferbestätigungen generieren zu können. Dies wird im deliverygen-Dokument realisiert. Das Generieren von Bestätigungen wird per Benutzerinteraktion ausgelöst, ähnlich wie bei der Freigabe von Bestellvorschlägen durch das Klicken auf einen Link, der innerhalb eines order-Elements für eine noch nicht abgeschlossene Bestellung eingebettet ist. Wird für eine Bestellung eine Lieferbestätigung erzeugt, so wird die Be-

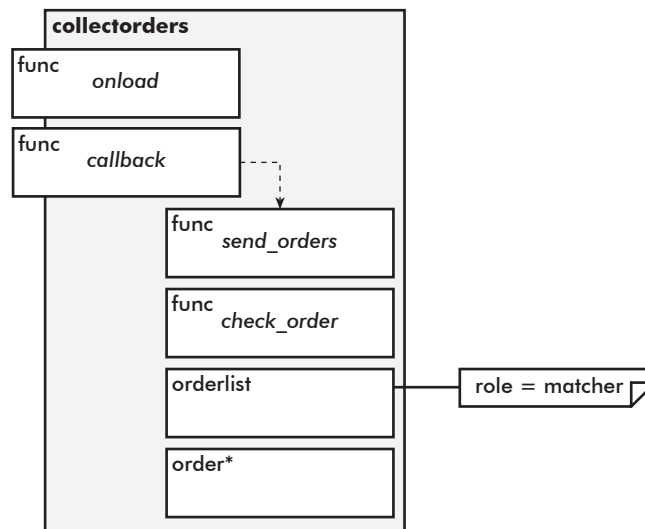


Abb. 104: Aufbau des collectorders-Dokuments

stellung aus dem deliverygen-Dokument entfernt. Die Lieferbestätigungen selbst werden an alle Personen übermittelt, die an einer Bestellung beteiligt waren, d.h. es werden gleichzeitig Bestätigungen an den Besteller, den Koordinator und den Entscheider versandt.

Die Funktionalität wird wiederum über das Muster des Laufzeitumgebungsmonitors bereitgestellt, demzufolge besitzt das deliverygen-Dokument eine Struktur entsprechend Abbildung 105.

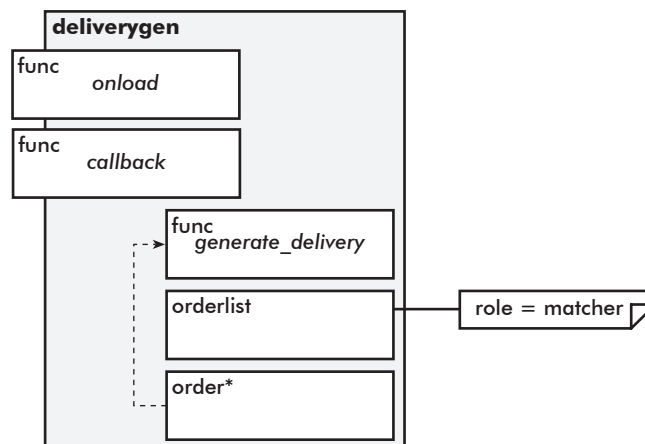


Abb. 105: Struktur der deliverygen-Komponente

Sammeln und Anzeigen von Lieferbestätigungen

Die Auswertung der Lieferbestätigungen soll für den Entscheider und den Koordinator nicht weiter vertieft werden, so daß dort nur einfache Komponenten zum Sammeln der Bestätigungen eingesetzt werden. Beim Besteller allerdings soll eine Anzeige der eingangenen Bestellungen ermöglicht werden. Diese beiden unterschiedlichen Ziele können mit zwei sehr ähnlichen Dokumenten erreicht werden, dem collectdelivery- und dem showdelivery-Dokument. Wie auch der Großteil der

bis hierher vorgestellten Komponenten nutzen sie den Laufzeitumgebungs-Monitor, um eingehende Dokumente zu verarbeiten. Der Aufbau entspricht demnach der in Abbildung 106 gezeigten Struktur. Die beiden Dokumente unterscheiden sich nur dadurch, daß im showdelivery-Dokument eine Aufbereitung der eingetroffenen Bestätigungen für die Anzeige erfolgt, indem Beschreibungselemente vor den Datenelementen eingefügt werden.

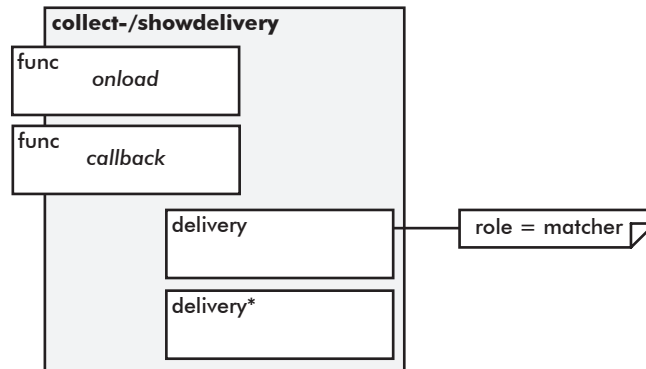


Abb. 106: Struktur von showdelivery- und collectdelivery-Dokument

Mobile Komponenten

Der Inhalt der mobilen Komponenten wurde bereits weiter oben definiert, so daß an dieser Stelle die Funktionalität innerhalb dieser Bestandteile spezifiziert werden soll. Zu den mobilen Komponenten zählen dabei die Dokumente für die Bestellvorschläge, Beschaffungsantragslisten und Lieferbestätigungen (orderproposal, orderlist und delivery) gemäß Tabelle 21.

Die benötigte Funktionalität umfaßt in erster Linie die Benachrichtigung des Monitor-Dokuments beim Eintreffen einer mobilen Komponente in einer Laufzeitumgebung. Diese kann durch einfaches Einbinden des Code-Fragments aus Abbildung 100 erreicht werden.

Die zweite Funktionalität besteht in der Möglichkeit, die mobilen Komponenten durch Benutzerinteraktion zu modifizieren. Wie bereits erwähnt, wird diese Funktionalität durch Einbetten von Elementen aus dem Namensraum der Präsentationselemente und Hinzufügen des Eingabefeld-Musters realisiert.

Wichtig bei der Implementierung der Funktionalitäten ist die Tatsache, daß die dazu benötigten Komponenten durch die Mechanismen des XML-Namespace-Standards von den eigentlichen Nutzdaten zu trennen sind, um die Verwendbarkeit der mobilen Komponenten auch in nicht-AHD-fähigen Umgebungen zu gewährleisten.

23.7 Erweiterungen und Modifikationen

Die bis hierher vorgestellte Realisierung einer Buchbestellungsanwendung stellt nur ein Beispiel für eine mögliche Umsetzung dar. So ist einerseits die durchgängige Verwendung von AHD-Techniken für die Veranschaulichung der dahinterliegenden Mechanismen gewählt worden, andererseits sind mögliche Erweiterungen nicht beschrieben worden.

Die Nutzung von AHD-Techniken wird im Regelfall nicht für alle Komponenten einer Web-basierten Anwendung sinnvoll sein. Deshalb soll nun untersucht werden, inwieweit sich AHD-Komponenten mit nicht-AHD-Komponenten kombinieren lassen. Dazu werden die folgenden zwei Szenarien entwickelt:

- Implementierung einzelner Netzwerkknoten durch nicht-AHD-Techniken
- Erweiterung der Funktionalität um zusätzliche Anwendungsbereiche durch Hinzufügen externer Komponenten

Austausch von Netzwerkknoten

Verfügt in der hier vorgestellten Anwendung der Einkäufer nicht über die Möglichkeit, eine AHD-Laufzeitumgebung zu nutzen, so sind zwei Abläufe neu zu gestalten: das Übermitteln von Beschaffungsanträgen vom Entscheider zum Einkäufer und das Erzeugen und Verschicken der Lieferbestätigungen an die beteiligten Personen.

Die Übermittlung von Beschaffungsanträgen kann grundsätzlich auf unterschiedlichen Wegen erfolgen, etwa durch Nutzung anderer Netzwerkprotokolle wie FTP oder E-Mail. Hier soll von einer E-Mail-basierten Lösung ausgegangen werden, da sie ohne größeren Aufwand auf der Empfängerseite zu realisieren ist. Zugute kommen dabei zwei Eigenschaften des AHDM: XML als zugrunde liegendes Datenformat ist hinreichend einfach auch durch den Endbenutzer les- und bearbeitbar, wenn nicht sogar durch Text-Editoren unterstützt. Außerdem wird im AHDM keine Annahme über die konkreten Werte der URIs bei der Kommunikation gemacht, so daß auch ein E-Mail-Transport durch die entsprechenden AHD-Funktionen ausgelöst werden kann. Als Änderung innerhalb der Anwendung ist demnach nur die URI, unter der der Einkäufer erreicht werden kann, durch eine E-Mail-URL auszutauschen. Dies führt dazu, daß der Einkäufer die Beschaffungsanträge per E-Mail empfängt.

Aufwendiger gestaltet sich sicherlich das Erstellen und Versenden von Lieferbestätigungen. Für das Erstellen ist auf jedem Fall ein XML-Editor empfehlenswert, mit dem die Lieferbestätigungen DTD-konform angelegt werden können. Zusätzlich ist auch eine einfache, programmgesteuerte Umwandlung der XML-Dokumente für die laufenden Beschaffungsanträge in dazugehörige Lieferbestätigungen mittels XSLT oder ähnlicher Techniken denkbar.

Auch das Versenden der Lieferbestätigungen soll per E-Mail erfolgen. Dazu soll eine Proxy-basierte Lösung implementiert werden, bei der eine Proxy-Komponente zwischen den unterschiedlichen Transportmechanismen transparent vermittelt (siehe hierzu das *Proxy Document* Muster, beschrieben in Unterkapitel 18.2). Die Proxy-Komponente nimmt dazu die Lieferbestätigungen per E-Mail entgegen, wertet diese aus und versendet diese an die beteiligten Personen. Das Auswerten und Versenden kann unter Nutzung der AHD-Funktionen realisiert werden. Das Empfangen von E-Mail muß allerdings über eine externe Instanz umgesetzt werden. Hierfür bieten sich beispielsweise ein Programm wie procmail [146] an, welches beim Eintreffen von E-Mail anhand von Filter-Regeln bestimmte Aktionen wie etwa das Starten von Programmen durchführt.

Die Funktionalität wird demnach folgendermaßen verteilt:

- Es wird eine Laufzeitumgebung implementiert, welche Zielort für das Versenden von Lieferbestätigungen ist. Diese wird über procmail beim Eintreffen einer Lieferbestätigung aktiviert und die Lieferbestätigung geladen.

- In der Laufzeitumgebung wird ein Monitor-Dokument genutzt, um die eintreffenden Lieferbestätigungen mittels eines Sender-Dokuments an die entsprechenden Personen zu verschicken
- Die Lieferbestätigungen enthalten eine Funktion, die das Monitor-Dokument beim Laden benachrichtigt. Diese Funktion soll wie gehabt per XLink eingebunden werden.

Daraus resultiert eine Struktur des Proxy-Netzwerkknotens gemäß Abbildung 107.

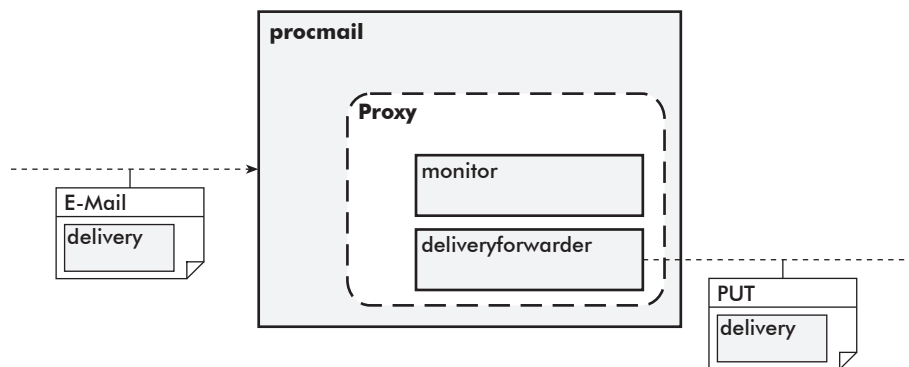


Abb. 107: Struktur des Proxy-Knotens für den Einkäufer

Die Adressierung des Proxy-Knotens per E-Mail-Adresse kann auf unterschiedlichen Wegen erfolgen. Eine einfache Lösung bestünde darin, eine dedizierte Adresse für diese und andere Anwendungen einzurichten, welche dann als Gateway zwischen E-Mail- und HTTP- bzw. AHD-Transport dient.

Untersucht man das Problem, welches sich bei dieser Erweiterung stellt, allerdings auf wiederkehrende und damit in einem Entwurfsmuster behandelbare Charakteristika, so läßt sich die Aufgabenstellung auf die Wahl eines passenden Transportmittels reduzieren. Eine generelle Lösung hierfür wäre die Festlegung alternativer Transportmechanismen für eine Ziel-URI. Dies könnte entweder an zentraler Stelle erfolgen, beispielsweise in einem LDAP-Verzeichnis, oder innerhalb der Anwendung, also als zusätzliche XML-Elemente neben den primären Elementen. Die Auswahl des passenden Transportmechanismus kann ebenso entweder innerhalb der Anwendung oder an zentraler Stelle durchgeführt werden.

Erweiterung um zusätzliche Anwendungsbereiche

Um die Möglichkeiten zur Interoperabilität der vorgestellten Lösung und des AHDM zu demonstrieren, soll die Funktionalität durch Einbindung externer Programme erweitert werden. Als Beispiel dient hierfür der Ablauf bei der Prüfung einer Bestellung durch den Entscheider.

Die Freigabe eines Beschaffungsantrages ist wie bereits erwähnt üblicherweise an eine Prüfung auf Budgetüberschreitungen, Doppelbestellungen, auf fachlich nicht relevante Bestellungen oder ähnliche Kriterien gebunden. An dieser Stelle soll nun eine mögliche Anbindung der Anwendung an eine Budgetverwaltung skizziert werden. Diese Budgetverwaltung soll die für Literaturbeschaffungen vorhandenen Mittel, bezogen auf bestimmte Personengruppen wie Lehrstühle, Fachbereiche oder Institute, verwalten. Dazu sind bei einer Bestellung folgende Informationen notwendig:

- Wieviele Mittel stehen einer Gruppe von Personen für die Beschaffung von Literatur insgesamt zur Verfügung?
- Wieviele Mittel wurden bereits verbraucht bzw. sind für laufende Beschaffungen vorgesehen?
- Wie werden die Besteller bzw. die Bestellvorschläge den Gruppen zugeordnet?

Während die ersten beiden Fragen innerhalb der Budgetverwaltung zu klären sind, betrifft die dritte Frage die Schnittstelle zwischen den beiden Anwendungsgebieten. Hierbei bieten sich zwei Möglichkeiten an: Nutzung der bestehenden Information, die in einem Bestellvorschlag vorhanden ist, etwa durch das orderer-Element, oder das Einfügen neuer Elemente, anhand derer ein Bestellvorschlag einem Budget zugeordnet werden kann. Die zweite Möglichkeit wird vorgezogen, da sie flexibler ist, XML-Namensräume eine inhaltliche Trennung zwischen dem Informationsraum der Beschaffungsanwendung und der Budgetverwaltung erlauben und schließlich das orderer-Element nur optional ist. So soll zum Zuordnen eines Bestellvorschlags durch den Besteller ein entsprechender eindeutiger Identifizierer in einem XML-Element mit dem Namen budgetid in den Bestellvorschlag eingefügt werden. Denkbar ist z.B. die Verwendung eines Zertifikats. Dieses soll zudem in den übrigen im weiteren Verlauf eingesetzten Dokumenten und Elementen (order, delivery) erhalten bleiben.

Zu Schnittstelle gehört auch der eigentliche Ablauf der Prüfung, d.h. das Zusammenspiel der beiden Anwendungen. Die Punkte, an denen eine Kopplung erfolgt, sind das Prüfen eines Beschaffungsantrags und das Eintreffen einer Lieferbestätigung beim Entscheider. Zur Anbindung kommt wie in der vorigen Erweiterung eine Kombination aus Proxy und Auftrag/Dienst-Vermittler. Die Prüfung eines Antrages soll vom collectorders-Dokument durch eine Proxy-Komponente an die eigentliche Budgetverwaltung weitergeleitet werden. Die Proxy-Komponente ist dabei durch den Delegationsmechanismus an das collectorders-Dokument gebunden. Die Auswertung der Lieferbestätigungen soll durch ein dem collectdelivery-Dokument ähnlichen Dokument an die Budgetverwaltung weitergeleitet werden. In beiden Fällen muß die Budgetverwaltung über eine entsprechende Schnittstelle verfügen, d.h. eine Möglichkeit, Funktionen über HTTP zu aktivieren. Eine mögliche Architektur zeigt Abbildung 108.

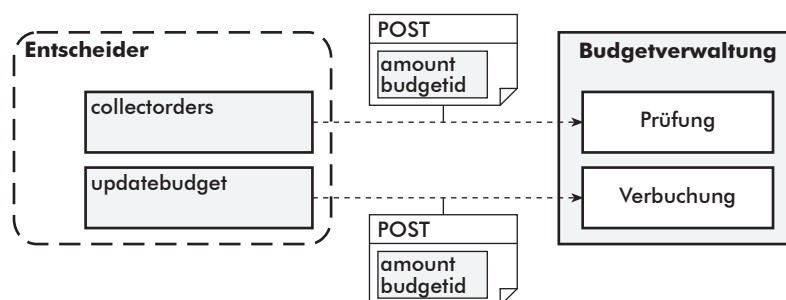


Abb. 108: Mögliche Anbindung einer Budgetverwaltung beim Entscheider

23.8 Auswertung

Die dritte hier vorgestellte Anwendung soll dazu dienen, die Eigenschaften mobiler, aktiver Dokumente zu demonstrieren. Beim Einsatz solcher mobiler Komponenten

können Agentensystem-ähnliche Anwendungen realisiert werden, die neben den Charakteristika von Agentensystemen eine Reihe weiterer Vorteile bieten.

Vorteile

Als wichtigste Vorteile beim Einsatz mobiler AHDs erscheinen die Offenheit der so erstellten Systeme und die lose Kopplung der Komponenten. Während die Offenheit bei der Interoperabilität mit anderen Systeme hilfreich ist, kommt die lose Kopplung einer höheren Flexibilität bei einer Änderung des Systems zugute.

Offenheit

Die Buchbestellungs-Anwendung profitiert insbesondere vom durchgängigen Einsatz von XML und XML-Namensräumen. Dadurch ist Offenheit an mehreren Stellen gegeben, in dieser Anwendung hauptsächlich zwischen den und innerhalb der Komponenten:

- Die zwischen den stationären Systemkomponenten ausgetauschten Daten sind jederzeit auch von anderen Anwendungen verarbeitet- und interpretierbar. So können beispielsweise die beim Entscheider eintreffenden Beschaffungsanträge auf unterschiedlichsten Wegen verarbeitet und zum Einkäufer weitergeleitet werden.
- Durch die Verwendung von XML-Techniken auch innerhalb der stationären Komponenten können diese Komponenten neben der Erbringung wesentlicher Teile der Funktionalität auch statisch betrachtet und von externer Stelle bearbeitet werden. Es lassen sich etwa in einem collectprops-Dokument vorhandene ungültige Bestellvorschläge durch ein externes Programm ergänzen und in gültige Vorschläge umwandeln.

Lose Kopplung

Die lose Kopplung der einzelnen Bestandteile untereinander wird durch die Mechanismen des AHDM unterstützt. So wird nicht explizit festgelegt, wie die unterschiedlichen angebotenen Dienste wie die Bearbeitung von Bestellvorschlägen oder Beschaffungsanträgen realisiert werden, sondern daß diese Dienste in Anspruch genommen werden. Dies geschieht durch das Transferieren der mobilen Bestandteile an bestimmte Stellen, hier die Laufzeitumgebungen der beteiligten Personen. Dadurch ist eine einfache Änderung der Implementierung der Dienste möglich, da der jeweilige Dienstnehmer keine Annahmen über die Weiterverarbeitung trifft.

Auf der Gegenseite werden zu bearbeitende Informationen nicht direkt als solche gekennzeichnet, sondern aufgrund struktureller Eigenschaften verarbeitet. Als Resultat einer solchen Strategie lassen sich bei einer Änderung des Systems etwa durch Hinzufügen von Daten die bestehenden mobilen Komponenten immer noch bearbeiten.

Voraussetzungen

Die Nutzung von Vorteilen wie Offenheit oder loser Kopplung ist an wenige zu erfüllende Voraussetzungen gebunden. Dazu zählt vornehmlich die Verwendung von XML und XML-Namespace, da sie die Basis für das AHDM bilden und die Grundlage für die Strukturierung und Kombination bzw. Separierung von Informationen stellen.

Bereits weniger verbindlich ist der Einsatz von HTTP als Transportmechanismus. Allein die Semantiken, die durch HTTP impliziert werden, sollten beim Entwurf ei-

nes Systems berücksichtigt werden, etwa die zustandslose Übertragung oder die Möglichkeit, entfernte Komponenten per POST-Methode zu aktivieren. Allerdings wurde in der Buchbestellungs-Anwendung darauf Rücksicht genommen, möglichst einfache Kommunikationsmechanismen zu nutzen. Als Folge davon ließe sich z.B. die PUT-Methode, welche für die Datenübertragung genutzt wird, durch andere Mechanismen wie etwa E-Mail ersetzen, da die Kommunikation nur in eine Richtung erfolgt.

Eine wichtige Implikation der hier gewählten Systemarchitektur ist zudem die Vereinbarung der angeforderten und erbrachten Funktionalität. Als Zugeständnis an die gewünschte lose Kopplung werden vom Dienstnehmer keine direkten Angaben über die gewünschte Verarbeitung gemacht. Es ist sicherlich denkbar, die beiden Seiten bei der Verarbeitung der Information näher zu beschreiben (etwa über Meta-Daten) und eine Abstimmung vorzunehmen. Allerdings erscheint hier das Ziel, eine Lösung mit möglichst hoher Interoperabilität zu implementieren, wichtiger, zumal der Aufwand für die genaue Spezifikation der gewünschten und angebotenen Dienste sowie deren Abgleich das System weitaus komplexer machen würde. Hierbei zeigt sich sicherlich ein Nachteil beim Einsatz solch grundlegender Modelle wie dem AHDM, da Funktionalitäten auf höheren Abstraktionsebenen nicht Kernbestandteil sind, sondern erst implementiert werden müssen. Dafür ergibt sich aber auf der anderen Seite eine erhöhte Flexibilität, da eben unterschiedliche Ansätze für solche Abstraktionen gewählt werden können.

Verwendete Muster

Wie auch bei der vorangegangenen Anwendungen spielen Muster eine wichtige Rolle bei der Implementierung der Beschaffungsanwendung. Kern der Anwendung ist das Muster der *Mobile Active Documents*, wobei die Beschaffungsaufträge als mobile Dokumente realisiert wurden, die zwischen stationären Netzwerkknoten ausgetauscht werden. Die Verarbeitung in den Netzwerkknoten beruht auf dem Einsatz zweier Muster: dem *Runtime Environment Monitor*, um eingehende Dokumente zu identifizieren, und dem *Structure-based Multicast*, um die Art der eingehenden Dokumente weiter zu unterscheiden und dementsprechend zu verarbeiten. Das Struktur-Matching wird durch den Einsatz von *Role-Identifier*-Attributen unterstützt, die in den verarbeitenden Dokumenten die Vorgabehierarchien für den Strukturvergleich kennzeichnen.

Erweiterungen des Verarbeitungsflusses lassen sich mit einem Muster des *Proxy Document* implementieren, bei dem für den Versender transparent eingehende Aufträge an externe Stellen wie etwa eine Budgetüberwachung versandt werden.

24 Diskussion

Bei den vorgestellten Anwendungen stand die Demonstration der grundlegenden Techniken im AHDM im Vordergrund, gefolgt von der Nutzung von Entwurfsmustern und dem damit verbundenen Einsatz des Vorgehensmodells aus dem vorigen Abschnitt. Die Anwendungen decken dabei den Bereich der eigenständigen, der kooperativen und der agentenähnlichen Anwendungen ab, welche jeweils eine bestimmte Auswahl der AHDM-Techniken und Entwurfsmuster nutzen.

Interoperabilität

Wie bei der Erweiterung der Beschaffungsanwendung allerdings bereits zu sehen war, werden AHD-Anwendungen in den selteneren Fällen durchgängig mit diesen Techniken realisiert. Vielmehr spielt die Interoperabilität mit anderen Techniken und Systemen eine besonders große Rolle. Dies ist unter anderem auch in den Charakteristika des Einsatzbereichs aktiver Hypertextdokumente begründet: Das Internet und insbesondere das World Wide Web weisen einen hohen Grad an Heterogenität auf, welche durch eine Anzahl von gemeinsamen Protokollen und Standards überbrückt werden kann. Sie decken einerseits die Kommunikationsinhalte und andererseits die Kommunikationswege ab.

Kommunikationsinhalte

Durch die Verwendung vom XML in AHDM-basierten Anwendungen kann im Regelfall die dort strukturierte Information auch in anderen Kontexten genutzt werden. So sind bei der Bookmark-Verwaltung und beim Kontaktmanager die Kernkomponenten auch als eigenständige Dokumente nutzbar. Die Weiterverwendbarkeit der Inhalte wird durch den XML-Namespace-Standard weiter gefördert. In der Erweiterung des Beschaffungsprozesses werden in die bereits bestehenden Dokumente neue Inhalte eingebunden, die die vorhandene Anwendung zum einen nicht beeinflussen, zum anderen aber die Implementierung einer neuen Funktionalität ermöglichen.

Kommunikationswege

Auch die Art der Kommunikation, wie sie in den vorgestellten Anwendungen genutzt wurde, unterstützt die Interoperabilität. So können durch das HTTP-Protokoll auch externe Systeme mit geringem Aufwand eingebunden werden, da gerade HTTP im Anwendungsumfeld AHD-basierter Systeme eine grundlegende Technik darstellt. Weiterhin ermöglicht die Nutzung impliziter Kommunikation wie der Delegations- und Matchingmechanismen flexible Erweiterungsmöglichkeiten.

Transparenz

Vergleicht man die unterschiedlichen Ausprägungen Web-basierter Systeme hinsichtlich der Nutzung offener Standards und der daraus resultierenden Interoperabilität, so fällt auf, daß Anwendungen, die auch intern solche Standards verwenden, die höchste Transparenz aufweisen und somit die Grenzen für Wiederverwendbarkeit und Erweiterbarkeit weiter gezogen werden können. Schließlich erlaubt der Einsatz von XML und HTTP innerhalb von AHD-basierten Systemen auch die Nutzung interner Komponenten durch externe Instanzen (siehe dazu Abbildung 109).

Systemstrukturen

Eine weitere wichtige Fragestellung ist die der Untersuchung der Anwendung des AHDM bei der Erstellung von Systemen sowie die dabei entstehenden Systemstrukturen unter software-technischen Gesichtspunkten. Dazu sollen die Beispielanwendungen zuerst gegen die Anforderungen, welche im Rahmen der Anforderungsanalyse in Kapitel 11 identifiziert wurden, geprüft werden. Die dort aufgezählten Anforderungen sollen getrennt nach funktionalen und nicht-funktionalen bzw. Qualitätsanforderungen untersucht werden. Auch die Anforderungen an die für die Entwicklung eingesetzten Methoden sollen untersucht werden.

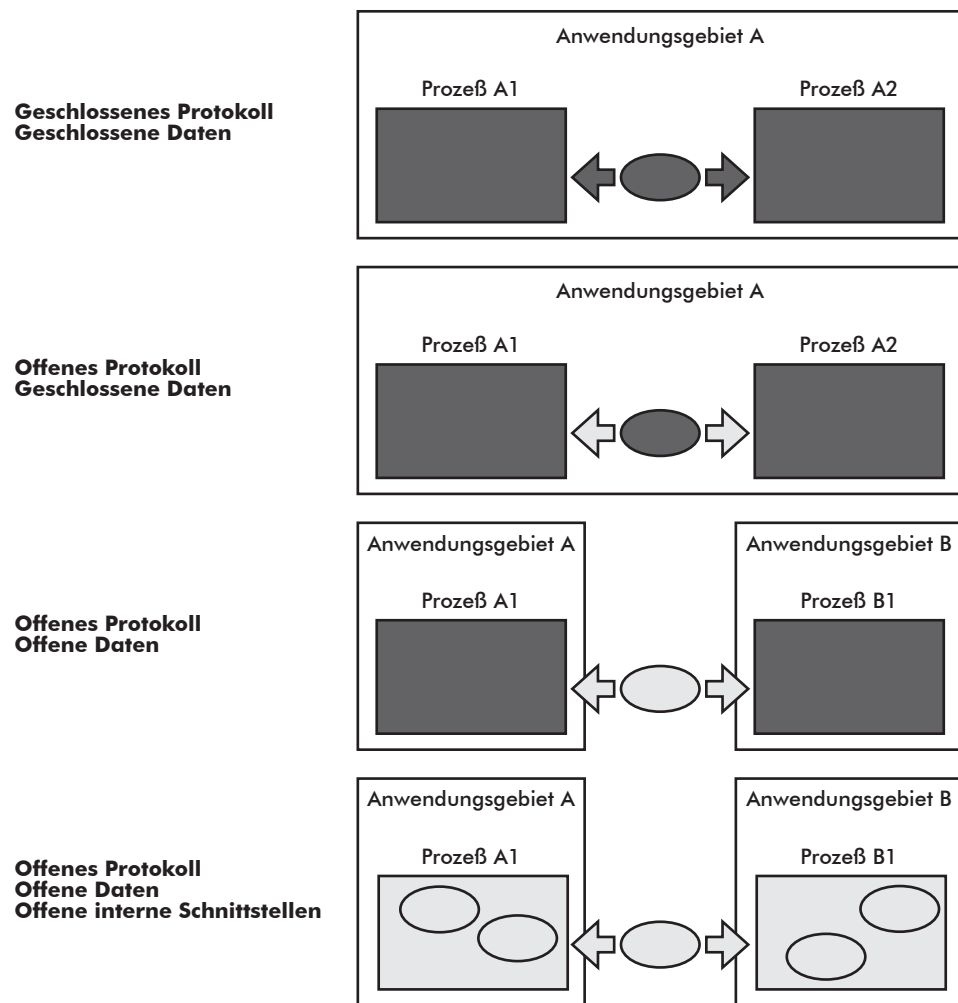


Abb. 109: Unterschiedliche Ausprägungen der Transparenz

Funktionale Anforderungen

Die Erfüllung der funktionalen Anforderungen ist in Unterkapitel 14.1 in gewissem Umfang bereits auf die Charakteristika der im AHDM eingesetzten Techniken zurückgeführt worden. Durch die konkrete Anwendung des Modells im Rahmen der Anwendungen sind dazu ergänzend weitere Aussagen möglich.

- Die Informationsvernetzung und -distribution, welche auf URIs basiert, erfordert für den sinnvollen Einsatz eine Zwischenschicht, die aussagekräftige Identifizierer für Netzwerkknoten oder Dokumentenbestandteile in URIs überführen kann.
- XML-Namensräume stellen ein taugliches Mittel dar, um einerseits die im AHDM verwendeten Informationsräume für Präsentation, Verhaltensbeschreibung und Nutz-Information in einem Dokument zu verbinden.
- Zusätzlich definierte Semantiken wie Entwurfsmuster oder Idiome stellen eine Basis für die Wiederverwendbarkeit von Komponenten dar und helfen dadurch bei der Konstruktion von Systembestandteilen.

Nicht-funktionale Anforderungen

Schwieriger als die Bewertung der Erfüllung funktionaler Anforderungen ist die Betrachtung der nicht-funktionalen Anforderungen, da sie oft schwer quantifizier- und exakt definierbar sind. Dennoch können auch hier einige Aussagen aufgrund der realisierten Anwendungen gemacht werden.

- Wie bereits erwähnt, besteht mit den XML-Namensräumen ein Mechanismus, um unterschiedliche AHDM-Informationsräume in einem Dokument zu kombinieren. Neben dieser Eigenschaft sind Namensräume aber auch sehr gut geeignet, Anwendungen derart zu erweitern, daß entweder neue Funktionalitäten realisiert werden können, oder daß eine Einbindung externer Systeme möglich ist. Dies ist möglich, da durch Namensräume bereits bestehende Strukturen nicht berührt werden. Ein Beispiel hierfür ist die Erweiterung der Beschaffungsanwendung, die zeigt, daß die XML-Namensräume ein System interoperabler machen.
- Durch die lose Kopplung über die AHDM-Delegationsmechanismen können Anwendungen flexibler modifiziert werden, wie etwa beim Austausch der Gewichtungsfunktionalität in der Bookmark-Verwaltung. Auch die Nutzung des Rollen-Attributs zur Referenzierung beteiligter Elemente in einem Entwurfsmuster wie dem Eingabefeld erleichtert das flexible Erweitern von Komponenten um neue Funktionalität.

Methode

Wie in Kapitel 16 beschrieben, wird bei den Anforderungen an die Entwurfsmethode zwischen Anforderungen an die Methode selbst sowie Anforderungen der dadurch produzierten Systeme unterschieden. Die Anforderungen an die Systeme bzw. deren Komponenten stellen dabei eine Ergänzung der nicht-funktionalen Anforderungen dar und sind teilweise schon weiter oben besprochen worden. So kann lose Kopplung durch Delegationsmechanismen erreicht werden, ebenso wie die Wiederverwendbarkeit. Wichtig erscheint zudem, daß die Methode bei der Erstellung der Beispielanwendungen konstruktiv eingesetzt werden konnte, d.h. durch die Anwendung der Methode wurden die Systeme aus den Einzelbestandteilen schrittweise zusammengesetzt. Auch war es mit der gewählten Notation möglich, die Systeme und Bestandteile hinreichend genau zu dokumentieren.

Abschnitt D

Werkzeuge



25 Aufgabenstellung

Das in den vorangegangenen Abschnitten vorgestellte Modell und die zugehörige Entwurfsmethode wurden im letzten Abschnitt durch Anwendungen evaluiert. Dazu sind als weitere Grundlage Werkzeuge nötig, die die definierten Mechanismen umsetzen. Die Entwicklung solcher Werkzeuge soll in diesem Abschnitt untersucht werden. Dabei wird zuerst ein Prototyp implementiert (die *Kino-2-Bibliothek*), der zur Umsetzung der beschriebenen Anwendungen und zur Untersuchung der wichtigsten Anforderungen dient. Diese Implementierung dient als „proof of concept“ und ist die voll-funktionsfähige Testplattform für die Beispiele des vorangegangenen Abschnitts. Dabei werden allerdings die beim prototypischen Einsatz weniger relevanten Aspekte wie z.B. XML-Validierung zugunsten wichtiger Aspekte etwa bei der CSS-Darstellung zurückgestellt.

Im Kapitel 27 sollen die so gewonnenen Erkenntnisse in die Implementierung eines Werkzeugs zur umfassenden Unterstützung der AHDM-Techniken einfließen (das *JKino-Java-Package*), wobei insbesondere auf die vollständige Implementierung der Basistechniken wie XML und DOM Wert gelegt wird. Dazu zählen u.a. validierende XML-Parser oder DOM-Implementierungen mit Namespace-Unterstützung. Diese Funktionalitäten können in vielen Fällen durch vorhandene state-of-the-art Software-Pakete realisiert werden, so daß die Integration von AHDM-Techniken im Vordergrund steht. Wie im Verlauf dieses Abschnitts jedoch deutlich wird, existieren bei den vorhandenen Paketen Defizite bei der Darstellung von XML-Daten mittels CSS.

Die Werkzeuge lassen sich in zwei Gruppen unterteilen. Erstens sind Werkzeuge notwendig, AHD-basierte Anwendungen zu erstellen, zweitens sind die Funktionen der AHD-Laufzeitumgebung zu implementieren. Die Werkzeuge zur Erstellung von aktiven Hypertextdokumenten sollen in dieser Arbeit allerdings nicht weiter betrachtet werden. Durch die Verwendung von XML als Dokumentenformat lassen sich hierfür beliebige XML-Editoren einsetzen. Eine sinnvolle Unterstützung ergibt sich aber dann, wenn die Wiederverwendung von Konzepten und Komponenten ähnlich der im vorigen Abschnitt beschriebenen Mustern unterstützt wird. Auch die Möglichkeit für die graphische Darstellung der Systemarchitektur sollte durch die eingesetzten Werkzeuge unterstützt werden. Die Beurteilung des AHDM für die Implementierung Web-basierter Informationssysteme ist hiervon nicht direkt betroffen. Alle in dieser Arbeit entwickelten Werkzeuge sind als Open-Source-Programme [139] frei verfügbar.

26 Implementierung eines Prototyps

Der vorgestellte Prototyp soll möglichst universell einsetzbar sein, weshalb als Entwicklungsplattform das Betriebssystem Unix gewählt wurde. Unix ist durch die Verbreitung der offenen Varianten Linux, FreeBSD, NetBSD und OpenBSD mittlerweile auf jeder wichtigen Hardwareplattform lauffähig. Durch diese weite Verbreitung sind viele der Unix-spezifischen Programmierschnittstellen mittlerweile als Standard (z.B. POSIX) oder de-facto-Standard (z.B. die Socket-Schnittstelle) anzusehen, was den Forderungen nach Plattformunabhängigkeit und Ausnutzung bestehender Standards bei der Entwicklung AHDM nahe kommt. Da in Unix-ähnlichen Betriebssystemen die Programmiersprache C zur Implementierung sowohl des

Kerns als auch aller wichtigen System- und Anwendungsprogramme benutzt wird und deshalb die Entwicklung mit C dementsprechen unterstützt wird, soll diese Sprache auch beim Prototyp zum Einsatz kommen. Der Prototyp wird zwecks vereinfachter Nutzung als Funktionsbibliothek und nicht als eigenständiges Programm realisiert. Da der Prototyp auf Erfahrungen und Bestandteilen des in [97] implementierten Kino-Widgets aufbaut, wird er hier mit dem Namen *Kino 2* bezeichnet. Er dient hauptsächlich der Implementierung der im vorigen Abschnitt beschriebenen Beispielanwendungen und zum Sammeln von Erfahrungen für die Umsetzung einer vollständigen AHD-Laufzeitumgebung.

Das Vorgehen bei der Implementierung des Prototyps zur Nutzung aktiver Hypertextdokumente ist wie auch die bereits vorgestellte Methode an die Business Object Notation angelehnt. Hier soll sehr verkürzt zuerst die Systemgrenze identifiziert, daran anschließend die Subsysteme beschrieben und schließlich die Module gebildet werden.

26.1 Systemgrenze

Die Systemgrenze für den Prototyp läßt sich unter anderem ausgehend der in Abbildung 34 gezeigten Übersicht festlegen. So ist ausgehend von der Verwendung des Prototyps im Web-Umfeld und durch die Anwendung durch Endbenutzer die Abgrenzung zur Benutzerschnittstelle einerseits und zum Netzwerkzugang andererseits notwendig.

26.2 Subsysteme

Die Schnittstellen zum Systemumfeld repräsentieren erste Kandidaten für Subsysteme. Weitere Kandidaten für Subsysteme sollen durch eine Analyse der zusätzlich benötigten Systemfunktionalität und der zugrundeliegenden Informationsstrukturen ermittelt werden.

Parser

Ausgehend vom Einsatz von XML-Dokumenten als Basis für aktive Hypertextdokumente ist die Repräsentation und Bearbeitung der Dokumentenstrukturen zu realisieren. Das Document Object Model ist in diesem Zusammenhang als Standard-API für die Nutzung von XML-Dokumentenstrukturen vorgesehen. Hinzu kommt noch die Funktionalität zur Erzeugung der Dokumentenstruktur aus XML-Quelltexten, welche durch einen XML-Parser erbracht werden soll. Der Einfachheit halber werden die DOM- und Parser-Funktionalität im Subsystem *Parser* zusammengefaßt.

GUI

Die Anbindung der Benutzerschnittstelle hat primär zwei Aufgaben: die Darstellung von XML-Dokumenten und die Weiterleitung von Benutzerinteraktionen an die aktiven Hypertextdokumente. Genauer ausgedrückt bedeutet dies die Formatierung der Dokumentenstrukturen gemäß CSS und die Erzeugung entsprechender Oberflächenelemente sowie die Implementierung eines Mechanismus zur Nachbildung des Intrinsic Event Model. Diese Aufgaben werden im Subsystem *GUI* erfüllt.

Eine notwendige Anforderung ist weiterhin noch die Kapselung plattformspezifischer Eigenschaften, da der Prototyp möglichst leicht an andere Systemumgebungen

anpaßbar sein soll. Neben den im Unix-Umfeld weitverbreiteten Mechanismen wie dem X Window System und den darauf aufbauenden unterschiedlichen Oberflächenbibliotheken käme hier etwa auch die Durckausgabe oder entsprechend der CSS-Spezifikation die Sprachausgabe in Frage.

AHD

Da der Prototyp eine einfache Struktur erhalten soll und die AHD-Funktionalität wie sie in der Laufzeitumgebung spezifiziert ist, eng mit der Netzwerkanbindung verbunden ist, soll im Subsystem *AHD* sowohl die Laufzeitumgebung als auch die Datenhaltung bzw. der Datentransport implementiert werden.

Zu den erforderlichen Funktionalitäten gehören daneben auch die Bereitstellung der Ausführungsumgebung für die in aktive Dokumente eingebetteten Programmfragmente und eine erweiterbare Architektur für die Anbindung unterschiedlicher Programmiersprachen für diese Programme.

Wrapper

Eine wichtige Aufgabe der Kino-2-Bibliothek ist der Export der implementierten Funktionalität für darauf aufbauende Anwendungen. Dies kann durch Einbinden der Bibliothek über den Linker erreicht werden. Da ein wesentlicher Teil der Bibliothek (die DOM-Funktionalität und die AHD-Laufzeitumgebung) aber auch innerhalb der aktiven Hypertextdokumente genutzt werden soll, müssen die Funktionen auch aus den dort eingesetzten Programmiersprachen heraus aufrufbar sein. Im Subsystem *Wrapper* wird ein Mechanismus implementiert, der einen Export der Funktionalität über eine generische Schnittstellenbeschreibung der Bibliothek erlaubt. Die Schnittstellenbeschreibung liegt in XML vor und wird über DOM-Funktionen in Interface-Dateien für den *Simplified Wrapper and Interface Generator* (SWIG) [13] umgewandelt. SWIG erzeugt aus diesen Dateien Programmcode für die Einbindung in Skriptsprachen wie Lua, Perl, Tcl oder Python.

26.3 Module

Die Implementierung wird durch die Bildung einer Anzahl von Module umgesetzt, die den Aufgabenfeldern in den Subsystemen zugeordnet werden. Es werden hier nicht alle Module beschrieben, sondern nur die für die Systemarchitektur wichtigsten Module erläutert.

Datenmodell

Das Datenmodell für AHDs in der Kino-2-Bibliothek ist wie bereits erwähnt auf dem DOM aufgebaut und implementiert die wichtigsten Funktionen daraus. Zusätzlich sind allerdings noch unterstützende Daten für die Formatierung von XML-Dokumenten mit CSS-Layoutvorlagen notwendig.

DOM

Der größte Teil der DOM-Funktionalität und der zugrundeliegenden Datenstrukturen werden im Modul *DOM* implementiert, dabei werden allerdings nur die Kernbestandteile der DOM-Spezifikation unterstützt. Es wird dabei versucht, die Vererbungsbeziehungen zwischen den DOM-Schnittstellen über eine simple Technik in C-Strukturen abzubilden, um einerseits den Anwendungscode zu vereinfachen

und andererseits eine gewisse Effizienz zu erreichen. Ähnlich wie bei der Implementierung des X Toolkits [61] werden dazu klassenähnliche C-Strukturen gebildet, welche immer ein Feld zur Identifikation des repräsentierten Typs (z.B. Node, Element oder Document) besitzen. Zugriff auf die typspezifischen Felder wird dann über *casting* erreicht. Dies geht auf Kosten der Typsicherheit der DOM-Implementierung. Da die Kino-2-Bibliothek aber nur als Prototyp ausgelegt ist, soll dieser Nachteil an dieser Stelle hingenommen werden.

Property

Bei der Unterstützung der Formatierung von XML-Dokumenten existieren zwei Alternativen, die sich so auch in den beiden existierenden unterschiedlichen Ansätzen für das Layout von XML-Dokumenten widerspiegeln: Während CSS relativ nah an die ursprüngliche Dokumentenstruktur gebunden ist und nur eine begrenzte Anzahl von nicht im Ausgangsdokument vorkommenden Pseudo-Elementen definiert, wird bei DSSSL und XSL auf einer vom ursprünglichen Dokument losgelösten Struktur von *flow objects* oder Layoutelementen aufgesetzt.

In der Kino-2-Bibliothek soll der erste Weg gewählt werden, da sich dadurch die Implementierung vereinfacht und die Zuordnung des dargestellten Dokuments zum darunterliegenden Datenmodell weniger aufwendig ist. Deshalb werden innerhalb der DOM-Datenstrukturen nicht nur die für die Repräsentation von XML-Dokumenten wichtigen Informationen abgelegt, sondern zusätzlich auch die CSS-Eigenschaften. Dementsprechend enthält eine Element-Struktur neben den Attributen aus dem DOM auch den vollständigen Satz von Eigenschaften aus CSS. Sie werden im Modul *Property* definiert.

StyleBase

Die CSS-Eigenschaften werden im Modul *StyleBase* von ihrer textuellen Repräsentation im XML-Quelltext in die tatsächlichen Werte entsprechend der Typdeklarationen im Modul *Property* umgewandelt. Da bestimmten CSS-Eigenschaften nur in Abhängigkeit von einem Ausgabemedium ein sinnvoller Wert zugewiesen werden kann, sind die Umwandlungsfunktionen in zwei Gruppen eingeteilt: erstens die Funktionen, die bereits CSS-Eigenschaften während des Parsens des Dokuments errechnen können (Längenangaben in Pixeln, Farbwerte, Textausrichtungen usw.), zweitens die Funktionen, die die medienabhängigen Werte berechnen (dazu zählen insbesondere alle Zeichensatzabhängige Maße). Die zweite Gruppe von Funktionen wird im Subsystem GUI realisiert (siehe unten).

Parser

Die Transformation vom XML-Quelltext in DOM-Strukturen ist Aufgabe des Parser-Subsystems. Dazu gehört neben dem Erkennen von Auszeichnungselementen im Quelltext auch die Analyse der Document Type Declaration und Definition und das Auflösen von Entity-Referenzen.

Parser

Bei der Entwicklung von Parsern im XML-Umfeld haben sich zwei teilweise komplementäre Ansätze herausgebildet, die die Aufgaben im Parse-Prozeß mit unterschiedlichen Schwerpunkten belegen. Die erste Gruppe von Parsern repräsentiert die sog. ereignisbasierten Parser. Dazu zählt als prominentester Vertreter das *Simple API for XML* (SAX) [114], das eine Schnittstelle für ereignisbasierte Parser definiert. Er-

ereignisbasierte Parser verarbeiten den XML-Quelltext als Datenstrom, wobei das Auftreten bestimmter Konstrukte im Datenstrom als Ereignis betrachtet wird.

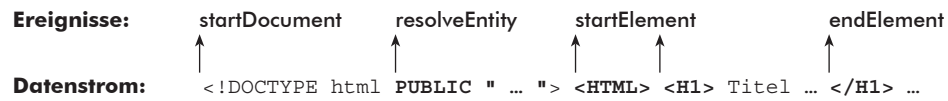


Abb. 110: Beispiel für Datenstrom und Ereignisse bei SAX-basierten Parsern

Die Konstrukte, die SAX erkennt, umfassen alle wichtigen Bestandteile von XML, z.B. Elemente, Entity-Referenzen, Processing Instructions oder Document Type Definitions. Eine Anwendung, die einen SAX-basierten Parser nutzen möchte, registriert *callback*-Funktionen beim Parser, die beim Auftreten der Parse-Ereignisse mit entsprechenden Daten aufgerufen werden. Der Vorteil ereignisbasierter Parser ist die Möglichkeit sehr effizienter Implementierungen, da der Zustandsraum des Parsers durch die Einfachheit von XML begrenzt ist, und was noch schwerer wiegt, während des Parsens vom Parser selbst keine zusätzlichen Datenstrukturen erzeugt werden. Ereignisbasierte Parser eignen sich so vor allem für den Einsatz in Systemen mit beschränkten Systemressourcen.

Die zweite Gruppe von Parsern sind die baumbasierten Parser, die einen Parse-Baum aus dem XML-Quelltext erzeugen. Im Regelfall wird hier eine DOM-Struktur erzeugt, auf die mittels der DOM-Schnittstellen zugegriffen werden kann. Baumbasierte Parser können mittels ereignisbasierter Parser implementiert werden, so daß die Erzeugung des Parse-Baums in den callback-Funktionen des baumbasierten Parsers erzeugt werden. Sie sind ressourcenintensiver als ereignisbasierte Parser, da für ein Dokument üblicherweise eine vollständige Baum-Repräsentation erzeugt wird. Allerdings sind sie auch Voraussetzung für alle Anwendungssysteme, die ein XML-Dokument nicht nur einmal verarbeiten sondern beliebige Modifikationen erlauben. Solche Anwendungen benötigen eine Strukturrepräsentation des Dokuments, da eine wiederholte Modifikation des Quelltextes mit jeweils anschließendem erneutem Parsen ineffizient ist. Auch wird die Implementierung validierender Parser vereinfacht, da die Validierung eines XML-Dokuments zusätzliche Kontextinformationen der einzelnen Elemente einbezieht.

Der Parser in der Kino-2-Bibliothek ist als baumbasierter Parser im Modul *Parser* realisiert, der auf einem ereignisbasiertem Parser aufsetzt. Der Vorteil dieser Zusammensetzung besteht darin, daß neben dem Erzeugen der DOM-Strukturen auch andere Aktionen während des Parsens ausgelöst werden können. Zu diesem Zweck nutzt der Parser SAX-ähnliche callback-Funktionen. Wichtigste Funktion ist dabei der Tag-callback. Sie wird in ein allgemeineres Rahmenwerk von callback-Funktionen eingepaßt, welches in der Kino-2-Bibliothek u.a. für Benutzerinteraktionen oder andere Ereignisse genutzt werden. Das callback-Rahmenwerk wird weiter unten beschrieben.

Der realisierte Parser ist nicht validierend und kann auch HTML-Dokumente verarbeiten. Da in HTML die Verwendung von Ende-Tags für einige Elemente optional ist, wird eine Heuristik genutzt, um auch in solchen Fällen einen korrekten DOM-Baum zu erzeugen. Dazu wird die CSS-Eigenschaft `display` herangezogen. Entsprechend der CSS-Spezifikation kann etwa ein inline-Element kein block-level-Element enthalten, so daß beim Auftreten eines block-level-Elements alle bis dahin nicht geschlossenen inline-Elemente abgeschlossen werden. Ähnlich können auch beim

Ende-Tag eines block-level-Elements alle darin enthaltenen inline- und block-level-Elemente geschlossen werden.

Handler

Die Signalisierung der Anwendung durch den ereignisbasierten Teil des Parsers erfolgt wie bereits angesprochen über callback-Funktionen. Daneben werden callback-Funktionen auch für drei weitere Fälle genutzt:

- Auftretende XLinks oder Entity-Referenzen auf externe Ressourcen werden über den Link-callback an die Anwendung weitergeleitet. Die Anwendung erhält bei Aufruf dieses callbacks alle Daten, die notwendig sind, um die externe Ressource zu laden und weiterzuverarbeiten. Dazu zählt u.a. auch die Möglichkeit, die Ressource an einer bestimmten Stelle in das aktuelle Dokument einzufügen.
- Benutzerinteraktionen und andere externe Ereignisse lösen den Event-callback aus. Hier werden der Anwendung die Quelle des Ereignisses (im Regelfall das Element, in dem die Benutzerinteraktion stattfand) sowie nähere Information über die Art des Ereignisses übermittelt.
- Ein wichtiger Bestandteil des AHDM ist die Ausführung von Verhaltensbeschreibungen. Da das AHDM keine Festlegung auf eine bestimmte Programmiersprache für die Verhaltensbeschreibungen trifft, können über den Script-callback Funktionen registriert werden, die die Verhaltensbeschreibungen z.B. in einem Skript-Interpreter ausführen. Dadurch wird eine offene Architektur realisiert, die um zusätzliche Programmiersprachen einfach erweitert werden kann. Den im Script-callback registrierten Funktionen wird der Quelltext des auszuführenden Programmfragments zusammen mit Kontextinformation (aufrufendes Element, umgebendes Dokument) und eventuell vorhandenen Funktionsparametern übergeben.

Die für die Datenübergabe notwendigen Strukturen sind im Subsystem Parser im Modul *Handler* definiert. Die callback-Funktionen selbst werden an mehreren Stellen in der Bibliothek verwaltet. Der Script-callback kann bei der unten besprochenen Laufzeitumgebung registriert werden. Die Event- und Link-callback-Funktionen werden beim erzeugten Dokument eingetragen. Der Tag-callback wird sowohl beim Parser als auch beim Dokument eingetragen. Die Verwaltung der callback-Funktionen in der Dokumentenstruktur erlaubt die Nutzung eines einmal erzeugten Dokuments auch ohne Parser und ohne Laufzeitumgebung. Dadurch wird die Abhängigkeit der Komponenten in der Kino-2-Bibliothek reduziert.

AHD-Laufzeitumgebung und Netzwerkanbindung

Die für die Unterstützung des AHDM wesentliche Funktionalität wird im Subsystem AHD realisiert, welches aus Vereinfachungsgründen auch die Netzwerkanbindung und Datenhaltung enthält.

AHD

Die in Unterkapitel 13.6 spezifizierte Laufzeitumgebung für aktive Hypertextdokumente ist im Modul *AHD* implementiert. Es stellt eine direkte Implementierung der Schnittstelle aus Abbildung 44 dar.

Die Aufgaben der Laufzeitumgebung umfaßt drei Bereiche: die Verwaltung der aktiven Hypertextdokumente, die Implementierung der Funktionen entsprechend der

definierten Schnittstelle und die Anbindung der Dokumente an die Ereignisquellen in der Anwendung und die Netzwerk-/Datenhaltungsschicht. Somit ist das Modul AHD wesentlich an der Implementierung des Kommunikationsmodells aus Unterkapitel 13.7 beteiligt. Konkret bedeutet dies eine Einbindung der Laufzeitumgebung an mehreren Stellen, die im folgenden erläutert werden sollen.

Die Laufzeitumgebung wird beim Parser über den Tag-callback registriert, um die Behandlung der func-, var-, und delegate-Tags zu übernehmen. Dazu wird die display-Eigenschaft dieser Elemente so gesetzt, daß sie nicht angezeigt werden. Zudem wird der textuelle Inhalt der Elemente für einen beschleunigten späteren Zugriff zwischengespeichert. Der Parser überträgt die registrierten callback-Funktionen beim Anlegen der DOM-Struktur eines Dokuments in das Dokument.

Die Benutzerinteraktionen werden durch das Registrieren der Laufzeitumgebung im in der Liste der Event-callbacks im Dokument verarbeitet. Zu beachten ist, daß das Dokument keine Ereignisse selbst erzeugt, sondern nur als Sammelpunkt für Benutzerereignisse dient. Die Ereignisse werden typischerweise von den Modulen im Subsystem GUI erzeugt und an das AHD weitergeleitet. Entsprechend des Ereignistyps (Mausbewegung, Mausklick, Tastendruck usw.) und der auslösenden Quelle wird gemäß des IEM bei Bedarf eine Funktion im AHD aktiviert. Dies geschieht durch Aufruf der im AHD registrierten Script-callback-Funktionen.

Um die beim Parsen oder beim Aktivieren eines Links im Dokument referenzierten Ressourcen zu laden, wird die Laufzeitumgebung auch im Link-callback des Dokuments registriert. Die Behandlungsfunktion im Modul AHD reicht aber die eintreffenden Anforderungen an die Netzwerkschicht weiter und verarbeitet nur die hiervon zurückgelieferten Daten.

Als letzte callback-Funktion wird im Modul AHD ein vordefinierter Script-callback implementiert, der eine beispielhafte Anbindung der Programmiersprachen Tcl und Lua enthält. Im Script-callback wird für jedes auszuführende Programmskript eine eigene Interpreterumgebung der jeweiligen Programmiersprache angelegt. So ist gewährleistet, daß auch eine nebenläufige Ausführung von Programmfragmenten möglich ist. Weiterhin werden die übergebenen Parameter dekodiert und als Variable entsprechend Tabelle 3 auf Seite 88 in der Interpreterumgebung angelegt. Der Rückgabewert wird der aufrufenden Funktion in der im Modul Handler definierten Struktur übermittelt.

Die Funktionen, die das AHDM ausmacht und die in der Schnittstellenspezifikation der Laufzeitumgebung enthalten sind, betreffen hauptsächlich die Aktivierung von Verhaltensbeschreibung und den Zugriff auf AHD-Zusandsvariable. Dazu wird im Modul die parent und remote delegation implementiert.

Die Verwaltung der AHDs entsprechend der Semantiken aktiv und inaktiv (siehe Unterkapitel 13.8) wird als Liste von Dokumenten realisiert, bei der jedes aktive AHD über die Ursprungs-URI identifiziert werden kann. Bei der Nutzung eines AHDs kann über diese Liste ermittelt werden, ob das Dokument bereits aktiv ist und gegebenenfalls ein Nachladen aus dem Dateisystem ausgelöst werden. Die Liste wird auch bei Referenzierung eines Dokuments über eine Netzwerkverbindung genutzt, so daß sich eine Art Zwischenspeicher aktiver Hypertextdokumente ergibt und Modifikationen eines aktiven Dokuments für alle externen Instanzen direkt sichtbar sind.

HTTPClient und HTTPServer

Die Realisierung der Netzwerkanbindung erfolgt in den Modulen *HTTPClient* und *HTTPServer*. Entsprechend der Systemarchitektur bei der Nutzung aktiver Hypertextdokumente wird die Trennung zwischen Client und Server aufgehoben.

Daraus ergeben sich zwei Schnittstellen zum restlichen System. Die AHD-Laufzeitumgebung nutzt die HTTP-Client-Funktionen um die AHD-Funktionen zu realisieren. Der HTTP-Server wiederum nutzt die AHD-Laufzeitumgebung, um Anforderungen, die sich auf AHDs beziehen, zu verarbeiten.

Die HTTP-Implementierung in beiden Modulen ist nur sehr rudimentär, da für den Prototyp keine erweiterten Möglichkeiten wie persistente Verbindungen, vollständige Fehlerbehandlung oder effiziente Ressourcenausnutzung erforderlich sind. So ist der HTTP-Server als einfacher, nebenläufiger Server realisiert, bei dem für jeden eintreffenden Request ein Thread gestartet wird. Dieser Mechanismus spiegelt die geforderte Semantik bei der Ausführung von Verhaltensbeschreibungen in aktiven Hypertextdokument (siehe dazu Unterkapitel 13.8) direkt wieder.

Präsentation

Die Gruppe der Module, die die Präsentation von HTML- und XML-Dokumenten in Verbindung mit CSS umsetzt, ist nur bei solchen Anwendungen notwendig, bei denen Endbenutzer involviert sind. In der Kino-2-Bibliothek sind die Module optional. Wie oben erwähnt existieren nur wenige Abhängigkeiten des Datenmodells vom Präsentationsmodell, hauptsächlich handelt es sich dabei um die Ermittlung von Werten für CSS-Eigenschaften, die vom jeweiligen Ausgabemedium abhängen.

Um im Prototyp die Portierbarkeit zu erleichtern, sind die Module der Präsentationsschicht in zwei Gruppen einteilbar. Es existiert ein plattformunabhängiges Präsentationsmodell und je nach Zielplattform (verwendetes Fenstersystem, Oberflächenbibliotheken usw.) vorhandene Anpassungsmodule.

Layouter

Die Umsetzung der CSS-Beschreibung für ein XML-Dokument in Layoutangaben wird im Modul *Layouter* durchgeführt. Die Komponenten, die beim Layout-Prozeß verarbeitet werden, sind mit den durch das Parsen erzeugten DOM-Objekten identisch. Es ergibt sich somit eine direkte Übereinstimmung von XML-Elementen und CSS-Blöcken. Der Layout-Vorgang wird anhand dieser Elemente nach einem einfachen Verfahren durchgeführt, bei dem block-level-Elemente mit inline-Elementen gefüllt werden. Die inline-Elemente und anderer Inhalt wie Text fließen dabei um eingebettete block-level-Elemente. Durch den Layout-Vorgang werden hauptsächlich die Positions- und Dimensionsangaben der Elemente errechnet, so daß für die spätere Anzeige schnell darauf zugegriffen werden kann.

HTMLFormTags

Eine notwendige Abstraktion der Ausgabeplattform im Präsentationsmodell umfaßt die Erzeugung von eingebetteten Oberflächenelementen für HTML-Forms, etwa Eingabefelder oder Buttons. Das Modul *HTMLFormTags* exportiert Funktionen, um diese Elemente zu erzeugen, die konkrete Umsetzung wird durch die angesprochenen Anpassungsmodule umgesetzt.

Die zur Verfügung stehenden Funktionen zur Erzeugung von Oberflächenelementen decken eine Untermenge der im HTML-Teil des DOM definierten Funktionen ab. Sie sind auch in Verhaltensbeschreibungen in einem AHD nutzbar.

Event

Das IEM wird im Modul *Event* gekapselt. Es stellt eine Schnittstelle zwischen umgebender Ausgabeplattform und den aktiven Dokumenten dar. Für jedes auftretende Ereignis werden im zugehörigen Dokument die dort registrierten Event-callback-Funktionen ausgelöst.

Style

Die tatsächliche Errechnung von CSS-Werten erfolgt im Modul *Style*, ergänzend zu den medienunabhängigen Berechnungen im Modul *StyleBase*. Das Modul *Style* enthält auch die CSS-Layoutvorlage für HTML-Dokumente, so daß solche Dokumente ohne CSS-Vorlage ausgegeben werden können.

Painter

Das Modul *Painter* implementiert die Ausgabe der vom Layouter vorbereiteten dokumenten-Struktur. Dabei wird auf die konkreten Funktionen der Zielplattform zurückgegriffen, d.h. für jede neue Zielplattform ist ein neues Painter-Modul zu erstellen.

Visual

Die übrigen plattformabhängigen Funktionen, die in der Präsentationsschicht benötigt werden, befinden sich im Modul *Visual*. Ähnlich wie das Modul *Painter* ist auch das Modul *Visual* an die Zielplattform anzupassen.

26.4 Gruppierung der Module

Werden die Module auf die entsprechenden Subsysteme verteilt, so ergibt sich die in Abbildung 111 dargestellte Systemarchitektur.

Die Architektur soll möglichst modular sein, so daß die Subsysteme durch andere Implementierungen ausgetauscht werden können. Lediglich innerhalb der Subsysteme ist eine starke Kopplung zwischen den Modulen vorhanden.

26.5 Anwendung der Bibliothek

Die Bibliothek kann auf verschiedenen Wegen genutzt werden. Im einfachsten Fall wird die im Modul AHD realisierte AHD-Laufzeitumgebung instanziiert, so daß aktive Hypertextdokumente geladen und ausgeführt werden können. Eine Reaktion auf externe Anfragen kann durch das Übergeben von Parametern wie Portnummer und Wurzelverzeichnis zum Starten des HTTP-Servers in der Laufzeitumgebung erreicht werden.

Durch den callback-Mechanismus kann das Verhalten der Bibliothek an definierten Stellen geändert werden. Es ist somit möglich, den Parse-Vorgang, die Ereignis-Behandlung, das Auflösen von externen Referenzen und die Ausführung von Verhaltensbeschreibungen zu steuern und zu ergänzen.

Eine Beispielanwendung ist der *Atlas*-Browser, der ein Minimalsystem für die Nutzung aktiver Hypertextdokumente darstellt. Im *Atlas*-Browser können AHDs gela-

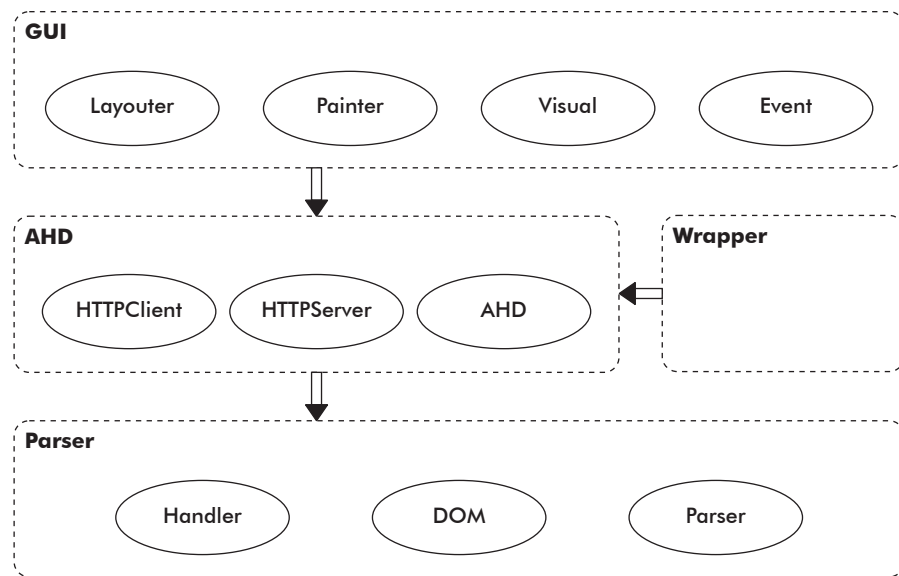


Abb. 111: Architektur der Kino-2-Bibliothek

den und dargestellt werden, die aktiven Dokumente lassen sich modifizieren und ihr Status kann überwacht werden. Der Browser nutzt den HTTP-Server der Kino-2-Bibliothek und kann so als eigenständiger Netzwerkknoten agieren. Der Atlas-Browser ist in Abbildung 112 abgebildet.



Abb. 112: Übersichtsfenster des Atlas-Browsers

27 Erweiterung bestehender Werkzeuge

Bei der Untersuchung bestehender Werkzeuge im XML- und WWW-Umfeld ist die weite Verbreitung von Java-basierten Implementierungen offensichtlich. Da dadurch alle wichtigen Aspekte und Standards unterstützt werden, soll an dieser Stelle die Erweiterung bestehender Java-basierter Werkzeuge im Vordergrund stehen. Dabei sollen die bei der Entwicklung des Prototypen gewonnenen Erkenntnisse berücksichtigt werden. Die erstellte Implementierung in Form einer Klassenbibliothek trägt den Namen *JKino*.

27.1 Vorhandene Werkzeuge

Im folgenden soll eine Auswahl für die Implementierung des AHDM zur Verfügung stehender Werkzeuge beschrieben werden. Deren Charakteristika bestimmen im weiteren Verlauf die Systemarchitektur und die zu ergänzende Funktionalität.

DOM-Schnittstellen

Die Schnittstellen für das DOM sind vom W3C bereits in der DOM-Spezifikation für die Programmiersprachen Java und JavaScript definiert worden. Die Java-Schnittstellen sind als Java-Interfaces realisiert, so daß eine konkrete Implementierung dieser Schnittstellen noch zu leisten ist.

XML4J

Das von der Firma IBM zur Verfügung gestellte Angebot an Werkzeugen für die Java-basierte Verarbeitung von XML-Dokumenten und darauf aufbauender Techniken ist sehr umfangreich. Die Komponenten werden dabei häufig als Java Beans gekapselt, was eine Integration in Entwicklungswerkzeuge und auch Anwendungssysteme erleichtert.

Die an dieser Stelle interessanteste Programmbibliothek ist der *XML Parser for Java* (XML4J) [80]. XML4J enthält sowohl einen ereignisbasierten als auch einen DOM-basierten Parser zusammen mit einer DOM-Implementierung. Der ereignisbasierte Parser ist SAX-konform und stellt die Grundlage für den darauf aufbauenen DOM-Parser dar. Beide Varianten existieren in einer validierenden und einer nicht-validierenden Ausführung. In JKino wird die nicht-validierende Klasse *DOMParser* eingesetzt. Das Zusammenspiel der zum Parsen eines XML-Dokuments nötigen Klassen zeigt Abbildung 113 (die Java-Packages sind zur Vereinfachung als nicht geschachtelte Subsysteme dargestellt).

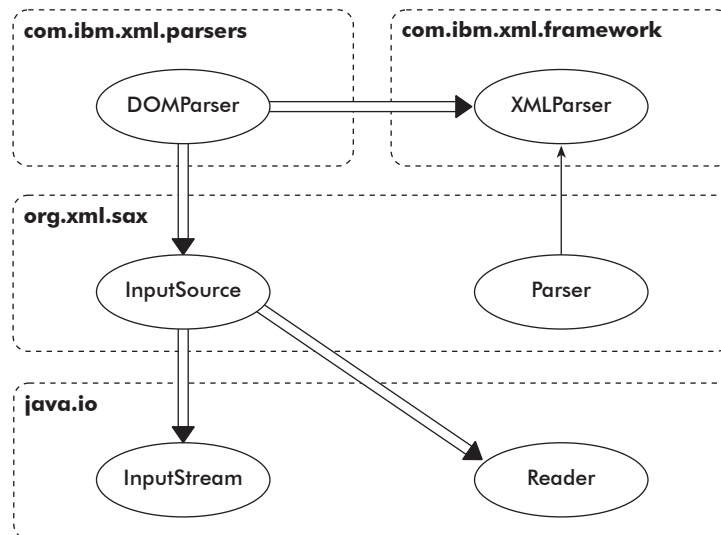


Abb. 113: Architektur bei der Nutzung des XML4J-DOM-Parsers

Eine interessante Möglichkeit, welche durch die XML4J-Implementierung eines DOM-Parsers und die DOM-Spezifikation gegeben ist, ist die Nutzung einer Fabrik-Klasse (entsprechend des im vorigen Abschnitt beschriebenen Fabrik-Musters) zur

Instanziierung eines konkreten DOM-Dokumentenobjekts. Da dies wiederum eine Fabrik-Klasse für die in einem DOM-Baum vorkommenden Knoten ist, können über eine angepaßte Fabrik-Klasse spezialisierte DOM-Bäume erzeugt werden. Dies wird weiter unten bei der Implementierung der AHDM-Funktionalität näher erläutert.

Java-Plattform-Bibliotheken

Wie bereits bei der Diskussion der verwandten Bereiche erwähnt stellt die Java-Plattform eine umfassende Basis für die Realisierung von Anwendungen unter Nutzung von Web-Techniken dar. Die Funktionalität wird dabei in zwei aufeinander aufbauenden Gruppen von Klassen umgesetzt. Grundlage bilden die Ein-/Ausgabeklassen im Paket *java.io*, zu nennen sind insbesondere die Stream- und Reader-Klassen, mit denen Datenquellen, -senken und verarbeitende Klassen implementiert werden können. Sie werden im Paket *java.net* genutzt, um z.B. Kommunikation über HTTP-Verbindungen zu ermöglichen, wobei alle HTTP-Methoden für eine Client-seitige Nutzung genutzt werden können. In JKino sollen sie für die Umsetzung des Kommunikationsmodell eingesetzt werden, d.h. die AHD-Laufzeitumgebung soll für die Realisierung der get-, put- und call-Funktionen auf den *java.net*-Klassen aufbauen.

Jetty

Die Realisierung der Server-Funktionalität erfordert eine offene Schnittstelle für die Bearbeitung von HTTP-Anfragen. Eine solche Schnittstelle kann einerseits durch die Verfügbarkeit eines erweiterbaren HTTP-Servers oder durch die Servlet-Schnittstelle erreicht werden. In dieser Arbeit soll der *Jetty*-Server [118] der Firma Mortbay genutzt werden, welcher beide Optionen bietet. Jetty ist ein Open-Source-Projekt, so daß der Quelltext des Servers zur Verfügung steht und auch modifiziert werden kann. Als Erweiterungsschnittstelle wird aber nur die Servlet-Schnittstelle genutzt, da damit die Erweiterungen auch in anderen, Servlet-fähigen Programmen eingesetzt werden kann.

Die Erweiterungen, die hier als Servlet implementiert werden sollen, entsprechen den im Prototyp bereits besprochenen Anbindung der AHD-Laufzeitumgebung, so daß der HTTP-Server bei entsprechenden Anfragen diese an die Laufzeitumgebung weiterleitet. Die Methoden, die das Servlet dafür implementieren muß, sind in Abbildung 114 gezeigt.

```
public interface Servlet {
    public void init (ServletConfig config);
    public ServletConfig getServletConfig ();
    public void service (ServletRequest req,
        SevletResponse res);
    public String getServletInfo ();
    public void destroy ();
}
```

Abb. 114: Servlet-Interface

Über dieses sehr einfache Interface können alle Servlet-fähigen HTTP-Server (d.h. solche Server, die eine *Servlet-Engine* implementieren) erweitert werden. In Jetty werden die Servlets dynamisch über eine Konfigurationsdatei den eintreffenden HTTP-Requests zugeordnet. Dieses Mapping erfolgt anhand der Pfad-Information im *ServletRequest*. Ein Servlet wird typischerweise nur einmal instanziiert und be-

handelt dann alle eintreffenden Anfragen konkurrent. Eine Änderung dieser Semantik läßt sich durch eine modifizierte Servlet-Engine erreichen, wird aber in JKino nicht benötigt. Ebenso sollen die in der Servlet-Spezifikation beschriebenen Mechanismen zum Sitzungsmanagement nicht verwendet werden. Dementsprechend ist die Laufzeitumgebung allein in der *service*-Methode eines Servlets anzubinden. Dazu steht zusätzlich im *javax.servlet.http*-Paket die Klasse *HttpServlet* zur Verfügung, die die HTTP-Anfragen GET, PUT, POST und DELETE an entsprechend zu implementierende Methoden weiterleitet.

Die Bearbeitung einer HTTP-Anfrage kann im Regelfall nur durch ein Servlet erfolgen. Die Koordination erfolgt über das an die *service*-Methode übergebene Objekt der Klasse *ServletResponse*. Erfolgt über dieses Objekt eine Ausgabe auf den Datenstrom zum Client, so können keine anderen Servlets mehr darüber Daten ausgeben. Andere Operationen können allerdings noch durchgeführt werden.

JACL

Die Ausführung der Verhaltensbeschreibungen in aktiven Hypertextdokumenten erfordert den Einsatz von Interpretern für verschiedene Programmiersprachen. Mittlerweile existieren in Java direkt und plattformunabhängig einsetzbare Interpreter für die wichtigsten Skriptsprachen, so etwa ECMAScript mittels des FESI-Interpreters [109] oder Python durch JPython [3].

Wegen seiner Einfachheit soll Tcl im Vordergrund stehen. Für die Ausführung von Tcl soll der JACL-Interpreter [161] in JKino integriert werden. JACL stellt eine fast vollständige Tcl-8-Umsetzung dar, so daß die Programmfragmente in den aktiven Dokumenten portabel ablauffähig sind.

Die Integration von Skriptsprachen in Anwendungen kann hier direkt über die Schnittstellen von JACL oder über das *Bean Scripting Framework* (BSF) [81] erfolgen. Da das BSF allerdings den Einsatz von Java Beans für die von den Skriptsprachen aus sichtbaren Java-Objekten vorschreibt, soll nur die direkte Anbindung eingesetzt werden. Hierfür wird die bereits bei der Kino-2-Bibliothek beschriebene Schnittstellendefinition der AHD- und DOM-Funktionen herangezogen. Aus dieser in XML vorliegenden Spezifikation wird mit Hilfe von Konvertierungsprogrammen entsprechender Zwischencode für die Bereitstellung der DOM-Funktionen von XML4J und der AHD-Funktionen der weiter unten beschriebenen AHDM-Implementierung erzeugt.

ICE Browser

Die Darstellung von HTML- und XML-Dokumenten mit CSS-Layoutvorlagen ermöglicht der *ICE Browser* [82] der Firma ICESoft. Der ICE Browser enthält auch eine DOM-Implementierung und läßt sich um einen JavaScript-Interpreter erweitern und IEM-Ereignisse behandeln. Die darzustellenden Dokumente können allerdings zur Zeit nur in textueller Form über Datenströme dem Browser übergeben werden. Eine Darstellung bereits aufbereiteter DOM-Strukturen (etwa aus einer Menge aktiver Hypertextdokumente in einer AHD-Laufzeitumgebung) ist allerdings für eine spätere Version fest vorgesehen.

Xalan

Um XML-Dokumente in HTML-Dokumente zu transferieren, können XSLT-Prozessoren eingesetzt werden. Der im Apache-Projekt entstandene XSLT-Prozessor *Xalan* [5] wird hier eingesetzt, um AHDs und XML-Dokumente auf der Server-Seite bei Bedarf in HTML-Dokumente unter Nutzung von XSLT-Stylesheets umzuwandeln.

27.2 Systemgrenze

Die Grenze des implementierten Systems kann enger gezogen werden als es bei der Entwicklung des Prototypen der Fall ist, da die oben beschriebenen bestehenden Werkzeuge genutzt werden sollen.

Konkret ist der XML4J-Parser in JKino einzubinden, zudem ist JKino in den Jetty-Server als Servlet zu integrieren. Weiter sind die AHD- und DOM-Funktionen von XML4J in den JACL-Interpreter aufzunehmen. Die Schnittstelle zur Netzwerkfunktionalität wird durch die Netzwerkklassen der Java-Plattform gebildet. Eine vorläufige Anbindung des ICE Browsers soll die Darstellung von AHDs erlauben, wobei allerdings die volle Funktionalität inklusive der Ausführung von Programmskripten in den AHDs nicht realisiert werden kann, da der ICE Browser entsprechende Schnittstellen noch nicht bietet.

27.3 Subsysteme

Im Vergleich zum Kino-2-Prototypen sind in JKino nur zwei Subsysteme vorhanden, da wesentlich mehr Systembestandteile bereits vorhanden sind. Als Subsysteme existieren die auch im Prototyp vorhandenen Cluster AHD und Wrapper.

27.4 Klassen

Analog zur Kino-2-Bibliothek wird die AHD-Funktionalität durch eine Klasse *AHD-Runtime* bereitgestellt, die Kapselung dieser Klasse und der DOM-Funktionen für den JACL-Interpreter erfolgt in der Klassen *JACLWrapper*, unterstützt durch das Interface *ScriptHandler*. Die Integration der Laufzeitumgebung in den Jetty-Server wird durch die Klasse *AHDServlet* realisiert. Die Nutzung des ICE Browsers wird in Unterkapitel 27.6 diskutiert

AHDRuntime

Die Klasse *AHDRuntime* ist eine direkte Implementierung der in Abbildung 44 gezeigten Schnittstelle. Eine Nutzung der AHD-Funktionalität ist durch Instanziierung eines Objekts dieser Klasse möglich, welches in JKino durch die Initialisierung des Servlets geschieht (siehe unten).

Im Gegensatz zum AHD-Modul des Prototypen deckt die *AHDRuntime*-Klasse nur die Verwaltung der aktiven Hypertextdokumente und die Realisierung der AHD-Funktionen ab. Die Anbindung an die umgebenden Systembestandteile wird durch die anderen hier besprochenen Klassen und die bereits verfügbaren Werkzeuge implementiert.

Die Erweiterung der Laufzeitumgebung ist in JKino vornehmlich auf das Hinzufügen neuer Skriptsprachen-Interpreter beschränkt, da der Parse-Prozeß und die dabei

aufzulösenden externen Referenzen durch den XML4J-Parser behandelt werden. Auch ist durch die nicht vollständige Anbindungsmöglichkeit des ICE Browsers eine erweiterte Ereignisbehandlung nicht vorgesehen. Dementsprechend wird in der AHDRuntime-Klasse nur eine Möglichkeit für die Registrierung von Skriptsprachen-Interpretern implementiert, welche durch die unten beschriebene Klasse JACLWrapper genutzt wird. Alternativ wäre bei einer Implementierung der Kernklassen in JKino als Java Beans auch der Einsatz des BSF denkbar.

ScriptHandler

Die Registrierung von Interpretern für die Ausführung von Programmskripten in aktiven Hypertextdokumenten erfordert die Definition einer gemeinsamen Schnittstelle für diese Interpreter. Sie wird durch das Java-Interface ScriptHandler beschrieben, welches die zu registrierenden Interpreter oder die dazugehörigen kapselnden Klassen implementieren müssen.

JACLWrapper

Die Klasse JACLWrapper stellt die Funktionalität zur Ausführung von Tcl-Code mit Nutzung der AHD- und DOM-Funktionen zur Verfügung. Dazu implementiert diese Klasse das ScriptHandler-Interface und kann so bei der Laufzeitumgebung als Skriptsprachen-Interpreter registriert werden.

Der Export der AHD- und DOM-Funktionen in einen Tcl-Interpreter kann auf unterschiedlichen Abstraktionsebenen erfolgen. Da die genannten Funktionen anders als bei den Modulen im Kino-2-Prototyp nicht isoliert stehen sondern durch Instanzen bestimmter Klassen realisiert werden, liegt eine objektorientierte Umsetzung auf der Tcl-Ebene nahe. So könnte der folgende Funktionsaufruf

```
ahd.call (current, "print", "text=Hello+World");
```

auf der Java-Ebene durch eine Nachahmung der Objekt-Semantik auf der Tcl-Ebene als

```
$ahd call $current print text=Hello+World
```

geschrieben werden. Es soll allerdings hier davon kein Gebrauch gemacht werden, da erstens die Schnittstellen wie sie im DOM beschrieben werden, keine Aussage über eine notwendigerweise objektorientierte Umsetzung machen, und zweitens die gemeinsame Grundlage für eine portable Implementierung in unterschiedlichen Skriptsprachen die prozedurale Schreibweise besser geeignet ist. So werden die exportierten Funktionen gemäß folgendem Beispiel im Tcl-Code genutzt werden:

```
AHDRuntimeCall $ahd $current print text=Hello+World
```

Diese Schreibweise ist zwar weniger kompakt, allerdings läßt sich die dahinter stehende prozedurale Umsetzung leichter analog in anderen Programmiersprachen implementieren.

AHDServlet

Um die Laufzeitumgebung in einen HTTP-Server zu integrieren, wird die Klasse AHDServlet als Adapter genutzt. Sie wird durch die Servlet-Engine des Servers instanziiert. Das Servlet wird für alle HTTP-Methoden und alle abrufbaren Pfade registriert. Im Servlet werden die eingehenden Anforderungen gefiltert und entsprechend der angeforderten Ressource und der darauf auszuführenden Operation behandelt oder ignoriert. Die Klasse AHDServlet ist von der Klasse HTTPServlet abgeleitet, so daß die Behandlung der Anfragen in den Methoden *doGet*, *doPut*, *doHead* und *doPost* erfolgt.

Die Verwendung eines Servlets erlaubt es, zusätzliche Verarbeitungsschritte bei der Behandlung von Anfragen durchzuführen. Dies wird der JKino-Bibliothek benutzt, um die nativen AHD-Dokumente bei Bedarf in HTML-Dokumente umzuwandeln, so daß sie in Standardbrowsern darstellbar sind. Dazu wird bei der Anfrage nach einem HTML-Dokument (d.h. bei einem GET-Request) ein gleichnamiges AHD gesucht. Existiert es, wird es durch den Xalan-XSLT-Prozessor umgewandelt. Dazu muß im AHD ein Verweis auf den zu benutzenden XSLT-Stylesheet in einer Processing Instruction existieren.

Die Behandlung von Anfragen entsprechend des AHDM werden durch ein Objekt der Klasse AHDRuntime durchgeführt. Dieses Objekt wird als Singleton (d.h. als Objekt einer nur einmal instanzitierbaren Klasse) durch das Servlet erzeugt.

27.5 Gruppierung der Klassen und Systemarchitektur

Die Aufteilung der Klassen auf die Subsysteme ergibt folgende Systemarchitektur:

AHD: Das Subsystem AHD enthält die Klassen AHDRuntime und AHDServlet.

Wrapper: Im Subsystem Wrapper befinden sich die Klassen ScriptHandler und JACLWrapper.

Abbildung 115 zeigt die Klassen und deren Beziehungen zu den vorhandenen Klassen (Die Nutzung der DOM-Funktionen über die Klasse JACLWrapper sowie die Einbindung des Xalan-XSLT-Prozessors ist aus Gründen der Übersichtlichkeit nicht dargestellt).

27.6 Anwendung der Klassenbibliothek

Die bis hierher beschriebenen Klassen können in unterschiedlichsten Anwendungen genutzt werden. Durch die gegebene Systemarchitektur ist eine Integration in ein Beispielprogramm ähnlich wie bei der Nutzung der Kino-2-Bibliothek möglich. Da der ICE Browser noch keine Verarbeitung bereits verarbeiteter XML-Dokumente in Form eines DOM-Baumes erlaubt, soll hier auf die Anbindung des Browsers verzichtet werden. Es wäre allerdings möglich, die textuellen Repräsentationen aktiver Hypertextdokumente durch den Browser darstellen zu lassen, und auf eine Synchronisierung der dargestellten Dokumente mit den zugrundeliegenden aktiven Hypertextdokumente zu verzichten. Zum anderen kann über einen XSLT-Stylesheet ein AHD in ein korrespondierendes HTML-Dokument umgewandelt werden, wobei allerdings auch hier beide Dokumente nicht synchronisiert sind.

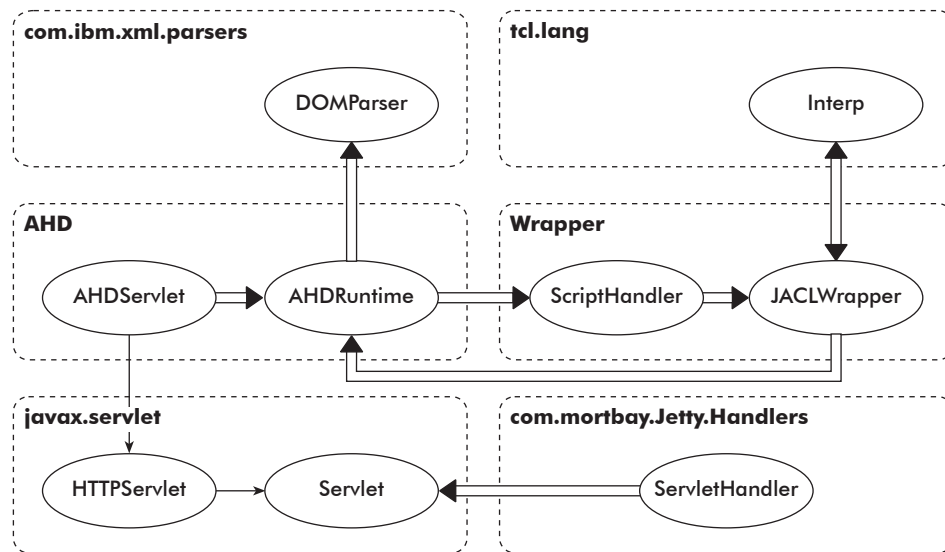


Abb. 115: Systemarchitektur von JKino

28 Diskussion

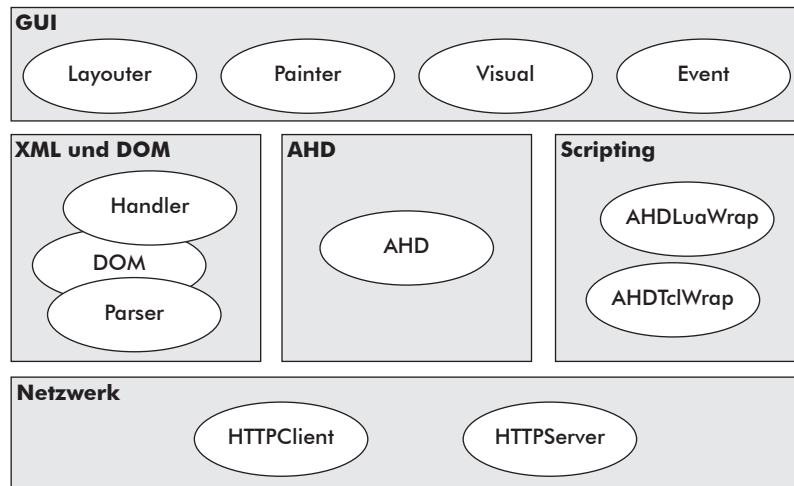
Die Anforderungen an Werkzeuge, welche die Nutzung von AHD-basierten Systeme erlauben, lassen sich anhand der Umsetzung des Kino-2-Prototypen untersuchen. Neben den Aufgabenbereichen, die auch in anderen Web-basierten Systemen abgedeckt werden müssen (etwa die Verarbeitung von XML- oder HTML-Dokumenten, die Netzwerkfunktionalität oder die Benutzerinteraktion) ist durch die im AHDM festgeschriebene Laufzeitumgebung ein weiterer Bereich definiert, welcher zu implementieren ist. Dabei erweist sich der Einsatz von IDL-Schnittstellenbeschreibungen als hinreichend zur Präzisierung der Funktionalität, u.a. auch deshalb, weil die Funktionalität zur Dokumentenmanipulation über das DOM ebenso per IDL-Beschreibungen definiert ist.

Überträgt man die Architektur des Kino-2-Prototypen auf eine Implementierung, die sich auf bestehende Werkzeuge stützt, so sind zwei Aufgaben zu erfüllen: Erstens ist die Kernfunktionalität wie sie in der Definition der Laufzeitumgebung beschrieben ist, zu realisieren, zweitens sind die einzelnen Systemkomponenten miteinander zu verbinden. Diese Verbindung erfolgt in der JKino-Klassenbibliothek über Schnittstellenklassen. Im Fall der Anbindung der DOM- und Laufzeitumgebung an die Skriptspracheninterpreter können sie weitgehend automatisiert erstellt werden. Sowohl im Kino-2-Prototypen als auch in JKino kommt dabei eine weitere Schnittstellenbeschreibung in Form einer XML-Datei zum Einsatz.

Es zeigt sich, daß die Ergänzung bestehender Komponenten um AHD-Funktionalität nur einen relativ geringen Mehraufwand bedeutet. In Abbildung 116 werden die Module und Klassen der beiden Werkzeuge den Funktionsbereichen zugeordnet, die in dieser Arbeit implementierten Komponenten sind dabei hervorgehoben. Wie aus der Abbildung zu sehen ist, wird JKino die eigentlich neue Funktionalität durch nur eine einzige zusätzliche Klasse implementiert (**AHDRuntime**), unterstützt durch zwei weitere Klassen (**AHDServlet** und **ScriptHandler**). Diese einfache Umsetzbarkeit ist

einerseits durch die weitgehend vorhandene, präzise Beschreibung der Schnittstellen für die Grundlagentechniken (wie etwa das DOM oder die HTTP-Server-Erweiterung über Servlets) möglich, andererseits zeigt sich dadurch auch die hohe Interoperabilität und Flexibilität des AHDm auch auf der Werkzeugebene.

Funktionsbereiche in Kino-2



Funktionsbereiche in JKino

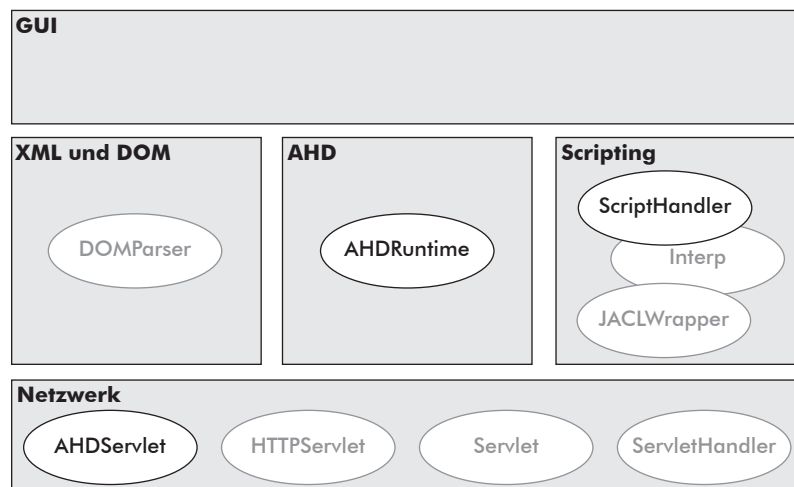


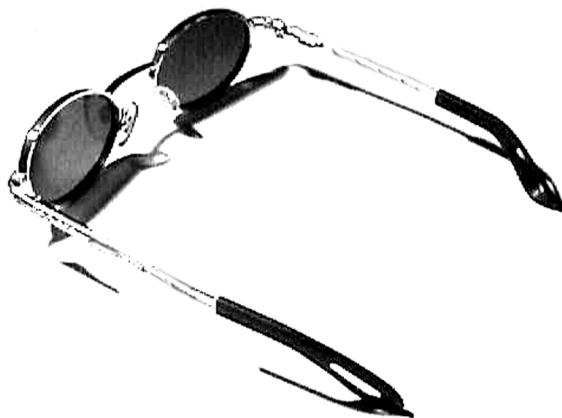
Abb. 116: Vergleich der Abdeckung der Funktionsbereiche

Bei der Umsetzung sind neben den hier gezeigten Werkzeugen auch andere Ausprägungen möglich. Der Kern der AHD-Funktionalität besteht in der Umsetzung der Funktionen zum Zugriff auf Zustandselemente und die Ausführung von Verhaltensbeschreibungen. Darüber hinaus vorhandene Möglichkeiten wie die Netzwerkfunktionalität sind nicht zwingend notwendig, dies gilt ebenso für die Anbindung von aktiven Dokumenten an eine Benutzerschnittstelle. In diesem Zusammenhang könnte ein formaler Vereinbarungsmechanismus zwischen AHD-Anwendung und umgebendem System die charakteristischen Eigenschaften der Umgebung die Adaption

einer Anwendung an unterschiedliche Umgebungen vereinfachen. Dazu könnte etwa eine RDF-Beschreibung der Fähigkeiten einer Laufzeitumgebung eingesetzt werden. In diesem Abschnitt wurde aus Gründen der Vereinfachung auf die Umsetzung von Werkzeugen verzichtet, die die Erstellung AHD-basierter Systeme und die in dieser Arbeit vorgeschlagene Entwicklungsmethode unterstützen. Dennoch lassen sich gewisse Anforderungen treffen, die bei einer Implementierung solcher Werkzeuge wichtig sind. Dazu zählt insbesondere die Verwaltung wiederverwendbarer Komponenten, die graphische Dokumentation von Systemen in der Entwurfsphase und nicht zuletzt auch die Unterstützung bei der Bearbeitung bereits bestehender Systeme. Interessant erscheint auch eine Anbindung an Web-basierte Entwicklungswerkzeuge wie *OzWeb* [91] oder *Hyperform* [188], zumal die Komponenten AHD-basierter Anwendungen in solchen Werkzeugen durch ihre XML-Basis einfach zu integrieren sind. Durch die Verwendung von offenen Standards als Basis für das AHDM ist es aber auch denkbar, Standardwerkzeuge wie XML-Editoren oder Site-Management-Systeme für die Erstellung von AHD-Systemen einzusetzen.

Abschnitt F

Ausblick



29 Zusammenfassung

Der letzte Abschnitt dieser Arbeit soll eine Bewertung der bisherigen Ergebnisse liefern, gefolgt von einer Darstellung der darüber hinausgehenden Perspektiven. Die hier stattfindende Bewertung unterscheidet sich dabei von den abschließenden Diskussionen der einzelnen Abschnitte durch eine übergreifende Sichtweise, bei der die Ergebnisse der Abschnitte im Gesamtzusammenhang beurteilt werden sollen.

29.1 Vorgehensweise in der Arbeit

Die Vorgehensweise, die für diese Arbeit gewählt wurde, sieht eine sequentielle Erarbeitung von Ergebnissen vor. Ausgehend von einer Aufstellung und Analyse von Anforderungen an Entwurfsmodelle für Web-basierte Informationssysteme wird ein Modell vorgestellt, welches über beispielhafte Anwendungen verifiziert werden soll. Unterstützend dazu werden eine Vorgehensmethode und Werkzeuge beschrieben, welche jeweils wiederum ausgehend von Anforderungsanalysen entworfen sind.

Im Rückblick erscheint das Modell aktiver Hypertextdokumente für die Entwicklung von Web-basierten Systemen verwendbar zu sein, wobei allerdings eine Unterstützung durch Methode und Werkzeuge wichtig ist, insbesondere um die sog. weichen Qualitätsanforderungen wie Wiederverwendbarkeit und Flexibilität zu erfüllen. Eine rein theoretische Überprüfung des Modells wäre im Vergleich zu einer Verifizierung anhand von praktischen Anwendungen sicherlich wesentlich schwieriger durchzuführen und würde wahrscheinlich weniger aussagekräftige Ergebnisse zur Folge haben. Andererseits kann durch die hier nur knapp dargestellten und nur einen bestimmten Bereich abdeckenden Beispielanwendungen sicherlich keine allgemeingültige Empfehlung zugunsten des AHDM gemacht werden. Dennoch sollen im folgenden die u.a. auch durch die Anwendung des Modells erkennbaren Charakteristika des AHDM und AHD-basierter Systeme zusammenfassend dargestellt werden.

29.2 Kontext des Modells

Die oben angesprochene sequentielle Vorgehensweise resultiert in einer Einbettung des AHDM entsprechend Abbildung 117.

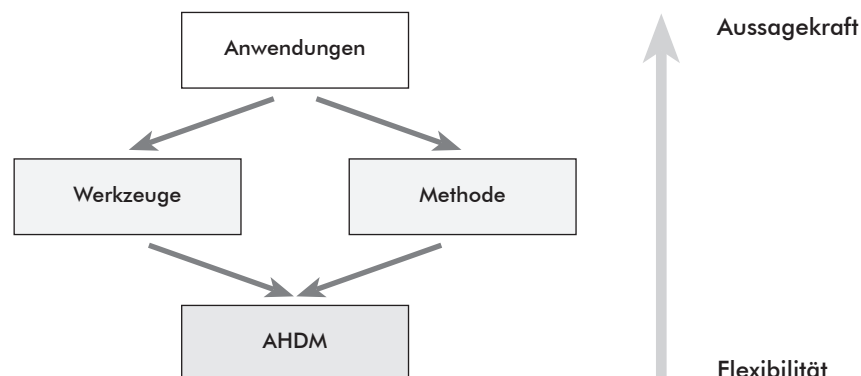


Abb. 117: Einbettung des AHDM

Das Modell selbst besitzt hierbei für sich alleine die größte Flexibilität, allerdings gewinnt es erst durch die vorgeschlagene Entwicklungsmethode und den Kontext der benötigten Werkzeuge an Ausdruckskraft. So sind die vorgestellten Semantiken in Form von Entwurfsmustern nicht fester Bestandteil des Modells, erhöhen aber die Bedeutung von bestimmten Konfigurationen. Zwar wird durch die Vorgabe solcher Muster und einer Entwurfsmethode die ursprüngliche Flexibilität des Modells eingeschränkt, dies spiegelt aber die oft anzutreffende Notwendigkeit des Abwägens zwischen beiden Eigenschaften wieder, die sich so etwa auch beim Entwurf von Programmiersprachen zeigt. Wie allerdings beim Abschluß der Vorstellung der Entwicklungsmethode erwähnt, soll das AHDM nicht fest an die Methode gebunden sein. Gleiches gilt für die Implementierungen der Werkzeugprototypen, vielmehr ist ja eine der ursprünglichen Anforderungen die Implementierungsunabhängigkeit des Modells. Allein die Konzepte, die hinter den Werkzeugen stehen, sind bindend (d.h. die Nutzung offener Standards oder die Art der Anbindung an Skriptspracheninterpreter).

29.3 Submodelle des AHDM

Ähnlich wie die Einbettung des AHDM in eine Schichtenarchitektur sind auch die einzelnen Kernkomponenten des AHDM aufeinander aufbauend angeordnet, wie es in Abbildung 118 zu sehen ist. Basis des AHDM ist das Informationsmodell, welches den Aufbau und die Bedeutung der Bestandteile aktiver Hypertextdokumente beschreibt. Aus dem Informationsmodell resultiert ein Objektmodell, welches die Zuordnung zwischen Funktions-, Variablen- und den übrigen Elementen in einem AHD über Delegationsmechanismen steuert. Die Verbindung aktiver Hypertextdokumente wird im Kommunikationsmodell geregelt, wobei mehrere Bereiche abgedeckt werden. So werden neben den beteiligten Kommunikationspartnern auch die transportierten Inhalte und die Adressierung der Komponenten beschrieben. Auf diesen beiden Modellen baut das Ausführungsmodell (realisiert durch die Laufzeitumgebung) auf, das beide Modelle zusammenfügt, d.h. es definiert, wie mittels des Kommunikationsmodells dem Informationsmodell entsprechende Komponenten Verhaltensbeschreibungen aktivieren können.

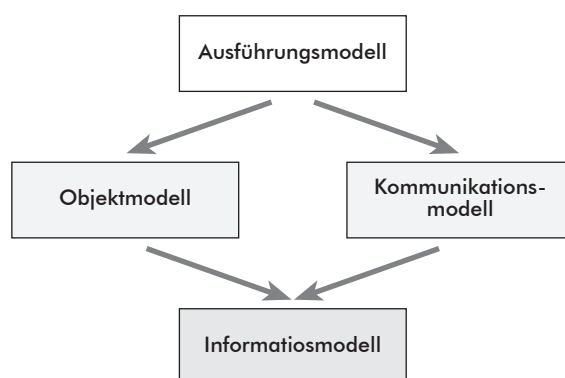


Abb. 118: Zusammenspiel der Submodelle im AHDM

Der Vorteil einer weiteren Aufteilung des AHDM in Submodelle liegt in der flexiblen Nutzung auch von einzelnen Submodellen. Dabei sind insbesondere Informations- und Kommunikationsmodell zu nennen, die durch den Einsatz offener

Standards auch in anderen Umgebungen zum Einsatz kommen können. Es ist z.B. denkbar, auch nicht-AHD-fähige Anwendungen über die Nutzung von HTTP entsprechend des Kommunikationsmodells einzubinden oder die Nutzdaten aktiver Hypertextdokumente von den nur im Kontext des AHDM relevanten Daten zu trennen.

29.4 Lose Kopplung der Systembestandteile

Untersucht man die Verbindungen der Bestandteile AHD-basierter Anwendungen, so ist auf jeden Fall die sehr lose Kopplung der Komponenten interessant. Sie wird vornehmlich durch den Einsatz von XML-Namensräumen und die lose Referenzierung von Elementen mittels Attributen oder Struktur-Matching erreicht.

XML-Namensräume lassen eine transparente Vermischung von XML-Elementen (samt Inhalt) aus unterschiedlichen Quellen zu. Die zusammengeführten Informationen können über die eindeutige Kennzeichnung der Herkunft aber genauso wieder getrennt werden. XML-Namensräume bilden durch diese Möglichkeiten eine wesentliche Grundlage für das Informationsmodell, zumal eine der Kernanforderungen an das AHDM die Interoperabilität ist, und Namensräume eine gezielte Verwendung der Information je nach Anwendungsgebiet ermöglichen.

Die zweite im AHDM genutzte Technik zur losen Kopplung ist die Nutzung von Attributwerten statt Elementnamen zur Referenzierung von Elementen. Dieses Vorgehen hat zwei Vorteile: Erstens können die benutzten Attribute unabhängig vom ursprünglichen Einsatzzweck der Elemente gewählt werden (vergleichbar mit XML-Namensräumen), zweitens können Attribute in DTDs transparent bestehenden Attributdefinitionen hinzugefügt werden, falls die Validität eines Dokuments gewährleistet sein soll.

Der Einsatz von Namensräumen wird im AHDM genutzt, um das Informationsmodell zu realisieren und die AHDM-spezifischen Elemente von den Nutzdaten eines Dokuments zu trennen. Attribute werden eingesetzt, um Elemente beim Zugriff über die Laufzeitumgebung oder im Rahmen von Entwurfsmustern wie dem Rollen-Muster zu referenzieren.

Durch die lose Kopplung der Komponenten ergibt sich ein wichtiger Unterschied zu objektorientierten Systemen, denen bestimmte Konzepte im AHDM ähneln, insbesondere die Zuordnung von Methoden und Zustandsvariablen zu einem XML-Element. Vergleicht man die Kohäsion zwischen Objekten und den zugeordneten Methoden oder Variablen in objektorientierten Programmiersprachen mit den Art und Weise, wie im AHDM objektähnliche Komponenten gebildet werden können, so ist zu bemerken, daß XML-Elemente mit den Mitteln des AHDM wesentlich einfacher um Methoden und Zustandsvariablen erweitert werden können, und daß diese Bindung genauso einfach wieder aufgelöst werden kann. So lassen sich virtuelle Objekte bilden, die nur bei Bedarf und je nach Einsatzzweck um Methoden und Variablen erweitert werden. In objektorientierten Programmiersprachen hingegen ist eine solche Zuordnung meist fest durch Klassenbeschreibungen gegeben. Nur in Sprachen, die ein dynamisches Objektmodell verwirklichen (z.B. XOTcl oder CLOS [166]) ist eine ähnlich große Flexibilität bei der Bildung von Objekten gegeben. Allerdings können im AHDM auch Elemente ohne Methoden und Zustandsvariable sinnvoll als Nutzdaten eingesetzt werden, während in objektorientierten Programmiersprachen Objekte ohne Methoden und Variable selten genutzt werden.

29.5 Infrastruktur

Im Vergleich zu Systemen und Techniken, die an spezielle Anwendungen angepaßt sind, wie etwa die bei der Erläuterung der verwandten Bereiche beschriebenen Client- und Server-seitigen Implementierungen oder anderen, implementierungsnahen Ansätzen ist zu bemerken, daß das AHDM weniger Ausdrucksstärke besitzt. Vielmehr müssen anwendungsspezifische Aufgaben mit mehr Aufwand umgesetzt werden. So enthält z.B. das ActiWeb-System ein Repository für die Verwaltung von mobilen Agenten, ein Mechanismus, der bei der Verwendung des AHDM nicht direkt zur Verfügung steht.

Wie aber weiter oben bereits angedeutet, sind in vielen Situationen Kompromisse zwischen Ausdrucksstärke und Flexibilität zu treffen. Im Fall des AHDM ergibt sich ein wichtiger Grund für den Vorzug von Flexibilität in der Ausrichtung auf die Bereitstellung einer Infrastruktur für unterschiedlichste Systeme. Dieses Ziel sollte bei einer Bewertung eines möglichen Einsatzes des AHDM berücksichtigt werden.

30 Offene Punkte

Nachdem die besonderen Charakteristika des AHDM herausgestellt worden sind und die Erfüllung von Anforderungen an das AHDM und die damit verbundenen Methoden und Werkzeuge in den einzelnen Abschnitten dieser Arbeit betrachtet worden ist, soll in diesem Kapitel noch ein Ausblick auf nicht behandelte bzw. problematische Bereiche gegeben werden.

30.1 Sicherheitsaspekte

Eine sehr wichtige Aufgabe bei der Implementierung von Systemen, die sich auf Informationsdistribution und mobilen Programmcode stützen, ist die Sicherstellung diverser Sicherheitsaspekte. Daneben existieren allgemeine Sicherheitsrisiken in Web-basierten Anwendungen. Einen Überblick über diese Thematik liefert [54], dort werden drei Bereiche genannt, die Sicherheitsrisiken ausgesetzt sein können: Die Information selbst, die Übertragungswege, über die die Information übermittelt wird, und schließlich das informationsverarbeitende System. Generell hängt dabei die Bedeutung der Informationssicherheit vom Grad der Offenheit des Systems ab, die höchste Bedeutung kommt den Sicherheitsaspekten bei offenen Systemen zu, die mit anderen, externen Systemen kooperieren müssen und bei denen Informationen und auch Programmcode über Systemgrenzen hinweg ausgetauscht wird.

Da dieser Bereich jedoch sehr umfangreich ist und der Schwerpunkt des AHDM auf der Bereitstellung eines Modells für die Implementierung einer Infrastruktur für Web-basierte Informationssysteme liegt, sollen hier nur ausgewählte Punkte diskutiert werden. Zudem bestehen oft unterschiedliche Möglichkeiten, Sicherheitskonzepte zu realisieren, etwa durch die Implementierung der Konzepte entweder auf Anwendungsebene oder auf niedrigeren Ebenen wie der Protokoll- oder der Programmiersprachenebene. Deswegen soll hier nur auf die grundlegenden Aufgaben eingegangen werden, die für die Realisierung von sicheren verteilten Systemen erfüllt werden müssen.

Integrität

Die Integrität von Daten spielt eine zentrale Rolle für die Sicherheit von Informationssystemen, da darauf weitere Konzepte aufgebaut sind. Bezogen auf das AHDM bedeutet Integrität von Daten den Schutz von aktiven Hypertextdokumenten vor unbeabsichtigter oder unzulässiger Veränderung.

Um die Integrität von Dokumenten zu sichern, können zusätzliche Informationen, die charakteristisch für ein Dokument sind und sich bei Modifikation des Dokuments ebenso ändern, angegeben werden. Üblicherweise werden hierzu Prüfziffern eingesetzt, wie sie etwa die MD5- [153] oder SHA-1-Algorithmen [122] liefern.

Ein interessanter Aspekt bei der Verwendung von XML als Basisformat für auszutauschende Dokumente ist die Strukturierbarkeit der Dokumente. Dadurch ist es z.B. möglich, bei Bedarf nur bestimmte Elemente eines Dokuments mit einer Prüfsumme zu versehen. Im einfachsten Fall kann die Prüfsumme als weiteres Attribut dem zu sichernden Element hinzugefügt werden. Im Zusammenhang mit dem AHDM würde sich anbieten, Prüfsummen über die func-Elemente zu bilden, um deren Inhalt vor Modifikationen zu schützen. Allerdings sind zur vollständigen Sicherung vor Modifikationen noch weitere Maßnahmen wie etwa asymmetrische Verschlüsselung notwendig, die weiter unten beschrieben werden.

Ein weiterer wichtiger Punkt bei der Sicherung der Integrität von XML-Dokumenten sind die unterschiedlichen möglichen physikalischen Repräsentationen eines XML-Dokumentes durch die Option, Leer- und Sonderzeichen zu ignorieren. Um dementsprechend für ein XML-Dokument die Prüfsumme zu errechnen, sollte es vorher in eine eindeutige (*kanonische*) Form gebracht werden. Details hierzu sind in [33] beschrieben.

Vertraulichkeit

Die Sicherung der Vertraulichkeit von Daten ist weniger für die aktiven Bestandteile eines aktiven Dokuments von Interesse, als vielmehr für die Nutzdaten. Vertraulichkeit soll hierbei gewährleisten, daß Informationen nur von einem bestimmten Adressaten gelesen werden können. Dies kann durch asymmetrische oder symmetrische Verschlüsselungsverfahren wie Blowfish [159] oder RSA [152] erreicht werden. Auch hier besteht die Möglichkeit, nur bestimmte Teile eines XML-Dokuments zu verschlüsseln. Gerade bei den rechenintensiven asymmetrischen Algorithmen und großen Dokumenten lohnt eine solche Unterteilung in kritische und nicht-kritische Information.

Neben der Verschlüsselung von Dokumenten können auch die Kommunikationswege gesichert werden, über die die Dokumente übertragen werden. Im AHDM umfaßt dies beispielsweise die Nutzung von HTTP für Methodenaufrufe oder Variablenzugriffe. Hier existiert mit dem *Secure Sockets Layer* (SSL) [64] ein Mechanismus, um Socket-basierte Netzwerkverbindungen (und damit auch die im AHDM benutzten HTTP-Verbindungen) abzusichern. Allerdings besteht bei der alleinigen Nutzung gesicherter Übertragungskanäle das Risiko der Offenlegung von Daten innerhalb des Anwendungssystems selbst, etwa durch einen Angriff innerhalb der AHDM-Laufzeitumgebung, weshalb eine Verschlüsselung auf Dokument- oder Elementebene bevorzugt werden sollte.

Authentisierung

Die eindeutige Zuordnung eines Dokuments zu einem Autor oder einer erzeugenden Quelle wird über Authentisierungsmechanismen gewährleistet. Auch hier können asymmetrische Verschlüsselungsverfahren genutzt werden, jedoch spielt die Infrastruktur, mit der die Schlüssel verwaltet und zugänglich gemacht werden, eine große Rolle. Im AHDM kann die Authentisierung genutzt werden, um zusammen mit der Sicherung der Integrität eindeutige Aussagen über die Herkunft und den Inhalt von Verhaltensbeschreibungen zu machen. Solche Aussagen werden für die Entscheidung der Authorisierung von Operationen benötigt, wie weiter unten dargestellt wird. Authentisierung und Integrität kann im XML-Umfeld über die *XML-Signature*-Spezifikation [12], welche zur Zeit in der Entwurfsphase ist, hergestellt werden. Sie enthält eine Beschreibung sog. digitaler Signaturen, die in XML verfaßt sind, und die Unterschrift beliebiger Ressourcen erlaubt.

Authorisierung

Eine wichtige Aufgabe in Verbindung mit dem AHDM ist die Kontrolle der Authorisierung aktiver Dokumente, insbesondere die Kontrolle über die Ausführung der Verhaltensbeschreibungen. Gerade bei verteilten Systemen, bei denen von der Mobilität aktiver Dokumente Gebrauch gemacht wird, ist dieser Aspekt verstärkt zu untersuchen.

Es kommen zwei mögliche Ansätze in Frage, die auch miteinander kombiniert werden können: Einerseits können unerwünschte Zugriffe durch Schaffung einer gesicherten Umgebung (*sandbox*) für die Ausführung von Programmen vermieden werden (vgl. dazu beispielsweise [26] oder [194]), andererseits können Operationen aufgrund einer Zugriffskontrollfunktion erlaubt werden. Im AHDM ist der erste Ansatz indirekt realisiert, indem nur die DOM- und AHDM-Schnittstellen zur Verwendung in den eingebetteten Programmfragmenten eines AHDs exportiert werden. Es handelt sich also um eine nach außen abgesicherte Umgebung mit einer definierten Menge verfügbarer Methoden. Allerdings ist durch die AHD-Schnittstellen der Zugang zur Netzwerkschicht der Laufzeitumgebung möglich, was in bestimmten Fällen aus Sicherheitsgründen unerwünscht sein kann. Auch können durch die eingesetzten Programmiersprachen Sicherheitslücken entstehen.

Sollen die Ausführung der Methoden zusätzlich kontrolliert werden, etwa um die Kommunikation mit externen Dokumenten zu verhindern, so böte sich die Einführung einer Zugriffskontrollfunktion an, die aufgrund des agierenden Subjekts (dem aktiven Hypertextdokument, identifiziert durch eine URI), der auszuführenden Operation (einer Methode aus der Schnittstelle der Laufzeitumgebung inklusive der DOM-Operationen) und dem benutzten Objekt (wiederum ein Dokument oder Teile davon, identifiziert entweder durch eine URI oder eine DOM-Referenz in Form eines programmiersprachenspezifischen Zeigers) die Ausführung der Methode verweigern oder zulassen kann [120].

Die Einführung einer solchen Zugriffskontrollfunktion ist transparent für die Implementierung der aktiven Hypertextdokumente, zudem ist eine Implementierung von geringer Komplexität. Allein die Konfiguration der Funktion durch die Definition erlaubter oder verbotener Kombinationen von Subjekt, Operation und Objekt ist innerhalb der Laufzeitumgebung notwendig. Da diese Konfiguration aber stark anwendungsabhängig ist, soll von einer weiteren Beschreibung an dieser Stelle ab-

gesehen werden. Sie ist Bestandteil eines Sicherheitsmanagements, das die Regeln für die Informationssicherheit applikationsweit steuern muß. Unabhängig von der Einführung einer solchen Zugriffskontrolle bleibt aber das in aktiven Hypertextdokumenten nutzbare API der Laufzeitumgebung gleich.

Um die Herkunft des Programmcodes sicherzustellen sind die weiter oben beschriebenen Verfahren zur Authentifizierung notwendig. Die einfachste Art der Authentifizierung würde ein generelles Ausführen von Code abhängig von der Herkunft und nicht differenziert nach einzelnen Operationen oder zu bearbeitenden Objekten erlauben. Die für das AHDM wohl angemessenste Form der Prüfung würde weitergehend alle AHD-Funktionen über eine Kontrollfunktion absichern, zumal die zu modifizierenden Daten durch eine leicht verwaltbare URI-Referenz identifiziert werden.

Neben der Zugriffs- und Ausführungskontrolle der aktiven Bestandteile eines AHDs kann auch der Zugriff auf die Nutzdaten abgesichert werden. Insbesondere die vernetzte Natur von XML-Dokumenten stellt hier andere Anforderungen an Sicherheitstechniken als herkömmliche Dokumente, da durch die Vernetzung virtuelle Dokumente bestehend aus mehreren verbundenen physikalischen Dokumenten gebildet werden können. Ein Autorisierungsmechanismus, der speziell auf diese Anforderungen ausgerichtet ist, wird in [53] beschrieben.

30.2 Plattformunabhängigkeit

Ein wichtiges Ziel beim Entwurf des AHDM ist die Plattformunabhängigkeit. Allerdings sind hierbei zwei Probleme zu beobachten. Zum einen muß eine Laufzeitumgebung auf jeder Zielplattform vorhanden sein, zum anderen muß für jede in den Programmfragmenten eingesetzte Programmiersprache ein Interpreter in die Laufzeitumgebung integriert werden. Um diese Probleme abzuschwächen oder zu lösen sind mehrere Ansätze denkbar.

Die Implementierung einer Laufzeitumgebung hängt stark von der zur Verfügung stehenden DOM-Implementierung ab. Einerseits macht die Integration der Laufzeitumgebung ohne eine DOM-Implementierung wenig Sinn, andererseits hilft die DOM-Implementierung bei der Integration. Ist eine solche Implementierung vorhanden und hinreichend stark in Komponentenform in ein bestehendes Werkzeug integriert, so kann im Regelfall die AHD-Laufzeitumgebung als darauf aufbauende Komponenten umgesetzt werden. Dies ist z.B. bei XML4J oder Surflit der Fall. Generell scheint daneben mit der Java-Plattform eine geeignete Umgebung für die komponentenbasierte Zusammenstellung von Werkzeugen zu existieren, wobei die Plattformunabhängigkeit hier durch die Design-Ziele von Java unterstützt wird. Neben der Verfügbarkeit wohldefinierter Schnittstellen vieler vorhandener Java-Komponenten spielt im weiteren Rahmen auch der Trend zu Open-Source-Software [139] eine wichtige Rolle, da damit bestehende Implementierungen anhand der offenen Quelltexte erweitert werden können. Die im vorigen Abschnitt vorgestellten Prototypen können in diesem Zusammenhang als Studie der benötigten Funktionalitäten dienen. So sind mittlerweile XML-Techniken und -Werkzeuge hinreichend weit verbreitet, um auf diversen Plattformen eingesetzt werden zu können. Auch die Netzwerkfunktionalität steht sehr oft in Form der angesprochenen Socket-Schnittstelle zur Verfügung. Allein die Präsentationsschicht, die das Ereignis- und Darstellungsmodell mittels IEM und CSS umsetzt, ist noch nicht in hinreichender Form platt-

formübergreifend implementiert. Hier entsteht jedoch mit dem plattformübergreifenden Mozilla-Projekt [119] ein umfassendes Rahmenwerk für XML-basierte Anwendungen, welches alle relevanten Standards wie DOM, CSS oder IEM abdecken soll.

Ein weiterer Punkt ist das Einsatzgebiet aktiver Hypertextdokumente, welches sich nicht nur auf interaktive Anwendungen unter Nutzung bereits verfügbarer Client-Programme beschränkt, sondern auch solche Systeme umfaßt, in denen die Werkzeuge vollständig neu zusammengestellt und mit AHD-Funktionalität angereichert werden können. Auch ist durch das Informationsmodell eine transparente Nutzung der Nutzdaten in nicht-AHD-fähigen Umgebungen weiter möglich.

Die Verwendung unterschiedlicher Programmiersprachen in aktiven Hypertextdokumenten ist problematischer, da im Gegensatz zu den fest vorgeschriebenen DOM- und AHD-Schnittstellen die in einer Laufzeitumgebung verwendbaren Programmiersprachen nicht fixiert sind. Grundsätzlich sollte über die Beschränkung auf die Nutzung der genannten Schnittstellen und Vermeidung des Einsatzes darüber hinausgehender Möglichkeiten der jeweiligen Programmiersprachen wie etwa Funktions- oder Klassenbibliotheken eine Kompatibilität wenigstens auf der funktionalen Ebene der Programmfragmente erreichbar sein. Dies entspricht auch dem Sinn der implementierungsunabhängigen Beschreibung der Schnittstellen über die IDL.

Neben der Beschränkung auf eine auf möglichst vielen Plattformen verfügbare Programmiersprache wie Tcl kann u.U. auch das bereits beschriebene Bean Scripting Framework im Java-Umfeld Basis für einen Lösungsansatz sein. Da das BSF eine wohldefinierte Schnittstelle für die Vermittlung zwischen Programmierspracheninterpreter und Anwendungsobjekten bereitstellt, ließen sich so auch dynamisch zur Laufzeit neue Interpreter in die Laufzeitumgebung integrieren. Allerdings würde hierfür eine Implementierung der Laufzeitumgebung und aller zugreifbarer Objekte als Java Beans nötig sein.

30.3 Laufzeitverhalten

Ein bei der Verwendung von XML und Interpretersprachen grundsätzlich zu betrachtender Aspekt ist das Laufzeitverhalten der damit umgesetzten Systeme. So ist die Verarbeitung von XML in der Regel nur über den (initialen) Zwischenschritt des Parsens möglich, der allerdings bei der Verwendung binärer Daten entfällt. Auch benötigt XML in seiner nicht-komprimierten Form mehr Speicherplatz (allerdings lassen sich XML-Dokumente sehr gut komprimieren), läßt sich so aber mit textbasierten Werkzeugen einfacher weiterverarbeiten.

Der Einsatz von interpretierten Programmiersprachen ist nicht zwingend im AHDM festgelegt, allerdings werden solche Sprachen bevorzugt, da sie einfach in aktive Hypertextdokumente eingebunden werden können und auch eine nachträgliche Veränderung der textbasierten Verhaltensbeschreibungen erlauben. Es gibt zudem einige Techniken wie *just-in-time*-Übersetzung oder Bytecodes, die die Ausführungsgeschwindigkeit von interpretierten Programmiersprachen erhöhen und transparent eingesetzt werden können.

Es ist offensichtlich, daß Systeme, die auf dem AHDM basieren, im Laufzeitverhalten nicht geeignet sind, Konzepte wie etwa die in [175] vorgestellten aktiven Netzwerknachrichten zu implementieren. Allerdings stellen Skriptsprachen eine flexible und einfach zu handhabende Basis für unterschiedlichste Zwecke zur Verfügung.

Weiterhin wiegen im AHDM die Eigenschaften von XML zur Unterstützung der Interoperabilität und zur Realisierung des Informationsmodells die Nachteile der aufwendigeren Verarbeitung und des erhöhten Speicherbedarfs auf. Zudem können zeitkritische Bestandteile einer Anwendung in Sprachen wie C realisiert und als externe Komponente eingebunden werden, wobei die Komponenten durch aktive Hypertextdokumente und den darin enthaltenen Skripten in der Art einer *glue language* [137] angesteuert werden. Die Verwendung von HTTP erleichtert diese Einbindung durch eine einfache Implementierbarkeit und weite Verbreitung. Damit ergibt sich der Vorteil der flexiblen Definition der Applikationssemantik auf AHD-Ebene und der Umsetzung laufzeitkritischer Bestandteile in externen Komponenten. Darüber hinaus ist eine Erweiterung der Laufzeitumgebung um wichtige, oft wiederverwendbare Funktionen denkbar. Zusammenfassend gesagt ist also beim Einsatz des AHDM zwischen Laufzeitverhalten und Flexibilität abzuwägen, wobei Freiheitsgrade in beide Richtungen bestehen.

31 Fazit

Zusammenfassend sollen noch einmal einige wichtige Punkte herausgestellt werden. Mit dieser Arbeit wurde der Versuch unternommen, ein umfassendes Modell zur Entwicklung von Systemen, basierend auf aktivem Hypertext, zu formulieren. Die Anwendung eines solchen Modells hat einige interessante Implikationen bezüglich der resultierenden Systemarchitektur. So wird eine eher dezentrale Struktur einer Client-/Server-Architektur bevorzugt, gerade auch durch die Vereinigung der Aufgaben von Client- und Server-Funktionalität in einer Instanz.

Einer der wichtigsten Schwerpunkte dieser Arbeit ist die Herausstellung der Strukturierung und Vernetzung der Informationen in einem System, ein Aspekt, der insbesondere bei offenen Systemen und sich ändernden Anforderungen erhöhte Bedeutung hat. Demzufolge ist das hier vorgestellte AHDM kein Modell, um wohldefinierte, geschlossene Einzelanwendungen zu realisieren, sondern eher eine Grundlage für flexible Systeme, die je nach Anwendungskontext unterschiedlich genutzt werden können. Es erweitert dabei aber den einfachen Dokumentenbegriff, ausgehend von maschinenverarbeitbaren zu ausführbaren Dokumenten mit der Option auf maschinenverstehbare Dokumente.

Um die Verwertbarkeit der Ergebnisse zu erhöhen, wurde auf eine gewisse Modularisierung sowohl des Kernmodells als auch der damit verbundenen Methode und der Werkzeuge geachtet. So kann etwa das Informationsmodell, welches u.a. die Trennung eines heterogenen Informationsraumes über Namensräume beschreibt, auch in anderen Kontexten eingesetzt werden. So ist zu hoffen, daß trotz der vorhandenen Einschränkungen des Modells zumindest die grundlegenden Erkenntnisse weiter verwendet werden können, etwa wenn in anderen Modellen oder Systemen ein ähnlicher, dokumentenzentrierter und auf offenen Datenaustausch ausgerichteter Ansatz gewählt wird.

Anhang

I Document Type Definitions

Definition der AHD-Elemente

```

<!ELEMENT var (#PCDATA)>
<!ATTLIST var
    name          CDATA          #REQUIRED>

<!ELEMENT func (#PCDATA)>
<!ATTLIST func
    name          CDATA          #REQUIRED
    type          CDATA          #REQUIRED
    returns       CDATA          "text/plain">

<!ELEMENT delegate EMPTY>
<!ATTLIST delegate
    href          CDATA          #REQUIRED>

<!ENTITY % role.att
    role          CDATA          #IMPLIED>

```

Definition wiederverwendbarer Präsentationselemente

```

<!ELEMENT heading ANY>
<!ELEMENT field ANY>
<!ELEMENT label ANY>
<!ELEMENT separator ANY>

```

Definition der Dublin-Core-Metadaten

```

<!ELEMENT title (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT contributor (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT relation (#PCDATA)>
<!ELEMENT coverage (#PCDATA)>
<!ELEMENT rights (#PCDATA)>

```

II IDL-Beschreibungen

```

interface AHDRuntime {

    void          put (      in Element      current,
                             in DOMString    varName,
                             in DOMString    varValue);

    DOMString     get (      in Element      current,
                             in DOMString    varName);

```



```

DOMString call (      in Element      current,
                      in DOMString    funcName,
                      in DOMString    paramString);

Document encode (    in DOMString    text);

Document here ( );

void lock (          in Document      document);

void unlock (        in Document      document);

Document fromText (  in DOMString    text);

DOMString toText (   in Document      document);

boolean match (      in Node          source,
                   in Node          pattern);

};

```

III Stylesheets

```

.document {
    display: block; float: none;
    font-size: 16pt; margin: 2em;
}
.heading1 {
    display: block; float: none;
    font-family: sans-serif; font-size: 172%;
    font-weight: bold; margin-top: 0.5ex;
    margin-bottom: 1ex; border-bottom-color: #49718e;
    border-bottom-width: 0.2ex
}
.heading2 {
    display: block; float: none; font-family: sans-serif;
    font-size: 144%; font-weight: bold;
    padding-top: 0.5ex; padding-bottom: 1ex;
    border-bottom-color: #49718e; border-bottom-width: 0.2ex
}
.heading3 {
    display: block; float: none; font-family: sans-serif;
    font-weight: bold; font-size: 120%; margin-top: 0.5ex;
    margin-bottom: 1ex; border-bottom-color: #49718e;
    border-bottom-width: 0.2ex
}
.heading4 {
    display: block; float: none; font-family: sans-serif;
    font-weight: bold; margin-top: 0.5ex; margin-bottom: 1ex;
    border-bottom-color: #49718e; border-bottom-width: 0.2ex
}
.para {
    display: block; float: none; font-family: serif;
    margin-top: 0.5ex; margin-bottom: 0.5ex;
}
.inline {
    display: inline;
}

```

```

}
.list {
    display: block; float: none; font-family: serif;
    margin-top: 0.5ex; margin-bottom: 0.5ex;
    margin-left: 2em;
}
.quote {
    display: block; float: none; font-family: serif;
    margin-top: 1ex; margin-bottom: 1ex;
    margin-left: 2em; margin-right: 2em;
    font-style: oblique;
}
.definition {
    display: block; float: none; font-family: serif;
    margin: 1.5em; border-width: 1pt;
    border-color: #000000; padding: 0.8em;
    background-color: #c0c0c0;
}
.label {
    display: inline; font-weight: bold;
    font-family: sans-serif; font-size: 90%;
    font-variant: roman;
}
.link {
    display: inline; color: #0000ff;
    text-decoration: underline;
}
func {display: none}
var {display: none}
hidden {display: none}
button {display: inline; vertical-align: middle}
input {display: inline; vertical-align: middle}
textarea {display: inline; vertical-align: top}

```

IV Wiederverwendbare Komponenten

```

<!DOCTYPE monitor [
<!ELEMENT monitor (documentlist)>
<!ELEMENT documentlist (document*)>
<!ELEMENT document (#PCDATA)>
<!ATTLIST document
    url          CDATA #REQUIRED
    callback     CDATA #REQUIRED>
]>
<monitor>
  <ahd:func name="register" type="text/tcl"><![CDATA[
    set list [DocumentGetElementsByTagName $ahd_document \
      documentlist]
    set list_elem [NodeListGetItem $list 0]
    NodeListDelete $list

    set doc_elem [ElementNew]
    ElementSetTagName $doc_elem "document"
    NodeAppendChild $list_elem $doc_elem
    ElementSetAttribute $doc_elem "url" $document
    ElementSetAttribute $doc_elem "callback" $callback
  ]]></ahd:func>

```

```
<ahd:func name="enter" type="text/tcl"><![CDATA[
    set list [DocumentGetElementsByTagName $ahd_document \
        document]
    for {set i 0} {$i < [NodeListGetLength $list]} {incr i} {
        set url [ElementGetAttribute [
            NodeListGetItem $list $i] "url"]
        set func [ElementGetAttribute [
            NodeListGetItem $list $i] "callback"]
        AHDRuntimeCall $ahd_ahd $ahd_element \
            $url\#$func document=$document
    }
    NodeListDelete $list
]]></ahd:func>

<documentlist>
</documentlist>
</monitor>
```


Literaturverzeichnis

- [1] Alexander, C.: „The Timeless Way of Building“, Oxford University Press, 1979.
- [2] Allaire Corp.: „ColdFusion Documentation“, <http://www.allaire.com/Documents/cf4docs.cfm>, Oktober 1999.
- [3] Angell, K. W.: „Examining JPython“. In: Doctor Dobb’s Journal, April 1999.
- [4] Apache Software Foundation: „Apache 1.3 User’s Guide“, <http://www.apache.org/docs/>, Dezember 1999.
- [5] Apache Software Foundation: „Xalan Java“, <http://xml.apache.org/xalan/>, Juni 2000.
- [6] Apple Computer Inc.: „The WebObjects Developer’s Guide“, <http://developer.apple.com/techpubs/webobjects/System/Documentation/Developer/WebObjects/DevGuide/WebObjectsDevGuide.pdf>, Oktober 1998.
- [7] Apparao, V., Eich, B., Guha, R., Ranjan, N.: „Action Sheets: A Modular Way of Defining Behaviour for XML and HTML“, W3C Note, <http://www.w3.org/TR/NOTE-AS>, Juni 1998.
- [8] Apparao, V., Glazman, D., Wilson, C.: „Behavioral Extensions to CSS“, W3C Working Draft, <http://www.w3.org/TR/becss>, August 1999.
- [9] Bakken, S., et al.: „PHP Manual“, <http://www.php.net/manual/>, August 1999.
- [10] Ball, S.: „Embeddable Components For Stand-Alone Web Applications“. In: Proceedings of the Third Australian World Wide Web Conference, Juli 1997.
- [11] Ball, S.: „Surfit! – A WWW Browser“, <http://surfit.anu.edu.au/>, August 1995.
- [12] Bartel, M., Boyer, J., Fox, B.: „XML-Signature Core Syntax and Processing“, W3C Working Draft, <http://www.w3.org/TR/xmlsig-core>, Januar 2000.
- [13] Beazley, D. M.: „SWIG : An Easy to Use Tool for Integrating Scripting Languages with C and C++“. In: Proceedings of the Fourth Annual Tcl/Tk Workshop, Juli 1996.
- [14] Beazley, D. M.: „Python Essential Reference“, New Riders, 1999.
- [15] Bentley, R., Horstmann, T., Sikkil, K., Trevor, J.: „Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System“. In: Proceedings of the Fourth International World Wide Web Conference, April 1995.
- [16] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., Secret, A.: „The World Wide Web“, In: Communications of the ACM Vol. 37 No. 8, August 1994.
- [17] Berners-Lee, T., Fielding, R., Frystyk, H.: „Hypertext Transfer Protocol – HTTP/1.0“, RFC 1945, Mai 1996.

- [18] Berners-Lee, T., Fielding, R., Masinter, L.: „Uniform Resource Identifiers (URI): Generic Syntax“, RFC 2396, August 1998.
- [19] Berners-Lee, T., Conolly, D.: „Web Architecture: Extensible Languages“, W3C Note, <http://www.w3.org/TR/NOTE-webarch-extlang>, Februar 1998.
- [20] Bieber, M., Vitali, F., Ashman, H., Balasubramanian, V., Oinas-Kukkonen, H.: „Fourth Generation Hypermedia: some missing links for the World Wide Web“. In: International Journal on Human Computer Studies, Vol. 47 No. 1, Juli 1997.
- [21] Bier, E. A.: „Documents as User Interfaces“, In: Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, September 1990.
- [22] Biron, P. V., Malhotra, A.: „XML Schema Part 2: Datatypes“, W3C Working Draft, <http://www.w3.org/TR/xmlschema-2>, September 1999.
- [23] Bonhomme, S., Roisin, C.: „Interactively Restructuring HTML Documents“, WWW5.
- [24] Borenstein, N.: „Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work“. In: Proceedings of the Conference on Computer-supported Cooperative Work, November 1992.
- [25] Borenstein, N., Freed, N.: „MIME (Multimedia Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies“, RFC 1521, September 1993.
- [26] Borenstein, N.: „EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail“. In: Proceedings of the IFIP International Conference on Upper Layer Protocols, Architectures and Applications, Mai 1994.
- [27] Bos, B., Lie, H., Lilley, C., Jacobs, I.: „Cascading Style Sheets, Level 2“, W3C Recommendation, <http://www.w3.org/TR/REC-CSS2>, März 1998.
- [28] Bosak, J.: „Namespaces in XML and XHTML“, Beitrag auf der xml-dev-Mailingliste, <http://www.lists.ic.ac.uk/hypermail/xml-dev/xml-dev-Sep-1999/0385.html>.
- [29] Boyer, J., Bray, T., Gordon, M.: „Extensible Forms Description Language (XFDL) 4.0“, W3C Note, <http://www.w3.org/TR/NODE-XFDL>, September 1998.
- [30] Bray, T.: „The Annotated XML Specification“, <http://www.xml.com/axml/axml.html>, 1998.
- [31] Bray, T., Paoli, J., Sperberg-McQueen, C. M.: „Extensible Markup Language (XML) 1.0“, W3C Recommendation, <http://w3c.org/TR/REC-xml>, Februar 1998.
- [32] Bray, T., Hollander, D., Layman, A.: „Namespaces in XML“, W3C Recommendation, <http://www.w3.org/TR/REC-xml-names>, Januar 1999.

- [33] Bray, T., Clark, J., Tauber, J.: „Canonical XML“, Version 1.0, W3C Working Draft, <http://www.w3.org/TR/xml-c14n>, November 1999.
- [34] Brickley, D., Guha, R. V.: „Resource Description Framework (RDF) Schema Specification“, W3C Proposed Recommendation, <http://www.w3.org/TR/PR-rdf-schema>, März 1999.
- [35] Brown, A. W., Wallnau, K. C.: „The Current State of CBSE“, IEEE Software Vol. 15 No. 5, September 1998.
- [36] Brown, M. H., Najork, M. A.: „Distributed Active Objects“. In: Proceedings of the Fifth International World Wide Web Conference, Mai 1996.
- [37] Buchanan, M. C., Zellweger, P. T., Pier, K.: „Multimedia Documents as User Interfaces“. In: Proceedings of INTERCHI-93: Human Factors in Computing, April 1993.
- [38] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: „A System of Patterns“, Wiley & Sons, 1996.
- [39] Ciancarini, P., Rizzi, A., Vitali, F.: „An extensible rendering engine for XML and HTML“. In: Proceedings of the Seventh International World Wide Web Conference, April 1998.
- [40] Clark, J. et al.: „A Proposal for XSL“, W3C Note, <http://www.w3.org/TR/NOTE-XSL.html>, August 1997.
- [41] Clark, J.: „Comparison of SGML and XML“, W3C Note, <http://www.w3.org/TR/NOTE-sgml-xml>, Dezember 1997.
- [42] Clark, J.: „Associating Style-Sheets with XML documents“, W3C Recommendation, <http://www.w3.org/TR/xml-stylesheets>, Juni 1999.
- [43] Clark, J., DeRose, S.: „XML Path Language (XPath) Version 1.0“, W3C Proposed Recommendation, <http://www.w3.org/TR/xpath>, Oktober 1999.
- [44] Clark, J.: „XSL Transformations (XSLT)“, W3C Recommendation, <http://www.w3.org/TR/xslt>, November 1999.
- [45] Coar, K. A. L., Robinson, D. R. T.: „The WWW Common Gateway Interface Version 1.1“, Internet Draft, <http://www.ietf.org/internet-drafts/draft-coar-cgi-v11-03.txt>, Juni 1999.
- [46] Cowan, J., Megginson, D.: „XML Information Set“, W3C Working Draft, <http://www.w3.org/TR/xml-infoset>, Mai 1999.
- [47] Deach, S.: „Extensible Stylesheet Language (XSL) Specification“, W3C Working Draft, <http://www.w3.org/TR/WD-xsl>, April 1999.

- [48] DeRose, S.: „XML XLink Requirements Version 1.0“, W3C Note, <http://www.w3.org/TR/NOTE-xlink-req>, Februar 1999.
- [49] DeRose, S., Orchard, D., Trafford, B.: „XML Linking Language“, W3C Working Draft, <http://www.w3.org/TR/xlink>, Juli 1999.
- [50] DeRose, S., Daniel, R.: „XML Pointer Language“, W3C Working Draft, <http://www.w3.org/TR/WD-xptr>, Juli 1999.
- [51] Digital Creations, Inc.: „Zope Content Manager’s Guide“, Version 2.1, <http://www.zope.org/Documentation/Guides/ZCMG/>, Oktober 1999.
- [52] Drake, F. L.: „The XML Bookmark Exchange Language“, <http://www.python.org/topics/xml/xbel/docs/html/xbel.html>, Oktober 1998.
- [53] Dridi, F., Neumann, G.: „Towards Access Control for Logical Document Structures“. In: Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, August 1998.
- [54] Dridi, F., Neumann, G.: „Managing Security in the World Wide Web: Architecture, Services, and Techniques“. In: Janczewski, L. (Hrsg.): „Internet and Intranet Security Management: Risks and Solutions“, Idea Group Publishing, 2000.
- [55] van Doorn, M., Eliëns, A.: „Integrating applications and the World Wide Web“. In: Proceedings of the Third International World Wide Web Conference, April 1995.
- [56] Dubinko, M., Silvester, S., Schnitzenbaumer, S., Raggett, D.: „XForms 1.0: Data Model“, W3C Working Draft, <http://www.w3.org/TR/xforms-datamodel/>, April 2000.
- [57] Dybvik, R. K.: „The Scheme Programming Language“, Prentice Hall, 1996.
- [58] English, P., M., Tenneti, R.: „Interleaf Active Documents“. In: Brailsford, D. F., Furuta, R. K. (Hrsg.): Electronic Publishing – Origination, Dissemination and Design Vol. 7 No. 2, Juli 1994.
- [59] European Computer Manufacturers Association: „ECMAScript Language Specification“, <ftp://ftp.ecma.ch/ecma-st/e-262-pdf.pdf>, August 1998.
- [60] Fielding, B. et al.: „Hypertext Transfer Protocol – HTTP/1.1“, RFC 2616, Juni 1999.
- [61] Flanagan, D. (Hrsg.): „Volume 5: X Toolkit Intrinsics Reference Manual“, 3. Auflage, O’Reilly & Associates, 1992.
- [62] Franklin, S., Graesser, A.: „Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents“. In: Proceedings of the Third Workshop on Agent Theories, Architectures and Languages“, August 1996.

- [63] Free Software Foundation: „Guile: Duct Tape For Bits“, <http://www.gnu.org/software/guile/guile.html>, September 1999.
- [64] Freier, A. O., Karlton, P., Kocher, P. C.: „The SSL Protocol Version 3.0“, Internet Draft, <http://www.ietf.org/internet-drafts/draft-freier-ssl-version3-02.txt>, November 1996.
- [65] Furuta, R., Stotts, P. D.: „Trellis: a Formally-defined Hypertextual Basis for Integrating Task and Information“, Department of Computer Science Technical Report No. TAMU-HRL-94-007, Texas A & M University, 1994.
- [66] Ghezzi, C., Jazayeri, M., Mandrioli, D.: „Fundamentals of Software Engineering“, Prentice Hall, 1992.
- [67] Girgensohn, A., Lee, A., Schlueter, K.: „Experiences in Developing Collaborative Applications Using the World-Wide-Web ‘Shell’“. In: Proceedings of the Seventh ACM Conference on Hypertext, März 1996.
- [68] Godland, Y. et al.: „HTTP Extensions for Distributed Authoring – WebDAV“, RFC 2518, Februar 1999.
- [69] Goedicke, M., Neumann, G., Zdun, U. : „Object System Layer“. In: Proceedings of EuroPLoP 2000 – Fifth European Conference on Pattern Languages of Programs, Juli 2000.
- [70] Goldberg, Y., Safran, M., Shapiro, E.: „Active Mail – A Framework for Implementing Groupware“, In: Proceedings of the Conference on Computer-supported Cooperative Work, November 1992.
- [71] Goldfarb, C. F.: „The SGML Handbook“, Oxford University Press, Oxford 1990.
- [72] Gosling, J., Joy, B., Steele, G.: „The Java Language Specification“, Addison Wesley Longman, 1996.
- [73] Greening, D. R.: „Self-Service Syndication with ICE“. In: Web Techniques, <http://www.webtechniques.com/archives/1999/11/greening/>, November 1999.
- [74] Guetari, R., Quint, V., Vatton, I.: „Amaya: an Authoring Tool for the Web“. In: Proceedings of the Maghrebien Conference on Software Engineering and Artificial Intelligence, Dezember 1998.
- [75] Güting, R. H., Zicari, R., Choy, D. M.: „An Algebra for Structured Office Documents“. In: ACM Transactions on Office Information Systems Vol. 7 No. 4, April 1989.
- [76] Hansen, W. J.: „Enhancing Documents with embedded programs: How Ness extends insets in the Andrew Toolkit“. In: Proceedings of the International IEEE Conference on Computer Languages, März 1990.

- [77] Hart, J. M., Rosenberg, B.: „Client/Server Computing for Technical Professionals“, Addison Wesley, 1995.
- [78] Heinrich, L. J., Pomberger, G., Schrefl, M., Stary, C.: „Profil der Wirtschaftsinformatik“, Beschluß der WKWi im Verband der Hochschullehrer für BWL e.V., 1993.
- [79] Hughes Technologies, Pty. Ltd.: „Lite & W3-mSQL Reference“, http://www.hughes.com.au/library/lite/manual_20/, August 1999.
- [80] IBM Corp.: „XML Parser for Java“, <http://www.alphaworks.ibm.com/tech/xml4j/>, Dezember 1999.
- [81] IBM Corp.: „Bean Scripting Framework“, <http://www.alphaworks.ibm.com/tech/bsf/>, Dezember 1999.
- [82] ICESoft A.S.: „ICEBrowser“, <http://www.icesoft.no/ICEBrowser/>, Dezember 1999.
- [83] Ierusalimschy, R., de Figueiredo, L. H., Fihlo, W. C.: „Lua – an extensible extension language“. In: Software: Practice & Experience Vol. 26 No. 6, Juni 1996.
- [84] International Organization for Standardization: „ISO 8879: Information processing – Text and office systems – Standard Generalized Markup Language (SGML)“, Genf, 1986.
- [85] International Organization for Standardization: „ISO/IEC 10740: Information processing – Hypermedia/Time-based Structuring Language (HyTime)“, Genf, 1992.
- [86] International Organization for Standardization: „ISO/IEC 10646-1993 (E): Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane“, Genf, 1993.
- [87] International Organization for Standardization: „ISO/IEC 10179-1996 (E): Information technology – Processing Languages – Document Style and Semantics Specification Language (DSSSL)“, Genf, 1996.
- [88] Isakowitz, T., Stohr, E. A., Balasubramanian, P.: „A Methodology for Structured Hypermedia Design“. In: Communications of the ACM, Vol. 38 No. 8, August 1995.
- [89] Jelliffe, R.: „How to Promote Organic Plurality on the WWW“, <http://www.ascc.net/xml/en/utf-8/monolith.html>, Juni 1999.
- [90] Jensen, K. K., Riksted, G. E.: „Linda – A Distributed Programming Paradigm“, Diplomarbeit, Universität Aalborg, 1989.
- [91] Kaiser, G. E., Dossick, S. E., Jiang, W., Yang, J. J.: „An Architecture for WWW-based Hypercode Environments“. In: Proceedings of the 19th International Conference on Software Engineering, Mai 1997.

- [92] Kalakota, R., Whinston, A. B.: „Frontiers of Electronic Commerce“, Addison-Wesley, 1996.
- [93] Khare, R., Rifkin, A.: „Capturing the State of Distributed Systems with XML“. In: Conolly, D. (Hrsg.): World Wide Web Journal Special Issue on XML Vol. 2 No. 4, 1997.
- [94] Kieser, A., Kubicek, H.: „Organisationstheorien“, Band 2, Kohlhammer, 1978.
- [95] Kistler, T., Marais, H.: „WebL – a programming language for the Web“. In: Proceedings of the Seventh International World Wide Web Conference, April 1998.
- [96] Knuth, D. E.: „Literate Programming“, CSLI Publications, 1992.
- [97] Köppen, E.: „Entwicklung eines erweiterbaren Widgets zur Anzeige von HTML-Dokumenten“, Diplomarbeit, Universität Gesamthochschule Essen, 1996.
- [98] Köppen, E., Neumann, G., Nusser, S.: „Cineast – An extensible Web Browser“. In: Proceedings of WebNet 97 World Conference of the WWW, Internet and Intranet, November 1997.
- [99] Köppen, E.: „A Practical Approach towards Active Hyperlinked Documents“. In: Proceedings of the Seventh International World Wide Web Conference, April 1998.
- [100] Kotok, A.: „Introduction to XML and EDI“. In: XML.com, http://xml.com/pub/1999/08/edi/index.html?wwwrrr_19990804.txt, April 1999.
- [101] Krishnamurthy, B., Mogul, J. C., Kristol, D. M.: „Key Differences between HTTP/1.0 and HTTP/1.1“. In: Proceedings of the Eighth International World Wide Web Conference, Mai 1999.
- [102] Kristol, D. M., Montulli, L.: „HTTP State Management Mechanism“, RFC 2109, Februar 1997.
- [103] Ladd, D., Ramming, J. C.: „Programming the Web: An Application-Oriented Language for Hypermedia Service Programming“. In: Proceedings of the Fourth International World Wide Web Conference, April 1995.
- [104] Lassila, O., Swick, R.: „Resource Description Framework (RDF) Model and Syntax Specification“, W3C Recommendation, <http://www.w3.org/TR/REC-rdf-syntax/>, Februar 1999.
- [105] Lehner, F., Hildebrand, K., Maier, R.: „Wirtschaftsinformatik. Theoretische Grundlagen“, Hanser Verlag, 1995.
- [106] Levy, J.: „A Tk Netscape Plugin“. In: Proceedings of the Fourth Annual USENIX Tcl/Tk Workshop, Juli 1996.

- [107] Lindholm, T., Yellin, F.: „The Java Virtual Machine Specification“, 2. Auflage, Addison-Wesley, 1999.
- [108] Lucco, S., Sharp, O., Wahbe, R.: „Omniware: A Universal Substrate for Web Programming“. In: Proceedings of the Fourth International World Wide Web Conference, April 1995.
- [109] Lugrin, J.-M.: „FESI EcmaScript Interpreter“, <http://home/worldcom.ch/jmlugrin/fesi/>, Januar 2000.
- [110] Maler, E., DeRose, S.: „XML Linking Language (XLink) Design Principles“, W3C Note, <http://www.w3.org/TR/NOTE-xlink-principles>, März 1998.
- [111] Mandry, T., Pernul, G., Röhm, A. W.: „Mobile Agenten auf elektronischen Märkten – Einsatzmöglichkeiten und Sicherheitsanalyse“. In: Röhm, A. W., et al. (Hrsg): „Sicherheit und Electronic Commerce“, Vieweg Verlag, 1999.
- [112] Masinter, L.: „Returning Values from Forms: multipart/form-data“, RFC 2388, August 1998.
- [113] Massy, D., Williams, S., Norlander, R., Kurata, L., Reardon, T.: „HTML Components“, W3C Note, <http://www.w3.org/TR/NOTE-HTMLComponents>, Oktober 1998.
- [114] Megginson, D. et al.: „SAX 1.0: The Simple API for XML“, <http://www.megginson.com/SAX/>, Mai 1998.
- [115] Merle, P., Gransart, C., Geib, J.-M.: „CorbaWeb: A Generic Object Navigator“. In: Proceedings of the Fifth International World Wide Web Conference, Mai 1996.
- [116] Microsoft Corp.: „The Component Object Model Specification“, <http://www.microsoft.com/COM/resources/comdocs.asp>, Oktober 1995.
- [117] Moats, P.: „URN Syntax“, RFC 2141, Mai 1997.
- [118] Mort Bay Consulting: „Jetty Java HTTP Server“, <http://www.mortbay.com/software/Jetty.html>, Dezember 1999.
- [119] Mozilla Organization: „mozilla.org at a glance“, <http://www.mozilla.org/mozorg.html>, Juli 2000.
- [120] National Computer Security Center: „A Guide to Understanding Discretionary Access Control In Trusted Systems“, NCSC-TG-003-87, September 1987.
- [121] Myers, J., Rose, M.: „Post Office Protocol - Version 3“, RFC 1939, Mai 1996.
- [122] National Institute of Standards and Technology: „The Secure Hash Algorithm (SHA-1)“, FIPS PUB 180-1, <http://www.itl.nist.gov/fipspubs/fip180-1.html>, April 1995.

- [123] Nelson, C.: „OpenDoc and Its Architecture“, <http://www-4.ibm.com/software/ad/opendoc/library/aixpert95.aug95.opendoc.html>. In: AIXpert, August 1995.
- [124] Nelson, T. H.: „Literary Machines“, The Distributors, 1987.
- [125] Netscape Communications Corp.: „NSAPI Programmer’s Guide“, <http://developer.netscape.com/docs/manuals/enterprise/nsapi/index.htm>, Dezember 1997.
- [126] Netscape Communications Corp.: „Plug-in Guide“, <http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm>, Januar 1998.
- [127] Netscape Communications Corp.: „Client-Side JavaScript Reference“, Version 1.3, <http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm>, Dezember 1999.
- [128] Netscape Communications Corp.: „Server-Side JavaScript Reference“, Version 1.4, <http://developer.netscape.com/docs/manuals/js/server/jsref/index.htm>, Dezember 1999.
- [129] Neumann, G., Nusser, S.: „Wafe – An X Toolkit based Frontend for Application Programs in Various Programming Languages“. In: Proceedings of the USENIX Winter Conference, Januar 1993.
- [130] Neumann, G., Zdun, U.: „XOTcl – an Object-Oriented Scripting Language“. In: Proceedings of the Seventh Usenix Tcl/Tk Conference, Februar 2000.
- [131] Newcomb, S. R., Kipp, N. A., Newcomb, V. T.: „The HyTime Hypermedia/Time-based Document Structuring Language“, In: Communications of the ACM Vol. 34 No. 11, November 1991.
- [132] Noltemeyer, H.: „Graphentheorie“, Walter de Gruyter, 1975.
- [133] Object Management Group: „The Common Object Request Broker: Architecture and Specification“. <http://www.omg.org/corba/corbbiop.htm>.
- [134] Orfali, R., Harkey, D., Edwards, J.: „The Essential Client/Server Survival Guide“, 2. Auflage, John Wiley & Sons, 1996.
- [135] Ousterhout, J. K.: „Tcl: An embeddable Command Language“. In: Proceedings of the SeventhUSENIX Winter Conference, Januar 1990.
- [136] Ousterhout, J. K.: „An X11 Toolkit based on the Tcl Language“. In: Proceedings of the Eight USENIX Winter Conference, Januar 1991.
- [137] Ousterhout, J. K.: „Scripting: Higher level programming for the 21st century“, IEEE Computer, Vol. 31 No. 3, März 1998.
- [138] Palay, A. J., Hansen, W. J. et al., „The Andrew Toolkit – An Overview“. In: Proceedings of the USENIX Winter Conference, Januar 1988.

- [139] Perens, B.: „The Open Source Definition“, Version 1.7, <http://www.opensource.org/osd.html>, April 2000.
- [140] Pescio, C.: „Principles versus Patterns“, IEEE Computer Vol. 30 No. 9, September 1997.
- [141] Phelps, T. A., Wilensky, R.: „Toward Active, Extensible, Networked Documents: Multivalent Architecture and Applications“. In: Proceedings of the First Conference on Digital Libraries, März 1996.
- [142] Picot, A., Reichwald, R., Wigand, R. T.: „Die grenzenlose Unternehmung“, 2. Auflage, Gabler, 1996.
- [143] Postel, J.: „Simple Mail Transfer Protocol“. RFC 821, August 1982.
- [144] Postel, J., Reynolds, J.: „File Transfer Protocol (FTP)“, RFC 959, Oktober 1985.
- [145] Pree, W. et al.: „Design Patterns for Object-Oriented Software Development“, Addison Wesley, 1995.
- [146] Procmail Foundation: „Procmail Homepage“, <http://www.procmail.org>, Oktober 1999.
- [147] Quint, V., Vatton, I.: „Making Structured Documents Active“. In: Brailsford, D. F., Furuta, R. K. (Hrsg.): Electronic Publishing – Origination, Dissemination and Design Vol. 7 No. 2, Juli 1994.
- [148] Raggett, D., Le Hors, A., Jacobs, I.: „HTML 4.01 Specification“, W3C Recommendation, <http://www.w3.org/TR/html4/>, Dezember 1999
- [149] Raymond, E. S.: „The Halloween Documents“, <http://www.opensource.org/halloween/>, November 1999.
- [150] Rees, O. et al.: „A Web of Distributed Objects“. In: Proceedings of the Fourth International World Wide Web Conference, April 1995.
- [151] Reisig, W.: „Petrinetze – Eine Einführung“, Springer 1986.
- [152] Rivest, R. L., Shamir, A., Adelman, L. M.: „A method for obtaining digital signatures and public-key cryptosystem“. In: Communications of the ACM. Vol. 21 No. 2, Februar 1978.
- [153] Rivest, R. L.: „The MD5 Message-Digest Algorithm“, RFC 1321, April 1992.
- [154] Roisin, C. et al.: „Thot structured editor“, <http://www.inrialpes.fr/opera/Thot/>, November 1997.

- [155] Rossi, G., Schwabe, D.: „Building Hypermedia Applications as Navigational Views of Information Model“. In: Proceedings of the Annual Hawaii International Conference on System Sciences, Januar 1995.
- [156] Rossi, G., Schwabe, D., Garrido, A.: „Design Reuse in Hypermedia Applications Development“. In: Proceedings of the Eighth International Hypertext Conference, April 1997.
- [157] Object Management Group Inc.: „OMG Unified Modelling Language Specification“, <http://www.omg.org/uml/>, Juni 1999.
- [158] Sánchez, J. A., Legget, J. J.: „HyperActive: Extending an Open Hypermedia Architecture to Support Agency“, In: ACM Transactions on Human-Computer Interaction Vol. 1 (1994) No. 4, April 1994.
- [159] Schneier, B.: „Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)“, In: Proceedings of the First Fast Software Encryption Workshop, April 1994.
- [160] Scriptics Corp.: „Tclet Demos“, <http://www.scriptics.com/products/tcltk/plugin/applets.html>, Dezember 1999.
- [161] Scriptics Corp.: „Tel Java Integration“, <http://www.scriptics.com/products/java/>, Dezember 1999.
- [162] Seffah, A., Khuwaja, R. A.: „An Architectural Framework for Developing Web-Based Interactive Applications“. In: Proceedings of WebNet 97 World Conference of the WWW, Internet and Intranet, November 1997.
- [163] Seiffert, H.: „Einführung in die Wissenschaftstheorie 1“, C.H. Beck, 1974.
- [164] Snelting, G.: „Paul Feyerabend und die Softwaretechnologie“. In: Informatik-Spektrum 21 (1998) 5, Springer-Verlag 1998.
- [165] Sperberg-McQueen, C. M., Burnard, L. (Hrsg.): „Guidelines for Electronic Text Encoding and Interchange“, Text Encoding Initiative, <http://www-tei.uic.edu/orgs/tei/p3/elect.html>, 1994.
- [166] Steele, G. L.: „Common Lisp the Language“, 2. Auflage, Digital Press, 1990.
- [167] Stevahan, R.: „Adding Style and Behavior to XML with a dash of Spice“, W3C Note, <http://www.w3.org/TR/NOTE-spice>, Februar 1998.
- [168] Sun Microsystems Corp.: „Java Beans“, Version 1.01, <http://java.sun.com/beans/docs/beans101.pdf>, Juli 1997.
- [169] Sun Microsystems Corp.: „Enterprise Java Beans Specification“, Version 1.1, <http://java.sun.com/products/ejb/docs.html>, Dezember 1999.

- [170] Sun Microsystems Corp.: „Applets“, <http://java.sun.com/applets/>, Dezember 1999.
- [171] Sun Microsystems Corp.: „Java Servlet Specification“, Version 2.2, <http://java.sun.com/products/servlet/2.2/>, Dezember 1999.
- [172] Sun Microsystems Corp.: „Java Server Pages Specification“, Version 1.1, <http://java.sun.com/products/jsp/>, November 1999.
- [173] Sun Microsystems Corp.: „Java 2 Platform Enterprise Edition Specification“, Version 1.2, <http://java.sun.com/j2ee/>, Dezember 1999.
- [174] Sweet, L. L.: „The new e-conomy“. In: InfoWorld.com, <http://www.infoworld.com/cgi-bin/displayStory.pl?features/990726ecomm.htm>, Juli 1999,
- [175] Tennenhouse, D. L., Wetherall, D. J.: „Towards an Active Network Architecture“. In: Computer Communication Review, Vol. 26 No. 2, April 1996.
- [176] Terry, D. B., Baker, D. G.: „Active Tioga Documents: an exploration of two paradigms“. In: Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, September 1990.
- [177] Thistlewaite, P., Ball, S.: „Active FORMs“. In: Proceedings of the Fifth International World Wide Web Conference, Mai 1996.
- [178] Thompson, H. S., Beech, D., Malony, M., Mendelson, N.: „XML Schema Part 1: Structures“, W3C Working Draft, <http://www.w3.org/TR/xmlschema-1/>, September 1999.
- [179] Tigue, J., Lavinder, J.: „WebBroker: Distributed Object Communication on the Web“, W3C Note, <http://www.w3.org/TR/1998/NOTE-webbroker>, Mai 1998.
- [180] Userland Software Inc.: „XML-RPC Specification“, <http://www.xml-rpc.com/spec>, Oktober 1999.
- [181] Vitali, F., Chiu, C.-H., Bieber, M.: „Extending HTML in a Disciplined Way With Displets“. In: Proceedings of the Sixth World Wide Web Conference, April 1997.
- [182] Wahl, M., Howes, T., Kille, S.: „Lightweight Directory Access Protocol (v3)“, RFC2251, Dezember 1997.
- [183] Waldén, K., Nerson, J.-M.: „Seamless Object-Oriented Software Architecture“, Prentice Hall, 1995.
- [184] Wall, L., Christiansen, T., Schwartz, R. L.: „Programming Perl“, 2. Auflage, O'Reilly, 1996.
- [185] Weck, W., Szyperski, C.: „Do We Need Inheritance?“, Proc. of the Workshop on Composability Issues in Object-Oriented Programming, ECOOP'96, Juni 1996.

- [186] Weibel, S., Kunze, J., Lagoze, C., Wolf, M.: „Dublin Core Metadata for Resource Discovery“, RFC 2413, September 1998.
- [187] White, J.: „Mobile Agents White Paper“, General Magic Inc., 1996.
- [188] Wiil, U. K., Leggett, J. J.: „Hyperform: A Hypermedia System Development Environment“. In: ACM Transactions on Information Systems Vol. 15 No. 1, Januar 1997.
- [189] Winer, D. et al.: „Simple Object Access Protocol (SOAP) 1.1“, W3C Note, <http://www.w3.org/TR/SOAP>, Mai 2000.
- [190] Wodaski, R.: „ASP Technology Feature Overview“, <http://msdn.microsoft.com/workshop/server/asp/aspfeat.asp>, August 1998.
- [191] Wood, L., et al.: „Document Object Model (DOM) Level 1 Specification“, W3C Recommendation, <http://www.w3.org/TR/REC-DOM-Level-1>, Oktober 1998.
- [192] Wood, L., et al.: „Document Object Model (DOM) Level 2 Specification“, W3C Working Draft, <http://www.w3.org/TR/WD-DOM-Level-2>, September 1999.
- [193] Yeager, N. J., McGrath, R. E.: „Web Server Technology“, Morgan Kaufmann Publishers, 1996.
- [194] Yellin, F.: „Low Level Security in Java“. In: Proceedings of the Fourth International World Wide Web Conference, April 1995.
- [195] Zdun, U.: „Entwicklung und Implementierung von Ansätzen, wie Entwurfsmustern, Namensräumen und Zusicherungen, zur Entwicklung von komplexen Systemen in einer objektorientierten Skriptsprache“, Diplomarbeit, Universität Gesamthochschule Essen, 1998.
- [196] Zdun, U.: „Entwurf und Entwicklung eines mobilen Objekt-Systems für Anwendungen im Internet“. Diplomarbeit, Universität Gesamthochschule Essen, 1999.
- [197] Zellweger, T. P.: „Scripted Documents: A Hypermedia Path Mechanism“. In: Proceedings of the ACM Conference on Hypertext, November 1989.