Diplomarbeit

Automatisierte WWW-Veröffentlichung auf der Basis formaler Auszeichnungssprachen

Stefan Mintert

Universität Dortmund

Fachbereich Informatik
Lehrstuhl für Künstliche Intelligenz
Januar 1999

Betreuer:
Prof. Dr. Katharina Morik
Dipl. Inform. Stefan Haustein

Automatisierte WWW-Veröffentlichung auf der Basis formaler Auszeichnungssprachen

Zusammenfassung

Das Ergebnis dieser Diplomarbeit ist ein System, das SGML/XML-kodierte Dokumente für die Veröffentlichung im World Wide Web aufbereitet. Der Benutzer, d.h. der Leser, hat die Möglichkeit, nach Stichworten in einer von mehreren Kategorien zu suchen. Suchbegriff und Suchkategorie ersetzen die häufig zu ungenaue Volltextsuche. Im Gegensatz zu den Ausgaben von herkömmlichen Suchfunktionen erhält er als Ergebnis keinen vorgefertigten Ausschnitt aus dem Dokument (z.B. ein Kapitel, einen Abschnitt usw.). Vielmehr wird der Ausschnitt in Abhängigkeit der Textstelle bestimmt, die bei der Suche als Treffer ermittelt wurde.

Die Definition solcher Dokumentausschnitte und der Suchkategorien findet mit Bezug auf den Dokumenttyp, nicht bezüglich des konkreten Textes statt. Diese Herangehensweise besitzt den Vorteil, notwendige Konfigurationen nur *einmal* je Dokumenttyp machen zu müssen, die dann für sämtliche Dokumente dieses Typs anwendbar sind.

Danksagung

Die Zahl der Menschen, die mich während der Bearbeitung dieser Diplomarbeit vorangetrieben und für meine geistige Unversehrtheit gesorgt haben, ist zu groß, als daß ich sie alle nennen könnte. Obwohl ich jedem einzelnen meinen Dank schulde, beschränke ich mich bei der namentlichen Nennung auf all diejenigen, die unmittelbaren fachlichen Einfluß auf meine Arbeit hatten. Dies sind vor allem meine Betreuer, Prof. Katharina Morik und Stefan Haustein, die gestaltend auf den Programmentwurf und den Text eingewirkt haben. Für die Ordnung meiner Gedanken in ungezählten Stunden der Diskussion danke ich Henning Behme (die Telekom dankt auch), Eduard Paul und Rainer Stoll (ich sag' nie wieder SGML in Eurer Gegenwart) sowie Sven Schröter (Küchentisch in Hörde). Frei nach Brecht gilt für alle: Ihr habt Vorschläge gemacht, ich habe sie angenommen; nicht jeden, aber viele.
Daß diese Diplomarbeit frei von Rechtschreibfehlern ist, kann ich nicht garantieren. Daß dem Leser viele erspart geblieben sind, danke ich den prüfenden Blicken von Eduard Paul und Rainer Stoll sowie Susanne Collet, die darüber hinaus auch noch mein Wissen über Makroökonomie auf knapp über Null gebracht hat; mehr ist nicht möglich, so viel weiß ich jetzt.
Thanks always to Jackson.

Let the music keep our spirits high

Für meine Familie

Inhaltsverzeichnis

Einleitung

1

1.1 Aufgabenbeschreibung

Innerhalb dieser Diplomarbeit geht es darum, ein System zu entwikkeln, das es erlaubt, umfangreiche Texte möglichst einfach im World Wide Web zu veröffentlichen, und das dem Leser gestattet, auf diese Texte in für ihn angenehmer Weise zuzugreifen. Um diese abstrakte Aufgabe zu erfüllen, gibt es zahlreiche existierende Wege, die verschiedene Schwächen aufweisen. Diese Schwächen möchte ich kurz ansprechen; daraus leiten sich die Anforderungen an diese Arbeit ab.

Für die Veröffentlichung *umfangreicher* Texte im Web gibt es zwei Herangehensweisen: Man kann die Texte zusammenhängend, etwa in PostScript oder PDF, anbieten. In diesem Fall hat man ein für den Druck geeignetes Format und nutzt das Web lediglich als *Transportmedium*. Die andere Möglichkeit besteht in der Aufbereitung eines Textes in eine web-taugliche Form. Allein wegen der beschränkten Bandbreite heißt das in der gegenwärtigen Praxis, das Dokument in mehrere Teile zu teilen (z.B. einzelne Kapitel) und über Hyperlinks das Blättern zwischen den Teilen zu ermöglichen. Je nach dem, welche Eigenschaften das Ausgangsformat hat, ist diese Aufbereitung entweder aufwendig oder schlecht (z.B. ist die Speicherung von Word-Texten als HTML qualitativ nicht sehr gut). Der Grund für die geringe Qualität ist meistens die Tatsache, daß Texte speziell für den Ausdruck (in einem bestimmten Format, etwa A4) vorbereitet werden. Texte, die Informationen über ihre *Struktur* und weitere *Metainformationen* beinhalten, lassen sich besser *medienunabhängig* behandeln. Damit ist die erste Randbedingung für das zu entwikkelnde System gefunden: *Das System soll mit strukturierten Texten umgehen können.*

Um lange Texte zu lesen, bevorzugen die meisten Menschen die ausgedruckte Form. Ein Vorteil beim Online-Publishing besteht in der Möglichkeit, Texte durchsuchen zu können, um bestimmte Stellen schneller zu finden. Der übliche Ansatz der Volltextsuche besitzt den Nachteil, daß häufig *zu viele* und *unerwünschte* Textstellen gefunden werden. Ist man beispielsweise auf der Suche nach dem Pro-

grammierbefehl »print«, so sind Rezepte für Aachener Printen und Texte über Print-Medien sicherlich nicht gewünscht. Das Problem besteht darin, daß man zwar den Such*begriff* angeben kann, nicht jedoch die Such*kategorie* (hier: Begriff »print«, Kategorie »Befehle von Programmiersprachen«). Die Ursache dafür ist, daß Texte meistens eben keine Metainformationen (wie die Kategorie) enthalten. Da ich oben bereits vorausgesetzt habe, daß Texte solche Metainformationen enthalten müssen, folgt die zweite Anforderung an das zu entwickelnde System: *Die Suche soll keine simple Volltextsuche sein, sondern innerhalb einer vom Benutzer ausgewählten Kategorie stattfinden (»qualifizierte Suche«).*

Die Frage ist nun: Was soll als Ergebnis geliefert werden? Wenn der Suchvorgang nur als Antwort liefert, ob der Begriff in der richtigen Kategorie in einem bestimmten Text gefunden wurde oder nicht, ist das *Ergebnis* mit den bekannten Suchmaschinen vergleichbar: Statische Texte werden auf einem Webserver angeboten und ausgeliefert. Diese Vorgehensweise berücksichtigt aber in keiner Weise, *an welcher Stelle* sich der gefundene Begriff befindet. Es wäre doch besser, das Ergebnis davon abhängig zu machen. Natürlich sollte nicht nur das gefundene Wort zurückgeliefert werden, sondern eine bestimmte Umgebung des Wortes. Die Umgebung sollte so beschaffen sein, daß der Text darin verständlich bleibt. Wenn der Leser dann diesen *Ausschnitt* als Ergebnis bekommt, trifft das seine Wünsche besser. Das ist die dritte Anforderung: *Das Ergebnis einer Suche soll ein möglichst sinnvoller Ausschnitt aus dem Dokument sein.*

Was fängt man mit gefundenen Textausschnitten an, die das Ergebnis von *mehreren* Suchvorgängen sind? Irgendwie müssen die Ausschnitte zusammengefaßt werden: *Der Benutzer soll die Möglichkeit bekommen, sich ein »eigenes«, neues Dokument aus mehreren Ausschnitten zusammenzustellen.*

Die bisherigen Fragestellungen orientierten sich an der Benutzersicht. Auch für den Webadministrator beziehungsweise den Verfasser von Texten sollte die Aufgabe erleichtert werden. Was kann man tun, um nicht für jeden einzelnen Text immer wieder den gleichen Aufwand betreiben zu müssen? Welche Gemeinsamkeit von Texten kann man verwenden? Die Antwort lautet einmal mehr: die Strukturinformation. Durch eine gemeinsame Struktur wird ein *Dokumenttyp* definiert, der Eigenschaften von Dokumenten beschreibt. Unter Ausnutzung dieser Eigenschaften gelangt man zur letzten Anforderung: *Die Definition eines Dokumentausschnitts, der Ergebnis einer Suche ist, soll mit Bezug auf den Dokumenttyp (z.B. Lehrbuch, Handbuch, Bedienungsanleitung, Brief, Artikel usw.) geschehen, um viele gleichartige Dokumente mit möglichst geringem Aufwand publizieren zu können.*

Die Anforderungen lassen sich stichpunktartig wie folgt zusammenfassen. Zentrale Bedeutung haben dabei die beiden ersten Punkte.

1. Das System soll die Suche nach Begriffen in den Texten erlauben. Die Suche sollte keine simple Volltextsuche sein, sondern innerhalb einer vom Benutzer ausgewählten Kategorie stattfinden (»qualifizierte Suche«).
2. Das Ergebnis einer Suche soll ein möglichst sinnvoller Ausschnitt aus dem Dokument sein.
3. Der Benutzer soll die Möglichkeit bekommen, sich ein »eigenes«, neues Dokument aus mehreren Ausschnitten zusammenzustellen.
4. Die Definition eines Dokumentausschnitts, der Ergebnis einer Suche ist, soll mit Bezug auf den Dokumenttyp (z.B. Lehrbuch, Handbuch, Bedienungsanleitung, Brief, Artikel usw.) geschehen, um viele gleichartige Dokumente mit möglichst geringem Aufwand publizieren zu können.

Als weitere Randbedingungen und Wünsche sind folgende Punkte zu nennen:

1. Das System soll mit strukturierten Texten umgehen können. Als geeignetes Format wurde SGML/XML ausgewählt.
2. Die Texte sollen in der SGML/XML-Form verbleiben, um sie weiterhin in einer Textverarbeitung nutzen zu können.
3. Das System soll die Publikation der Texte über das World Wide Web ermöglichen.
4. Das System soll die Grundlage bilden, um durch Weiterentwicklung und unter Zuhilfenahme von Verfahren des maschinellen Lernens das benutzeradaptive Publizieren im World Wide Web möglich zu machen. Dieser Schritt wird in einer weiteren Diplomarbeit ausgeführt.

Als Anwendung für diese Diplomarbeit wurde Textmaterial im Umfang zweier Bücher zur Verfügung gestellt. Bei dem Material handelt es sich um volkswirtschaftliche Texte über makroökonomische Modelle. Ein Hindernis bestand darin, daß die Dateien im Format der Macintosh-Textverarbeitung *WriteNow* oder alternativ im Rich Text Format (RTF) vorlagen. Die angesprochene implizite Struktur fand sich leider in keiner Weise in den Auszeichnungen der Texte wieder, die sich ausschließlich auf Attribute beschränkten, die der Formatierung und visuellen Darstellung dienen. Teil meiner Diplomarbeit war es also auch (neben der Hauptaufgabe, ein System zu entwikkeln, das die obigen Anforderungen erfüllt), das vorliegende Material auf strukturelle Einheiten zu analysieren und Auszeichnungen zu entwerfen, die die Struktur darstellen. Selbstverständlich sollte das

zu entwickelnde System aber mit beliebigen Texten arbeiten, solange sie nur geeignet ausgezeichnet sind.

1.2 Lösungsansatz

Für das gegebene Textmaterial sieht der Lösungsansatz so aus, daß es einmalig in die SGML-Struktur überführt wird und diese dann als Basis für die weitere Editierung und für die Veröffentlichung im Web dient (vgl. Abbildung 1).

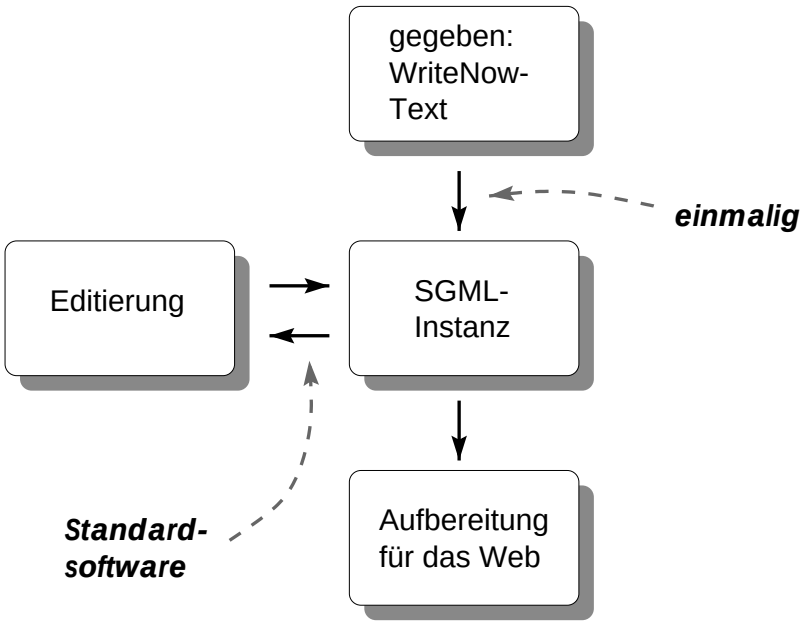

Abbildung 1: Der Ansatz zur Behandlung der gegebenen Texte: einmalige Umwandlung in SGML und darauf aufbauende Verarbeitung

Die Aufgabe der Konvertierung von umfangreichen Texten von RTF in SGML ist zwar keineswegs einfach[1], stellt jedoch nicht den Kern der Diplomarbeit dar. Von zentraler Bedeutung ist der Entwurf und die Implementation des Systems zur Publikation im Web. Bei der Entwicklung habe ich zunächst verschiedene Sichtweisen des Web-Publishings betrachtet (vgl. Abbildung 2).

1 Da es sich um eine sogenannte »up-translation« (auch: »up-conversion« oder »upward-transformation«) handelt, bei der fehlende (Struktur- und logische) Information hinzugefügt werden muß.

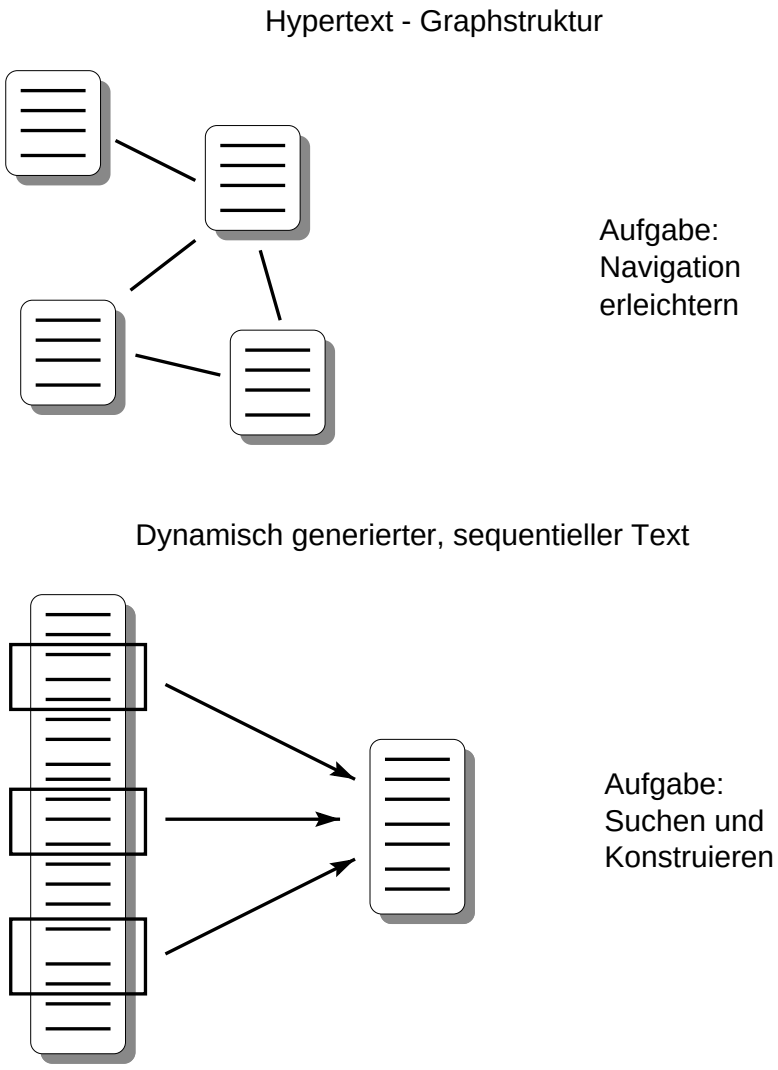

Abbildung 2: Sichtweisen des Web-Publishing

Eine Sichtweise ist das Hypertextmodell, in dem Dokumentteile als Knoten eines Graphen repräsentiert sind. Die Kanten des Graphen stellen Navigationspfade dar, die der Leser verfolgen kann. Die Aufgabe, die sich dem Verfasser eines Hypertextes stellt, ist, *jedem* Leser die »optimale« Navigation durch seinen Text zu erlauben. Für die meisten Schreiber ist es schon anstrengend genug, einen guten sequentiellen Text zu verfassen[2]. Aus diesem Grund sollte der Autor nicht damit belastet werden, den höheren Aufwand des Hypertextentwurfs leisten zu müssen, weshalb ich die Hypertextsichtweise zurückgestellt habe. Dennoch kann man einen Vorteil des Hypertextes nicht bestreiten: Der Leser ist in der Lage, sich seinen eigenen Text(ablauf) konstruieren zu können. Dies macht er einfach dadurch, daß er einigen Pfaden folgt, anderen nicht. Diesen Vorteil mit

2 Jeder Diplomand wird das wissen.

der Anforderung zu vereinen, dem Autor zu erlauben, seinen gewohnten sequentiellen Schreibstil einzusetzen, ist ein Ziel des im Rahmen dieser Diplomarbeit entwickelten Systems. Zu dessen Erreichen bekommt der Leser die Möglichkeit, Dokumentausschnitte zu einem neuen Text zusammenzufügen.

Die Gemeinsamkeit beider Ansätze ist, daß ein Leser einen Text stets sequentiell aufnimmt. Unterschiedlich ist nur die Auswahl der gelesenen Textausschnitte. Ein weiterer Unterschied ist, daß beim allgemeinen Hypertext keine kanonische Reihenfolge existiert, sobald der Grad der Verknüpfung hoch genug ist. Im Gegensatz dazu gibt es etwa bei einem Buch immer eine kanonische Reihenfolge, die durch die Anordnung der Seiten festgelegt ist. Bei dem hier verfolgten Ansatz bleibt die »richtige Reihenfolge« der Dokumentausschnitte erhalten. Wenn also einmal die Ausschnitte bekannt sind, ist der Schritt des Zusammenfügens einfach: Die Ausschnitte werden in ihrer ursprünglichen Ordnung aneinandergereiht. Dies betrifft die im letzten Abschnitt genannte Anforderung 3.

Die Anforderung der qualifizierten Suche gehe ich hier in einer Form an, die ich zuvor schon angedeutet habe: Der Benutzer kann Begriffe in vorgegebenen Kategorien suchen. Diese Kategorien werden unter Berücksichtigung der Struktur des Textes definiert. In der Praxis könnte das zum Beispiel so aussehen, daß der Benutzer »einen Brief an Herrn Müller« finden möchte und dazu in der Kategorie »Adressat« den Namen »Müller« eingibt. Hinter den Kulissen hat der Administrator festgelegt, daß die Zeichenkette »Müller« im Element »Nachname« gesucht wird, welches sich im Element »Adressat« befindet, das im Element »Brief« enthalten ist.

Bei der Frage, welcher Ausschnitt zu einer bestimmten Stelle des Dokuments zurückgeliefert werden soll, greife ich noch einmal auf die Struktur zurück. Mein System schafft die Möglichkeit, zu einem Element eine strukturelle Umgebung zu definieren, die als zugehöriger Ausschnitt angezeigt wird. So könnte man in Fortsetzung des obigen Beispiels zwei Kategorien definieren: »Briefe an einen bestimmten Adressaten« und »Adresse eines bestimmten Adressaten«. In beiden Fällen erfolgt die *Suche* nach dem Namen des Adressaten. Im ersten Fall wird jedoch der *gesamte Brief* als Ergebnis geliefert, im zweiten Fall nur die *Adresse*.

1.3 Aufbau der Diplomarbeit

Die Ausführungen bis zu diesem Punkt haben einen Eindruck von dem entwickelten System, das den Namen IP4W3 (Intelligent Publishing for the World Wide Web) bekommen hat, vermittelt. Zum Abschluß der Einleitung möchte ich kurz den weiteren Aufbau der Diplomarbeit vorstellen.

Im folgenden Kapitel 2 gebe ich eine kurze Einführung in die Konzepte und Begrifflichkeit der Techniken, die ich bei der Realisierung meines Programmsystems eingesetzt habe. Dies ist notwendig, da ich innerhalb dieser Diplomarbeit durchgängig die Terminologie verwende, die im SGML-Standard [ISO86] bzw. in der XML-Spezifikation [W3C98D],[W3C98J] benutzt wird. Des weiteren benutze ich an Stellen, die die Formatierung betreffen, Begriffe aus DSSSL [ISO96B]. Der zweite Teil des Kapitels ordnet meine Entwicklung in die Palette der bestehenden Lösungen ein und versucht, den Sinn meiner Arbeit herauszustellen[3].

Das anschließende Kapitel beschreibt das gesamte System und seine Komponenten. Ich gehe dabei zwar in die Tiefe, verzichte an dieser Stelle aber auf Implementationsdetails oder auf Details zur Konfiguration.

Kapitel 4 zeigt eine Anwendung von IP4W3. Dabei nehme ich sowohl die Sichtweise des Systembetreuers als auch den Standpunkt des Benutzers ein. Am Ende dieses Kapitels sollte es dem Leser möglich sein, den Zweck und den praktischen Nutzen meines Programmsystems zu beurteilen.

Das folgende Implementationskapitel zeigt, wie ich die zuvor beschriebenen Funktionen realisiert habe. Dazu gehört an einigen Stellen auch eine Beschreibung der Entscheidungsprozesse, die ich durchlaufen habe, bevor ich eine bestimmte Lösung gewählt habe.

In Kapitel 6 zeichne ich die Schritte nach, die zur Umwandlung der gegebenen Textdaten in SGML nötig waren. Für die Funktionsweise meines Programms spielt dies zwar keine Rolle, so daß die Beschreibung auch in einem Anhang ihren Platz finden könnte, jedoch halte ich die ausschlaggebende Motivation, die von dieser Anwendung ausging, für Grund genug, ihr einen Platz im Hauptteil einzuräumen.

Das abschließende Kapitel 7 diskutiert einige Verbesserungen und Erweiterungen, die an IP4W3 durchgeführt werden können.

Im Anhang A sind Bedienungsanleitungen für die ausführbaren Programme von IP4W3 zu finden (Manualseiten). Der folgende Anhang enthält DTDs und Stylesheets, die ich im Rahmen dieser Arbeit geschrieben habe.

---

3 Um es marktgerecht zu formulieren: Ich versuche *Alleinstellungsmerkmale* anzuführen.

Grundlagen und Einordnung in das Themengebiet

2

In diesem Kapitel beschreibe ich die Technologien, die für diese Diplomarbeit von Bedeutung sind, gebe einen Überblick über vergleichbare Systeme und ordne das von mir entwickelte System ein.

2.1 SGML und XML

Dieser Abschnitt vermittelt grundlegende Informationen über die Standard Generalized Markup Language (SGML) und die Extensible Markup Language (XML). SGML und XML sind hier von Bedeutung, da ich sie als Eingabeformat für IP4W3 ausgewählt habe. Sie erlauben, die Struktur- und Metainformationen, von denen im ersten Abschnitt die Rede war, in Texten zu repräsentieren. Beide Sprachen konzentrieren sich auf diese Art von Auszeichnungen und erlauben keinerlei Formatierung. Damit besitzen sie genau den Schwerpunkt, der hier benötigt wird.

2.1.1 Ideen und Konzepte

Im Umfeld von SGML geht es stets um Textdokumente, d.h. um solche Dokumente, die zum überwiegenden Teil aus Text bestehen, die aber auch Graphiken, Tabellen und so weiter enthalten dürfen. Textdokumente können etwa in folgende Komponenten aufgegliedert werden.

Der Inhalt

: umfaßt neben dem Text auch die Abbildungen, Tabellen usw.

Die Struktur

: meint Gliederungen, wie sie technische und wissenschaftliche Dokumente in der Regel besitzen (z.B. Kapitel, Abschnitte usw.). Im Falle von mathematischen Abhandlungen geht diese Unterteilung bis auf die Ebene von (mathematischen) Sätzen, Definitionen und ähnlichem. In der Prosa sind solche scharfen Trennungen nicht zu finden, während lyrische Texte eine Gliederung etwa in Verse besitzen.

Ein weiteres, beliebtes Beispiel sind Briefe. Sie besitzen eine weitgehend einheitliche Struktur der Art Absender, Adressat, Datum, Betreff, Anrede, Rumpf. Die Konventionen für die Gestaltung des Briefkopfes sind jedoch von Land zu Land sehr verschieden.

Die Information über die visuelle Darstellung

wird in der Literatur sehr unterschiedlich bezeichnet. Neben dem Begriff *visual markup* ist auch häufig der Ausdruck *layout markup* vertreten.
Zu verstehen ist darunter die Festlegung der Schriftart, der Zeichensatzgröße, aber auch des Satzspiegels, also die Aufteilung der Seite und die Plazierung von Elementen auf der Seite.

Die logische Information

geht auf die Bedeutung von Textstellen und von einzelnen Worten ein. Anders ausgedrückt kann man auch von semantischer Information sprechen.
Ist ein Wort ein Name oder ein Fachbegriff? Stellt ein Absatz ein Zitat oder eine These dar? - Diese Information soll hier *logische Information* heißen.

Generic Markup versus Visual Markup

Bei der Texterstellung unterscheidet man zwischen Generic Markup und Visual Markup. Letzteres dürfte dem Leser vertrauter sein, denn als ein Spezialfall des Visual Markup ist das Konzept des »What you see is what you get« (WYSIWYG) weitgehend bekannt.

Die Arbeit an einem Dokument konzentriert sich dabei auf die Erfassung des Textes und die Festlegung der äußeren Form. Das für den Verfasser auf den ersten Blick so einfache Vorgehen - er sieht den Text ständig in seiner »endgültigen« Form - birgt auch einige Probleme. So muß er selbst darauf achten, daß gleichartige Information (Name, Begriff usw., s.o.) auch stets gleich aussieht. Bei der Speicherung geht die logische Information und die Strukturinformation verloren. Lediglich die äußeren Attribute werden gespeichert.

Diesen Mißstand zu beheben, hat sich das Generic Markup vorgenommen. Die Idee ist einfach: Der Autor wird von der Bürde befreit, sich während der Texterstellung Gedanken über das Aussehen machen zu müssen und bekommt darüber hinaus die Möglichkeit, strukturorientiert zu arbeiten. Das heißt er kann die Gliederung seines Textes, die er beim Schreiben ohnehin im Kopf hat, in sein Dokument übernehmen, ohne den Umweg über die Formatierung, das Layout usw. machen zu müssen. Die Formatierung wird dann in einem eigenen Schritt mit Bezug auf die logische Information und die Struktur ausgeführt.

Document Type Definition

Für die Umsetzung dieser Ideen ist bei SGML der Dokumenttyp wichtig. Gemeint ist damit folgendes: Wenn man sich verschiedene Texte hinsichtlich ihres Aufbaus und ihrer Struktur ansieht, sind gewisse Gemeinsamkeiten und Unterschiede feststellbar. Je nachdem wie groß diese Differenzen sind, zählt man verschiedene Dokumente zum selben oder zu unterschiedlichen Typen.

Bei dem Versuch, den Dokumenttyp »Buch« näher zu charakterisieren, fällt die Gliederung in Kapitel, Abschnitte, Unterabschnitte und letztlich Absätze auf. Die Anzahl und die Tiefe der Gliederungen variiert von Buch zu Buch. Trotz dieser Unterschiede erscheint es sinnvoll, zwei Bücher als gleichartig anzusehen, auch wenn die Anzahl der Kapitel voneinander abweicht.

Eine formale Typisierung von Dokumenten muß also die Gemeinsamkeiten festlegen, zugleich aber genügend Flexibilität für den konkreten Fall lassen. In SGML erfolgt die Charakterisierung eines Dokumenttyps in der Document Type Definition (DTD)[1]. Die DTD legt fest, welche logischen Elemente ein Text enthalten darf, welche Elemente unbedingt vorhanden sein müssen und in welcher Reihenfolge sie erscheinen dürfen. Um das Beispiel »Buch« weiter zu verfolgen: Ein Buch muß in jedem Fall einen Titel haben, der auch zwingend am Anfang erscheinen muß. Ein Index hingegen ist optional.

Eine Verknüpfung mit einem bestimmten Layout ist in der DTD nicht enthalten. Für die Verarbeitung von SGML-Dokumenten gibt es einen weiteren ISO-Standard mit dem handlichen Namen »Document Style Semantics and Specification Language« (DSSSL, gesprochen »Dissel«, ISO 10179). Die Trennung von Struktur und Aussehen beschert den Dokumenten, neben den genannten Vorteilen, eine weitere positive Eigenschaft: Da ihre Formatierung in keiner Weise feststeht, sind sie vollkommen unabhängig vom Ausgabemedium. Ob also ein Text auf Papier (beliebiger Größe) gedruckt, auf dem Bildschirm (in beliebiger Farbe, Schriftart usw.) ausgegeben oder von einem Sprachsynthesizer gesprochen wird, ist vollkommen offen.

2.1.2 SGML–Syntax – Ein Beispiel

Document Type Definition

Eine DTD für einen selbsterdachten Dokumenttyp zeigen die folgenden Zeilen:

---

1 Die theoretische Grundlage hierfür bilden die regulären Grammatiken. Für Details siehe [GOLD90] und [WIHE96].

```
<!--      Element Min Inhaltsmodell      -->
<!ELEMENT notiz   - -
          (titel, autor, datum, absatz+)   >
<!ELEMENT titel   - - (#PCDATA)            >
<!ELEMENT autor   - - (#PCDATA)            >
<!ELEMENT datum   - - (#PCDATA)            >
<!ELEMENT absatz  - - (#PCDATA | wichtig)* >
<!ELEMENT wichtig - - (#PCDATA)            >
```

Dieses einfache Beispiel zeigt eine DTD für eine »Notiz«, die besteht aus

- ✗ einem »Titel«,
- ✗ einem »Autor«,
- ✗ einem »Datum«
- ✗ sowie einem oder mehreren (dafür steht das Pluszeichen in der Beschreibung des Elementes »notiz«) »Absätzen«.

Die ersten drei Elemente enthalten reinen Text, in SGML-Terminologie *Parsed Character Data* (PCDATA) genannt. Ein Absatz kann wahlweise (dafür steht der senkrechte Strich) aus Character Data oder aus »wichtigem« Text bestehen und zwar beliebig kombinierbar (dafür steht der Stern). Eine DTD läßt sich auch in einer Baumstruktur darstellen. Für das Beispiel zeigt folgende Abbildung eine Baumdarstellung.

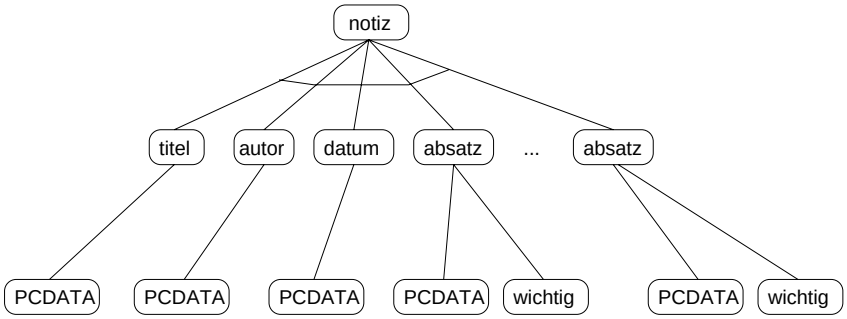

Abbildung 3: Die Notiz–DTD als Baumdarstellung

Dokumentinstanz

Ein SGML-Dokument wird auch »Instanz« genannt, abgeleitet von dem englischen »instance«, zu Deutsch »Beispiel«[2]. Eine Instanz ist also ein Beispiel, ein Vertreter aus einer Klasse von Dokumenten, die durch die Document Type Definition bestimmt ist.

2 Obwohl der Begriff »Instanz« im Deutschen eine andere Bedeutung hat, werde ich mich dem üblichen Gebrauch im SGML-Umfeld anschließen und ein Dokument auch Instanz nennen.

Für die obige DTD sieht eine Instanz etwa wie folgt aus.

```
<!DOCTYPE notiz SYSTEM "/usr/home/sm/dtd/notiz.dtd" >
<notiz>
 <titel>Termine</titel>
 <autor>Stefan Mintert</autor>
 <datum>1. Februar 1999</datum>
 <absatz>
    <wichtig>Abgabe der Diplomarbeit</wichtig>
 </absatz>
 <absatz>    ...
 </absatz>
</notiz>
```

In der ersten Zeile, der sogenannten *Doctype-Deklaration*, nennt die Instanz diejenige DTD, zu der sie konform ist. Die weiteren Zeilen sind das eigentliche Dokument. Die Syntax ist recht einfach zu verstehen: Eine Instanz besteht aus hierarchisch verschachtelten Elementen. Ein Element beginnt mit einem *Start-Tag* (z.B. <notiz>) und endet mit einem *End-Tag* (z.B. </notiz>). Der englische Begriff »tag« bedeutet so viel wie Schildchen oder Etikett. Verbindet man anschaulich damit die Auszeichnung von Waren mit einem Preisschild, so ist sofort klar, weshalb der englische Oberbegriff »tagging language« mit »Auszeichnungssprache« übersetzt wird.

Zusammen mit der gezeigten DTD verifiziert man leicht, daß das vorliegende Beispiel eine gültige Instanz ist. Für komplexere Fälle gibt es Werkzeuge (Parser), die diese syntaktische Verifikation durchführen.

2.1.3 Hypertext Markup Language

Die Hypertext Markup Language (HTML) ist nicht etwa eine Konkurrentin für SGML, sondern eine Anwendung davon. Formal ist HTML nichts weiter als eine DTD[3]. HTML ist zwar die lingua franca des Web, jedoch für die Benutzung als Eingabeformat für IP4W3 nicht ausreichend, da die benötigten Metainformationen nicht oder nur unter unzumutbaren Einschränkungen in HTML-Instanzen kodiert werden können[4].

2.1.4 Extensible Markup Language

Bei der Extensible Markup Language (XML) handelt es sich um eine *Teilmenge von SGML*. Damit sind alle *hier* für SGML gemachten Aussagen auch für XML gültig. Die technischen Unterschiede zwischen XML und SGML spielen innerhalb dieser Diplomarbeit eine unterge-

3 Genaugenommen gibt es für HTML 4 mittlerweile drei DTDs mit verschiedenen Aufgaben.

4 Die Informationen lassen sich zwar als Wert des Attributs `class` in ein Dokument aufnehmen, jedoch sind die Attributwerte in keiner Weise beschränkt. Es fehlt also eine formale Definition des Dokumenttyps.

ordnete Rolle. Wichtig sind die damit verbundenen Konzepte, die bei beiden Sprachen gleich sind (zumindest soweit wie ich sie hier benutze). Aus diesem Grund unterscheide ich im folgenden in der Regel nicht zwischen XML und SGML. Dort wo es notwendig ist, erwähne ich es ausdrücklich. Ein Vergleich beider Sprachen ist in [CLAR97] zu finden[5]. Da XML zweifellos einen großen Einfluß auf die Zukunft des Web haben wird und möglicherweise SGML verdrängen wird, ist XML hier von ebenso großer Bedeutung wie SGML.

Eine wesentliche, durch XML eingeführte Neuerung ist, daß Dokumente auch *ohne* DTD existieren können. Sie müssen nur noch abgeschwächten Bedingungen genügen. In diesem Fall spricht man von der *Wohlgeformtheit* eines Dokuments. Halten sie sich darüber hinaus auch noch an die Regeln einer DTD, spricht man von *Gültigkeit*. Ich halte diese Innovation für die schlechteste Veränderung gegenüber SGML, die XML dem Verfasser eines Dokuments zugesteht. Als Folge davon sind nämlich Dokumente zugelassen, die völlig freie Elementnamen besitzen. Unterliegen die Namen jedoch keiner Konvention, so sind sie wertlos. Sämtliche Programme (wie auch Stylesheets) müssen für ein konkretes Dokument geschrieben werden, nicht für eine Klasse von Instanzen. Programme, die nach Informationen in einem Dokument suchen, können die Metainformationen nicht nutzen, weil sie die Elementnamen nicht kennen. Die strukturelle Korrektheit des Dokuments läßt sich nicht verifizieren, da es keine Spezifikation der Abhängigkeiten gibt. Da ich beim Entwurf meines Systems die Existenz einer DTD vorausgesetzt habe, ist die Wohlgeformtheit von keiner großen Bedeutung. Ich verwende sie allerdings *intern* an einer Stelle, um eine Effizienzsteigerung zu erzielen.

2.2 DSSSL und XSL

Die Verarbeitung von SGML-Dokumenten zum Zwecke der Formatierung mit der Document Style Semantics and Specification Language (DSSSL) soll hier nur kurz angerissen werden. Ich verwende DSSSL zwar, um Dokumente in HTML umzuwandeln, jedoch liegt die wesentliche Leistung meiner Arbeit im Bereich SGML/XML, so daß für das Verständnis der folgenden Kapitel schon ein geringes Maß an DSSSL-Kenntnissen genügt.

In der folgenden Abbildung sind die wesentlichen Stationen, die ein Dokument durchläuft, skizziert.

5 Eine sehr lesenswerte Arbeit zum Vergleich der Inhaltsmodelle beider Sprachen ist [KILP98].

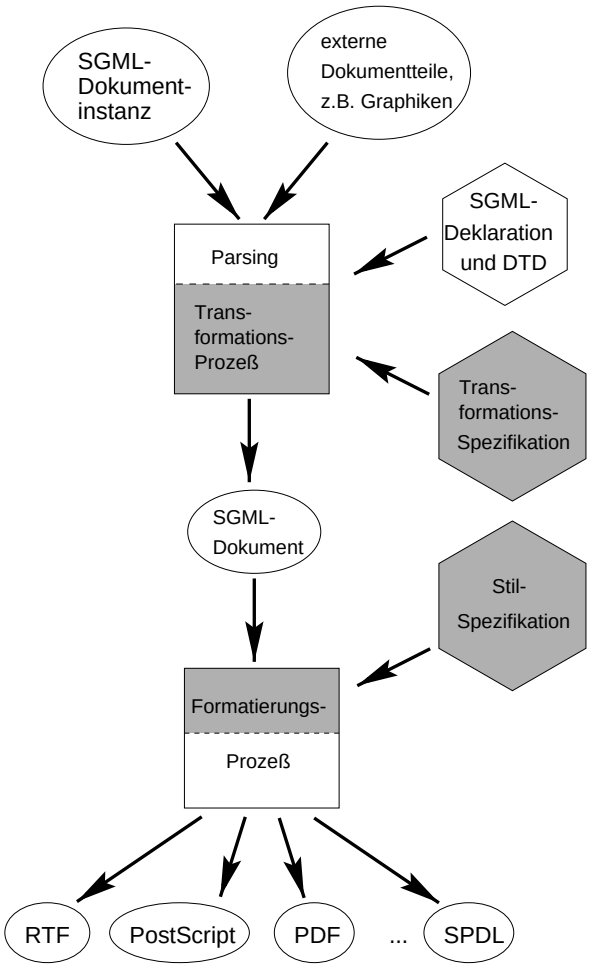

Abbildung 4: Schematische Darstellung der Verarbeitung von SGML-Dokumenten mit DSSSL. Die grauen Teile unterliegen dem Einfluß von DSSSL.

Ausgangspunkt ist natürlich die SGML-Instanz als Eingabeobjekt. Die erste Prüfung führt ein Parser aus, der zu diesem Zweck die DTD benötigt. In einem weiteren Schritt, dem *SGML Tree Transformation Process*, werden Operationen »auf dem Baum« der Elemente durchgeführt. Möchte man etwa im obigen Beispiel nur den Autor einer Notiz extrahieren, so könnte der SGML Tree Transformation Process - bildlich gesprochen - die anderen Äste aus dem Baum abschneiden und nur den Autor übrig lassen. Dieser Vorgang kann als ein Vorverarbeitungsschritt angesehen werden. Sein Ergebnis ist wieder eine SGML-Instanz, die aber gegebenenfalls zu einer anderen DTD konform ist. Die Formatierung geschieht dann im *SGML Tree Formatting Process*, an dessen Ende ein Dokument im gewünschten Ausgabeformat steht.

Die obigen Prozesse werden in DSSSL mit der Transformation Language beziehungsweise der Style Language beschrieben. Zur Ausführung ist ein Interpreter nötig, der auch als DSSSL-Maschine bezeichnet wird[6]. Ihre Aufgabe geht über die in der Grafik grau dargestellten

Teile hinaus. Eine DSSSL-Maschine muß nämlich die Formatierungsanweisungen von DSSSL in das konkrete Ausgabeformat umsetzen. DSSSL-Programme (auch »Stylesheet«) sind *unabhängig* vom Ausgabeformat. Sie können es sein, weil die Sprache typographische Objekte (*Flow Object*s), wie Seiten, Absätze, Tabellen usw. zur Verfügung stellt, die nicht an ein bestimmtes Format gebunden sind. Die DSSSL-Maschine wandelt dann zum Beispiel den Abschluß des Flow Objects »Seite« in die konkrete Anweisung `\newpage` (TeX), `showpage` (PostScript) usw. um[7].

Zu beachten ist, daß in der Abbildung 4 unter den Ausgabeformaten nicht HTML genannt ist. Zwar handelt es sich bei HTML de facto um das Ausgabeformat für Web-Browser, jedoch ist die Hypertext Markup Language formal nur eine SGML-Anwendung. Um HTML zu erzeugen, ist also nur eine Transformation der Eingabeinstanz in eine HTML-Instanz durchzuführen. Diese dopelte Rolle von HTML (Zwischenformat in der DSSSL-Verarbeitungskette und Ausgabeformat für's Web) sorgt dafür, daß in der Praxis zwei DSSSL-Programme benötigt werden, eines für die HTML-Ausgabe und eines für die Druckausgabe.

2.2.1 Extensible Style Language

Die Extensible Style Language (XSL) ist die »Formatierungssprache für XML«. Sie verhält sich ungefähr so zu DSSSL, wie sich XML zu SGML verhält. Allerdings ist XSL schon aus syntaktischen Gründen *keine Teilmenge* von DSSSL. XSL wird zur Zeit vom World Wide Web Consortium (W3C) entwickelt. Innerhalb von IP4W3 kommt XSL nicht zum Einsatz. Ich erwähne sie an dieser Stelle, weil sie auf den ersten Blick die Sprache der Wahl ist, wenn es darum geht, XML-Instanzen zu formatieren. Unter funktionalen Gesichtspunkten ist DSSSL aber *mindestens* genau so gut geeignet. Diese Einschätzung kann in der Zukunft anders ausfallen, wenn XSL fertig ist und möglicherweise breitere Unterstützung findet.

2.3 Bestehende Systeme

Um das vorliegende System IP4W3 in die Menge der bestehenden Systeme einzuordnen, stütze ich mich auf die Klassifikation des Whirlwind Guide von Steve Pepper ab [PEPP98]. Dabei handelt es sich um die seit Jahren umfangreichste und stets aktuellste Übersicht über

6 Eine freie Implementation ist James' DSSSL Engine (Jade).

7 Die TeX- und PostScript-Beispiele sind nicht mehr als Beispiele. Ob eine DSSSL-Maschine tatsächlich *genau diese* Ausgabe erzeugt, kann ich nicht sagen, da dies auch durch DSSSL nicht festgelegt ist. Es bleibt der Maschine überlassen, *wie* sie ihr Ziel erreicht.

SGML- und mittlerweile auch XML-Software. Pepper benutzt in seiner Übersicht die folgende grobe Kategorisierung:

editing and composition

Werkzeuge zur interaktiven Erzeugung, Modifikation und Komposition von SGML-Dokumenten.

electronic delivery

Werkzeuge zum elektronischen Versenden von SGML-Dokumenten.

conversion

Werkzeuge für die skript-gesteuerte Erzeugung und Modifikation von SGML-Dokumenten.

document storage and management

Werkzeuge zur Speicherung und Verwaltung von SGML-Dokumenten.

control information development

Werkzeuge zur Erzeugung, Modifikation und Dokumentation von DTDs, DSSSL-Spezifikationen, FOSIs usw.

parsers and engines

Werkzeuge zur Erstellung von SGML- und XML-Parsern, HyTime-Maschinen und DSSSL-Maschinen.

IP4W3 ordnet sich nach dieser Einteilung in die Kategorien »document storage and management« und »electronic delivery« ein, integriert jedoch auch Komponenten aus der Kategorie »conversion«. Der Unterschied zu den meisten von Pepper aufgeführten Produkten ist, daß IP4W3 kein Programm ist, das eindeutig einer dieser Kategorien zuzuordnen ist und auch nicht nur *eine* Aufgabe erfüllt. Vielmehr handelt es sich bei IP4W3 um ein integriertes System, dessen Funktionsumfang ein Querschnitt der oben genannten Kategorien ist. Zu den kommerziellen Produkten, die Funktionen nur einer Kategorie anbieten, kann und will IP4W3 keine Konkurrenz sein. Statt dessen habe ich versucht, Funktionen aus verschiedenen Kategorien zu implementieren, um ein neues Konzept umzusetzen. Am besten läßt sich das dadurch erläutern, daß ich die Unterkategorien von Peppers Whirlwind Guide vorstelle, die von IP4W3 ganz oder teilweise abgedeckt werden.

Die Kategorie »document storage and management« unterteilt sich in »SGML document and component manager«, »SGML document manager«, »SGML DMS middleware« und »SGML document management utility«. IP4W3 enthält einen großen Component-Management-Anteil, was von Pepper folgendermaßen beschrieben wird: »[...] Systeme,

die Grove-Komponenten für eine beliebige DTD speichern und verwalten. [...] Die Speicherungseinheit kann das Dateisystem des Betriebssystems, eine Datenbank, ein Netzwerk oder eine Kombination der drei sein.« Reine Vertreter dieser Klasse bieten nur diese Verwaltung von Dokumenten-Komponenten und den Zugriff darauf. Kommerzielle Produkte, wie etwa Astoria von Chrystal Software, bieten eine programmierbare Schnittstelle (API), auf die weitergehende Funktionen aufgesetzt werden können. IP4W3 stellt für den Zugriff auf Dokumenten-Komponenten sogenannte Suchmuster zur Verfügung.

Als nächste relevante Unterkategorie ist »SGML document management utility« zu nennen: »Diese Werkzeuge stellen einige Aspekte von SGML-Dokumenten- oder Komponentenverwaltung (z.B. Indexierung, Suche, Versionskontrolle, Datenbankanbindung) zur Verfügung [...]« IP4W3 gehört dank seiner Indexierung und Suchfunktion auch in diese Kategorie.

Die äußerlich hervorstechendste Eigenschaft von IP4W3 ist die Benutzung von Web-Browsern als Schnittstelle zum Benutzer. Aus diesem Grund gehört das System auch in Peppers Kategorie »electronic delivery/SGML library web server«. Bekanntester kommerzieller Vertreter ist das Programm DynaWeb.

Da IP4W3 die Produkte *CoST*, *Jade* (DSSSL-Maschine) und *nsgmls* (SGML-Parser) benutzt, integriert das System Funktionen aus den folgenden Kategorien/Unterkategorien:

conversion/general s-converter

»Werkzeuge, [...] die SGML-Dokumente verarbeiten und SGML- oder nicht-SGML-Ausgabe erzeugen.«

parsers and engines/DSSSL engine

IP4W3 arbeitet mit DSSSL-Programmen, um Dokumente bzw. Ausschnitte zu formatieren.

conversion/composition converter

»Werkzeuge zur Konvertierung von SGML-Dokumenten in ein Format, das für die Verarbeitung durch ein Kompositionsprogramm geeignet ist.« Als »composition converter« kommt *Jade* zum Einsatz. Die Komposition von einem Ausgabedokument wird von IP4W3 auf Basis einer Stichwortsuche durchgeführt.

IP4W3 im Vergleich zu anderen Systemen

IP4W3 läßt sich nicht eindeutig in eine Kategorie einordnen. Allein diese Tatsache stellt ein Alleinstellungsmerkmal des Systems dar. Sofern der Vergleich ausschließlich auf die Kategorie beschränkt ist, in der das Vergleichsprodukt eingeordnet ist[8], kann IP4W3 im direkten Vergleich mit kommerziellen Produkten in vielen Fällen nicht be-

stehen. Der Grund dafür ist, daß die Produkte in Peppers Kategorisierung Spezialisten in der jeweiligen Kategorie sind. Das ist nicht überraschend, da die von Pepper betrachteten Produkte selbst erst die Kategorien definieren. Es gibt nur sehr wenige Programme, die sich nicht eindeutig einordnen ließen.

Es wäre weder inhaltlich sinnvoll noch wissenschaftlich interessant gewesen, gegen einen dieser Spezialisten anzutreten. Besonders deutlich zeigt sich das an der Kategorie »SGML document manager«, die sich in Peppers Interpretation kaum von klassischen Dokumenten-Datenbanken unterscheidet: »Systeme, die SGML auf Dokumentenebene oder als große Stücke speichern und verwalten, und die Entities oder Elemente auf hoher Ebene der Dokumentenhierarchie repräsentieren.« Im Gegensatz dazu ist der oben schon genannte Bereich der Komponentenverwaltung viel interessanter. IP4W3 sieht Dokumente deshalb nicht als monolithisches Gebilde an, sondern berücksichtigt die interne Struktur und nutzt sie aus, um neue und bessere Funktionen darauf anzubieten.

Zusammenfassend lassen sich als Vorteile von IP4W3 gegenüber vorhandenen Systemen folgende Eigenschaften nennen:

- ✗ IP4W3 ermöglicht das Retrieval von Dokumentausschnitten unter Berücksichtigung der Struktur des Dokumenttyps (durch Suchmuster, die auf der DTD definiert werden), der Struktur der Instanz sowie einer *qualifizierten* Benutzereingabe; das ist eine Eingabe, die nicht nur aus einem Suchbegriff, sondern auch aus einer Suchkategorie besteht.
- ✗ Das Ergebnis einer Suche kann vom Benutzer verfeinert werden.
- ✗ Über eine an Warenkorb-Systemen orientierte Schnittstelle kann der Benutzer Dokumentausschnitte kombinieren, die das Ergebnis von *mehreren* Stichwortsuchen sind.
- ✗ IP4W3 ermöglicht den Zugriff auf SGML/XML-Datenbestände über eine WWW-Schnittstelle. Das heißt, es können herkömmliche Web-Browser verwendet werden, die auf jedem relevanten Betriebssystem zur Verfügung stehen.
- ✗ Es gibt kein System, das die von IP4W3 implementierten Funktionen in vergleichbarer Weise anbietet[9].

8 Ich verstehe das allerdings als unzulässigen Vergleich.
9 Grundlage für diese Aussage ist die Kategorisierung des Whirlwind Guide [PEPP98].

Beschreibung des Gesamtsystems

3

In diesem Kapitel beschreibe ich das von mir entwickelte System IP4W3, seinen Aufbau und die einzelnen Komponenten. Die Implementierung und die Bedienung lasse ich hier aus, da beides in späteren Kapitel ausführlich erläutert wird.

3.1 Überblick

Im Konzept von IP4W3 gibt es drei Personen, den Autor, der das zu veröffentlichende Material stellt, den Benutzer (»Web-Surfer«), der das Material einsehen möchte, sowie den Administrator, der den Server betreibt und die technische Wartung übernimmt.

Während die Trennung zwischen dem Benutzer auf der einen Seite und Autor/Administrator auf der anderen Seite sofort einleuchtend ist, verdient die Einteilung in Autor und Administrator einige Erklärungen: Der Autor soll im Idealfall nichts weiter tun als seinen Text liefern. Er soll nicht mit den technischen Details belastet werden, die die Veröffentlichung im Web mit sich bringt. Statt dessen soll der Administrator diese Aufgaben erledigen. Bis zu diesem Punkt entspricht das dem üblichen Szenario des Webpublishing. Neu ist in diesem Fall die Aufgabe, die Konfiguration von IP4W3 zu übernehmen. Dazu muß der Administrator in der Lage sein, (a) eine strukturelle Beschreibung von Textstellen auf Basis der DTD zu formulieren, in denen der Benutzer suchen können soll und (b) Instanzen der DTD in HTML zu transformieren. Letzteres ist keine Besonderheit von IP4W3, sondern ist auch bei der »herkömmlichen« Vorgehensweise notwendig; IP4W3 sieht hierfür DSSSL als Formatierungssprache vor. Der Administrator muß daher neben Grundkenntnissen von SGML/XML auch Vorwissen über DSSSL mitbringen.

Um das vorliegende IP4W3-System zu verstehen, bietet sich zunächst die Erklärung aus Benutzersicht an (vgl. Abbildung 5). Der Benutzer erhält mit IP4W3 die Möglichkeit, innerhalb von Texten nicht einfach eine Volltextsuche durchzuführen, sondern den gesuchten Begriff in einer von mehreren Kategorien suchen zu lassen. Als Ergebnis bekommt er eine von Suchmaschinen bekannte Liste der Treffer. Bei den Treffern handelt es sich jedoch nicht um verschiedene Dateien,

die den Suchbegriff enthalten, sondern es sind *Ausschnitte* eines einzelnen durchsuchten Textes. Aus dieser Liste kann er eine oder mehrere Textstellen aussuchen und »vollständig«[1] anzeigen lassen. Die Größe der dargestellten Textausschnitte kann er weiter kontrollieren, bis er schließlich ein für ihn zufriedenstellendes Ergebnis vor sich hat. Prinzipiell besteht keine Einschränkung des Ausgabeformats.

Darüber hinaus kann der Benutzer die Resultate mehrerer Suchanfragen zusammenfassen. Zu diesem Zweck bietet IP4W3 eine an Warenkorb-Systemen orientierte Schnittstelle an: Hat der Benutzer einen Ausschnitt gefunden, der seinen Ansprüchen genügt, so kann er den Browser anweisen, sich den Ausschnitt zu »merken«. Er kann ihn in seinen virtuellen Warenkorb legen und eine weitere Suche starten. Nach und nach füllt er seinen Warenkorb mit Dokumentausschnitten. Hat er alles was er braucht, so kann er alle »gemerkten« Ausschnitte zusammen anzeigen und drucken lassen. Formularelemente, die sonst zur Steuerung in die Webseite integriert sind, werden bei der Anzeige der »gemerkten« Ausschnitte unterdrückt.

Die Informationen, die das System aus dem Verhalten des Benutzers, insbesondere dem Verwerfen bzw. dem Behalten von Textausschnitten, gewinnt, werden die Grundlage darstellen, um mit Mitteln des maschinellen Lernens Verbesserungen des Suchergebnisses zu erzielen. Für diese Weiterentwicklung schafft meine Diplomarbeit die Basis (die Umsetzung ist nicht Teil meiner Arbeit; siehe dazu auch Kapitel 7).

1 Die Bedeutung dieses Begriffs muß später genau geklärt werden. Sie spielt hier eine wichtige Rolle.

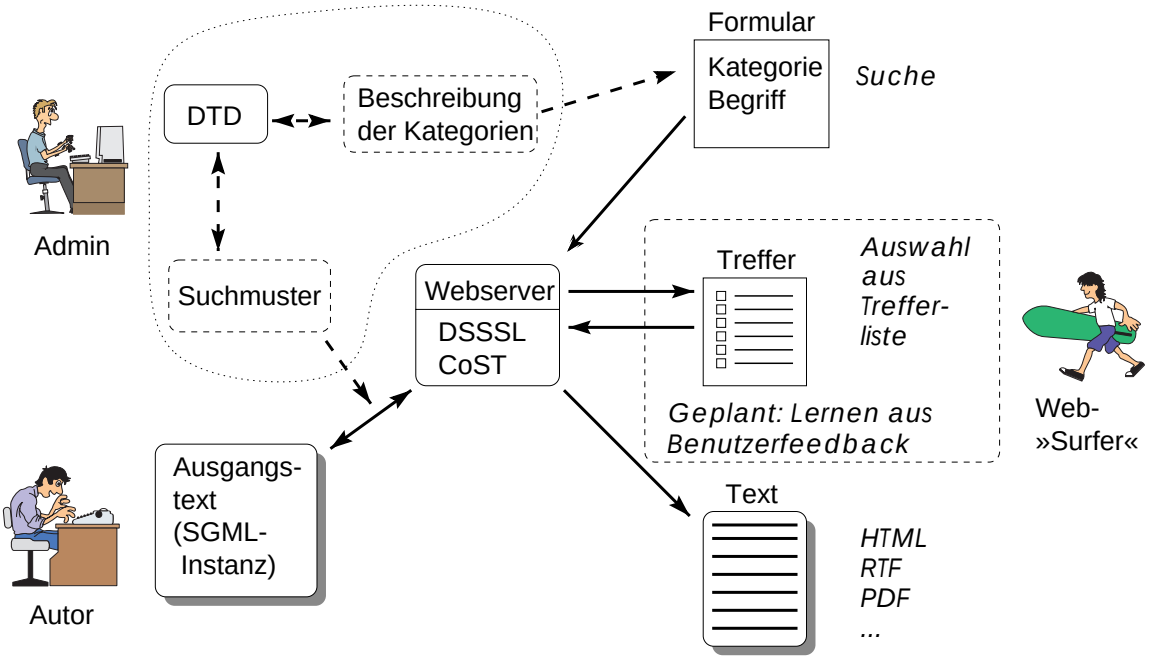

Abbildung 5: Konzeptskizze von IP4W3

Hinter den Kulissen spielt sich folgendes ab: Die Kategorien, in denen der Benutzer suchen kann, sind Elemente des Textes, die sich in einem bestimmten Kontext befinden. Zum Beispiel kann ein Element »Name« eine andere Bedeutung haben, je nach dem, ob es im Element »Adressat« oder im Element »Absender« enthalten ist. In IP4W3 werden solche Konstellationen als *Suchmuster* bezeichnet. Ein Suchmuster gibt an, in *welchem Element* nach dem fraglichen Begriff gesucht werden soll und in *welchem Kontext* dieses Element stehen muß.

Die Aufgabe des Administrators ist es, Suchmuster auf Grundlage der DTD zu formulieren und sie als Kategorien zu beschreiben, die für den Benutzer verständlich sind. Die Bezugnahme auf die DTD sorgt dafür, daß die Arbeit *nur einmal* pro DTD, also nicht für *jedes* Dokument erledigt werden muß und daß alle Instanzen dieser DTD anschließend ohne zusätzlichen Aufwand in IP4W3 veröffentlicht werden können.

Nach der Auswahl der gewünschten Textstellen durch den Benutzer erfolgt die *Formatierung* mit Hilfe der DSSSL-Maschine *Jade* (James' DSSSL Engine). Zu diesem Zweck muß der Administrator (oder der Autor) ein DSSSL-Stylesheet zur Verfügung stellen.

Eine zentrale Frage besteht darin, *welcher Textausschnitt* zu einem Treffer ausgegeben werden soll. Es ist sicher nicht sinnvoll, nur das Element auszugeben, in dem der Suchbegriff gefunden wurde. Im Einzelfall könnte es sich dabei nämlich allein um den Begriff handeln. Die Idee zur Lösung dieses Problems besteht in der Annahme,

daß es zu jedem Treffer einen gewissen inhaltlichen Kontext gibt, der für sich allein verständlich ist und der das vom Benutzer gewünschte Ergebnis darstellt. Diesem *inhaltlichen* Kontext sollte je nach Güte der DTD (und auch der Instanz) ein *struktureller* Kontext entsprechen, der benutzt werden kann, um ebenfalls auf Basis der DTD zu formulieren, welcher Textausschnitt zu einem Treffer zurückgeliefert werden soll. Ein solcher Textausschnitt wird in diesem Zusammenhang als *Atom* bezeichnet. Ein Atom ist also ein Stück Text, das nicht weiter unterteilt werden soll, um das Verständnis des Textstücks zu gewährleisten.

Zusammenfassend möchte ich noch einmal die Begriffe »Suchmuster« und »Atom« voneinander abgrenzen: Das Suchmuster dient dazu, einen strukturellen Ausschnitt von Instanzen zu beschreiben, in denen nach Begriffen gesucht wird. Ist eine solche Stelle erst einmal gefunden worden, so muß zu diesem Treffer ein Textausschnitt als Ergebnis geliefert werden, der alleinstehend verständlich ist - das ist das Atom.

In der vorliegenden Implementation von IP4W3 wird der Spezialfall realisiert, daß das Atom mit dem Suchmuster übereinstimmt. Das heißt, daß das Suchmuster zwei Funktionen erfüllt. Bei einem Treffer bestimmt es nicht nur, ob der Treffer im »richtigen« Kontext stattgefunden hat, sondern im positiven Fall auch, daß dieser Kontext das Suchergebnis darstellt.

3.2 Komponenten

Dieser Abschnitt beschreibt, aus welchen Komponenten IP4W3 besteht und welche Funktionen sie erfüllen. Bei den Komponenten handelt es sich um Suchmuster/Atome, Formatierungs-Rahmen und -Rumpf, Stichwortindex, Formularmasken und CGI-Skripte.

3.2.1 Suchmuster

Bei der Suche wird ein vorgegebener Elementkontext berücksichtigt. Dieser Kontext wird hier als *Suchmuster* bezeichnet, angelehnt an die *Patterns* in der XSL-Note [W3C97E] (siehe auch Abschnitt 5.1). Ein Suchmuster besteht aus einem *Zielelement*, das in eine Umgebung anderer Elemente eingebettet ist. Die Umgebung erfüllt zwei Funktionen:

1. Sie schränkt die Suche ein, da nur solche Zielelemente als Treffer geliefert werden, die sich in der passenden Umgebung befinden.
2. Sie stellt den Teil des Dokuments dar, der als Suchergebnis für das darin befindliche Zielelement angezeigt wird. (Dies ist, wie zuvor beschrieben, eigentlich die Aufgabe des Atoms.)

IP4W3-Suchmuster werden als SGML-Instanz formuliert[2]. Das *Zielelement* (`target-element`) bildet den Kern des Suchmusters. In diesem Element wird später bei der Suche nach dem vom Benutzer eingegebenen Begriff gesucht. Eingebettet ist dieses Zielelement in eine Umgebung weiterer Elemente eines *festzulegenden Typs* (`element type="..."`). Soll der Typ nicht festgelegt werden, so steht dafür die Angabe `any`, was bedeutet, daß im Dokument ein Element eines beliebigen Typs stehen darf. Zur Veranschaulichung ein einfaches Beispiel:

```
<!-- Suchmuster -->
<element type="absatz">
    <target-element type="definition">
    </target-element>
</element>
```

Obiges Suchmuster sucht eine Definition (`target-element type="definition"`), die in einem Absatz (`element type="absatz"`) *direkt* eingebettet ist. Zum Beispiel paßt dazu ein Dokumentausschnitt der folgenden Form:

```
<absatz>
   ...
   <definition>Ein Begriff</definition>
   ...
</absatz>
```

Falls Elemente des Typs `definition` in Absätzen auch tiefer verschachtelt werden dürfen, empfiehlt sich folgendes Suchmuster:

```
<!-- Suchmuster -->
<element type="absatz">
  <any>
    <target-element type="definition">
    </target-element>
  </any>
</element>
```

Dieses Suchmuster würde, im Gegensatz zum ersten Suchmuster, zu den folgenden *beiden* Ausschnitten passen:

```
<absatz> ...
    <em>
        <definition>Ein Begriff</definition> ...
    </em>
</absatz>


<absatz> ...
    <strong>
        <definition>Ein Begriff</definition> ...
    </strong>
</absatz>
```

---

2 Im Gegensatz zu den Patterns der XSL-Note, die XML-Syntax nutzen. Ich halte mich hier an die *Reference Concrete Syntax* von SGML, da sich dadurch die Angabe einer SGML-Deklaration beim Parsing erübrigt.

Das Element em in der ersten Instanz paßt in diesem Fall zu dem Element any im Suchmuster. Gleiches gilt für strong im unteren Teil.

Elemente können in Suchmustern beliebig tief verschachtelt werden. Sollen zusätzlich auch noch Attributwerte von Elementen in der Instanz überprüft werden, so ist dies mit dem leeren Element attribute in einem Suchmuster möglich. Es besitzt ein notwendiges Attribut name, das den Attributtyp bezeichnet, sowie ein optionales Attribut value, das den Wert angibt. Ein Beispiel für ein solches Suchmuster:

```
<!-- Suchmuster -->
<element type="absatz">
  <attribute name="art" value="einleitung">
  <any>
    <target-element type="definition">
    </target-element>
  </any>
</element>
```

Ein dazu passender Ausschnitt aus einem Dokument ist etwa:

```
<absatz art="einleitung"> ...
    <em>
        <definition>Ein Begriff</definition> ...
    </em>
</absatz>
```

Hier werden also nur »einleitende« Absätze betrachtet.

Im Suchmuster ist die Angabe des Attributwerts optional. Folgendes Suchmuster ist deshalb zulässig und paßt zu allen Absätzen, die ein Attribut id besitzen, unabhängig von dessen Wert:

```
<!-- Suchmuster -->
<element type="absatz">
  <attribute name="id">
  <any>
    <target-element type="definition">
    </target-element>
  </any>
</element>
```

Ein passender Dokumentausschnitt:

```
<absatz id="x345f"> ...
    <em>
        <definition>Ein Begriff</definition> ...
    </em>
</absatz>
```

Die unten folgende Tabelle 1 zeigt alle Elementtypen, die in Suchmustern vorkommen, im Überblick.

Suchmuster stellen die interne Repräsentation der Kategorien dar, in denen der Benutzer später suchen kann. Für die Suche wird aus den Suchmustern ein HTML-Formular erzeugt, das den Zugriff über einen Web-Browser gestattet. Zu diesem Zweck muß der Administrator zusätzliche Informationen angeben, die IP4W3 mitteilen, wie

ein Suchmuster in ein Formularelement umzuwandeln ist. Diese Informationen bilden den Inhalt des Elements `target-element`. Im einzelnen handelt es sich um

- ✗ Informationen, welches HTML-Element für ein Suchmuster benutzt werden soll, und
- ✗ eine Beschreibung, die zur Ausgabe auf der Formularseite verwendet wird.

Letzteres ist ganz einfach realisiert: Innerhalb von `target-element` muß als erstes ein Element vom Typ `beschreibung` enthalten sein. Darin ist einfacher Text (PCDATA) enthalten. Etwas komplexer kann das nachfolgende Element `form` ausfallen. Zwei Möglichkeiten stehen dem Administrator zur Verfügung:

- ✗ Er läßt im HTML-Formular eine freie Texteingabe zu, ergänzt um die Option, den Wertebereich des einzugebenden Begriffs zu beschränken.
- ✗ Er legt eine endliche Menge von Werten fest, aus denen der Benutzer einen auswählen kann (Aufzählungstyp).

Für die Texteingabe gibt es den Elementtyp `textinput`. Er besitzt das Attribut `wertebereich`, dessen mögliche Werte `beliebig`, `buchstaben` und `posinteger` sind. Der Vorgabewert ist `beliebig`.

Der Aufzählungstyp ist durch den Elementtyp `select` realisiert, der fast identisch zum gleichnamigen Typ aus HTML ist. Ein `select`-Element nimmt ein oder mehrere Optionen auf (`option`). Der Inhalt dieser Elemente wird als der Wert des Formularfelds angenommen, es sei denn, das Attribut `value` gibt explizit einen anderen Wert an.

Wann immer die Menge der möglichen Begriffe einer bestimmten Kategorie bekannt ist, sollte der Aufzählungstyp verwendet werden, da die browserseitige Benutzerschnittstelle in diesem Fall keine Fehleingaben zuläßt.

Es folgen zwei Beispiele für vollständige Suchmuster, die auch die Beschreibungen und Formularangaben enthalten:

```
<!-- Suchmuster -->
<suchmuster id="def">
  <element type="absatz">
    <any>
      <target-element type="definition">
        <beschreibung>Begriffsdefinitionen</beschreibung>
        <form>
          <textinput wertebereich="beliebig">
        </form>
      </target-element>
    </any>
  </element>
</suchmuster>
```

```
<!-- Suchmuster -->
<suchmuster id="def">
  <element type="absatz">
    <any>
      <target-element type="definition">
        <beschreibung>Begriffsdefinitionen</beschreibung>
        <form>
          <select>
            <option>Intelligenz</option>
            <option>Bewußtsein</option>
            <option>Verstand</option>
            <option
             value="Bewußtsein">consciousness</option>
        </form>
      </target-element>
    </any>
  </element>
</suchmuster>
```

Tabelle 1: Elementtypen von Suchmustern

Elementtyp	Beschreibung	Inhaltsmodell
`suchmuster`	Wurzelelement für ein Suchmuster	Enthält entweder ein `element` oder ein `target-element`.
`element`	Steht für ein Element, dessen Elementtyp über das Attribut `type` angegeben werden kann.	Enthält die Angabe ein oder mehrerer `attribute`, gefolgt von entweder einem `element` oder `any` oder einem `target-element`.
`any`	Steht für ein Element beliebigen Typs.	Wie `element`, aber ohne Kindelemente `attribute`. Die Angabe von Attributen macht keinen Sinn, da sie fast immer an einen bestimmten Elementtyp gebunden sind.
`attribute`	Gibt an, welches Attribut ein Element im Dokument besitzen muß und gegebenenfalls, welchen Wert dieses Attribut besitzen muß.	Leer

Tabelle 1: Elementtypen von Suchmustern

`target-element`	Das Zielelement bei der Suche. Der Begriff, den der Benutzer in das Webformular eingibt, wird in diesem Element gesucht.	Eine `beschreibung` zur Ausgabe auf der Webseite, die Angabe, welches Formularelement (`form`) für das Zielelement generiert werden soll und eine optionale Folge von `attributen`.
`beschreibung`	Textuelle Beschreibung der Kategorie, die durch das Suchmuster repräsentiert wird.	Text (PCDATA)
`form`	Enthält Informationen über die Formulardarstellung des Suchmusters.	Entweder `textinput` oder `select`.
`textinput`	Bewirkt, daß für das Suchmuster ein Texteingabefeld im Formular erzeugt wird. Der `wertebereich` läßt sich über das gleichnamige Attribut steuern.	Leer
`select`	Bewirkt, daß für das Suchmuster ein Auswahlfeld im Formular erzeugt wird.	Ein oder mehrere `optionen`.
`option`	Stellt eine `option` innerhalb eines `select`-Elements dar. Als Wert wird der Inhalt des Elements benutzt, es sei denn, das Attribut `value` überschreibt diesen.	Text (PCDATA)

3.2.2 Atome

Unter dem Begriff *Atom* verstehe ich hier den strukturellen Kontext eines Elements, genauer eines Zielelements aus einem Suchmuster, der hinreichend ist, um alleinstehend verstanden zu werden. Es handelt sich um ein zentrales Konzept dieser Diplomarbeit. Mit ihm ist die Idee verbunden, umfangreiches Textmaterial in kleine Stücke,

die Atome, zu zerlegen, die für den Benutzer eine möglichst optimale Antwort auf seine Anfrage darstellen. In diesem Zusammenhang heißt optimal, daß die Antwort so klein wie möglich, aber so groß wie nötig ausfällt. Wie bereits zuvor erwähnt (vgl. 3.1), werden in der vorliegenden Implementation von IP4W3 Atome direkt aus den Suchmustern abgeleitet.

3.2.3 DSSSL-Formatierungsrahmen und -rumpf

Die von IP4W3 gefundenen Textstellen werden mit Hilfe des DSSSL-Formatierers *Jade* in HTML transformiert. Zu diesem Zweck muß der Administrator (oder der Autor) ein DSSSL-Stylesheet (hier *DSSSL-Rumpf* genannt) für jede in IP4W3 benutzte DTD zur Verfügung stellen. Dieses Stylesheet führt die Formatierung durch. Ein Bestandteil von IP4W3, der *DSSSL-Rahmen*, sorgt dafür, daß genau die Ausschnitte formatiert werden, die der Benutzer sehen möchte. In Kurzfassung: Der Rahmen führt den Transformationsprozeß durch, der Rumpf den Formatierungsprozeß.

3.2.4 Stichwortindex

Zu jedem in IP4W3 integrierten Dokument gibt es einen Stichwortindex, der dazu verwendet wird, die Wortsuche durchzuführen. Die Suche liefert eine Liste von Elementen zurück, die zu der Suchanfrage passen. Elemente werden über eine interne ID eindeutig bezeichnet. Der Administrator kommt bei der normalen Benutzung des Systems nicht mit dem Stichwortindex in Berührung.

3.2.5 Formularmasken für die Benutzereingabe

Wie bereits im Abschnitt über Suchmuster erwähnt, erzeugt ein Formulargenerator aus den Informationen, die in den Suchmustern enthalten sind, automatisch ein HTML-Formular. Das Formular selbst muß vom Administrator nicht verändert werden. Jedoch ist es üblicherweise sinnvoll, die HTML-Seite, in der das Formular enthalten ist, manuell zu erstellen, um etwa Farben, Schriften und erläuternden Text zu ergänzen. IP4W3 erzeugt nur das Formular ohne Verzierungen.

Die generierten HTML-Seiten enthalten JavaScript-Anweisungen, die die Eingabe des Benutzers auf offensichtliche Fehler prüfen. Um gefundene Dokumentausschnitte client-seitig zu speichern, verwendet IP4W3 JavaScript und Frames. Die Anforderungen an den Browser umfassen daher JavaScript 1.1 und HTML 4 (darin sind Frames und JavaScript-Attribute definiert). Falls der Browser diesen Anforderungen nicht genügt, ist die Suche in IP4W3 aber uneingeschränkt

möglich. Auf das Merken von Dokumentausschnitten und Eingabeüberprüfung muß der Benutzer dann aber verzichten.

3.2.6 CGI–Skripte

IP4W3 arbeitet mit jedem beliebigen WWW-Server zusammen, der das Common Gateway Interface (CGI) unterstützt. Die Schnittstelle zwischen Web-Server und den Programmen von IP4W3 (Stichwortsuche, Formatierer usw.) bilden zwei CGI-Skripte. CGI wurde von den Entwicklern des CERN- und des NCSA-Servers entwickelt [NCSA98], um Programme, die von einem Web-Server aufgerufen werden, zwischen den beiden Servertypen austauschen zu können. Es handelt sich dabei um eine Konvention, wie Daten, die z.B. in einem Formular eingegeben werden, vom Browser an den Server und von diesem an ein Programm übergeben werden. Heute beherrschen alle bekannten Web-Server das Common Gateway Interface.

Bei den beiden CGI-Skripten von IP4W3 handelt es sich um ein Skript, das die Stichwortsuche durchführt und die Trefferliste ausgibt und eines, das die Formatierung erledigt. Beide Skripte rufen andere Programme von IP4W3 auf, die den »schwierigen« Teil der Arbeit übernehmen.

Anwendung am Beispiel der XML-Spezifikation

4

Dieses Kapitel beschreibt eine getestete Anwendung, um die praktische Einsetzbarkeit von IP4W3 zu veranschaulichen und zu belegen. Meine Absicht ist es, hier einen Eindruck von dem laufenden Programmsystem zu geben, so weit das in gedruckter Form möglich ist.

Als Anwendungsfall verwende ich hier nicht das in der Einführung genannte und in Kapitel 6 detailliert beschriebene Material über Makroökonomie. Der Grund dafür ist, daß das Material zum gegenwärtigen Zeitpunkt noch in einem mangelhaften Zustand ist. In Kapitel 6 führe ich diesen Punkt etwas genauer aus. Obwohl die Qualität der zur Verfügung gestellten Texte nicht sehr gut ist, hinterlassen die Testläufe mit IP4W3 einen positiven Eindruck. Sichtbare Schwächen waren direkt auf in der Instanz fehlende Auszeichnungen zurückzuführen, so daß sich diese Anwendung eignet, um Erfahrungen über benötigte Auszeichnungen zu erlangen.

Im folgenden beschreibe ich die Anwendung von IP4W3 auf die deutsche Übersetzung der XML-Spezifikation [W3C98J]. Dieses Dokument ist aus mehreren Gründen besonders interessant:

- ✗ Der Inhalt besteht u.a. aus Syntaxregeln für XML, die über den Text verstreut sind und in allen verfügbaren Fassungen (dies gilt auch für den englischen Originaltext) nicht adäquat zu durchsuchen oder zu extrahieren sind. IP4W3 ermöglicht das erstmals.
- ✗ Im Text sind viele Begriffsdefinitionen enthalten, die nicht durch eine besondere visuelle Umsetzung hervorgehoben werden. Sie sind aber in der Datei als »Definition« ausgezeichnet.
- ✗ Die übersetzte Spezifikation ist gemäß einer DTD ausgezeichnet, die keine besondere Unterstützung für die im IP4W3-System realisierten Funktionen mitbringt. In diesem Sinne handelt es sich also um einen Testfall, der dem Einsatz »unter realen Bedingungen« nahe kommt. Daß die DTD sehr allgemein gehalten ist, erschwert die Aufgabe für IP4W3 zusätzlich. Es handelt sich zwar nicht um den »worst case«, aber Instanzen und DTDs sollten doch besser sein als dieser Testfall.

Die folgenden Beschreibungen folgen den Schritten, die zur Integration der XML-Spezifikation in IP4W3 notwendig waren. Im Überblick waren dies:

Schritt 1: Initialisierung

Hier wird IP4W3 darüber informiert, daß ein neues Dokument aufgenommen werden soll. Das System legt dafür neue Verzeichnisse an.

Schritt 2: Das Dokument

Das Dokument (nebst DTD) und der DSSSL-Rahmen (zur Formatierung) müssen vom Administrator gestellt werden.

Schritt 3: Erstellen der Suchmuster

IP4W3 braucht Suchmuster für das neue Dokument.

Schritt 4: Aufnehmen des neuen Dokuments

Mit den letzten fehlenden Informationen vom Administrator bereitet das System die Instanz, den Suchindex, die Suchmuster und die HTML-Seite auf.

Schritt 5: Modifikation der generierten HTML-Datei

Bei Bedarf modifiziert der Administrator die von IP4W3 erzeugte HTML-Seite.

4.1 Schritt 1: Initialisierung

Die Initialisierung dient dazu, die Verzeichnisstruktur für das neue Dokument anzulegen, damit für die nächsten Schritte definierte Verhältnisse vorliegen. Unterstützt wird der Administrator dabei von dem Programm `ip4w3-manager`, das ihm nicht nur die Arbeit erleichtern soll, sondern ihn auch anleiten soll (siehe Abbildung 6). Bei der hervorgehobenen Textstelle handelt es sich um die Antwort, die der Administrator auf die Frage von `ip4w3-manager` eingegeben hat. Das Verzeichnis `/home/kimo-d/mintert/DA/web/dokumente/ip4w3` ist in diesem Fall das Installationsverzeichnis von IP4W3. Dokumente befinden sich im Unterverzeichnis `/dokumente`.

Abbildung 6: Die Initialisierung eines neuen Dokuments

In diesem Schritt ist als einzige Frage zu beantworten, welchen Dokumentbezeichner der Administrator für das neue Dokument vergeben möchte (hier »xmlspec«). Er dient als *eindeutige* Bezeichnung innerhalb von IP4W3.

4.2 Schritt 2: Das Dokument

Das Dokument für die deutsche XML-Spezifikation ist gemäß einer DTD für die Buchherstellung ausgezeichnet (vgl. [BEMI98]). Aufgrund des Umfangs der DTD werde ich sie hier nicht vorstellen. Sie ist auch nicht Teil meiner Diplomarbeit, sondern wurde (genau wie die Instanz) schon vor der Arbeit an IP4W3 von mir geschrieben. Gleiches gilt für das DSSSL-Stylesheet, das ich zur Umwandlung von Instanzen dieser DTD in HTML geschrieben habe. Es kommt in IP4W3 als DSSSL-Rumpf zum Einsatz. Der in HTML gewandelte Ausgangstext kann im Web unter der Adresse `http://www.mintert.com/xml/trans/REC-xml-19980210-de.html` eingesehen werden.

4.3 Schritt 3: Erstellen der Suchmuster

Die Definition der drei verwendeten Suchmuster besitzt in diesem Fall folgende Gestalt. Zur besseren Lesbarkeit habe ich die Datei unterteilt, um jedes Suchmuster einzeln beschreiben zu können.

```
<!DOCTYPE ip4w3project
          PUBLIC "-//mintert.com//DTD IP4W3 Project 1.0//DE">
<ip4w3project>
  <suchmuster id="def">
    <element type="absatz">
      <any>
        <target-element type="def">
```

```
          <beschreibung>Begriffsdefinitionen</beschreibung>
          <form>
            <textinput>
          </form>
        </target-element>
      </any>
    </element>
  </suchmuster>
```

Das Muster mit der ID »def« paßt zu allen Zielelementen des Typs `def`, die beliebig tief in einem `absatz` eingebettet sind. Für Definitionen im Fließtext wird also der umgebende Absatz als Atom benutzt.

```
<suchmuster id="keys">
  <element type="absatz">
    <any>
      <target-element type="kbd">
        <beschreibung>Schlüsselworte im
                  Fließtext</beschreibung>
        <form>
          <textinput>
        </form>
      </target-element>
    </any>
  </element>
</suchmuster>
```

Das Muster mit der ID »keys« paßt zu allen Zielelementen des Typs `kbd`, die beliebig tief in einem `absatz` eingebettet sind. Ein komplexeres Suchmuster ist das folgende:

```
<suchmuster id="prod">
  <element type="table">
    <attribute name="class" value="spec">
    <any>
      <element type="table">
        <attribute name="class" value="prod">
        <any>
          <target-element type="td">
            <beschreibung>XML-Syntaxregeln</beschreibung>
            <form>
              <textinput>
            </form>
          </target-element>
        </any>
      </element>
    </any>
  </element>
</suchmuster>
</ip4w3project>
```

Das Muster mit der ID »prod« paßt zu allen »Produktionsregeln der XML-Syntax«. Diese befinden sich innerhalb des Textes in verschachtelten Tabellen der unten dargestellten Form. Das Suchmuster berücksichtig die Tabellenstruktur.

Im folgenden Auszug aus dem Dokument korrespondieren die hervorgehobenen Stellen mit den entsprechenden Stellen im obigen Suchmuster. Daran ist zu erkennen, daß das Suchmuster den Do-

kumentausschnitt finden würde (wenn nach einem der Begriffe `document`, `prolog`, `element` oder `Misc` gesucht würde).

```
<table class="spec" rules="groups">
  <tr>
    <th>Dokument</th>
  </tr>
  <tr>
   <td><table cols="4" class="prod">
    <tr>
          <td>[1]</td>
          <td>document</td>
          <td>::=</td>
          <td>prolog element Misc*</td>
    </tr>
    </table>
   </td>
  </tr>
</table>
```

4.4 Schritt 4: Aufnehmen des neuen Dokuments

Nachdem alle Dateien vom Administrator in das Dokumentverzeichnis kopiert wurden, liegt folgende Situation vor: Die Datei `AWLINET.DTD` enthält die DTD und die Datei `awl2w3c.dsl` das zugehörige DSSSL-Stylesheet für die Transformation in HTML. Beides war bereits vorhanden und mußte für diese Anwendung nicht neu geschrieben werden. Bei der Suchmusterdatei `spec-suchmuster.sgml` handelt es sich um die im vorhergehenden Abschnitt beschriebene Datei. Die Instanz `xmlspec.sgml` ist das Ausgangsdokument[1].

Zur Aufnahme muß noch einmal `ip4w3-manager` aufgerufen werden `ip4w3-manager neu` (siehe Abbildung 7). Bei den hervorgehobenen Textstellen handelt es sich wieder um die Antworten, die der Administrator auf die Fragen von `ip4w3-manager` eingegeben hat. Abgesehen von einem Dokumenttitel muß er lediglich die Namen der zuvor genannten Dateien eingeben.

1 Tatsächlich ist es ein Kapitel aus dem Buch [BEMI98].

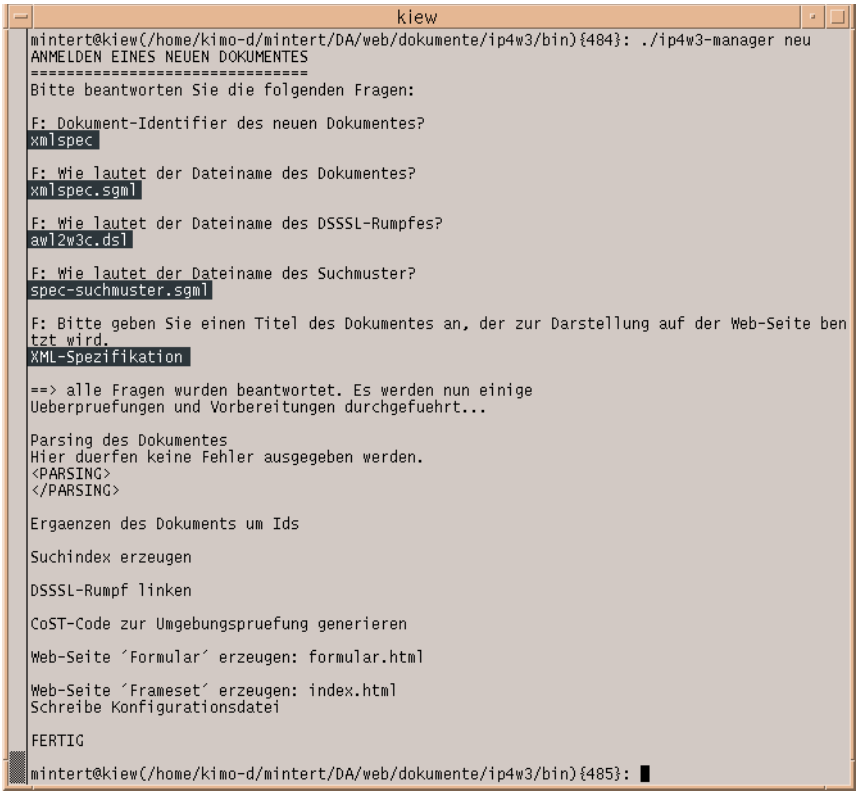

Abbildung 7: Die Anmeldung eines neuen Dokuments

4.5 Schritt 5: Modifikation der generierten HTML-Datei

Bereits der im letzten Schritt erreichte Zustand ist voll funktionsfähig. Im Dokumentverzeichnis befinden sich nun zwei Dateien: `index.html` und `formular.html`. Es ist gängige Praxis, daß die Vorgabeseite für ein Verzeichnis `index.html` heißt. Wenn man nun mit einem Web-Browser auf das Dokument zugreift, so wird diese Seite ausgeliefert. Es handelt sich um den folgenden Frameset. Die Ansicht im Web-Browser zeigt Abbildung 8.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
  <head>
    <title>XML-Spezifikation</title>
  </head>
  <frameset rows="*,10%">
    <frame name="hauptrahmen"
           src="formular.html">
    <frame name="steuerrahmen"
           src="/ip4w3/lib/html/steuerung.html?xmlspec">
  </frameset>
</html>
```

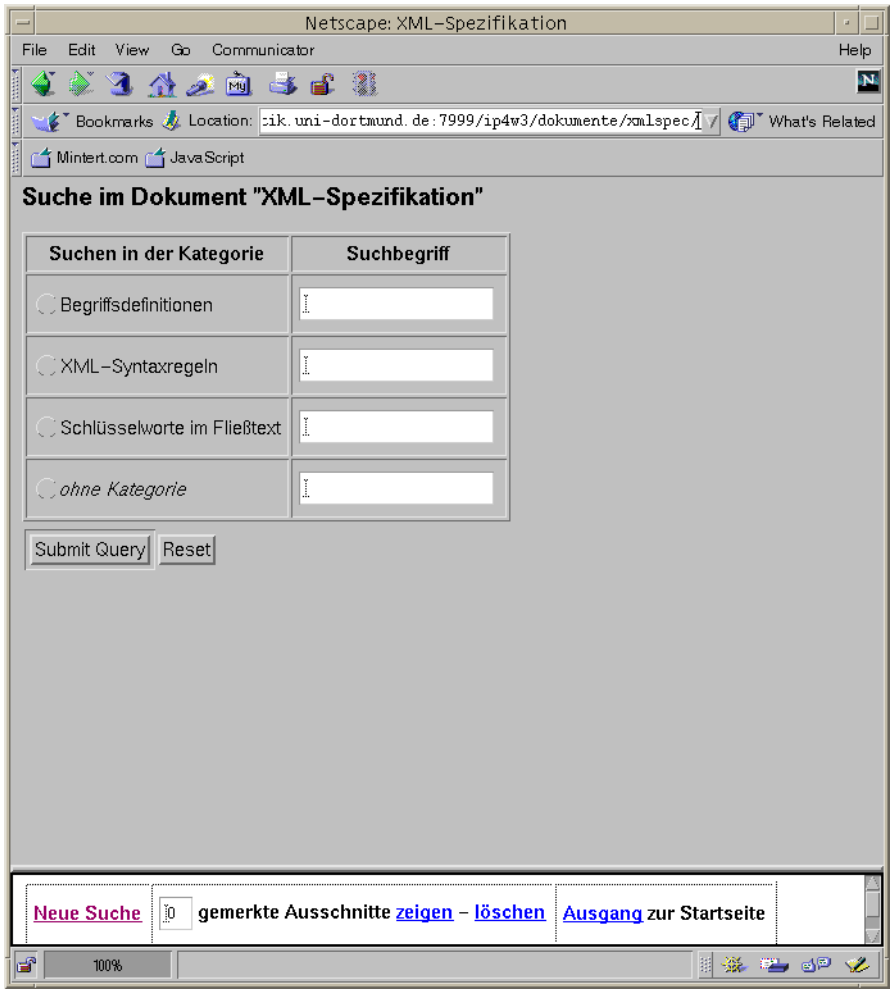

Abbildung 8: Die von IP4W3 erzeugte Webseite: etwas schmucklos, aber voll funktionsfähig. In der Adress-Zeile ist zu erkennen, daß der vom Administrator gewählte Dokumentbezeichner (»xmlspec«) in den URL eingeflossen ist.

Der obere Frame enthält nur ein HTML-Fragment, nämlich das »nackte« Suchformular[2]. Diese Datei sollte in eine HTML-Datei integriert werden, so daß Einstellungen wie Farben und Schriftarten manuell vorgenommen werden können. Für das Anwendungsbeispiel der XML-Spezifikation habe ich eine weitere HTML-Datei geschrieben, die das Formular einbindet (siehe Abbildung 9).

2 Formal ist es ein Element des Typs `form` gemäß HTML-DTD.

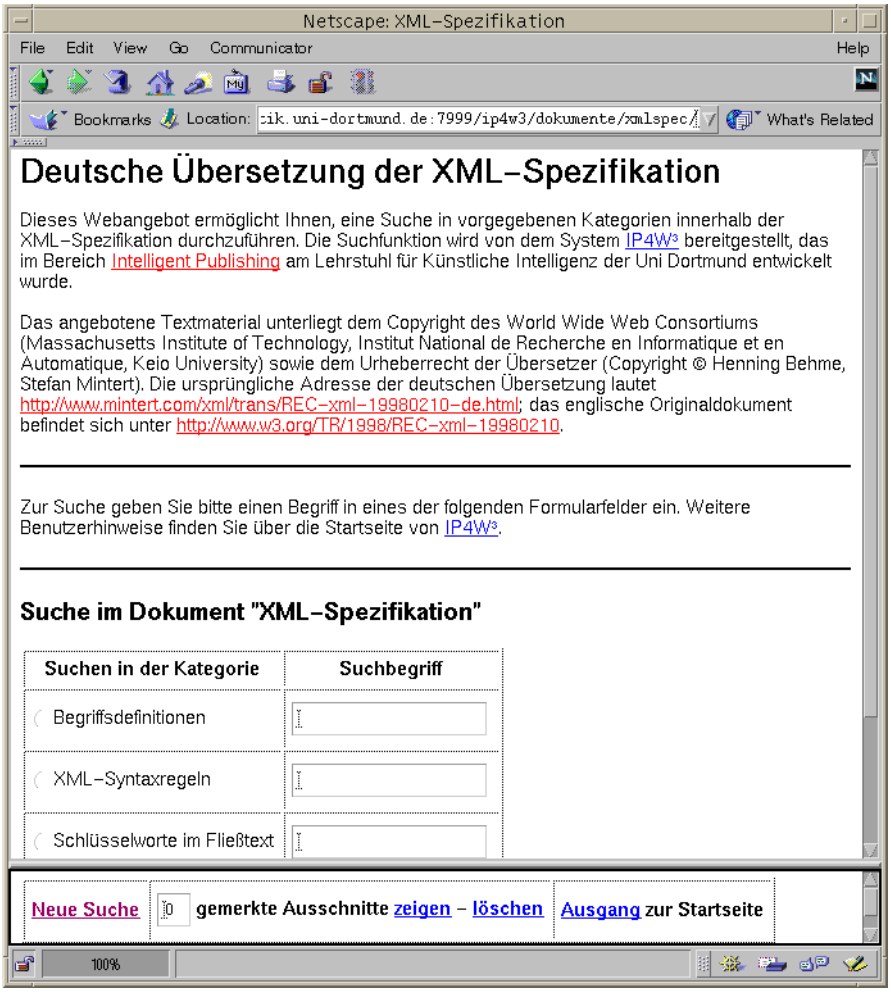

Abbildung 9: Das Suchformular nach der Modifikation

Der untere Frame enthält die Steuerung, die für alle Dokumente in IP4W3 identisch ist. Über einen Parameter erhält sie die Dokumenten-ID (das ist in obigem Listing zu sehen: `?xmlspec`), um die Links korrekt zu erzeugen. Die Funktionsweise der Steuerung beschreibe ich im folgenden Abschnitt.

Für den Administrator ist damit die Arbeit erledigt. Er mußte lediglich die Suchmuster erstellen (falls sie für die DTD nicht schon vorhanden waren) und die Webseite um etwas Text ergänzen. Ich denke, daß das ursprünglich gesetzte Ziel der einfachen Handhabung damit erreicht wurde.

4.6 Anwendung aus Benutzersicht

Bei der Betrachtung der Benutzersicht muß man davon ausgehen, daß der Benutzer mit der Thematik des jeweiligen Textes vertraut ist. Andernfalls würde er sich wohl kaum für den Text interessieren.

Um die folgenden Ausführungen nachvollziehen und bewerten zu können, muß auch beim Lesen dieser Arbeit das Vorwissen des Benutzers berücksichtigt werden. Da es sich bei der beschriebenen Anwendung um die XML-Spezifikation handelt, hoffe ich, daß die anfänglichen Ausführungen zu SGML/XML den nötigen Hintergrund geschaffen haben.

Ein Beispiellauf

Zuerst wird nach der Definition des Begriffs »Zeichen« (vgl. Abbildung 10) gesucht. Die Klein/Großschreibung wird dabei nicht berücksichtigt. Sobald das Eingabefeld aktiviert wird, wird automatisch der Schalter für die zugehörige Kategorie »Begriffsdefinition« ausgewählt. Diese Kategorie korrespondiert mit dem in 4.3 gezeigten Suchmuster »def«.

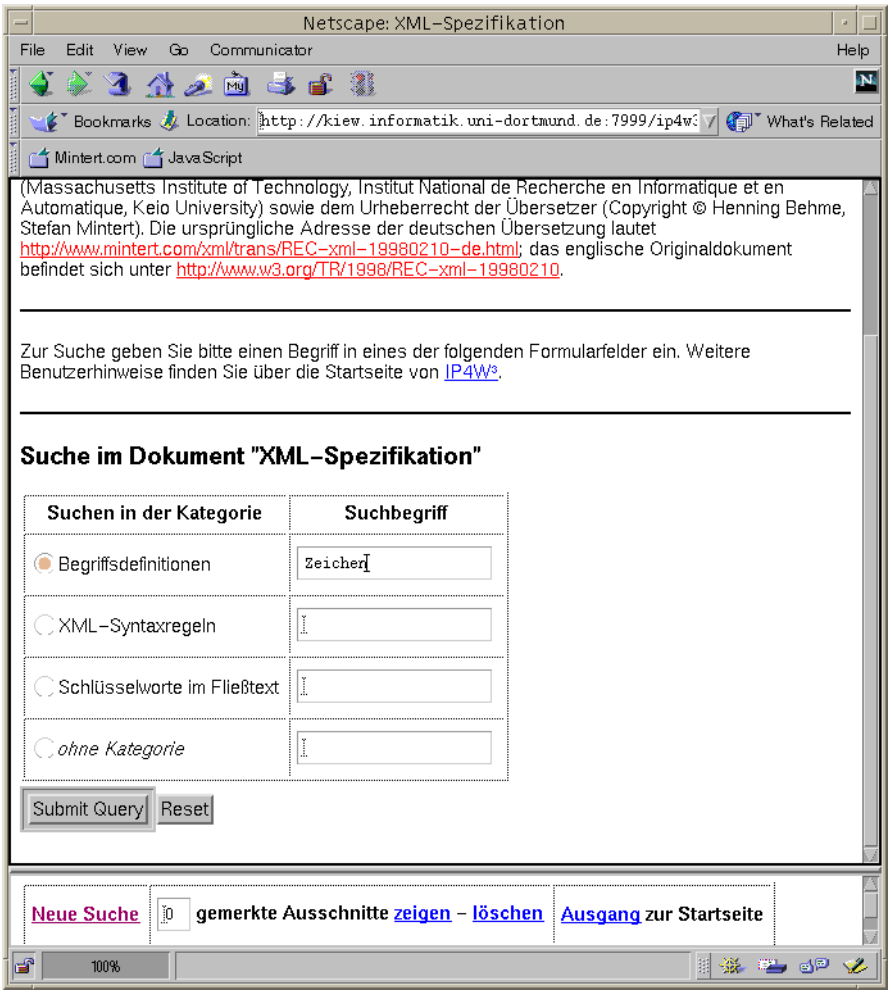

Abbildung 10: Eingabe eines Suchbegriffs

In den »Begriffsdefinitionen« findet IP4W3 das Wort »Zeichen« dreimal (vgl. Abbildung 11). Der Benutzer kann nun einzelne Treffer auswählen oder mit dem Schalter »Alle auswählen« alle drei Aus-

schnitte markieren. In diesem Fall erhält er die formatierte Ansicht in Abbildung 12.

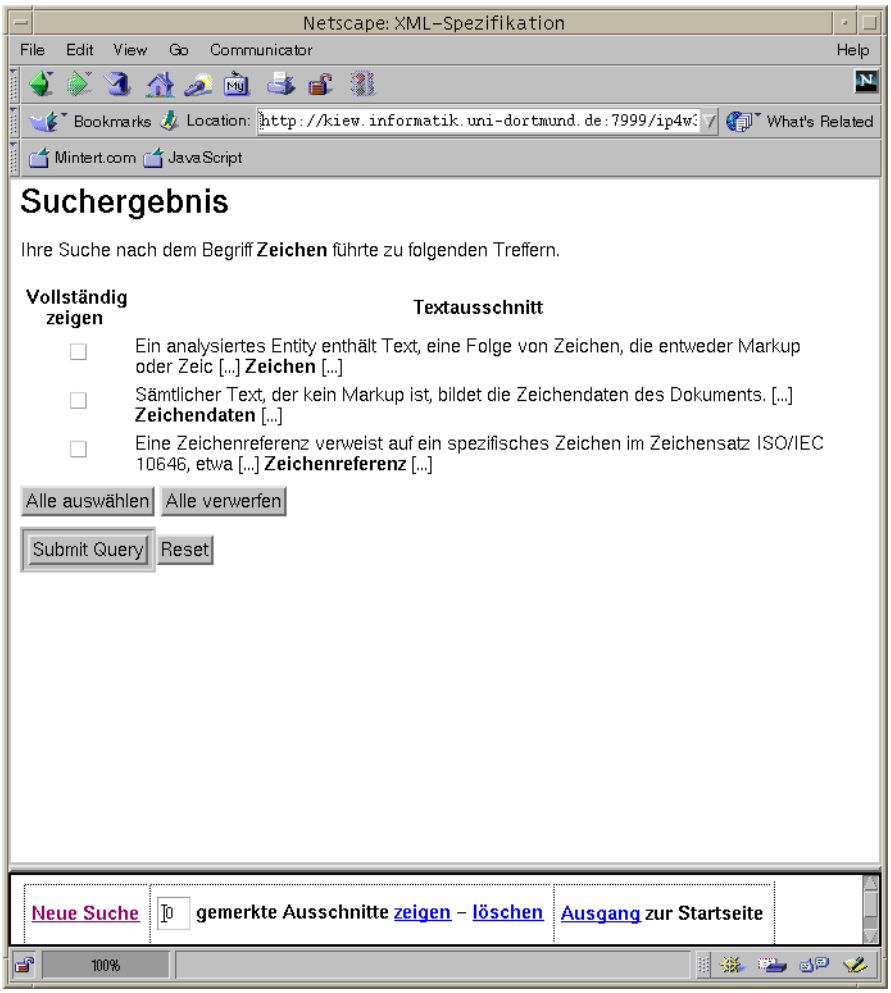

Abbildung 11: Die HTML-Trefferliste der Suche

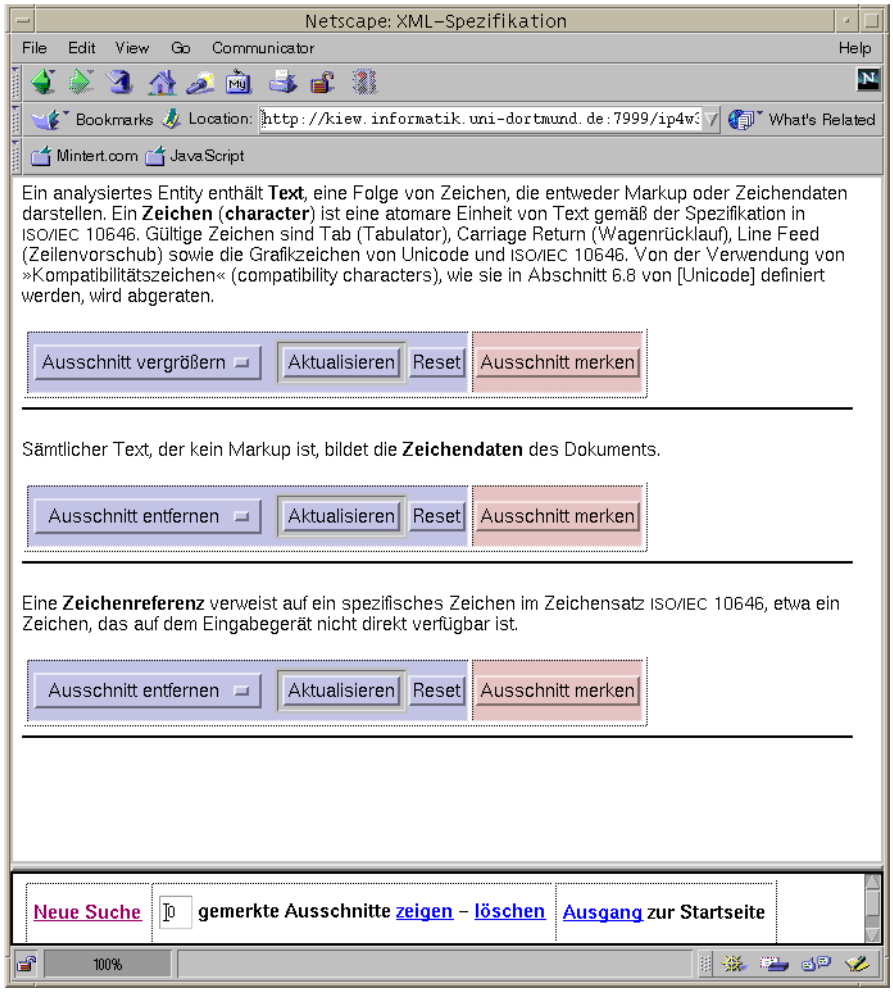

Abbildung 12: Die formatierte Ansicht dreier Atome

Jedes angezeigte Atom kann nun einzeln behandelt werden. Der Benutzer kann den Ausschnitt vergrößern, unverändert behalten oder verwerfen. Beim Vergrößern liefert IP4W3 das Vater-Element zum gerade sichtbaren Element als Ergebnis. In diesem Beispiel wird der erste Ausschnitt vergrößert und die beiden anderen verworfen. Das Ergebnis ist in Abbildung 13 zu sehen. Es handelt sich dabei um den Abschnitt 2.2 aus der XML-Spezifikation, der neben der umgangssprachlichen Definition von »Zeichen« auch den formalen Zeichenbereich in XML definiert. Dieses Element wird nun durch Anklicken von »Ausschnitt merken« in den virtuellen Warenkorb gelegt. Hier tritt erstmals der Steuerrahmen am unteren Bildrand in Aktion: Der Zähler für die gemerkten Atome wurde um eins erhöht.

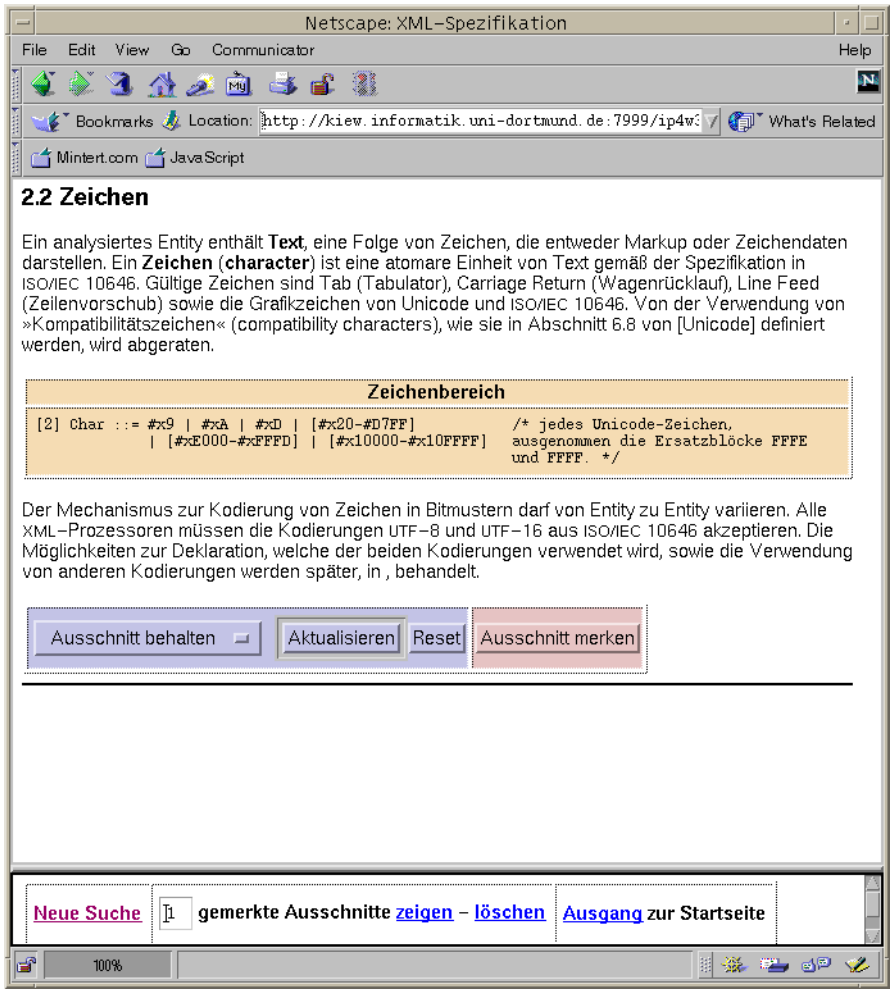

Abbildung 13: Das gewünschte Ergebnis der Suche

Durch eine zweite Suche nach dem Begriff »attlist« in der Kategorie »XML-Syntaxregeln« (Suchmuster »prod« in Abschnitt 4.3) erhält der Benutzer alle Stellen der XML-Syntax, die sich mit Attributlisten befassen (vgl. Abbildung 14). Als gesuchten Ausschnitt merkt er sich hier den zweiten Treffer.

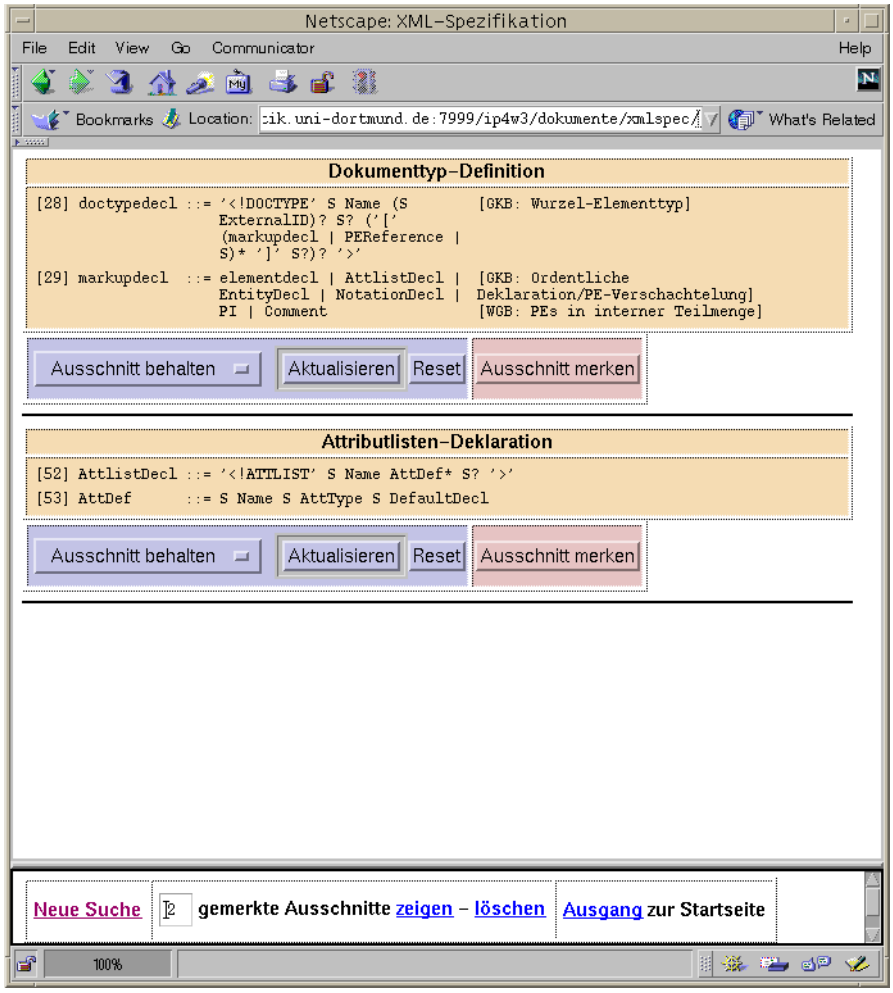

Abbildung 14: Das Ergebnis der zweiten Suche nach »attlist« in der Kategorie »XML-Syntaxregeln«

Nun befinden sich im Warenkorb zwei Atome unterschiedlicher Kategorien, die das Ergebnis von zwei unabhängigen Suchvorgängen sind. Um sie zusammen anzeigen zu lassen, genügt ein Klick auf den Link »gemerkte Ausschnitte zeigen« im Steuerrahmen. Der Browser zeigt anschließend die Ansicht in Abbildung 15. Hier ist zu bemerken, daß bei der Formatierung der gemerkten Ausschnitte die Schalter zur Ausschnittwahl unterdrückt werden, um das »fertige« Dokument drucken zu können. Die Ausschnitte behalten ihre Reihenfolge aus dem Ursprungsdokument, unabhängig davon, in welcher Reihenfolge sie gefunden wurden.

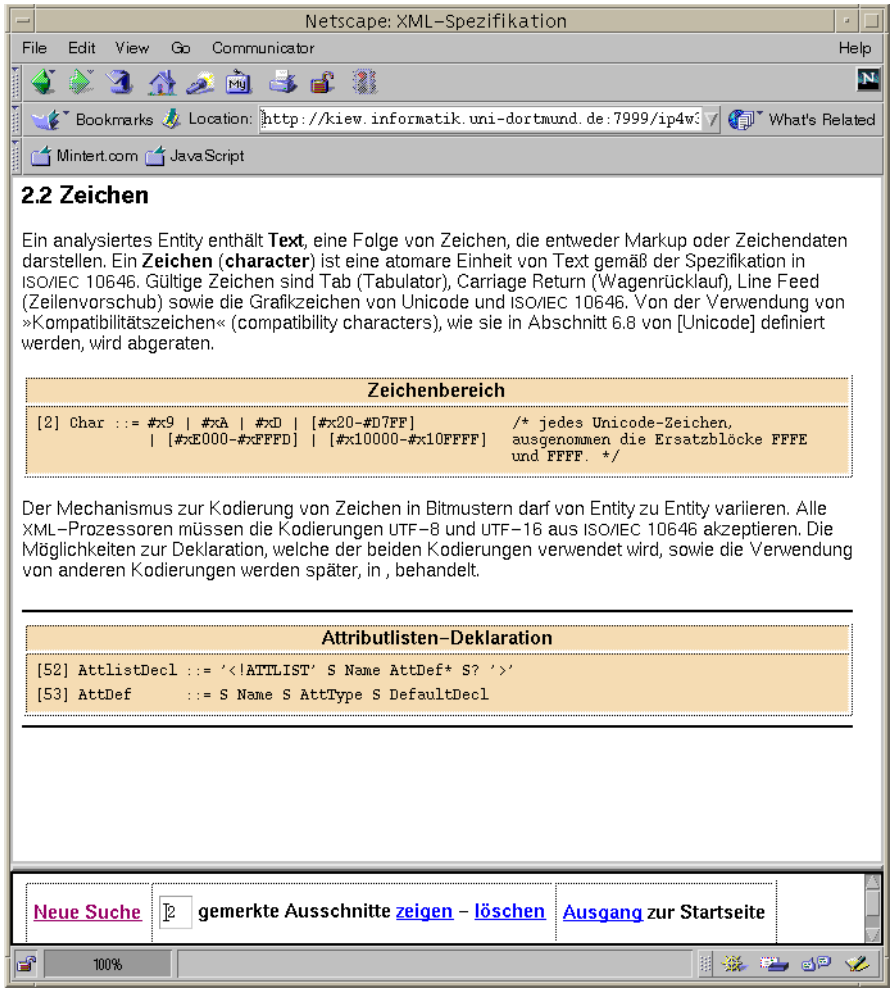

Abbildung 15: Atome unterschiedlicher Kategorien, die das Ergebnis von zwei unabhängigen Suchen sind

Implementation

5

Dieses Kapitel beschreibt, *was* ich *wie* und *warum* gemacht habe. Die Begründung meiner Entscheidungen betrifft insbesondere den Entwurf der Suchmuster, da diese ein zentrales Konzept meiner Arbeit darstellen. Die weiteren Ausführungen beschreiben die Implementation der Systemkomponenten.

Am Ende des Kapitels ist in Abschnitt 5.5 eine Übersicht der Programmabläufe zu finden. Ich habe diesen Abschnitt an's Ende gestellt, da für sein Verständnis die Kenntnis der einzelnen Komponenten notwendig ist. Um einen ersten Eindruck der Abläufe zu bekommen, kann es aber auch sinnvoll sein, diesen Abschnitt zuerst zu lesen. Die Entscheidung sei dem Leser überlassen.

5.1 Entwurf der Suchmuster

Bei der Frage, welche Möglichkeiten IP4W3 für die Formulierung der Suchmuster zulassen soll, habe ich mich an den Möglichkeiten orientiert, die Formatierungssprachen im SGML/XML-Umfeld bieten. Im folgenden begründe ich diese Entscheidung und beschreibe die resultierenden Suchmuster.

Die Aufgabe, die ein Suchmuster erfüllen muß, besteht darin, ein Element einer Dokumentinstanz in Abhängigkeit seiner strukturellen Umgebung zu identifizieren. Obwohl es darum geht, Elemente einer Instanz zu benennen, darf sich die Formulierung nur auf die DTD abstützen. Andernfalls müßten Suchmuster für jedes Dokument maßgeschneidert werden. Zwar kann es sinnvoll sein, dies zu tun[1], der Regelfall ist jedoch, sich auf die DTD zu beziehen.

Es liegt hier eine Situation vor, die in gleicher Weise bei der Formatierung von Dokumenten auftritt. Formatierungssprachen müssen in der Lage sein, Elemente in Abhängigkeit ihrer Umgebung zu behandeln. Beispielsweise hängt die Darstellung eines Elements

1 Dies gilt insbesondere für Mehrzweck-DTDs im Sinne von [BEMI98], da solche DTDs nur wenige Informationen bieten, die für IP4W3 von Nutzen sind.

»Überschrift« davon ab, ob es sich innerhalb eines Kapitels, eines Abschnitts usw. befindet. Abstrakter kann man sowohl bei IP4W3 als auch bei Formatierungssprachen davon sprechen, daß bestimmte Elemente (»was«) in bestimmter Weise (»wie«) behandelt werden. Das »wie« weicht in den beiden Fällen voneinander ab (Formatieren versus Durchsuchen), aber das »was« ist gleich (welches Element in welcher Umgebung?). Es ist also angebracht auch zu untersuchen, welche Mittel aus dem Anwendungsfeld der Formatierungssprachen für IP4W3 geeignet sind.

5.1.1 Elementidentifikation in anderen Systemen

Für den Entwurf der Suchmuster habe ich die (zum Teil noch im Entwurfsstadium befindlichen) Standards DSSSL [ISO96B], HyTime [ISO92], XPointer [W3C98F] und XSL [W3C97E] betrachtet.

Elementidentifikation in DSSSL [ISO96B]

Elemente können in DSSSL nur über den Namen ihres Elementtyps (`gi`) oder zusätzlich unter Berücksichtigung ihrer Vorgängerelemente (`qualified-gi`) angesprochen werden. Bei den Vorgängern muß es sich jedoch stets um eine Folge von *direkten* Vorgängern handeln (Eltern-Kind-Bezug). Es ist nicht möglich, in einem `qualified-gi` eine Hierarchiestufe auszulassen, um etwa Großeltern-Enkel-Bezüge zwischen Elementen auszudrücken[2]. Kompliziertere Elementstrukturen und selbst die Berücksichtigung von Attributen und ihren Werten müssen in DSSSL durch Abfragen mit der sehr mächtigen, aber komplizierten *Standard Document Query Language* (*SDQL*) behandelt werden. Dies wäre jedoch für den Administrator von IP4W3 ein zu großer Aufwand.

Ein Beispiel für einen `qualified-gi`, der ein Element `definition` innerhalb eines Elements `absatz` anspricht, ist:

```
absatz definition
```

Hier ist also `definition` ein Kind von `absatz`. Sollte `definition` ein Enkel sein, so ließe sich dies nicht mit einem `qualified-gi` ausdrükken.

Elementidentifikation in XPointer [W3C98F]

XPointer sind Bestandteil der neuen Hyperlink-Entwicklung des W3C und nicht für die Formatierung entwickelt worden. Vielmehr stellen sie universelle Zugriffsmöglichkeiten auf gesuchte Doku-

2 Formal gilt: Die Elementidentifikation erfolgt in DSSSL innerhalb der Klausel [168] `element-construction-rule`. Ein Element kann dabei entweder als [169] `gi` (Generic Identifier) oder als [170] `qualified-gi`, der aus einer Folge von `gi` besteht, identifiziert werden. Eine `element-construction-rule` bietet damit nur sehr schwache deklarative Ausdrucksmöglichkeiten.

mentstellen zur Verfügung. Es ist deshalb nicht überraschend, daß sie mehr Funktionen bieten als etwa bei der Formatierung benötigt werden.

Bisher gibt es keine (frei verfügbaren) Programme, die die Verarbeitung oder Erstellung von XPointer-Ausdrücken unterstützen. Selbst wenn es sie gäbe, müßte die Verwendung von XPointern in IP4W3 eingeschränkt werden, da ihre Mächtigkeit die Bedürfnisse in diesem Anwendungsfall übersteigt. Eine Implementation in IP4W3 wäre also nur eine Teilimplementation, was zwar kein zwingendes Gegenargument ist, was die Entscheidung für XPointer aber auch nicht begünstigt. Der Funktionsumfang der XPointer ist für IP4W3 ausreichend.

Das obige Beispiel hier noch einmal in XPointer-Notation:

```
child(all,absatz).child(all,definition)
```

Eine kleine Schwäche dieses Ausdrucks ist, daß das Element `absatz` hier als Kind aufgefaßt wird. Zwar wird es in der Praxis ein Kind irgendeines Elements sein, jedoch ist dieser Ausdruck nicht äquivalent zu vorangegangenem DSSSL-Konstrukt. Besser wäre eine Ausdrucksmöglichkeit der folgenden Form:

```
element(absatz).child(all,definition)
```

XPointer bieten dafür jedoch kein Schlüsselwort an[3], da ein XPointer stets von einem bestimmten Quellelement (in der Regel die Wurzel) ausgeht und von dort eine relative Navigation erlaubt.

Zusammenfassend kann man XPointer als geeignet, jedoch als nicht optimal bezeichnen.

Elementidentifikation in HyTime [ISO92]

HyTime ist ein auf SGML basierender Standard für Hypertext und Hypermedia. In HyTime lassen sich Dokumentstellen mit einem sogenannten *Locator* adressieren. Locator werden in SGML-Syntax ausgedrückt[4]. Für den Einsatz in IP4W3 sind HyTime-Locator nicht unmittelbar geeignet, da sie primär dazu dienen, Elemente in einer *konkreten* Instanz zu adressieren. Beispielsweise lassen sich Elemente nur unter Angabe einer ID (`nameloc`-Form) oder über ihre Position im Elementbaum (`treeloc`-Form) adressieren. Auch die `proploc`-Form hilft nicht weiter, da man mit ihr zwar Zugriff auf Eigenschaften (properties), wie zum Beispiel den Generic Identifier, eines Objekts hat, jedoch muß das Objekt schon gegeben sein. Was in IP4W3 benötigt wird, ist folgendes: Elemente auf Basis der DTD, also für *beliebige* Instanzen dieser DTD ansprechen zu können. IDs oder Posi-

3 Der Ausdruck `element(absatz)` ist Fiktion.

4 HyTime definiert bestimmte Elementnamen, die auch mit einer festgelegten Semantik belegt sind. Über ein von HyTime eingeführtes Konzept namens »Architectural Forms« werden diese Namen auf die Elementnamen abgebildet, die in einer beliebigen DTD verwendet werden.

tionen können in IP4W3 bei der Elementidentifikation nicht berücksichtigt werden, da sie nicht bekannt sind.

Um mit HyTime die gewünschten Ausdrucksmöglichkeiten zu erhalten, muß die *HyTime Query Language* (*HyQ*) bemüht werden. Das obige Absatz-Definition-Beispiel sieht dann so aus:

```
<HyQ>
  Select(DOMTREE
        and(Eq(Proploc(CAND GI) "absatz")
           Select(Relloc(Treeloc(DOMROOT 1) DOMROOT CHILDREN )
                 Eq(Proploc(CAND GI) "definition"))
        )
  )
</HyQ>
```

Bei Verwendung von HyQ bleibt nicht mehr viel von der SGML-Syntax übrig, so daß dieser Vorteil nicht mehr besteht. Des weiteren gilt auch hier, noch mehr als bei XPointer, daß HyQ so mächtig ist, daß für den Einsatz in IP4W3 nur eine Teilmenge sinnvoll wäre. Der Standard müßte also eingeschränkt werden; ich halte es jedoch für erstrebenswerter, eine »schlankere« Syntax vollständig zu implementieren als eine mächtige Sprache nur zum Teil.

Elementidentifikation in XSL Note [W3C97E]

An dieser Stelle unterscheide ich zwischen dem aktuellen Zustand von XSL und dem ursprünglichen Entwurf [W3C97E], der nur den Status einer »Note« bekommen hat. Gemein ist beiden, daß als zugrundeliegende Syntax XML zum Einsatz kommt. Es gibt jedoch Unterschiede, wie Elemente identifiziert werden. Da ich meine Arbeit begonnen habe als die »Note« aktuell war, beziehe ich mich im folgenden darauf.

XSL erlaubt, wie DSSSL, die Elementidentifikation über den Namen des Elementtyps (in sogenannten *Patterns*). Verschachtelte Elemente lassen sich ebenfalls formulieren. Vorgängerrelationen können in XSL, im Gegensatz zu DSSSL, auch Hierarchiestufen überspringen, um z.B. Großeltern-Enkel-Bezüge zwischen Elementen zu beschreiben. Darüber hinaus kann XSL ohne großen Aufwand die Existenz von Attributen und ihren Werten überprüfen. Aufgrund der Tatsache, daß die Syntax XML ist, lassen sich die Eigenschaften leicht formal in Form einer DTD fassen.

Das obige Beispiel nimmt hier folgende Gestalt an:

```
<!-- XSL Note Pattern in XML-Syntax -->
<element type="absatz">
  <target-element type="definition"/>
</element>
```

5.1.2 Elementidentifikation in IP4W3

Für die Elementidentifikation in IP4W3 habe ich mich an den XSL-Patterns orientiert. Sie stellen alle Funktionen zur Verfügung, die für die Bezeichnung von Elementstellen zur Suche in IP4W3 benötigt werden. Außerdem stellt die Syntax für den Administrator kein Hindernis dar; in Verbindung mit einer DTD ist sogar automatisch eine Benutzerschnittstelle vorhanden, die dem Administrator vertraut ist und mit einem SGML-Editor für einen guten Bedienkomfort sorgt. Die XSL-Note [W3C97E] enthält auch eine DTD für die in XML formulierten Patterns. Da absehbar war, daß die XSL-Note eine ungewisse Zukunft vor sich hat, habe ich nicht direkt diese DTD übernommen, sondern eine eigene geschrieben, die sich jedoch sehr stark an die DTD der Note anlehnt. Durch diesen Schritt konnte ich diesen Teil von IP4W3 von den zu erwartenden Veränderungen im XSL-Umfeld entkoppeln. Außerdem konnte ich so auch Erweiterungen einbringen, die für IP4W3 unbedingt benötigt werden, wie zum Beispiel Informationen über das zu generierende HTML-Formular.

Ein Beispiel für ein IP4W3-Suchmuster:

```
<element type="absatz">
    <target-element type="definition">
    </target-element>
</element>
```

Weitere Beispiele habe ich bereits in Abschnitt 3.2 gezeigt. Deshalb verzichte ich an dieser Stelle darauf.

Die Gründe für die Verwendung von XSL-basierten Suchmustern in IP4W3 lassen sich wie folgt zusammenfassen:

- ✗ Die SGML-Syntax ist dem Administrator vertraut.
- ✗ Die Bereitstellung einer DTD für die Suchmuster ermöglicht die Verwendung eines SGML-Editors als Benutzerschnittstelle.
- ✗ Die eigens für IP4W3 entworfene DTD ist den speziellen Bedürfnissen und Möglichkeiten des Systems angepaßt. Sollte der Funktionsumfang des Systems durch Unterstützung komplexerer Suchmuster erweitert werden, läßt sich die DTD leicht anpassen.
- ✗ Keiner der anderen genannten Ansätze bietet alle diese Vorteile.

5.2 Programme

Dieser Abschnitt beschreibt die Implementierung der Programme von IP4W3. Ihre Aufgaben umfassen die Erzeugung von HTML-Formularen, die Indexierung und Indexsuche in Dokumenten, das Feststellen von Übereinstimmungen mit Suchmustern, die Ermittlung eines Atoms, das Formatieren von Dokumentausschnitten sowie die Unterstützung des Administrators bei der Bedienung von IP4W3.

5.2.1 Formulargenerator

Die Aufgabe des Formulargenerators besteht darin, die Suchmuster einzulesen und daraus ein HTML-Abfrageformular zu erzeugen. Er akzeptiert als Eingabe die analysierte Form von Suchmustern (d.h. einen ESIS-Strom [ISON1035], der von einem SGML/XML-Parser wie *SP* geliefert wird), wie sie in Abschnitt 3.2.1 beschrieben sind. Die Ausgabe des Generators ist ein HTML-`form`-Element gemäß HTML-4-DTD[5].

Der Formulargenerator ist in der Tool Command Language (Tcl) implementiert, basiert auf dem SGML-Verarbeitungsprogramm CoST und ist in der Datei `suchmuster2formular.cost` zu finden. Ich habe CoST als Grundlage gewählt, weil dieses Programm eine programmierbare Schnittstelle (über Tcl) für den Zugriff auf SGML-Instanzen (in diesem Fall die Suchmuster) bietet.

Funktionsweise des Programms

Die Informationen zur Formularerzeugung sind in Suchmustern in den Elementen vom Typ `target-element` und darin im Elementtyp `form` enthalten. Es sind zwei mögliche Formularkomponenten zugelassen:

- ✗ Textfelder (Elementtyp `textinput`), denen mit dem Attribut `wertebereich` einer der folgenden Wertebereiche zugewiesen werden kann: `beliebig`, `buchstaben`, `posinteger`.
- ✗ Auswahlfelder (Elementtyp `select`), die dazu dienen, eine 1-aus-n-Auswahl zu ermöglichen. Dazu enthält ein `select`-Element mindestens ein Element vom Typ `option`.

Die Überprüfung des Wertebereichs findet client-seitig statt. Die in Abschnitt 5.4 beschriebene JavaScript-Funktion `pruefe_wert()` realisiert die Validierung. Für den Formulargenerator bleibt hierbei nur die Aufgabe, die JavaScript-Bibliothek in die HTML-Seite zu integrieren und die Formularelemente über HTML-Ereignisattribute mit den entsprechenden Funktionen zur Ereignisbehandlung zu verbinden.

Die beiden beschriebenen Formularkomponenten orientieren sich wegen ihres Verwendungszwecks an HTML-Formularen. Aus diesen Grund kann der Formulargenerator eine relativ einfache Abbildung durchführen, die der folgende Pseudo-Code zeigt:

---

5 Die Doctype-Deklaration könnte folgende Gestalt besitzen: `<!DOCTYPE form PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">`. Auf die Ausgabe dieser Zeile wird verzichtet, weil das Formular nicht für sich verwendet werden soll, sondern als Bestandteil einer HTML-Instanz.

```
Ausgabe des Formularkopfes;
Ausgabe des Script-Element zum Laden der JavaScript-Bibliothek;
// Schleife für jedes target-element:
Für jedes Element T vom Typ target-element
begin
  // Behandlung von textinput:
  Für das Kindelement F von T vom Typ form
  begin
    Für das Kindelement I von F vom Typ textinput
    begin
      Ausgabe des HTML-Elements zur Texteingabe
    end;
  end;

  // Behandlung von select:
  Für das Kindelement F von T vom Typ form
  begin
    Für das Kindelement S von F vom Typ select
    begin
      Ausgabe des HTML-Elements SELECT
      // alle Optionen ausgeben
      Für jedes Kindelement O von S vom Typ option
      begin
        Ausgabe des HTML-Elements OPTION
      end;
    end;
  end;
end;
```

5.2.2 Indexierung von und Indexsuche in SGML/XML-Dokumenten

Die Erzeugung eines Stichwortverzeichnisses passiert in zwei Schritten. Zunächst muß das Dokument so verändert werden, daß ein Element eindeutig bezeichnet werden kann. Im zweiten Schritt wird dann ein Verzeichnis erstellt, das es erlaubt, jedes Wort auf den Elementbezeichner des Elements abzubilden, in dem es vorkommt.

Um die Elementbezeichner in eine Instanz einzubringen, habe ich ein DSSSL-Programm geschrieben (`ergaenze-ids.dsssl`), das jedes Element einer Instanz mit einem Attribut namens `internal-id` ausstattet. Als Wert des Attributs wird eine fortlaufende Nummer, beginnend beim Wurzelelement vergeben. Die Ausgabe dieses Prozesses ist eine wohlgeformte, aber nicht gültige XML-Instanz, d.h. die Instanz besitzt keine Doctype-Deklaration mehr. Auf die Gültigkeit der Instanz kann beziehungsweise sollte aus folgenden Gründen verzichtet werden:

✗ Die ursprüngliche Instanz wird bereits beim Ergänzen der Elementbezeichner von einem Parser auf Korrektheit überprüft. Bei der weiteren Benutzung der veränderten Instanz ist deshalb gewährleistet, daß sie syntaktisch korrekt ist. Dies gilt natürlich nur, wenn (a) beim ersten Parsing kein Fehler entdeckt wird und wenn (b) die programmgesteuerte Veränderung keine neuen Feh-

ler einbaut. Ersteres wird dem Administrator beim Parsing gemeldet, letzteres ist durch mein Programm gewährleistet.

✗ Durch die nicht mehr stattfindende Validierung der Instanz gegen die DTD sind nachfolgende Verarbeitungsschritte (z.B. die Formatierung) schneller als sie es mit Validierung wären.

✗ Würde man die Gültigkeit der ausgegebenen Instanz verlangen, so wäre eine Manipulation der DTD notwendig. Es müßte zumindest die Attributlistendeklaration *jedes* Elementtyps verändert werden, um das Attribut `internal-id` zu deklarieren. Soweit es XML betrifft, ist dies noch relativ leicht zu bewerkstelligen, da es für jeden Elementtyp *mehrere* Attributlistendeklarationen geben darf, die vom Parser zusammengefaßt werden. Im allgemeinen SGML-Fall ist dies jedoch nicht so, da diese Möglichkeit erst durch den »Annex K, Web SGML Adaptations« [ISO97] am 4. Dezember 1997 als optionale Möglichkeit in SGML eingeführt wurde[6]. Diese Operation ist deshalb nicht ganz einfach zu implementieren. Dank der angeführten Gründe war es auch nicht notwendig oder gar sinnvoll.

Ich habe DSSSL für diese Aufgabe gewählt, weil die Sprache zusammen mit den Erweiterungen der DSSSL-Maschine Jade ein bequemes Werkzeug zur Transformation von SGML-Instanzen (Eingabe SGML, Ausgabe SGML) darstellt.

Für den Aufbau des Wortindex habe ich ein CoST-Programm geschrieben (`makeindex.cost`), das eine Liste von Worten mit dazugehörigen Elementbezeichnern (also den Werten des Attributs `internal-id`) erzeugt. CoST besitzt einen Verarbeitungsmodus, der die ereignisgesteuerte Verarbeitung von SGML-Instanzen erlaubt. Das Programm durchläuft die baumartige Darstellung des Dokuments im Tiefendurchlauf und erzeugt bei Erreichen oder Verlassen von Knoten ein Ereignis. Eines dieser Ereignisse ist das Erreichen eines CDATA-Knotens (Character Data), also eines Knotens, der Text (im Gegensatz zu beispielsweise Start- oder End-Tags) enthält. Der Event-Handler meines Programms kann nun auf den Inhalt eines CDATA-Knotens zugreifen. Dieser Inhalt steht als Tcl-Zeichenkette zur Verfügung. Um die einzelnen Worte zu erhalten, breche ich die Zeichenkette an vorgegebenen Trennzeichen (Punkt, Komma, Bindestrich usw.) auf und gebe die so erhaltenen Worte zusammen mit dem zuvor ermittelten Elementbezeichner aus. Der Elementbezeichner bezieht sich dabei auf den nächsten Vorfahren, der ein Attribut namens `internal-id` besitzt. In Pseudo-Code nimmt der Event-Handler folgende Gestalt an:

---

6 Es ist kein Zufall, daß die »Proposed Recommendation« für XML nur vier Tage später erschienen ist. Der Annex K diente einzig dem Zweck, SGML so zu verändern, daß XML eine Teilmenge davon sein würde.

```
// Event-Handler für CDATA-Knoten

// Inhalt des Knotens ermitteln
var inhalt    := ermittle Inhalt des Knotens;

// Inhalt in Worte auftrennen
var wortliste := trenne inhalt an den zeichen { ,.\t-()"!;/:};

// Element-ID des nächsten Vorfahren ermitteln
var elemID    := Ermittle den Attributwert
                 des Attributs "internal-id"
                 des nächsten Vorfahren;

// alle Worte mit Element-ID ausgeben
Für jedes Wort W in wortliste
begin
  Ausgabe von W und elemID;
end;
```

Anschließend liegt der Index als zweispaltige Tabelle vor. Die linke Spalte enthält ein Wort und in der rechten Spalte steht der zugehörige Elementbezeichner.

CoST war hier das Programm der Wahl, weil mit der zugrundeliegenden Sprache Tcl auch die leichte Verarbeitung von Komponenten möglich ist, die außerhalb der Reichweite von SGML liegen. In diesem Fall meine ich damit die einzelnen Worte. Die Modellierung von SGML/XML endet bei Parsed Character Data (PCDATA). Ob es sich bei PCDATA um einen Buchstaben, ein Wort, einen Satz oder ganze Absätze handelt, ist in SGML/XML nicht darstellbar und mit einigen Werkzeugen zur Verarbeitung nicht oder nur mit großem Aufwand zu handhaben. Zum Aufbau eines Wortindex sind die Möglichkeiten von CoST/Tcl hingegen sehr gut geeignet.

Bei der Stichwortsuche wird der so aufgebaute Index vom CGI-Skript (`search.pl`) sequentiell durchlaufen und das gesuchte Wort wird mit den Worten auf der linken Seite verglichen. Jeder Treffer liefert den Elementbezeichner auf der rechten Seite zurück. Bei diesen Treffern handelt es sich zunächst nur um potentielle Treffer, da in einem weiteren Schritt geprüft werden muß, ob sich der Treffer im gewünschten Elementkontext (festgelegt durch ein Suchmuster) befindet. Dieser Vorgang wird im folgenden Abschnitt 5.2.3 beschrieben.

Diskussion des Suchvorgangs

Die Stichwortsuche ist relativ einfach implementiert. Eine sequentielle Suche scheint mit Blick auf die Laufzeit eine schlechte Wahl zu sein. Eine naheliegende Verbesserung wäre der Einsatz eines Hashing-Verfahrens zur Effizienzsteigerung. Aus mehreren Gründen habe ich dies nicht implementiert.

- ✗ Da sich IP4W3 noch in der Testphase befindet, ist nicht absehbar, welche Möglichkeiten bei der Suche benötigt werden. Beispielsweise ist die Suche nach regulären Ausdrücken mit dem her-

kömmlichen Hashing nicht möglich. Die vorliegende Implementierung der sequentiellen Suche mit Perl läßt sich direkt auf reguläre Ausdrücke ausweiten.

✗ Die Effizienzsteigerung bei der Suche mit einem Hashverfahren ist zwar unbestritten, jedoch kostet der Aufbau der Hashtabelle genau so viel Zeit wie die sequentielle Suche. Den Vorteil des Hashing kann man nur dann ausnutzen, wenn die Tabelle *nur einmal* aufgebaut wird und wenn anschließend *mehrere* Suchen damit ausgeführt werden. Dies ist hier nicht möglich, da das CGI-Skript *bei jeder Suche* vom Web-Server neu aufgerufen wird. Würde es dann zunächst die Indexdaten sequentiell einlesen und daraus die Hashtabelle aufbauen, so könnte man in der gleichen Zeit die sequentielle Suche durchführen.
Eine Alternative wäre es, eine binäre Suche in der sortierten Wortliste auf der Festplatte durchzuführen. Dazu müßte jedoch die Größe eines Feldes (und damit die maximale Wortlänge eines Stichworts) beschränkt werden. Sollte sich die Suche als zu langsam herausstellen, wäre das die beste Alternative. Nachteile wären auch hier der Verzicht auf die Suche nach regulären Ausdrükken sowie der erhöhte Platzbedarf, der aus der festen Feldgröße resultiert.
Bei den Testläufen war die Laufzeit der Suche verschwindend gering, so daß die tatsächliche Antwortzeit stärker von Einflüssen abhängt, die außerhalb von IP4W3 liegen[7].

5.2.3 Prüfen auf Übereinstimmung mit dem Suchmuster und Ermitteln des passenden Atoms

Die bei der Indexsuche gefundenen Treffer sind zunächst nur Übereinstimmungen von Suchbegriff und Indexeintrag. Es muß noch geprüft werden, ob die Treffer auch in der richtigen Umgebung liegen. Das heißt, das gefundene Wort muß sich innerhalb des gewünschten Zielelements und dieses innerhalb der richtigen umgebenden Elemente befinden. Der Benutzer wählt dazu eine Kategorie aus, in der er die Suche durchführen möchte. Das zu einer Kategorie korrespondierende Suchmuster enthält die Informationen, die zur Umgebungsprüfung benötigt werden. Um die Prüfung möglichst schnell durchführen zu können, werden die Suchmuster einmalig in Programmcode umgewandelt, den das Programm zur Umgebungsprüfung und Atomermittlung (`search-atoms.cost`) zur Laufzeit einlesen

7 Beeinflussende Faktoren sind: Welcher Webserver wird benutzt? Läuft er alleinstehend (standalone) oder wird er bei jeder Anfrage über den Inet-Daemon gestartet? Liegen die Daten auf einer lokalen Festplatte oder auf einer, die per Network File System (NFS) angeschlossen ist (Qualität der Netzverbindung)? Welche Vorteile bringt FastCGI? Der alles überdeckende Einfluß ist aber die Performanz der Netzverbindung durch alle Schichten (von TCP/IP bis HTTP). Beherrscht der Browser zum Beispiel die HTTP-Keep-Alive-Methode?

und direkt ausführen kann (vgl. Abbildung 16). Die Umwandlung wird von `suchmuster2umgebungspruefung.cost` durchgeführt. Als Eingabe bekommt dieses CoST-Programm[8] eine Instanz mit Suchmustern und erzeugt als Ausgabe ein assoziatives Array mit den CoST-Programmanweisungen zur Umgebungsprüfung. Als Index für das Array wird die ID des Suchmusters verwendet, die laut DTD notwendig (`REQUIRED`) ist.

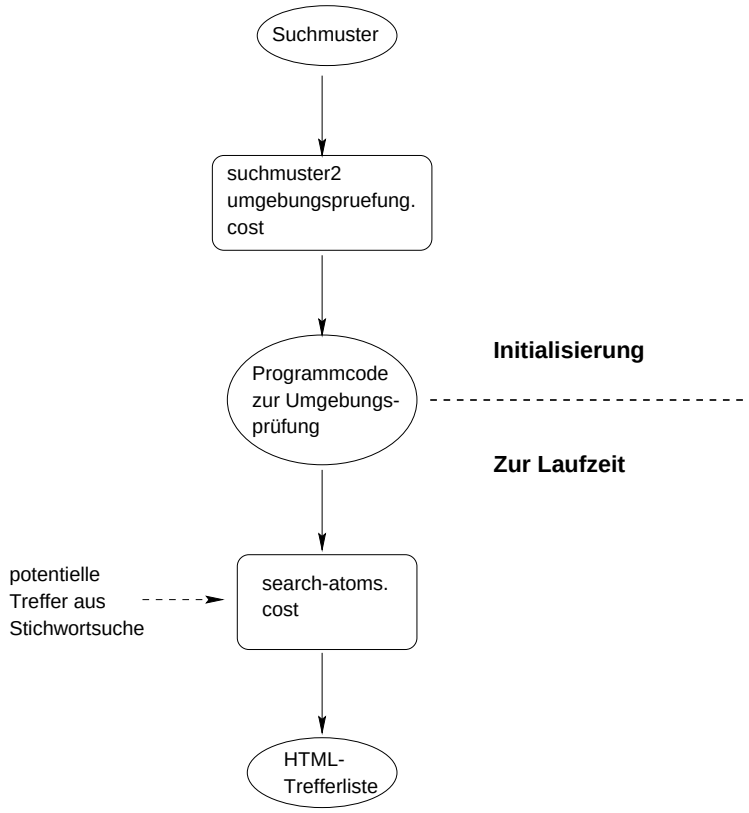

Abbildung 16: Umgebungsprüfung: Ablauf zur Initialisierung und zur Laufzeit

Bei der Umwandlung eines Suchmusters in Programmcode muß berücksichtigt werden, daß die Überprüfung in einem Blatt des Dokumentbaums stattfindet. Dies liegt daran, daß in SGML/XML-Dokumenten nur in Blättern Text enthalten sein kann, also kann auch nur dort ein Treffer auftreten. Zur Umgebungsprüfung navigiert mein Programm (`search-atoms.cost`) deshalb zu diesem Blatt und »schaut« in der Hierarchie nach oben. Diese Sichtweise spiegelt sich im Algorithmus wider, mit dem `suchmuster2umgebungspruefung.cost` ein Suchmuster verarbeitet, um die Programmanweisungen zur Um-

8 CoST wurde aus den zuvor schon genannten Gründen zur Implementierung benutzt, insbesondere sind dies die Flexibilität von Tcl in Verbindung mit CoSTs komfortabler SGML-Schnittstelle.

gebungsprüfung zu erzeugen. Um das Suchmuster »bottom-up« zu durchlaufen, führt das Programm eine *Tiefensuche* durch und gibt bei Erreichen eines *End-Tags* den entsprechenden CoST-Programmcode aus. Durch dieses Vorgehen ist die gewünschte Sichtweise gewährleistet. Es folgt der zuständige Event-Handler als Pseudo-Code:

```
// Event-Handler, der bei Erreichen eines
// *End-Tags* ausgeführt wird
if aktuelles Element E hat Start-Tag der Form <ELEMENT TYPE=X>
then
    Ausgabe: Programmcode,
             der zum Vaterelement vom Typ x navigiert;

    // Kindelemente namens attribute
    Für jedes Kindelement K von
    E vom Typ <ATTRIBUTE NAME=y VALUE=z>
    begin
       Ausgabe: Programmcode, der prüft, ob es ein Attribut
                  namens y mit Wert z gibt;
    end;
elseif aktuelles Element E hat Start-Tag der Form <ELEMENT>
then
    Ausgabe: Programmcode, der zum Vaterelement navigiert;

    // Kindelemente namens attribute
    Für jedes Kindelement K von
    E vom Typ <ATTRIBUTE NAME=y VALUE=z>
    begin
       Ausgabe: Programmcode, der prüft, ob es ein Attribut
                namens y mit Wert z gibt;
    end;
elseif aktuelles Element E hat
       Start-Tag der Form <TARGET-ELEMENT TYPE=X>
then
    Ausgabe: Programmcode, der prüft, ob das aktuelle Element
             vom Typ x ist;

    // Kindelemente namens attribute
    Für jedes Kindelement K von
    E vom Typ <ATTRIBUTE NAME=y VALUE=z>
    begin
       Ausgabe: Programmcode, der prüft, ob es ein Attribut
                namens y mit Wert z gibt;
    end;
elseif aktuelles Element E ist vom Typ ANY
then
    Ausgabe: Programmcode,
             der nacheinander zu allen Vorfahren navigiert;
fi;
```

Es werden nur die Elementtypen `element`, `target-element`, `any` und `attribute` betrachtet. Andere Typen, die in Suchmustern vorkommen, haben keine Bedeutung für die Umgebungsprüfung. Als Beispiel sei hier ein Suchmuster aus dem Anwendungsbeispiel »XML-Spezifikation« gezeigt, gefolgt von dem CoST-Programmcode, den der obige Event-Handler beim Tiefendurchlauf generiert.

```
<suchmuster id="def">
  <element type="absatz">
    <any>
      <target-element type="definition">
        <beschreibung>Begriffsdefinitionen</beschreibung>
        <form>
          <textinput wertebereich="beliebig">
        </form>
      </target-element>
    </any>
  </element>
</suchmuster>
```

Es treten nacheinander die Ereignisse für die End-Tags `</target-element>`, `</any>`, `</element>` ein (im Suchmuster hervorgehoben). Der Programmcode, der durch obige Ereignisroutine erzeugt wird, sieht wie folgt aus:

```
element "definition" ancestor parent element "absatz"
```

Gespeichert wird diese Ausgabe in einem Arrayeintrag, der den Index »def« (das ist die ID des Suchmusters) besitzt. Je nach Anzahl der Suchmuster gibt es entsprechend viele Arrayeinträge.

Die Ausführung des obigen Programmcodes durch CoST läßt sich umgangssprachlich folgendermaßen interpretieren: Wenn das aktuelle Element vom Typ `definition` ist (`element "definition"`), wähle den nächsten Vorfahren (`ancestor`) aus (in CoST ist ein Element auch Vorfahre von sich selbst), wähle dann das Vaterelement (`parent`); ist es vom Typ `absatz` (`element "absatz"`)? Sobald diese letzte Frage positiv beantwortet wird, wird die Abarbeitung beendet. Andernfalls würde über einen Backtracking-Mechanismus zu einer vorhergehenden Abfrage zurückgesprungen, für die es eine weitere Lösung gibt. In diesem Fall kann `ancestor` mehr als eine Lösung haben, da `ancestor` schrittweise alle Vorfahren bis zur Wurzel auswählt.

Das Programm `search-atoms.cost` liest die Ausgabe von `suchmuster2umgebungspruefung.cost` ein und greift über die ID des Suchmusters direkt auf den Programmcode zu, der zur Umgebungsprüfung ausgeführt wird. Dabei wird ein boolescher Wert als Ergebnis geliefert. Der logische Wert »wahr« bedeutet, daß der fragliche Treffer aus der Stichwortsuche tatsächlich in der richtigen Umgebung stattgefunden hat. Andernfalls trat die Wortübereinstimmung an einer anderen Stelle im Dokument auf. Auf eine detaillierte Darstellung als Pseudo-Code kann hier verzichtet werden, weil die Vorgehensweise sehr einfach ist: `search-atoms.cost` bekommt als Parameter die vom Benutzer ausgewählte Kategorie übergeben. Im obigen Beispiel wäre das »def«. Die Auswertung der Anweisung

```
query? element "definition" ancestor parent element "absatz"
```

liefert bereits den gesuchten booleschen Wert. Die eigentliche Arbeit besteht also darin, die Suchmuster in die CoST-Anweisungen zu

überführen, die dann nur (mit dem Präfix `query?` versehen) ausgeführt werden.

Die möglichen Fälle bei der Suche/Umgebungsprüfung werden nun folgendermaßen behandelt:

Kein Treffer bei der Stichwortsuche

Der Benutzer erhält eine entsprechende Rückmeldung.

Treffer bei der Stichwortsuche in der richtigen Umgebung

Der Benutzer erhält eine Kurzansicht der Treffer (HTML-Trefferliste) und kann daraus auswählen.

Treffer bei der Stichwortsuche, aber nicht in der richtigen Umgebung

Der Benutzer erhält eine Meldung darüber, daß es Treffer außerhalb der von ihm gewählten Kategorie gab. Er kann dann eine Suche ohne Kategorie ausführen.

Ermittlung des Atoms

Wurde ein Treffer in der richtigen Umgebung gefunden, so ermittelt `search-atoms.cost` das zugehörige Atom. Wie bereits früher ausgeführt, stimmen Atome und Suchmuster in der vorliegenden Implementation von IP4W3 überein. Aus diesem Grund wird als Atom unmittelbar das Element zurückgeliefert, das bei der Umgebungsprüfung als äußerstes Element gefunden wurde.

Interessant ist noch die Frage, was passiert, wenn eine Suche *ohne* Kategorie, also ohne Suchmuster durchgeführt wird. In diesem Fall wird jeder Treffer als in der richtigen Umgebung befindlich angesehen. Für die Ermittlung des Atoms verwende ich in diesem Fall folgende Heuristik: Ausgehend von dem Zielelement, das den gesuchten Begriff enthält, gehe ich in der Elementhierarchie so weit nach oben, bis ich auf ein Element treffe, das *keinen gemischten Inhalt* (mixed content) enthält. Umgangssprachlich heißt das, ich suche einen Vorfahren, der selbst keinen Text, sondern nur Kindelemente enthält. Ein solches Element wird auch als Container-Element bezeichnet. Container-Elemente sind als Vorgabeatome geeignet, da meine Betrachtung verschiedener Dokumenttypen zeigt, daß Container meist Dokumentteile enthalten, die auch alleinstehend verständlich sind. Da die Suche ohne Kategorie eine Ausnahme darstellt, ist diese Heuristik meiner Meinung nach adäquat, was auch durch die betrachteten Anwendungsbeispiele unterstützt wird.

Generierung der HTML-Trefferliste

Zu jedem gefundenen Atom erzeugt `search-atoms.cost` auch die HTML-Ausgabe für die Trefferliste. Dabei handelt es sich um eine Tabelle, die in jeder Zeile einen Treffer enthält. Jede Zeile besitzt eine

HTML-Checkbox, die der Benutzer anklicken kann, um den Treffer vollständig zu sehen. Daneben befindet sich eine Kurzfassung des Atoms. Diese Kurzfassung besteht aus jeweils n Zeichen des Atoms und des Zielelements. Vom Atom werden die ersten n Zeichen angezeigt und vom Zielelement die ersten n/2 Zeichen vor und nach dem gesuchten Begriff. Der Wert von n ist momentan auf 100 vorgegeben.

5.2.4 Formatierer (DSSSL-Rahmen)

Bei der Formatierung geht es in diesem Zusammenhang primär (aber nicht nur) um die Transformation von beliebigen SGML/XML-Dokumenten in HTML zur Darstellung im Web-Browser. Eine Formatierung für den Ausdruck soll dabei nicht ausgeschlossen sein.

Aufgrund dieses allgemeinen Ansatzes muß es eine Schnittstelle geben, die dem Benutzer erlaubt, Formatierungsanweisungen für einen Dokumenttyp in IP4W3 zu integrieren. Folgende Ziele wurden dabei verfolgt:

- ✗ Berücksichtigung von offenen Standards
- ✗ Eignung für Online- (HTML) und Papierausgabe

Im Umfeld von SGML/XML und dem World Wide Web stehen folgende Alternativen zur Verfügung:

- ✗ Cascading Style Sheets (CSS, [W3C96A] und [W3C97A])
- ✗ Extensible Style Language (XSL, [W3C98K])
- ✗ Document Style Semantics and Specification Language (DSSSL, [ISO96B])

Die Cascading Style Sheets sind eng an HTML gebunden und auch in ihrem Level 2 nicht mächtig genug, um den Ansprüchen in diesem Kontext zu genügen. Gegen XSL spricht, daß die Sprache noch nicht endgültig verabschiedet ist. Das W3C will die Entwicklung Mitte 1999 abschließen[9]. Ein praktisches Argument gegen XSL ist die mangelhafte Implementation; ein frei verfügbares Programm (*XT*) ist sehr instabil und nicht einsetzbar. DSSSL ist unter den genannten Alternativen zum gegenwärtigen Zeitpunkt die beste und wurde aus folgenden Gründen für IP4W3 gewählt:

- ✗ DSSSL ist eine ISO-Norm, die 1996 verabschiedet wurde.
- ✗ DSSSL ist für die Formatierung auf Papier geeignet und in der Implementierung *Jade* auch hervorragend für die HTML-Ausgabe geeignet.

---

9 Im Herbst 1998 liegt die zuständige Arbeitsgruppe einen Monat hinter ihrem früher bekanntgegebenen Zeitplan zurück.

✗ DSSSL-Formatierungen sind unabhängig vom Dateiformat[10] (RTF, Postscript, PDF).

DSSSL-Rahmen

Innerhalb von IP4W3 besteht der Formatierer aus einem DSSSL-Rahmen (`format.dsssl`), der zur Anwendung durch ein Rumpf-Programm für die spezielle DTD ergänzt werden muß. Der Rahmen navigiert zu den auszugebenden Elementen, die über ihre eindeutigen Elementbezeichner adressiert werden, und übergibt dann die Kontrolle an den Rumpf.

Beispiel (vgl. Abbildung 17): Wenn die Elemente A und B, die sich an beliebiger Stelle im Dokument befinden, formatiert werden sollen, sieht die Verarbeitungsreihenfolge so aus: Der DSSSL-Rahmen durchläuft das Dokument (Tiefensuche) und prüft in jedem Knoten, ob es sich um das Element A oder B handelt. Sobald er das erste Element findet (hier A), formatiert der Rumpf den gesamten Teilbaum, dessen Wurzel A ist. Da DSSSL auch hierbei eine Tiefensuche ausführt, übergibt der Rumpf die Kontrolle wieder an den Rahmen, nachdem der gesamte Teilbaum abgearbeitet ist. Der Rahmen wird also nicht noch einmal in den Teilbaum A hinabsteigen, sondern seine Suche nach dem Knoten B in dem nicht abgearbeiteten Teil fortsetzen.

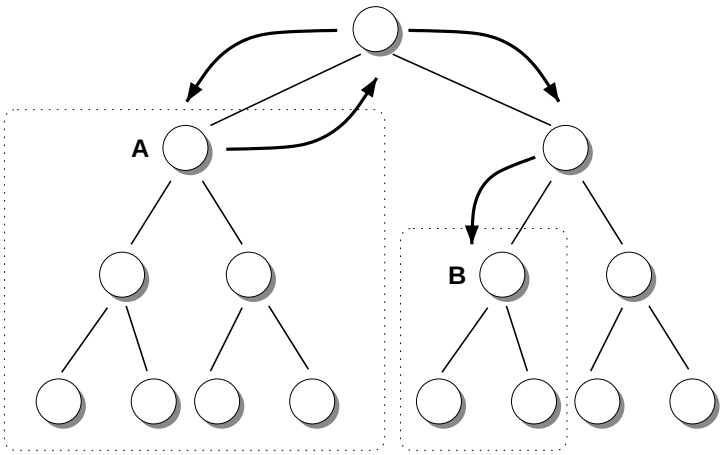

Abbildung 17: Aufgabe des Formatier-Rahmens: Navigation zu den gesuchten Elementen

Problem bei dieser Vorgehensweise

Falls der DSSSL-Rumpf nicht speziell für den Einsatz in IP4W3 entwickelt wurde[11], muß man bedenken, daß sich Anweisungen im Rumpf auf Grundeinstellungen (bei Webseiten zum Beispiel die Hin-

10 Allerdings sind die Dateiformate davon abhängig, welche Formate die benutzte DSSSL-Maschine unterstützt.

tergrundfarbe, bei Papierausgabe zum Beispiel das Seitenformat) abstützen, die für die Dokumentenwurzel gemacht werden. Da der Rumpf die Kontrolle nun erst später erlangt, kann es sein, daß die Grundeinstellungen noch nicht durchgeführt wurden.

Bei der Numerierung von Abschnitten treten keine unerwünschten Nebeneffekte auf, da die Formatierung im Kontext des gesamten Dokuments durchgeführt wird.

Navigation zu den zu formatierenden Elementen

Das Aufsuchen der zu formatierenden Elemente ist für die Geschwindigkeit des Vorgangs sehr wichtig. In der Regel ist die Anzahl der auszugebenden Elemente sehr klein im Vergleich zur Anzahl aller Elemente im Dokument. Das liegt daran, daß die Anzahl der Treffer möglichst gering sein soll. Des weiteren stehen die Treffer meist weit unten in der Hierarchie der Elemente, um kleine Dokumentausschnitte als Treffer zu erhalten. Ich habe deshalb die von DSSSL durchgeführte Tiefensuche modifiziert, um das Durchsuchen ganzer Teilbäume zu vermeiden, sofern die Aussage möglich ist, daß sich in einem fraglichen Teilbaum kein Treffer befindet[12]. Dazu habe ich ausgenutzt, daß als Elementbezeichner ausschließlich fortlaufende Zahlen benutzt werden, deren Ordnung die Reihenfolge der Elemente beim Tiefendurchlauf angibt. Die genaue Vorgehensweise des DSSSL-Rahmens zeigt das folgende Listing im Pseudo-Code, das ich anschließend noch erläutere.

```
var atom_ids := Liste der zu formatierenden Element-IDs

// Behandlung eines Elements
if Element-ID des aktuellen Elements E ist in atom_ids enthalten
then Formatiere aktuelles Element  // Wechsel zu DSSSL-Rumpf (1)
else
    if aktuelles Element E hat keinen
       Bruder rechts im Elementbaum
    then (3)
        Verarbeite rekursiv alle Kinder K
        des aktuellen Elements E;
    else (4)
        Min := Element-ID des aktuellen Elements + 1;
        Max := Element-ID des Bruders - 1;
        // [Min,Max] ist also das Interval der
        // Element-IDs aller Nachfahren
```

11 Diese Situation kann als der Regelfall angesehen werden, da es ein Ziel bei der Entwicklung von IP4W3 war, daß der Anwender möglichst geringe Vorarbeit für die Benutzung leisten muß. Man sollte also den Fall berücksichtigen, daß es sich bei dem DSSSL-Rumpf um ein Programm handelt, das der Anwender zur Formatierung von vollständigen Instanzen geschrieben hat.

12 Abgeschafft werden kann die Tiefensuche nicht, da sie untrennbar mit DSSSL verwoben ist. Weiter unten wird das noch erkennbar sein.

```
        if es gibt ein Element X in atom_ids: Min <= X <= Max
           // das heißt mindestens ein Nachfahre
           // ist zu formatieren
        then (5)
            Verarbeite rekursiv alle Kinder K
            des aktuellen Elements E;
        else (6)
            Liefere leeres Flow Object als Ergebnis;
        fi
    fi
fi
```

Falls das aktuelle Element zu den zu formatierenden gehört, wird die Kontrolle, wie oben beschrieben, an den Rumpf übergeben (1). Im anderen Fall (2) hängt das weitere Vorgehen davon ab, ob es einen jüngeren Bruder gibt. Das ist ein Element, das den gleichen Vater wie das aktuelle Element besitzt und in der Instanz unmittelbar auf das aktuelle Element folgt. Bei SGML/XML muß man noch unterscheiden, ob Elemente Geschwister sind, wenn sie nicht vom gleichen Typ sind. In DSSSL ist dieser Unterschied von Bedeutung. In IP4W3 spielt der Elementtyp *keine* Rolle. Die Überprüfung wird deshalb mit der Funktion `absolute-last-sibling?` und nicht mit `last-sibling?` ausgeführt (siehe 10.2.4.4 in [ISO96B]). Falls es einen solchen Bruder gibt (4), ist es möglich, das Intervall zu bestimmen, in dem die Elementbezeichner der *Nachfahren* liegen. Falls es ein Element in der Liste der zu formatierenden Elementbezeichner gibt, das innerhalb des Intervalls liegt, so werden rekursiv alle *Kinder* abgearbeitet (5). Im anderen Fall ist klar, daß kein Nachfahre formatiert werden muß. Also kann der gesamte Teilbaum ausgelassen werden (6). Für den Fall, daß es gar keinen jüngeren Bruder gibt (3), ist es nicht möglich, das Intervall der Elementbezeichner der Nachfahren unmittelbar zu bestimmen[13]. Aus diesem Grund werden auch dann alle Kinder abgearbeitet. Der rechte Zweig eines Teilbaums wird also stets betrachtet.

Verhindern von mehrfacher Ausgabe von Dokumentausschnitten

Das Umschalten zwischen Rahmen und Rumpf sorgt auch dafür, daß zwei zu formatierende Elemente, die ineinander verschachtelt sind, nicht mehrfach ausgegeben werden.

13 Es gibt keine DSSSL-Funktion, die die benötigte Information liefert.

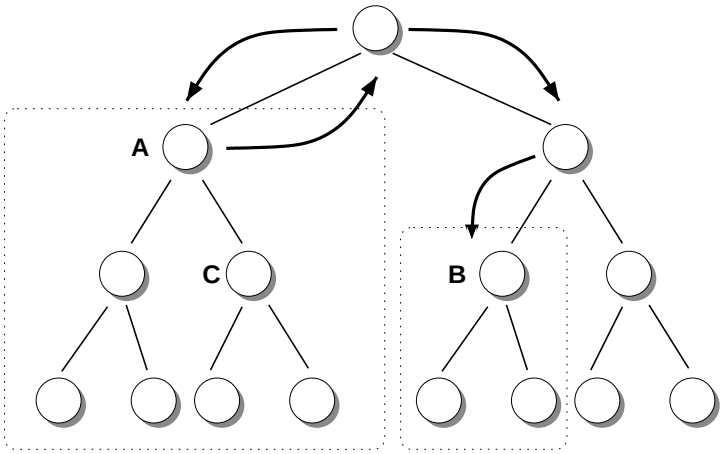

Abbildung 18: Formatierung von verschachtelten Elementen

Abbildung 18 illustriert diese Situation: Neben den Elementen A und B soll auch noch das Element C ausgegeben werden, das ein Nachfahre von A ist. Der Ablauf ist hier folgendermaßen: Die DSSSL-Steuerung läuft bis zum Element A, das der Rahmen als gesuchtes Element identifiziert. Dann läuft wie zuvor beschrieben die Formatierung des Teilbaums A ab. Im Anschluß daran wird der Teilbaum A nicht noch einmal behandelt. Ohne daß weitere Vorkehrungen notwendig sind, wird das Element C also nicht zweimal ausgegeben. Aus diesem Grund kann auf die aufwendige Überprüfung, welche bei der Suche gefundenen Elemente ineinander enthalten sind, verzichtet werden.

Diese positive Eigenschaft wird durch den DSSSL-Rahmen in keiner Weise berührt. Sowohl die unveränderte Tiefensuche von DSSSL als auch meine zuvor beschriebene effizientere Variante besitzen die Eigenschaft. Meine Lösung verhält sich funktional neutral, was für den Autor eines DSSSL-Rumpfes sehr wichtig ist, um keine unerwarteten Resultate zu riskieren.

5.2.5 Administrations-Werkzeug

Das Administrations-Werkzeug `ip4w3-manager` hat folgende Aufgabe: es erfragt vom Administrator die Informationen, die nötig sind, um ein Dokument in IP4W3 aufzunehmen. Diese Informationen werden benutzt, um die anderen Programme mit den korrekten Parametern aufzurufen. Außerdem speichert `ip4w3-manager` die Daten ab, damit sie bei einem erneuten Aufruf verfügbar sind[14].

14 Ein erneuter Aufruf kann beispielsweise bei einer Veränderung des Dokuments notwendig sein, da unter anderem der Suchindex neu generiert werden muß.

Da das Administrations-Werkzeug im wesentlichen eine nette Verpackung[15] für die anderen Programme ist, kam es bei der Implementierung darauf an, den Aufruf von Programmen und die Parameterübergabe möglichst einfach zu realisieren. Die von `ip4w3-manager` erfüllte Funktion ist keine sehr komplexe, so daß ich in diesem prototypischen Stadium des Gesamtsystems auf eine graphische Oberfläche verzichten konnte. Aus diesen Gründen habe ich das Programm als Bourne-Shell-Skript implementiert.

5.3 CGI-Skripte

In IP4W3 kommen zwei CGI-Skripte zum Einsatz, `preview.pl` und `search.pl`. Letzteres Skript führt die Stichwortsuche durch, die, wie oben beschrieben, eine einfache sequentielle Suche ist. Für die Formatierung ist `preview.pl` zuständig.

Im wesentlichen haben beide Skripte die Aufgabe, die Schnittstelle zwischen Web-Server und den zuvor beschriebenen Programmen zu bilden. Sie müssen dazu Parameter gemäß Common-Gateway-Interface-Konvention akzeptieren, auswerten und beim Aufruf an die anderen Programme weiterreichen. Die Behandlung der CGI-Parameter ist in einem Perl-CGI-Modul gekapselt, das als frei verfügbares Paket im Comprehensive Perl Archive Network (CPAN) erhältlich ist. Dies war der Hauptgrund für die Verwendung von Perl. Zwar läßt sich ein CGI-Skript auch relativ leicht selbst implementieren, jedoch ist hier nicht nur aus funktionalen Gründen auf eine korrekte Implementierung zu achten, sondern aus Gründen der Systemsicherheit auch auf eine sichere. Diese Anforderung macht die CGI-Programmierung schwierig, da es (natürlich?) möglich ist, eine korrekte aber mit Sicherheitslücken behaftete Umsetzung zu schreiben. Die Ausführung eines solchen Programms jedem WWW-Benutzer zu gestatten, ist riskant. Durch Benutzung eines ausgetesteten und zuverlässigen CGI-Moduls wird das Risiko minimiert.

Neben dem Aufruf von nsgmls, Jade und CoST erzeugen beide Skripte auch den HTTP-Kopf (Hypertext Transfer Protocol), der dem Web-Browser mitteilt, daß er als Datenformat HTML zu erwarten hat. Des weiteren geben die CGI-Skripte HTML-Code aus, in den die Ausgabe der anderen Programme eingebettet wird.

15 Ich erlaube mir, den »wrapper« in dieser Form in's Deutsche zu übertragen.

5.4 JavaScript-Funktionsbibliothek

Die JavaScript-Funktionsbibliothek `ip4w3.js` stellt folgende Funktionen zur Verfügung, die den Endanwender beim Ausfüllen der Suchformulare unterstützen.

function pruefe_wert(formularfeld,wertebereich)

prüft, ob der Wert (value) des `formularfelds` nur Zeichen der durch die Zeichenkette `wertebereich` angegebenen Menge enthält. Diese Funktion realisiert die Einschränkung des Wertebereichs, die der Administrator bei der Definition der Suchmuster angeben kann.

function alle_checkboxen(aktion,formular)

beeinflußt den Status sämtlicher Checkboxen im `formular`. Als `aktion` stehen zur Verfügung:

`"einschalten"`

aktiviert alle Checkboxen.

`"ausschalten"`

deaktiviert alle Checkboxen.

`"umschalten"`

invertiert den Zustand aller Checkboxen.

Alle beschriebenen Funktionen benötigen ausschließlich den Sprachumfang der JavaScript-Version 1.0 und sind damit zu allen JavaScript-fähigen Browsern kompatibel. Die Auslagerung in eine externe Bibliothek verlangt jedoch, daß der Browser in der Lage ist, das `src`-Attribut des Elements `script` zu verstehen. Dieses wurde durch Netscape mit dem Navigator 3 in Verbindung mit JavaScript 1.1 eingeführt. Es ist seit Version 4 Bestandteil des HTML-Standards. Der Formulargenerator von IP4W3 (siehe 5.2.1) sorgt dafür, daß für Browser, die externe JavaScript-Bibliotheken nicht einladen, die notwendigen Funktionen zumindest definiert sind, damit keine Fehlermeldung erscheint.

Speichern der gemerkten Ausschnitte

Die Speicherung der gemerkten Dokumentausschnitte (»Warenkorb«) geschieht in einem JavaScript-Array. Jeder Klick auf »Ausschnitt merken« überträgt die Nummer des Elements in das Array. Beim Anzeigen der gemerkten Ausschnitte wird aus den Array-Inhalten ein neuer URL generiert, der alle Elementnummern als Parameter enthält. Das Formatieren läuft dann genauso ab, wie bei Ausschnitten, die nach einer Suche formatiert werden.

5.5 Programmabläufe

Nachdem ich nun die Implementation der einzelnen Komponenten beschrieben habe, möchte ich in diesem Abschnitt das Zusammenspiel und die Abläufe mit Hilfe von Abbildungen beschreiben.

Den Anfang macht die Ablaufskizze für das Anmelden eines neuen Dokuments (vgl. Abbildung 19). Eine wichtige Position nimmt dabei der `ip4w3-manager` ein, da er dem Benutzer die Ausführung der Einzelschritte abnimmt. Aus dem ursprünglichen Dokument wird durch Hinzufügen von eindeutigen Elementbezeichnern eine veränderte Instanz erzeugt (`ergaenze-ids.dsssl`). Diese dient als Grundlage für den Suchindex (`makeindex.cost`). Aus den Suchmustern wird das HTML-Formular generiert (`suchmuster2formular.cost`) und der Programmcode, der bei der Suche benötigt wird (`suchmuster2umgebungspruefung.cost`).

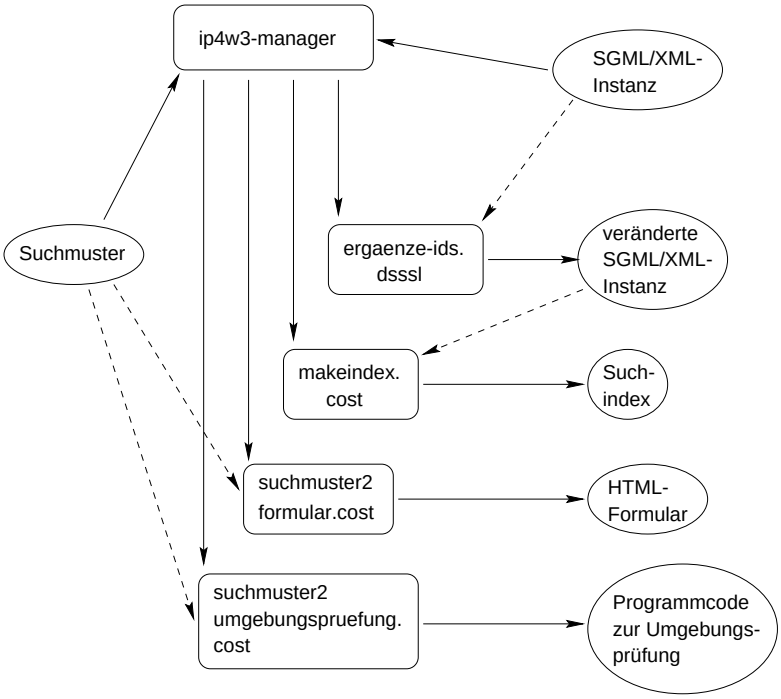

Abbildung 19: Anmelden eines Dokuments: Zusammenspiel der einzelnen Programme und der von ihnen gelesenen und geschriebenen Dateien

Bei der Suche (vgl. Abbildung 20) passiert folgendes: Die vom Benutzer in das Formular eingetragenen Daten übergibt der Web-Server gemäß CGI-Konvention an das Suchskript `search.pl`. Dieses führt die Stichwortsuche im Suchindex aus und läßt die gefundenen potentiellen Treffer vom Programm `search-atoms.cost` prüfen. Dazu muß das Programm die veränderte Instanz einlesen, um zu verifizieren, daß sich die Treffer in der richtigen Umgebung befinden. Außerdem benötigt es den im vorherigen Schritt erzeugten Programm-

code. Die Ausgabe ist die HTML-Trefferliste der ermittelten Atome (Kurzform der Dokumentausschnitte).

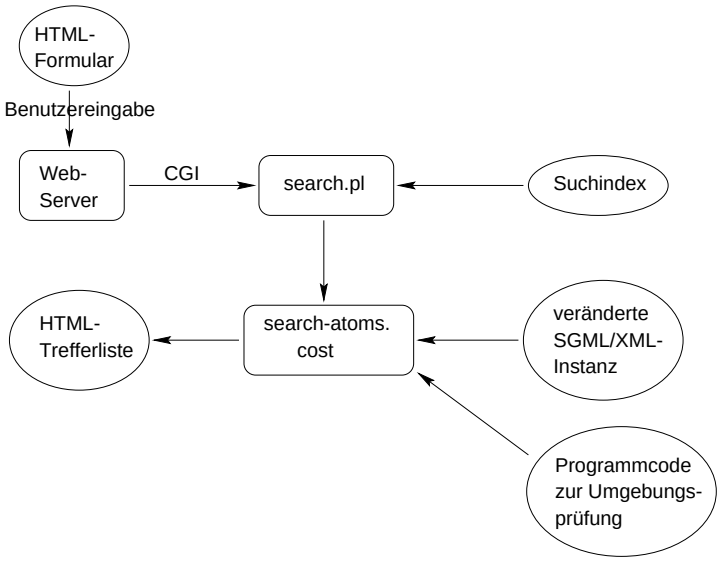

Abbildung 20: Programmablauf bei der Suche

Zur Formatierung (vgl. Abbildung 21) wird die Benutzerauswahl an das Skript `preview.pl` übergeben, das lediglich als CGI-Hülle für den Formatierer dient. Er bedient sich des DSSSL-Rahmens und -Rumpfes sowie (natürlich) der veränderten Instanz. Die Ausgabe wird an den Browser des Benutzers zur Darstellung übermittelt. Der Formatierungsschritt wird gegebenenfalls mehrfach durchlaufen, wenn der Benutzer die Größe der Dokumentausschnitte modifiziert und sie neu anzeigen läßt.

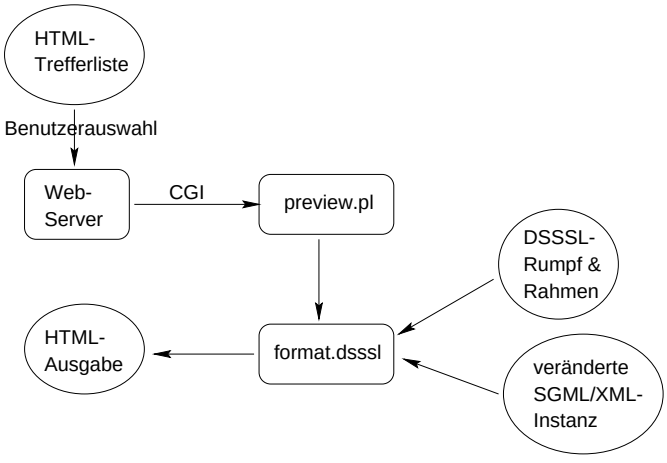

Abbildung 21: Programmablauf beim Formatieren

Umwandlung der gegebenen Daten in SGML/XML-Form

6

In diesem Kapitel beschreibe ich die Schritte zur Umwandlung der gegebenen Texte über makroökonomische Modelle in eine für IP4W3 geeignete Form. Zentrale Bedeutung kommt dabei der strukturellen Analyse zu (Abschnitt 6.2), die zum Entwurf von Auszeichnungen für die Dokumente führte (Abschnitt 6.4).

6.1 Ausgangsdaten

Die gegebenen Texte wurden mit der Macintosh-Textverarbeitung *WriteNow* geschrieben. Da keine Informationen über das Dateiformat dieses Programms verfügbar waren[1], wurde die Exportfunktion in das Rich Text Format (RTF) [MSFT94] benutzt.

Als erschwerend stellte sich heraus, daß ein großer Teil der Daten aus Grafiken bestand, die nach dem RTF-Export als Windows Metafile (WMF) in die Texte eingebettet waren. Dabei handelte es sich neben Abbildungen vor allem um komplexe mathematische Funktionen. Einfache Symbole wie Exponenten und Indizes waren hingegen durch Hoch- bzw. Tiefstellen von Textzeichen eingegeben worden.

Die Texte enthielten ausschließlich typographische und visuelle Auszeichnungen, jedoch keine Metadaten, die logische Informationen über Textabschnitte hätten liefern können.

6.2 Strukturelle Analyse

Die Analyse der gegebenen Texte wurde von mir anhand der gedruckten Bücher [UEBE92] und [UEBE95] durchgeführt. In den Dateien vorhandene Auszeichnungen trugen nicht zur Strukturierung und Auszeichnung der Texte bei. Das größte Problem bestand bei der Analyse darin, daß ich aufgrund des schwierigen und für mich weitgehend unverständlichen Inhalts nicht in der Lage war, den In-

1 Eine E-Mail an den Vertrieb ist bis heute unbeantwortet.

halt so stark zu berücksichtigen, wie es sinnvoll gewesen wäre. Diese Arbeit kann nur mit einem guten Verständis des Textes geleistet werden. Ein weiteres Hindernis war, daß eine informelle Beschreibung von mathematischen Gleichungen, von Tabellen und von Grafiken fehlte. Soweit es die gedruckte Form betrifft, ist das auch selbstverständlich, da z.B. Grafiken das Verständnis bei einem fachkundigen Leser fördern und eine zusätzliche verbale Erläuterung meist nicht notwendig ist. Möchte man jedoch eine Begriffsuche etwa in Gleichungen durchführen, so ist eine solche Beschreibung unbedingt notwendig, da die Suche andernfalls beispielsweise schon an abweichenden Variablennamen scheitert. Beschreibungen dieser Art sollten nach Absprache mit dem Autor von ihm eingebracht werden. Zum gegenwärtigen Zeitpunkt ist das Textmaterial in dieser Hinsicht immer noch verbesserungswürdig. Die Folge ist, daß die Suche in Gleichungen momentan gar nicht möglich ist, beziehungsweise keine brauchbaren Ergebnisse liefert.

Bei meiner Analyse habe ich mich darauf konzentriert, die Struktur der Texte, d.h. die Elemente auf den höheren Hierarchieebenen, zu erkennen und Auszeichnungen sowie deren Abhängigkeiten zu definieren. Auf den tieferen Ebenen, also auf Block- und Zeichenebene, habe ich versucht, so viel Metainformation in die Auszeichnungen zu bringen, wie an der gedruckten Vorlage und mit begrenztem Verständnis möglich war. Die von mir vorgeschlagenen Auszeichnungen sind in tabellarischer Form im Abschnitt 6.4 zu finden.

Als abschließende Bewertung läßt sich zur Analyse folgendes sagen: Die Struktur des Textes ist auch ohne Verständnis hinreichend gut zu erfassen. Zur Definition von Suchmustern und Atomen ist diese Strukturierung ausreichend, wie die Tests gezeigt haben. Die Auswahl der Zielelemente ist jedoch nach einer Auszeichnung ohne Textverständnis meiner Meinung nach nicht ausreichend. Der Qualitätssprung, der bei einer angemessenen Behandlung des Textes zu erwarten ist, läßt sich nur durch fachlich kompetente Annotationen erzielen. Dies gilt umso mehr bei fehlenden Informationen, wie den oben angesprochenen Gleichungsbeschreibungen.

6.3 Umwandlung

Wegen der fehlenden logischen Metadaten war eine automatische Konvertierung in SGML/XML nicht möglich. Nach der zuvor beschriebenen strukturellen Analyse habe ich Auszeichnungen festgelegt, die von einem Mitarbeiter des Autors, Herrn Götz Uebe, in den Text eingearbeitet wurden. Die so aufbereiteten Texte habe mit den enthaltenen Grafiken per RTF-Import in Word eingeladen.

Überraschenderweise erwies sich die Umwandlung der WMF-Grafiken als ein unerwartet großes Problem, da viele Programme dieses

Format nur unzureichend unterstützen. Letzlich habe ich die Exportfunktion für die WWW-Ausgabe von Word benutzt. Da kein bekannter Web-Browser das Format WMF beherrscht, hatte ich die Hoffnung, daß Word eine Konvertierung durchführt. Tatsächlich liefert Word eine Ausgabe im Graphic Interchange Format (GIF), was von jedem graphischen Browser angezeigt wird.

Die manuell bearbeiteten Texte habe ich auf fehlerhafte Auszeichnungen hin untersucht. Einige Fehler ließen sich automatisch beheben, da sie in immer gleicher Form auftraten. Zu diesem Zweck habe ich die Programme *sed* und *lynx* zu Hilfe genommen[2]. In einem letzten Schritt habe ich verbliebene Fehler unter Verwendung eines SGML-Parsers und -Editors manuell entfernt. Zusammenfassend läßt sich sagen, daß der Prozeß der Umwandlung ein aufwendiger, aber unbedingt notwendiger Schritt war.

6.4 Auszeichnungen

Dieser Abschnitt enthält die Auszeichnungen, die ich zusammen mit einer Beschreibung für die Aufbereitung der Texte geschrieben habe. Im wesentlichen handelt es sich um drei Stufen: Struktur des Buches, Elemente auf Block- und Elemente auf Zeichenebene.

Buch-Struktur

```
<Buch>
      <Titelei>, <Rumpf>, <Anhangteil>
<Titelei>
      enthält Vorwort usw.
<Rumpf>
      ein oder mehrere Kapitel
<Anhangteil>
      <AnhangKeywords>, <AnhangModelle>,
      ggf. weitere <Kapitel>
<Kapitel>
      <Ueberschrift>, ein oder mehrere <Keywords>,
      ggf. einleitender Textblock, ein oder mehrere
      <Abschnitte>
<Abschnitt>
      <Ueberschrift>, ggf. einleitender Textblock,
      ein oder mehrere <Unterabschnitte>
<Unterabschnitt>
      <Ueberschrift>, ein oder mehrere Textblöcke
<AnhangKeywords>
      <KeywordEintrag>
<AnhangModelle>
      <ModellTabelle>
```

2 Den »UNIX Power Tools« [PORL93] schulde ich in diesem Zusammenhang meinen Dank!

Blöcke und Zeichenelemente

Innerhalb von (Unter-)Abschnitten ist der Text in *Blöcke* unterteilt. Sie zeichnen sich durch eine vertikale Abgrenzung aus (*vertical flow*). Beispiele für Blöcke sind *Absätze, Tabellen, Definitionen, Abbildungen, Aufzählungen* usw.

Innerhalb eines Blocks gibt es die feinste Strukturierung in Form von *Zeichenelementen*. Dies sind Elemente, die einzelne Worte, Sätze usw. markieren. Beispiele für Zeichenelemente sind *Hervorhebungen, Keyword, Acronym, Modellname, Zitate* usw.

Die folgenden Tabellen nennen alle Blöcke und Zeichenelemente, aber auch Merkmale, an denen ein Textstück zu erkennen ist. Um das Verständnis zu verbessern, habe ich die Bedeutung einer Auszeichnung angegeben.

Tabelle 2: Blöcke

Block	Inhalt	Bedeutung	Merkmale im vorhandenen Material
<Blockzitat>	Zeichenelemente	abgesetztes, längeres Zitat	abgesetztes, längeres Zitat
<Definition>	<DefUeberschrift>, Blöcke	math. (o.ä.) Definition	durch »Definition« eingeleitet
<Gleichungssystem>	<Gleichung>		math. Gleichungen
<Gleichung>	inhaltliche Beschreibung mit Keywords <Gleichungsbeschreibung>, Formel oder <Bild> mit einer Formel		ggf. nebengestellte Zahl in runden Klammern
numerierte Liste <NL>	Listenelemente <LE>		numerierte Liste
unnumerierte Liste <UL>	Listenelemente <LE>		unnumerierte Liste mit Spiegelstrichen
Begriffsliste <BL>	Begriffselemente <BE> und Begriffsbeschreibung <BB>		Begriffe mit eingerückter Erklärung

Tabelle 2: Blöcke

<Tabelle>	Tabellenzeilen <TR> mit Tabellenzellen <TH> und <TD>		Tabelle
Bibliographie-Liste <BibListe>	Liste von Bibliographie-Einträgen <BibEintrag>		Liste von Bibliographie-Einträgen mit vorangestellter Ziffer in eckigen Klammern
Bibliographie-Eintrag <BibEintrag id="...">	<Autor>, <Titel>, <Verlag>, <Jahr>	Enthält einen Bibliographie-Eintrag, der mit einem eindeutigen Namen versehen ist (an Stelle der drei Punkte). Leider sind die Namen im Buch nicht eindeutig. Die dort verwendete Numerierung beginnt in jedem Kapitel bei 1. Es bietet sich deshalb an, den Buchnummern die Kapitelnummer voranzustellen, um einen eindeutigen Namen zu erhalten; z.B. <BibEintrag id="bib3.5"> (das fünfte Buch im dritten Kapitel)	Bibliographie-Eintrag bestehend aus Autor, Titel, Verlag, Jahr
<Absatz>	Zeichenelemente		Absatz
<Illustration>	<Ueberschrift>, Block		durch das Wort »Illustration« eingeleitet

Tabelle 2: Blöcke

<Abbildung id="...">	<Bild>, <Bildtext>	Eine Abbildung. An Stelle der drei Punkte wird ein eindeutiger Name des Bildes eingefügt, um später darauf verweisen zu können. Es bietet sich an, die Namen in der Form "fig1.2.3" zu bilden, wobei die erste Zahl das Kapitel, die zweite Zahl den Abschnitt und die dritte Zahl die Nummer der Abbildung bezeichnet.	Abbildung
<Atom>	Blöcke	Dient der Zusammenfassung von Blöcken, die in der späteren Ausgabe nicht getrennt werden sollen.	noch nicht vorhanden
<KeywordEintrag>	<Keyword>, Keyword-Thema <KeyThema> (in 5 Sprachen), Keyword-Beschreibung <KeyBeschreibung> (in 5 Sprachen), ggf. weiterer Block mit Erläuterungen		ein Eintrag aus dem Anhang 1
<ModellTabelle>	<Land>, Modell-Einträge <Modell-Eintrag>		eine Tabelle aus dem Anhang 2
<ModellEintrag>	Modell bestehend aus <No>, <Autor>, <Equ>, <Sto>, <Def>, <Mon>, <IO>, <Exo>, <Per>, <ModellFamilie> <Inuse>		Tabellenzeile aus dem Anhang 2

Tabelle 2: Blöcke

<ModellFamilie>	entweder <Autor> und/oder <Modelltyp> (ggf. mehrere)		Letzte Spalte der Tabellen in Anhang 2
<Quelle>	genau ein <BibEintrag>		durch das Wort »Source« eingeleitet

Tabelle 3: Zeichenelemente

Zeichenelement	ggf. Bedeutung / Beschreibung	Merkmale im vorhandenen Material
Hervorhebung	wichtige Textstelle	kursiv
Starke Hervorhebung	besonders wichtige Textstelle	fett
<Fussnote>	Anmerkung	hochgestellte Nummer, Text am Seitenende
<Keyword>		Wort in GROSSBUCHSTABEN
<KeyThema>	Thema eines Keyword in 5 Sprachen: <de> <en> <fr> <es> <it>	Eintrag in Anhang 1
<KeyBeschreibung>	Beschreibung eines Keyword in 5 Sprachen: <de> <en> <fr> <es> <it>	Eintrag in Anhang 1
<de>		Text in Deutsch
<en>		Text in Englisch
<fr>		Text in Französisch
<es>		Text in Spanisch
<it>		Text in Italienisch
<BibRef idref="...">	Verweis auf ein Buch	Zahl in eckigen Klammern
math. <Formel>		math. Formeln im Fließtext (nicht in Gleichungen)
<Acronym>		GROSSBUCHSTABEN
<Person>	Name einer Person	Name einer Person
<Land>	Name eines Landes	Name eines Landes
<Zitat>	Zitat	in Anführungszeichen eingeschlossen

Tabelle 3: Zeichenelemente

<Modellname>	Name eines Modells	?
<Modelltyp>	Typ eines Modells: CGE IO LP M NLP SIM VAR	Typ eines Modells: CGE IO LP M NLP SIM VAR
<Querverweis idref="...">	Querverweis	fett, Nummer eines Abschnitts
<Gleichungsreferenz idref="...">	Verweis auf eine Gleichung	Nummer einer Gleichung in runden Klammern
<Modellreferenz idref="...">	Stellt eine Querverbindung zu einer Modellbeschreibung im grünen Buch her	noch nicht vorhanden
<Ueberschrift>	Überschrift eines Kapitels, eines Abschnittes usw.	Überschrift
<UNBEKANNT>	markiert eine Stelle im Text, die einer genaueren Bestimmung durch einen Experten bedarf	
<DefUeberschrift>	Überschrift einer Definition	In runden Klammern hinter dem Wort »Definition«
<Bild>		Bild, Zeichnung oder Formel ohne umgebende Beschreibung
<Bildtext>		Text zu einer Abbildung
<Gleichungsbeschreibung>	inhaltliche Beschreibung einer Gleichung	noch nicht vorhanden
Listenelemente <LE>		Ein Element einer Liste
Begriffselement <BE>		Begriff einer Begriffsliste
Begriffsbeschreibung <BB>		Beschreibung eines <BE>
Tabellenzeile <TR>		Tabellenzeile
Tabellenfeld mit Ueberschrift <TH>		fett gedrucktes Tabellenfeld
normales Tabellenfeld <TD>		normales Tabellenfeld
<Autor>		Autor

Tabelle 3: Zeichenelemente

<Buchtitel>		Buchtitel
<Verlag>		Verlag
<Jahr>		Jahr
<bildref id-ref="...">	Verweis auf ein Bild. An die Stelle der drei Punkte tritt der Name des Bildes (vgl. <Abbildung>)	Verweis auf ein Bild.

6.5 DSSSL-Rumpf zur Transformation in HTML

Als Teil meiner Diplomarbeit habe ich ein DSSSL-Programm geschrieben, das Instanzen der Uebe-DTD in HTML transformiert. Es übernimmt innerhalb von IP4W3 die Rolle des DSSSL-Rumpfes. Ein Listing des DSSSL-Stylesheets ist in Abschnitt B.3 zu finden.

Ausblick

7

In diesem Kapitel zähle ich einige Dinge auf, die ich aus verschiedenen Gründen (vor allem die begrenzte Zeit) nicht in IP4W3 implementieren konnte. Ich diskutiere aber auch Fragen, die sich bei der Benutzung vielleicht stellen, und deren mögliche Lösungen. Die einzelnen Abschnitte dieses Kapitels sind voneinander unabhängig und als Sammlung zu verstehen.

7.1 Suchen in mehreren Dokumenten

Eine Funktion, die vielleicht als erste in IP4W3 vermißt wird, ist die Suche in mehreren Dokumenten. Falls es sich bei den fraglichen Dokumenten um Instanzen von verschiedenen Dokumenttypen handelt, gibt es einen einfachen Grund dafür: Die Definition der Suchmuster und auch der Formatierungsrumpf sind untrennbar mit der DTD verbunden. Aus diesem Grund gibt es keine gemeinsamen, durch die Suchmuster definierten Kategorien, in denen die Suche stattfinden kann.

Handelt es sich jedoch um eine Sammlung von gleichartigen Dokumenten, d.h. Instanzen des selben Dokumenttyps, die gegebenenfalls auch in einem inhaltlichen Zusammenhang stehen, so ist der Wunsch nach einer Suchmöglichkeit in mehreren Dokumenten naheliegend.

Für diesen Fall braucht IP4W3 keine besondere Unterstützung anzubieten, da SGML/XML eigene Konstrukte zur Lösung dieses Problems bietet. Auf den ersten Blick ist dies eine typische Anwendung des Subdocument-Features von SGML[1]. Aufgrund des zuvor gesagten ist es jedoch in IP4W3 nicht notwendig oder sinnvoll, Dokumente unterschiedlicher Typen als Subdocument zu behandeln. Außerdem ist das Subdocument-Feature in XML nicht enthalten[2]. Das Problem kann viel einfacher durch den Entity-Mechanismus von SGML/XML

1 Zum Verständnis siehe Annex C.3.2 aus [ISO86], auch zu finden in [GOLD90], Seite 89.
2 Die SGML-Deklaration für XML enthält den Eintrag SUBDOC NO [CLAR97].

gelöst werden, indem mehrere Dokumente zu einem zusammengefaßt werden. Zur Erklärung diene folgendes Beispiel.

Es sollen drei Bücher, die Instanzen des selben Dokumenttyps sind, in IP4W3 integriert werden. Die Bücher besitzen die folgenden PIDs[3]:

```
"-//DEK//DOCUMENT Fundamental Algorithms//DE"
"-//DEK//DOCUMENT Seminumerical Algorithms//DE"
"-//DEK//DOCUMENT Sorting and Searching//DE"
```

Der PID der zugrundeliegenden DTD sei:

```
"-//Addison-Wesley//DTD cs professional//DE"
```

Das Wurzelelement dieses Dokumenttyps sei `buch`, die Doctype-Deklaration der drei Instanzen sieht also wie folgt aus[4]:

```
<!DOCTYPE buch
          PUBLIC "-//Addison-Wesley//DTD cs professional//DE">
```

Um diese drei Bücher nun zu einer Instanz zusammenzufassen, ist folgendes zu tun: Es muß eine DTD erstellt werden, die eine Folge von Elementen des Typs `buch` erlaubt und es muß eine Instanz erstellt werden, die die drei Bücher als Entities lädt.

Beide Fälle sind einfach zu handhaben. Die DTD sieht so aus[5]:

```
<!ELEMENT bibliothek - - (buch)+>
<!ENTITY % buch-dtd
           PUBLIC "-//Addison-Wesley//DTD cs professional//DE">
%buch-dtd;
```

Nach dieser Definition besteht eine `bibliothek` einfach aus einer Folge von Büchern. Die Instanz nimmt nun folgende Form an:

```
<!DOCTYPE bibliothek [
<!ELEMENT bibliothek - - (buch)+>
<!ENTITY % buch-dtd
           PUBLIC "-//Addison-Wesley//DTD cs professional//DE">
%buch-dtd;
]>
<!ENTITY aocp1
          PUBLIC "-//DEK//DOCUMENT Fundamental Algorithms//DE">
<!ENTITY aocp2
          PUBLIC "-//DEK//DOCUMENT Seminumerical Algorithms//DE">
<!ENTITY aocp3
          PUBLIC "-//DEK//DOCUMENT Sorting and Searching//DE">
```

---

3 Es handelt sich um fiktive PIDs für Donald E. Knuth' Serie »The Art of Computer Programming«. Es ist nicht unwahrscheinlich, daß Knuth seine Bücher *nicht* in SGML/XML geschrieben hat. Insofern ist das Beispiel ein weniger gutes.

4 Auf den System-Bezeichner verzichte ich in diesem Beispiel. Formal handelt es sich also nur um eine gültige SGML-Doctype-Deklaration. Um XML-Konformität zu erzielen muß ein System-Bezeichner ergänzt werden.

5 Auch hier handelt es sich um die SGML-Syntax gemäß Reference Concrete Syntax [ISO86]. Um XML-konform zu sein, muß auf die Markup-Minimierung (- -) verzichtet werden. In der nachfolgenden Instanz sollte dann noch die XML-Deklaration ergänzt werden.

```
<bibliothek>
&aocp1;
&aocp2;
&aocp3;
</bibliothek>
```

Soweit stellt die SGML-seitige Behandlung kein Problem dar. Es ist aber noch zu diskutieren, wie sich Suchmusterdefinitionen und DSSSL-Skripte verhalten, die für die Buch-DTD geschrieben wurden und nun zusammen mit der Bibliothek-DTD eingesetzt werden. In beiden Fällen sind Schwierigkeiten ausgeschlossen, die durch die Zusammenfassung der Instanzen in der beschriebenen Weise hervorgerufen werden. Der Grund besteht darin, daß sich sowohl Suchmuster als auch DSSSL-Skripte auf Elemente konzentrieren, die »unten« in der Hierarchie stehen. Im Gegensatz zu Suchmustern kann man in DSSSL zwar gezielt die Wurzel ansprechen, jedoch ist auch dadurch kein neues Problem entstanden. Vielmehr stellen sich hier Fragen, die auch bei einzelnen Instanzen auftreten. Die Diskussion wurde bereits in 5.2.4 geführt.

Abschließend kann man feststellen, daß keine Notwendigkeit besteht, eine besondere Unterstützung für die Behandlung von mehreren Dokumenten in IP4W3 vorzusehen. Die Mittel, die SGML/XML bereitstellen, sind sowohl ausreichend als auch einfach genug, um in IP4W3 den Standpunkt einzunehmen, daß man grundsätzlich genau ein Dokument vor sich hat.

7.2 Konformität zu Standards

Die Frage, in wie weit IP4W3 konform zu den verschiedenen Standards ist, ist relativ leicht zu beantworten. Da ich für den Zugriff auf Instanzen und die Interpretation von DSSSL-Stylesheets externe Programme verwende, leitet sich die Konformität direkt aus den Eigenschaften dieser Programme ab.

Soweit es das Parsing von SGML/XML betrifft, ist in IP4W3 durch die Benutzung der Programme *nsgmls* und *jade* die bestmögliche Einhaltung der entsprechenden Standards gewährleistet (dies sind [ISO86] und [W3C98D]). Eine eigene System-Deklaration für IP4W3 ist nicht nötig; implizit verwendet das System die entsprechende Deklaration des Parsers (`http://www.jclark.com/sp/sysdecl.htm`).

Bei der Behandlung von Suchmustern ist darauf zu achten, daß das Programm *Cost* case-insensitive arbeitet. Dieses Verhalten entspricht der Reference Concrete Syntax von SGML. XML ist jedoch case-sensitive. Der Autor von CoST, Joe English, arbeitet zur Zeit an einer Aktualisierung, die XML-konform sein wird. Sofern er dabei keine Änderungen am Verhalten seines Programms einbaut, sollten meine Programme unmittelbar lauffähig sein.

Als DSSSL-Maschine verwende ich *Jade*. Der Programmierer James Clark ist auch der Verfasser der DSSSL-Spezifikation [ISO96B], so daß man auch hier von Einhaltung des Standards ausgehen kann. Allerdings ist Jade keine vollständige Implementierung. Ausführlich ist dies in der Dokumentation nachzulesen [CLAR98B]. Insbesondere fehlt die Transformationssprache. Clark hat als Ersatz einige neue Flow Objects definiert, von denen ich bei der Transformation in HTML Gebrauch mache.

Bei der Web-Übertragung ist der Server für die Einhaltung des Protokolls (HTTP) verantwortlich. Der von IP4W3 ausgegebene HTTP-Kopf (`Content-type`) ist HTTP-1.0-konform und meine Skripte halten sich an die CGI-Konvention. Sie sind daher mit jedem Web-Server lauffähig, der CGI beherrscht.

7.3 Persistenz von URLs

Die Persistenz von URLs ist von Bedeutung, wenn man eine Verknüpfung (Link) zu einer Seite herstellen möchte, die von IP4W3 generiert wurde. Auf die grundsätzliche Problematik der Namen und Adressen möchte ich hier nicht eingehen. Dazu empfehle ich den Aufsatz »The Myth of Names and Addresses« von Tim Berners-Lee [BERN96A].

IP4W3 führt keine mutwillige Änderung von URLs durch. Das heißt, wenn das System keine Veränderung von außen erfährt, sind die von IP4W3 erzeugten URLs auch in Zukunft gültig. Zwei Probleme gibt es aber:

- ✗ Die von IP4W3 erzeugte Seite ist üblicherweise in einen Frame eingebettet. Setzt man einen Link auf den oberen Frame, der die gewünschte Seite enthält, so fehlt der untere Frame mit der Steuerung. Das Problem besteht darin, daß der Schalter »Ausschnitt merken« nur funktioniert, wenn der untere Frame existiert. Das Problem wird abgeschwächt, wenn der URL benutzt wird, den man erhält, wenn man in der Steuerung den Link »Gemerkte Ausschnitte zeigen« benutzt. In diesem Fall sind in dem generierten Dokument keine Formularfelder mehr enthalten.
- ✗ Das zweite Problem tritt auf, wenn am Ausgangsdokument Änderungen gemacht werden. Sobald sich dadurch die Numerierung der Elemente verändert, sind URLs, die sich auf den ehemaligen Zustand beziehen, ungültig, weil in ihnen veraltete Elementnummern enthalten sind.

7.4 Erweiterungen und Änderungen

7.4.1 Wertebereiche in Formularen

Als zulässige Wertebereiche in Textfeldern von Formularen habe ich `beliebig`, `buchstaben` und `posinteger` zugelassen. Ich verstehe diese Bereiche nur als exemplarische Implementation, da sich in der späteren Anwendung herausstellen muß, welche Wertebereiche tatsächlich benötigt werden. Aus diesem Grund habe ich auf eine einfache Erweiterbarkeit geachtet. Sollte also Bedarf für weitere Wertebereiche bestehen, so sind folgende Änderungen durchzuführen.

Änderung der Suchmuster–DTD

In der Suchmuster-DTD sind die Wertebereiche als zulässige Attributwerte aufgezählt[6]. Die *Erweiterung* ist problemlos möglich, alle älteren Instanzen bleiben auch bezüglich der veränderten DTD gültig. Es ist nicht notwendig, den Public Identifier (PID) der DTD zu ändern. Bei einer *Entfernung* eines Wertebereichs werden natürlich alle Suchmuster-Instanzen ungültig, die diesen Bereich benutzen. In diesem Fall sollte eine neue DTD mit einem neuen PID erzeugt werden.

Erweiterung der JavaScript–Bibliothek

Die Überprüfung der Wertebereiche wird von der Funktion `pruefe_wert()` durchgeführt. Bei der Einführung neuer Wertebereiche muß die Funktion angepaßt werden. Sie erhält den Namen des Wertebereichs als Zeichenkette in Großbuchstaben beim Aufruf übergeben.

7.4.2 Druckausgabe mit DSSSL

Um eine Druckausgabe mit IP4W3 zu realisieren, muß zu jedem Dokument ein passender DSSSL-Rahmen vorhanden sein. Dies ist eine Anforderung an den Administrator, nicht an das System. IP4W3 muß dem Benutzer erlauben, ein anderes Ausgabeformat als HTML, z.B. RTF, PostScript usw., zu wählen. Hierfür bietet es sich an, ein Formularelement zur Auswahl vom DSSSL-Rahmen ausgeben zu lassen (`format.dsssl`). Das ausgewählte Format muß vom CGI-Skript (`preview.pl`) erkannt und beim Zusammensetzen des Rahmens und des Rumpfes berücksichtigt werden.

---

6 SGML: Der [145] `declared value` ist eine [68] `name token group`; XML: Der [54] `AttType` ist eine [59] `Enumeration`

7.5 IP4W3 und Hypertext

Wie in der Einleitung beschrieben, habe ich den Hypertextansatz in IP4W3 zurückgestellt, um stattdessen ein Dokument aus Ausschnitten zusammenzusetzen. Ich möchte nun nicht den grundsätzlichen Ansatz in Frage stellen. Die Überschrift dieses Abschnitts erweckt vielleicht den Eindruck einer Grundsatzdiskussion über das Konzept des Hypertextes. Zweifellos sind die Ideen in der Tradition von Bush [BUSH45], Engelbart (z.B. [ENGE62]) und Nelson (Xanadu-Projekt) noch längst nicht angemessen realisiert, jedoch ist hier nicht der Platz, sie zu diskutieren. Vielmehr möchte ich an dieser Stelle auf einige *offene* Fragen hinweisen, die sich in Verbindung von IP4W3 mit der (bescheideneren) Form des Hypertextes stellen, die heute im World Wide Web Realität ist oder in Kürze sein wird.

Wie sind externe Links zu handhaben?

Bei meiner Arbeit konnte ich die Handhabung von externen Links ignorieren, da IP4W3 keine solchen Verknüpfungen herstellt. Der Autor eines DSSSL-Rahmens ist dazu aber durchaus in der Lage. Ein naheliegender Ansatz wäre, einen HTML-Link mit dem Attribut `target="_top"` auszustatten, um aus dem Frameset von IP4W3 auszubrechen.

Wie sind interne Links zu handhaben?

Dokumentinterne Links gehen in HTML davon aus, daß das Dokument nur eine *statische* HTML-Seite umfasst. Da IP4W3-Seiten aber dynamisch, als Ergebnis einer Suche, generiert, sind interne Links in der bekannten Form nicht möglich.

Welche Möglichkeiten werden die neuen XLinks für IP4W3 bieten?

Momentan arbeitet das W3C an einem gegenüber HTML stark erweiterten Link-Konzept. Es ist zu prüfen, ob die zukünftigen Möglichkeiten Nutzen für IP4W3 bieten.

7.6 Lernen aus Benutzerverhalten

IP4W3 wurde mit der Idee entworfen, daß das System im Rahmen einer weiteren Diplomarbeit ergänzt wird, um aus dem Verhalten der Benutzer zu lernen. Bei der dadurch zu erzielenden Verbesserung kann es sich zum Beispiel darum handeln, welche Treffer zu einer Suchanfrage in welcher Reihenfolge zurückgeliefert werden. Treffer, die in der Vergangenheit als »richtige« Treffer angesehen wurden, könnten bei einer neuen Suchanfrage als erste in der Trefferliste erscheinen. Eine weitere Verbesserung betrifft die Ermittlung der Atome. Es ist nicht ausgeschlossen, daß die auf Grundlage der DTD

definierten Atome nicht bei allen Instanzen der DTD für optimale Ergebnisse sorgen. Der Benutzer wird dies dadurch kompensieren, daß er die gelieferten Atome verwirft oder den Ausschnitt vergrößert. Dieses Verhalten kann benutzt werden, um bei der nächsten Suche »bessere« Atome zurückzuliefern. Letzlich kann man die genannten Ansätze auf jeden Benutzer individuell anwenden oder man kann die Menge aller Benutzer gemeinsam behandeln. Dahinter stünde die Annahme, daß zwei verschiedene Benutzer die gleichen Wünsche bezüglich der Trefferliste und der Atome haben.

Ich möchte hier zwei Punkte besprechen, die zur technischen Umsetzung nötig sind:

Logging des Benutzerverhaltens

Da Web-Transaktionen zustandslos sind, muß man selbst dafür sorgen, nacheinander eintreffende Anfragen an den Webserver eindeutig einem Benutzer zuzuordnen. Zu diesem Zweck bietet es sich an, jedem Benutzer eine eindeutige ID zu geben. Es stehen zwei Möglichkeiten zur Verfügung: Kodierung der ID in versteckte Formularfelder (etwa `<input type="hidden" value="xyz" name="UID">`). Dazu müßte jedes Formular dynamisch generiert werden. Die betroffenen Komponenten von IP4W3 sind `search.pl`, `preview.pl` und `format.dsssl`. Die Alternative besteht in der Verwendung von Cookies. Dazu wird dem Benutzer beim Einstieg in das System eine ID als Cookie gegeben, die der Browser speichert und bei Zugriff auf vordefinierte Adressen wieder an den Server schickt. Der erste Cookie sollte dann von `search.pl` vergeben werden.

Was ist ein Endergebnis?

Wenn man versucht, eine Beziehung zwischen der ursprünglichen Anfrage und dem Endergebnis herzustellen, stellt sich die Frage, was ein Endergebnis ist beziehungsweise wie man es erkennt? Abstrakt kann man sagen: Das Endergebnis ist das Dokument, das der Benutzer als angemessene Antwort auf seine Suchanfrage ansieht. Da man nicht erkennen kann, ob der Benutzer das System irgendwann frustriert und erfolglos verlassen hat, ist es nicht unbedingt sicher, daß die letzte betrachtete Seite die gewünschte war. Es bietet sich jedoch an, die Dokumentausschnitte, die sich der Benutzer gemerkt hat, als gutes Ergebnis anzusehen. IP4W3 bekommt diese Information, wenn der Benutzer auf »gemerkte Ausschnitte zeigen« klickt. Dann nämlich wird das Skript `preview.pl` mit dem CGI-Parameter `keinesteuerung=true` aufgerufen (was dafür sorgt, daß die unschönen Formularelemente verschwinden). Da es nicht ausgeschlossen ist, daß ein Benutzer diesen Link mehrfach anklickt, sollte der letzte Zugriff gewertet werden.

7.7 Trennung von Suchmustern und Atomen

Konzeptionell unterscheide ich zwischen Suchmustern und Atomen. In der Implementierung werden Atome jedoch direkt aus Suchmustern abgeleitet. Die Praxis muß zeigen, ob ein komplexerer Atombegriff notwendig ist und ob dieser dann von den Suchmustern abweicht, oder ob diese mitwachsen.

Ein Beispiel für die Differenzierung beider Begriffe sei genannt: Es soll in einer Sammlung von Briefen die Suche nach dem Namen des Adressaten möglich sein. Das Resultat einer Suche soll aber weder der Name noch die Adresse des Empfängers sein, sondern der Brieftext, der an ihn geschickt wurde. Die Struktur eines Briefes sehe so aus:

```
<brief>
 <absender>...</absender>
 <adressat>...</adressat>
 <text>...</text>
</brief>
```

Das Suchmuster zu definieren, ist problemlos möglich. Jedoch ist das gewünschte zugehörige Atom (`text`) kein Nachfahre und auch kein Vorfahre des Suchmusters, so daß es in der Suchmuster-Definition gar nicht vorkommt.

Ob dieses Szenario realistisch ist und ob es wirklich ein Problem darstellt, ist sicher diskussionswürdig. Eine naheliegende Lösung für das Beispiel wäre es sicher, den gesamten Brief als Ergebnis zurückzuliefern. Es ist nicht ganz unwahrscheinlich, daß der Benutzer von einem Suchergebnis verwirrt wäre, in dem der Suchbegriff gar nicht vorkommt. Die Verknüpfung beider Konzepte über den Suchvorgang in IP4W3 spricht nach meiner Auffassung gegen eine Trennung von Suchmustern und Atomen. Die Situation könnte sich ändern, sobald Web-Browser die neuen, erweiterten Hyperlinks XLink und XPointer unterstützen. Mit XPointern lassen sich Dokumentstellen ähnlich adressieren, wie es momentan implizit durch Atome in IP4W3 geschieht. Mit einer Unterstützung durch die Browser könnte der Suchvorgang von IP4W3 in Zukunft Hyperlinks (verschiedene Ausprägungen von XLinks) als Ergebnis liefern, wobei die Atome als XPointer realisiert wären. Diese Idee bietet einen Ansatz, IP4W3 mit einem Hypertextkonzept zu verbinden.

Benutzung des Systems

A

In diesem Anhang beschreibe ich die Benutzung der ausführbaren Programme von IP4W3 im Stil von Unix-Manualseiten. Alle anderen Komponenten werden nicht über die Kommandozeile gestartet, sondern durch den Web-Server beziehungsweise die CGI-Skripte aufgerufen.

A.1 ip4w3-manager

Name

`ip4w3-manager` - Administrationswerkzeug für IP4W3

Aufruf

```
ip4w3-manager [init|neu|update DokID]
```

Beschreibung

Der `ip4w3-manager` dient dazu, ein neues Dokument in IP4W3 zu integrieren beziehungsweise IP4W3 darüber zu informieren, daß sich ein Dokument geändert hat.

Benutzung

`ip4w3-manager init` führt die Initialisierung für ein neues Dokument durch. Bei diesem Schritt muß ein eindeutiger Bezeichner für das Dokument genannt werden. Der Bezeichner wird als Verzeichnisname und innerhalb eines URLs verwendet und unterliegt den entsprechenden Beschränkungen. Es empfiehlt sich, einen Bezeichner mit einem Buchstaben zu beginnen und nur aus Buchstaben und Ziffern zusammenzusetzen. Darüber hinaus sollten nur kleine Buchstaben benutzt werden.

`ip4w3-manager neu` nimmt das neue Dokument in IP4W3 auf. Neben der Angabe des Dokumentbezeichners fragt das Programm nach dem Dateinamen der Instanz, dem Dateinamen des DSSSL-Rumpfes, dem Dateinamen der Suchmuster-Instanz sowie dem Titel des Dokuments.

`ip4w3-manager update DokID` aktualisiert alle internen Informationen des Dokuments mit dem Bezeichner `DokID`. Dies ist notwen-

dig, wenn am Dokument oder den Suchmustern Änderungen vorgenommen wurden.

A.2 makeindex

Name

`makeindex` - erzeugt ein Stichwortverzeichnis für eine SGML/XML-Instanz

Aufruf

```
makeindex sgml-instanz [nsgmls-parameter]
```

Beschreibung

`makeindex` erzeugt ein Stichwortverzeichnis, das jedem Wort einer gegebenen Instanz den Wert des Attributs `internal-id` des umgebenden Elements zuordnet.

Benutzung

Der Parameter `sgml-instanz` ist zwingend erforderlich und gibt den Dateinamen der Instanz an. Die weiteren Parameter werden an nsgmls weitergereicht. Die Ausgabe erfolgt auf der Standardausgabe.

A.3 ergaenze-ids

Name

`ergaenze-ids` - fügt IDs in SGML/XML-Instanzen ein

Aufruf

```
ergaenze-ids sgml-instanz [-o ausgabedatei] [jade-parameter]
```

Beschreibung

`ergaenze-ids` fügt in jedes Element einer gegebenen Instanz das Attribut `internal-id` mit einem eindeutigen Wert ein.

Benutzung

Der Parameter `sgml-instanz` ist zwingend erforderlich und gibt den Dateinamen der Instanz an. Mit dem optionalen Parameter `-o ausgabedatei` kann die Ausgabe in eine Datei umgelenkt werden; andernfalls erfolgt die Ausgabe auf der Standardausgabe. Die weiteren optionalen Parameter werden direkt an Jade weitergeleitet.

Dokumenttyp-Definitionen und DSSSL-Stylesheets

B

Dieser Anhang enthält die Sammlung der DTDs und Stylesheets, die ich im Rahmen dieser Diplomarbeit entworfen haben. Um die Lesbarkeit des Textes zu verbessern, habe ich sie an das Ende der Arbeit plaziert.

B.1 DTD für Suchmuster

Das folgende Listing zeigt die DTD für ein Suchmuster.

```
<!--


  DTD für Suchmuster

  Teil einer Diplomarbeit am
  Lehrstuhl für Künstliche Intelligenz
  Fachbereich Informatik
  Uni Dortmund
  Germany


  (c) 1998 Stefan Mintert


  Benutzung:

  <!DOCTYPE PUBLIC
            "-//mintert.com//DTD IP4W3 Suchmuster 1.0//DE">


  Formal Public Identifier (PID):
  "-//mintert.com//DTD IP4W3 Suchmuster 1.0//DE"

-->

<!-- ======== Entity-Deklarationen ======== -->

<!ENTITY % elemcont "element | any">
<!ENTITY % elematt  "type  CDATA  #IMPLIED">
<!ENTITY % telematt "type  CDATA  #REQUIRED">
<!ENTITY % attatt   "name  CDATA  #REQUIRED
                     value CDATA  #IMPLIED"  >
```

```
<!-- ======== Element-Deklarationen ======== -->

<!ELEMENT suchmuster - -  (element | target-element) >
<!ATTLIST suchmuster id         ID      #REQUIRED >
<!ELEMENT element    - -  (attribute*, (%elemcont; |
                           target-element) ) >
<!ATTLIST element
    %elematt;
    >
<!ELEMENT target-element - -    (beschreibung, form,
                                 attribute*)>
<!ATTLIST target-element
    %telematt;
    >


<!-- informelle Beschreibung des Suchmusters;
     sollte kurz sein, dient zur Ausgabe auf
     der HTML-Seite
-->
<!ELEMENT beschreibung - - (#PCDATA) >

<!--
Das Form-Element enthält informationen darüber, wie das
Suchmuster in ein korrespondierendes HTML-Formular umgewandelt
werden soll; zwei Möglichkeiten: 1-aus-n-Auswahl,
oder freie Eingabe (Text)
-->
<!ELEMENT form - - (select|textinput) >

<!-- select ist ein auswahlfeld und besteht aus... -->
<!ELEMENT select - - (option+) >

<!--
...ein oder mehr Optionen. Jede Option benötigt eine textuelle
Beschreibung, die im HTML-Formular erscheint sowie einen
optionalen Wert (Attribut value), nach dem gesucht wird. Falls
value fehlt, wird der Inhalt als Wert angenommen.
-->
<!ELEMENT option - - (#PCDATA) >
<!ATTLIST option
          value CDATA #IMPLIED >


<!--
textinput erlaubt die freie Eingabe eines Suchbegriffs;
über das Attribut wertebereich läääßt sich der Wertebereich
festlegen
-->
<!ELEMENT textinput - - EMPTY >
<!ATTLIST textinput
          wertebereich
                   (beliebig|buchstaben|posinteger) beliebig >


<!ELEMENT attribute - - EMPTY>
<!ATTLIST attribute %attatt;>
<!ELEMENT any - - (%elemcont; | target-element)>
```

Da eine Anwendung in IP4W3 normalerweise mehr als ein Suchmuster benötigt, muß es ein Element geben, das mehrere Elemente vom Typ `suchmuster` aufnehmen kann. Zu diesem Zweck habe ich folgende DTD geschrieben, die lediglich das Wurzelelement `ip4w3project` deklariert. Für den modularen Aufbau der DTDs habe ich mich entschieden, um zukünftige Änderungen zu begünstigen.

```
<!--


    DTD für das System IP4W3

    Teil einer Diplomarbeit am
    Lehrstuhl für Künstliche Intelligenz
    Fachbereich Informatik
    Uni Dortmund
    Germany


    (c) 1998 Stefan Mintert


    Benutzung:

    <!DOCTYPE PUBLIC
              "-//mintert.com//DTD IP4W3 Project 1.0//DE">


    Formal Public Identifier (PID):
    "-//mintert.com//DTD IP4W3 Project 1.0//DE"

-->
<!ELEMENT ip4w3project - - (suchmuster)+>
<!ENTITY % suchmuster-dtd
          PUBLIC "-//mintert.com//DTD IP4W3 Suchmuster 1.0//DE">
%suchmuster-dtd;
```

B.2 DTD für die Uebe-Bücher

Aufbauend auf der in Kapitel 6 beschriebenen Textanalyse habe ich folgende Dokumenttyp-Definition entwickelt. Sie orientiert sich an den vorgegebenen Büchern [UEBE92] und [UEBE95], ist jedoch auch geeignet, andere Schriften des gleichen Typs zu beschreiben. Zusammen mit dieser DTD lassen sich die bisher umgewandelten Kapitel fehlerfrei verarbeiten (insbesondere einwandfreies Parsing). Dennoch ist diese DTD als unvollständig anzusehen, da die noch fehlenden Kapitel für ihren Entwurf nicht herangezogen werden konnten.

Die DTD stellt keine besonderen Anforderungen an die SGML-Deklaration. Beim Parsing mit *nsgmls* habe ich die implizite System-Deklaration des Parsers verwendet (siehe `www.jclark.com`). Sollen die Instanzen als XML-Instanzen betrachtet werden, so ist lediglich die

Markup-Minimierung (- -) aus der DTD zu entfernen. Sämtliche Inhaltsmodelle sind XML-konform.

```
<!ENTITY % block " Abbildung | absatz | gleichungssystem |
tabelle | nl | ul | bl | bibliste | illustration | definition |
UNBEKANNTERBLOCK " >


<!ENTITY % zeichen "#PCDATA | em | br | strong | Fussnote | Key-
word | KeyThema | KeyBeschreibung | BibRef | Formel | Acronym |
Person | Land | Zitat | Modellname | Modelltyp | Querverweis |
Gleichungsref | Modellreferenz | Autor | UNBEKANNT | Buchtitel |
Verlag | Jahr | bildref| tabref | sub | sup | bild | stadt  |
defref" >



<!-- ================== Parameter Entities ================ -->



<!ENTITY % coreattrs
 "id          ID      #IMPLIED  -- document-wide unique id --"
  >

<!ENTITY % refattrs
 "idref      IDREF #REQUIRED  -- document-wide unique id --"
  >

<!-- ================== Gliederung ======================== -->


<!ELEMENT kapitel  - - (ueberschrift , keyword* , (%block;)* ,
                        abschnitt+) >
<!ATTLIST kapitel
  %coreattrs;
>


<!ELEMENT abschnitt  - - (ueberschrift , (%block;)* ,
                          unterabschnitt*) >
<!ATTLIST abschnitt
  %coreattrs;
>
<!ELEMENT unterabschnitt  - - (ueberschrift , (%block;)* ,
                               unterunterabschnitt*) >
<!ATTLIST unterabschnitt
  %coreattrs;
>
<!ELEMENT unterunterabschnitt  - - (ueberschrift , (%block;)+) >
<!ATTLIST unterunterabschnitt
  %coreattrs;
>


<!-- ================== Blöcke ========================== -->

<!ELEMENT absatz  - - (%zeichen;)* >
```

```
<!-- Listen -->
<!ELEMENT nl  - - (le+) >
<!ELEMENT le  - - ((%zeichen;)* | (%block;)*) >
<!ELEMENT ul  - - (le+) >
<!ELEMENT bl  - - (be,bb)+ >
<!ELEMENT be  - - (%zeichen;)* >
<!ELEMENT bb  - - ((%zeichen;)* | (%block;)*) >


<!-- Tabelle -->
<!ELEMENT tabelle  - - (tabellentext?,tr+) >
<!ATTLIST tabelle
  %coreattrs;
>
<!ELEMENT tr  - - (th|td)+ >
<!ELEMENT th  - - ((%zeichen;)* | (%block;)*) >
<!ELEMENT td  - - ((%zeichen;)* | (%block;)*) >
<!ELEMENT tabellentext  - - (%zeichen;)* >

<!-- mathematische Texte -->
<!ELEMENT gleichungssystem  - - (Gleichung+) >
<!ELEMENT Gleichung  - - (%zeichen;)* >
<!ATTLIST Gleichung
  %coreattrs;
>


<!ELEMENT Definition  - - (defUeberschrift, (%block;)+) >
<!ATTLIST Definition
  %coreattrs;
>
<!ELEMENT Blockzitat  - - (%zeichen;)* >



<!ELEMENT bibliste     - - (bibeintrag+) >
<!ELEMENT bibeintrag   - - (#PCDATA|titel |verlag|jahr|autor|
                            person )* >
<!ATTLIST bibeintrag
  %coreattrs;
>
<!ELEMENT titel   - - (#PCDATA) >
<!ELEMENT Verlag  - - (#PCDATA) >
<!ELEMENT Jahr    - - (#PCDATA) >

<!ELEMENT illustration  - - (ueberschrift, (%block;)+) >
<!ATTLIST illustration
  %coreattrs;
>



<!ELEMENT abbildung  - - (bild,bildtext?) >
<!ATTLIST abbildung
  %coreattrs;
>
<!ELEMENT bild  - o EMPTY >
<!ATTLIST bild
  src   CDATA#REQUIRED
>
```

```
<!ELEMENT bildtext  - - (%zeichen;)* >


<!-- ================== Zeichen =========================== -->
<!ELEMENT em               - - (#PCDATA) >
<!ELEMENT br               - o EMPTY >
<!ELEMENT strong           - - (#PCDATA) >
<!ELEMENT Fussnote         - - (%zeichen;)* >
<!ELEMENT Keyword          - - (#PCDATA) >
<!ELEMENT KeyThema         - - (#PCDATA) >
<!ELEMENT KeyBeschreibung  - - (#PCDATA) >
<!ELEMENT Formel           - - (#PCDATA) >
<!ELEMENT Acronym          - - (#PCDATA) >
<!ELEMENT Person           - - (#PCDATA) >
<!ELEMENT stadt            - - (#PCDATA) >
<!ELEMENT Land             - - (#PCDATA) >
<!ELEMENT Zitat            - - (#PCDATA) >
<!ELEMENT Modellname       - - (#PCDATA) >
<!ELEMENT Modelltyp        - - (#PCDATA) >

<!ELEMENT BibRef  - o EMPTY >
<!ATTLIST bibref
  %refattrs;
>
<!ELEMENT Querverweis  - o EMPTY >
<!ATTLIST Querverweis
  %refattrs;
>
<!ELEMENT Gleichungsref  - o EMPTY >
<!ATTLIST Gleichungsref
  %refattrs;
>
<!ELEMENT Modellreferenz  - o EMPTY >
<!ATTLIST Modellreferenz
  %refattrs;
>
<!ELEMENT bildref  - o EMPTY >
<!ATTLIST bildref
  %refattrs;
>
<!ELEMENT tabref  - o EMPTY >
<!ATTLIST tabref
  %refattrs;
>
<!ELEMENT defref  - o EMPTY >
<!ATTLIST defref
  %refattrs;
>

<!ELEMENT Autor           - - (#PCDATA) >
<!ELEMENT Buchtitel       - - (#PCDATA) >
<!ELEMENT ueberschrift    - - (%zeichen;)* >
<!ELEMENT sup             - - (#PCDATA) >
<!ELEMENT sub             - - (#PCDATA) >
<!ELEMENT defUeberschrift - - (%zeichen;)* >
```

```
<!-- alles, wass noch zu klären ist: -->
<!ELEMENT UNBEKANNT         - - ANY >
<!ELEMENT UNBEKANNTERBLOCK  - - ANY >
<!ELEMENT TODO              o o ANY >




<!-- ================================================ -->

<!-- Character References -->

<!ENTITY amp     CDATA "&#38;"
                 -- ampersand, u+0026 ISOnum -->
<!ENTITY lt      CDATA "&#60;"
                 -- less-than sign, u+003C ISOnum -->
<!ENTITY gt      CDATA "&#62;"
                  -- greater-than sign, u+003E ISOnum -->
```

B.3 DSSSL-Stylesheet für die Uebe-DTD

Das folgende Listing zeigt das DSSSL-Stylesheet, das ich für die Uebe-DTD geschrieben habe.

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style
Sheet//EN">

(declare-flow-object-class element
  "UNREGISTERED::James Clark//Flow Object Class::element")
(declare-flow-object-class empty-element
  "UNREGISTERED::James Clark//Flow Object Class::empty-element")
(declare-flow-object-class entity-ref
  "UNREGISTERED::James Clark//Flow Object Class::entity-ref")
(declare-flow-object-class document-type
  "UNREGISTERED::James Clark//Flow Object Class::document-type")
(declare-characteristic preserve-sdata?
  "UNREGISTERED::James Clark//Characteristic::preserve-sdata?"
  #t)

(define (copy-attributes #!optional (nd (current-node)))
  (let loop ((atts (named-node-list-names (attributes nd))))
    (if (null? atts)
        '()
        (let* ((name (car atts))
               (value (attribute-string name nd)))
          (if value
              (cons (list name value)
                    (loop (cdr atts)))
              (loop (cdr atts)))))))


(default
      (process-children))

; ==================================== Xrefs

(mode xref
```

```
(element KAPITEL
  (process-matching-children 'ueberschrift))

(element (KAPITEL ueberschrift)
  (literal (ancestor-kind-nummer "Kapitel")))


(element Abschnitt
  (process-matching-children 'ueberschrift))

(element (Abschnitt ueberschrift)
  (literal (string-append
        (let ((kapnum (ancestor-kind-nummer "Kapitel")))
      (string-append kapnum "."))
        (let ((absnum (ancestor-kind-nummer "Abschnitt")))
      absnum))))

(element Unterabschnitt
  (process-matching-children 'ueberschrift))

(element (Unterabschnitt ueberschrift)
  (literal (string-append
        (let ((kapnum (ancestor-kind-nummer "Kapitel")))
      (string-append kapnum "."))
        (let ((absnum (ancestor-kind-nummer "Abschnitt")))
      (string-append absnum "."))
        (let ((uabsnum (ancestor-kind-nummer
                        "Unterabschnitt")))
      uabsnum))))


(element Unterunterabschnitt
  (process-matching-children 'ueberschrift))

(element (Unterunterabschnitt ueberschrift)
  (literal (string-append
        (let ((kapnum (ancestor-kind-nummer "Kapitel")))
      (string-append kapnum "."))
        (let ((absnum (ancestor-kind-nummer "Abschnitt")))
      (string-append absnum "."))
        (let ((uabsnum (ancestor-kind-nummer
                        "Unterabschnitt")))
      (string-append uabsnum "."))
        (let ((uuabsnum (ancestor-kind-nummer
                         "UnterUnterabschnitt")))
      uuabsnum))))

(element bibeintrag
  (literal (string-append "[" (element-nummer
                                (current-node)) "]")))

(element abbildung
  (process-matching-children 'bildtext))

(element (abbildung bildtext)
  (literal (string-append "figure "
                    (element-nummer (current-node)))))

(element tabelle
  (process-matching-children 'tabellentext))
```

```
  (element (tabelle tabellentext)
    (literal (string-append "table "
                 (element-nummer (current-node)))))

  (element (gleichungssystem gleichung)
    (literal (string-append "("
        (element-nummer (current-node)) ")")))

  )


(element bibref
  (let ((ziel-id (attribute-string "idref")))
    (with-mode xref (process-element-with-id ziel-id))))


(element Gleichungsref
  (let ((ziel-id (attribute-string "idref")))
    (with-mode xref (process-element-with-id ziel-id))))


(element bildref
  (let ((ziel-id (attribute-string "idref")))
    (with-mode xref (process-element-with-id ziel-id))))


(element tabref
  (let ((ziel-id (attribute-string "idref")))
    (with-mode xref (process-element-with-id ziel-id))))


;(element ref
;  (let ((ziel-id (attribute-string "idref")))
;    (with-mode xref (process-element-with-id ziel-id))))




; ==================================== Fussnoten

(mode kap-fussnoten
  (default (process-node-list
        (select-elements
         (descendants
          (current-node))
         "fussnote")))
  (element fussnote
    (let ((zieladresse
       (string-append
        "fussnote"
        (element-nummer (current-node)))))
      (make element gi: "tr"
        (make sequence
          (make element gi: "td"
            attributes: (cons (list "valign" "top")
                      '())
            (make element gi: "A"
              attributes: (cons (list "NAME" zieladresse)
```

```
                          '())
                (literal (element-nummer (current-node)))))
            (make element gi: "td"
              (with-mode #f (process-children)))))))
  )

(element fussnote
  (let ((zieladresse (string-append
              (string-append "#" "fussnote")
              (element-nummer (current-node)))))
;    (make element gi: "A"         ; keine Hyperlinks
;     attributes: (cons (list "HREF" zieladresse)
;               '())
      (make element gi: "SUP"
        (make element gi: "FONT"
              attributes: (attribute
                   (list
                    "SIZE"
                    "-2"
                    "class"
                    "fussnotennummer"))
              (literal (element-nummer (current-node))))
        )
;     )
       )
)

; ==================================== RUMPF

(element KAPITEL
;  (make element gi: "body"
;   attributes: (attribute (list "bgcolor" "white"))
    (make sequence
      (process-children)
      (if (node-list=? (empty-node-list)
               (select-elements
                (descendants
                 (current-node))
                "fussnote"))
          (empty-sosofo)
          (make sequence
        (footseprule)
        (make element gi: "strong"
              (literal "Footnotes"))
        (br)
        (make element gi: "table"
              (with-mode kap-fussnoten (process-children)))
        (chapseprule))))
;   )
  )


(element abschnitt
  (process-children))
(element unterabschnitt
  (process-children))
(element unterunterabschnitt
  (process-children))
```

```
; ==================================== Ueberschriften

(element (KAPITEL ueberschrift)
  (let ((kapnummer (ancestor-kind-nummer "Kapitel"))
    (kaptext   (if (node-list=?
            (ancestor "ANHANGTEIL")
            (empty-node-list)) ; nicht im Anhang
               "Chapter"
               "Appendix")))
    (make sequence
      (make element gi: "H1"
        attributes: (attribute (list "class" "kapnummer"))
        (make element gi: "A"
          attributes: (attribute (list "name" kapnummer ))
          (make sequence
            (literal (string-append kaptext " "))
            (literal kapnummer))))
      (make element gi: "H1"
        (make element gi: "span"
          attributes: (attribute (list "class" "kaptitel"))
          (process-children))))))

(element (KAPITEL Abschnitt ueberschrift)
  (make element gi: "H2"
    (let ((praefix (string-append
            (string-append (ancestor-kind-nummer "Kapitel") ".")
            (ancestor-kind-nummer "Abschnitt"))))
      (make element gi: "A"
        attributes:(cons (list "name" praefix )
                 '()
                 )
        (make sequence
          (literal praefix)
          (literal " ")
          (process-children))))))



(element (KAPITEL Abschnitt Unterabschnitt ueberschrift)
  (make element gi: "H3"
    (let ((praefix (string-append
            (let ((kapnum (ancestor-kind-nummer "Kapitel")))
              (string-append kapnum "."))
            (let ((absnum (ancestor-kind-nummer "Abschnitt")))
              (string-append absnum "."))
            (let ((uabsnum (ancestor-kind-nummer
                  "Unterabschnitt")))
              uabsnum))))
      (make element gi: "A"
        attributes:(cons (list "name" praefix )
                 '()
                 )
        (make sequence
          (literal praefix)
          (literal " ")
          (process-children))))))

(element (KAPITEL Abschnitt Unterabschnitt
          Unterunterabschnitt ueberschrift)
  (make element gi: "H4"
```

```
    (let ((praefix (string-append
            (let ((kapnum (ancestor-kind-nummer "Kapitel")))
              (string-append kapnum "."))
            (let ((absnum (ancestor-kind-nummer "Abschnitt")))
              (string-append absnum "."))
            (let ((uabsnum (ancestor-kind-nummer
                   "Unterabschnitt")))
              (string-append uabsnum "."))
            (let ((uuabsnum (ancestor-kind-nummer
                   "unterunterabschnitt")))
              uuabsnum))))
      (make element gi: "A"
        attributes:(cons (list "name" praefix )
                 '()
                 )
        (make sequence
          (literal praefix)
          (literal " ")
          (process-children))))))

; ==================================== normale Bloecke

(element absatz
  (make element gi: "p"))

(element gleichungssystem
  (make element gi: "div"
    attributes: (attribute (list
                "align"
                "center"
                "class"
                "gleichungssystem"))
    (make element gi: "table"
          attributes: (attribute (list
                      "align"
                      "center")))))

(element (gleichungssystem gleichung)
  (make element gi: "tr"
    attributes: (attribute (list "class" "gleichung"))
    (make sequence
      (make element gi: "td"
        (literal (string-append
              "("
              (element-nummer (current-node)) ")")))
      (make element gi: "td"
        (process-children)))))



(element abbildung
  (make element gi: "div"
    attributes: (attribute (list
                "align"
                "center"
                "class"
                "abbildung"))))

(element (abbildung bild)
  (make element gi: "p"
```

```
    (make element gi: "img"
          attributes: (copy-attributes))))
(element (abbildung bildtext)
  (make element gi: "p"
    (make sequence
      (make element gi: "strong"
        (literal (string-append
              "Figure "
              (element-nummer (current-node))
              ": ")))
      (process-children-trim))))


(element definition
  (make element gi: "blockquote"
    attributes: (attribute (list "class" "definition"))))

(element (definition defUeberschrift)
  (make sequence
    (make element gi: "strong"
        (literal (string-append
              "Definition "
              (element-nummer (current-node)) ": ")))
    (process-children)))


(element illustration
  (make element gi: "div"
    attributes: (attribute (list "class" "illustration"))))

(element (illustration ueberschrift)
  (make sequence
    (make element gi: "strong"
        (literal (string-append
              "Illustration "
              (element-nummer
               (current-node)) ": ")))
    (process-children)))


(element unbekannt
  (make element gi: "pre"))

(element unbekannterblock
  (make element gi: "pre"))

(element bild
  (make element gi: "img"
    attributes: (copy-attributes)))

; ==================================== Listen
(element nl
  (make element gi: "ol"))
(element ul
  (make element gi: "ul"))
(element bl
  (make element gi: "dl"))
(element le
  (make element gi: "li"))
(element be
```

```
  (make element gi: "dt"
    (make element gi: "strong")))
(element bb
  (make element gi: "dd"))


; ==================================== Tabellen


(element tabelle
  (make element gi: "div"
    attributes: (attribute (list "align" "center"))
    (make element gi: "table"
          attributes: (attribute (list "border" "1")))))

(element tr
  (make element))
(element th
  (make element))
(element td
  (make element))
(element (tabelle tabellentext)
  (make element gi: "caption"
    (make sequence
      (make element gi: "strong"
        (literal (string-append
              "Table "
              (element-nummer (current-node)) ": )))
      (process-children-trim))))



(element bibliste
  (make element gi: "table"))
(element bibeintrag
  (make element gi: "tr"
    (make sequence
      (make element gi: "td"
        attributes: (attribute (list "valign" "top"))
        (literal (string-append
              "["
              (element-nummer
               (current-node)) "]")))
      (make element gi: "td"
        (process-children)))))
(element (bibeintrag titel)
  (make element gi: "em"
    attributes: (attribute (list "class" "buchtitel"))))
(element (bibeintrag autor)
  (make element gi: "strong"
    attributes: (attribute (list "class" "buchautor"))))



; ==================================== Zeichen

(element sub
  (make element))
(element sup
  (make element))
```

```
; ======================================================
; Hilfsfunktionen
;

; (attribute aliste) wandelt die aliste in eine Form um,
; die fuer die attributes-Charakteristik eines
; Flow-Objects geeignet ist
(define (attribute aliste)
  (if (null? aliste)
      (list)
      (cons (list (car aliste) (car (cdr aliste)))
        (attribute (cdr (cdr aliste))))))

;
; (kind-nummer node) liefert die child-number in
; formatierter Form
; d.h. bei Kapiteln im Anhangteil werden Buchstaben verwendet,
; sonst Ziffern
;
(define (kind-nummer node)
  (let ((nummer (child-number node)))
    (if (string=? (gi node) "KAPITEL")
    (if (node-list=?
         (ancestor "ANHANGTEIL")
         (empty-node-list)) ; nicht im Anhang
        (format-number nummer "1")
        (format-number nummer "A"))
    (format-number nummer "1"))))


; (ancestor-kind-nummer string-name)
; liefert die (kind-nummer) des naechsten
; Vorfahren mit dem GI string-name
(define (ancestor-kind-nummer string-name)
  (kind-nummer (ancestor string-name)))

; (element-nummer node)
; liefert die (element-number) des Knotens in
; formatierter (arabischer) Schreibweise
(define (element-nummer node)
  (let ((nummer (element-number node)))
    (format-number nummer "1")))


; Trennlinien
(define (chapseprule)
    (make empty-element gi: "HR"
      attributes: (attribute (list "size" "1" "noshade"
                                   "noshade"))))
(define (footseprule)
    (make empty-element gi: "HR"
      attributes: (attribute (list "size" "1" "noshade"
                                    "noshade" "width" "33%"
                                    "align" "left"))))

; Zeilenumbruch
(define (br)
  (make empty-element gi: "BR"))
```

Bibliographie

[ABEL91] Abelson, Harold; Gerald Jay Sussman, Julie Sussman: *Struktur und Interpretation von Computerprogrammen.* Berlin u.a., Springer 1991

[ADOB97] Adobe: *FrameMaker+SGML Developer's Guide.* Adobe 1997

[BEMI98] Behme, Henning; Stefan Mintert: *XML in der Praxis.* Bonn, Addison Wesley Longman 1998

[BERN96A] Berners-Lee, Tim: *The Myth of Names and Addresses.* 1996, http://www.w3.org/DesignIssues/NameMyth.html

[BOSA96] Bosak, Jon: *DSSSL Online Application Profile.* 1996, http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm

[BOSA97A] Bosak, Jon (Sun Microsystems): *XML, Java, and the future of the Web.* 1997, http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm

[BOSA97B] Bosak, Jon: *XML Part 3: Style [NOT YET] Version 1.0.* 1997, http://sunsite.unc.edu/pub/sun-info/standards/dsssl/xs/xs970522.ps.zip

[BOSA97C] Bosak, Jon: *Extensible Style Sheet Language (XSL) Version 0.2.* 1997, http://sunsite.unc.edu/pub/sun-info/standards/xsl/0.2/xsl-19971010.ps.zip

[BUSH45] Bush, Vannevar: *As We May Think.* In *The Atlantic Monthly*, 7/1945, http://www.isg.sfu.ca/~duchier/misc/vbush/

[CLAR97] Clark, James: *Comparison of SGML and XML.* 1997, http://www.w3.org/TR/NOTE-sgml-xml-971215

[CLAR98A] Clark, James: *Associating stylesheets with XML documents.* 1998, http://www.w3.org/TR/NOTE-xml-stylesheet

[CLAR98B] Clark, James: *Jade - James' DSSSL Engine.* 1998, http://www.jclark.com/jade/

[CONN97] Connolly, Dan: *XML: Principles, Tools and Techniques.* O'Reilly 1997

[CONN97A] Connolly, Dan; Rohit Khare, Adam Rifkin: *The Evolution of Web Documents.* In [CONN97]

[CULS97] Culshaw, Stuart; Michael Leventhal, Murray Maloney: *XML and CSS.* In [CONN97]

[DEDU94] DeRose, Stephen; David G. Durand: *Making Hypermedia Work.* Kluwer Academic Press 1994

[DRAK96] Drakos, Nikos: *All About LaTeX2HTML.* 1996, http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html

[ENGE62] Engelbart, Douglas C.: *Augmenting Human Intellect.* 1962, http://www.histech.rwth-aachen.de/www/quellen/engelbart/ahi62index.html

[FRMK95] Frame Technology Corporation: *FrameMaker Benutzerhandbuch Version 5.* Frame Technology Corporation 1995

[GERM97] Germán, Daniel M.: *An Introduction to DSSSL.* 1997, http://csg.uwaterloo.ca/~dmg/dsssl/tutorial/tutorial.html

[GOLD90] Goldfarb, Charles F.: *The SGML Handbook.* Oxford University Press 1990

[GOLD97] Goldfarb, Charles F.: *The SGML History Niche.* 1997, http://www.sgmlsource.com/history/index.htm

[GMS94] Goossens, Michel; Frank Mittelbach, Alexander Samarin: *The LaTeX Companion.* Reading, Addison-Wesley 1994

[GUKA92] Gulbins, Jürgen; Christine Kahrmann: *Mut zur Typographie.* Springer-Verlag 1992

[ISO86] International Organization for Standardization: *ISO 8879.* 1986

[ISON1035] International Organization for Standardization: *ISO/IEC JTC1/SC18/WG8/N1035.* http://www.sgmlsource.com/8879rev/n1035.htm

[ISO92] International Organization for Standardization: *Hypermedia/Time-based Structuring Language (HyTime), ISO/IEC 10744:1992.* 1992, http://www.ornl.gov/sgml/wg8/hytime/is10744r.html

[ISO96A] Goldfarb, Charles F. (International Organization for Standardization): *TC for Extended Naming Rules.* 1996, http://www.sgmlsource.com/8879rev/n1896rev.htm

[ISO96B] International Organization for Standardization: *Document Style Semantics and Specification Language, ISO/IEC 10179:1996.* 1996, ftp://ftp.ornl.gov/sgml/wg8/dsssl

[ISO97] Goldfarb, Charles F. (International Organization for Standardization): *ISO 8879 TC2.* 1997, http://www.sgmlsource.com/8879rev/n1955.htm

[KILP98] Kilpeläinen, Pekka: *SGML & XML Content Models.* Universität Helsinki 1998, http://www.cs.Helsinki.FI/TR/C-1998/12/

[LIBO97] Lie, Håkon Wium; Bert Bos: *Cascading Style Sheets.* Bonn, Addison-Wesley 1997

[MICR98] Microsoft: *XSL Tutorial.* 1998, http://www.microsoft.com/xml/xsl/tutorial/tutorial.htm

[MINT95] Mintert, Stefan: *Leise Revolution.* In *iX*, 7/1995, Seite 126

[MINT97] Mintert, Stefan: *Auszeichnungssprachen im WWW.* Universität Dortmund 1997

[MSFT94] Microsoft: *Rich Text Format (RTF) Specification.* 1994

[NCSA98] National Center for Supercomputing Applications: *The Common Gateway Interface.* 1998, http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

[NELS97] Nelson, Theodor: *Embedded markup considered harmful.* In [CONN97]

[PORL93] Peek, Jerry; Tim O'Reilly, Mike Loukides: *UNIX Power Tools.* Sebastopol/New York, O'Reilly & Associates/Random House 1993

[PEPP98] Pepper, Steve: *The Whirlwind Guide to SGML & XML Tools and Vendors.* 1998, http://www.infotek.no/sgmltool/guide.htm

[PRES97] Prescod, Paul: *Introduction to DSSSL.* 1997, http://itrc.uwaterloo.ca/~papresco/dsssl/tutorial.html

[RAGG98] Raggett, Dave; Jenny Lam, Ian Alexander, Michael Kmiec: *HTML 4.* Bonn, Addison-Wesley 1998

[RFC1738] Berners-Lee, Tim; Larry Masinter, Mark McCahill (Network Working Group): *Uniform Resource Locators (URL).* 1994, http://rfc.fh-koeln.de/rfc/html_gz/rfc1738.html.gz

[SPBU94] Sperberg-McQueen, Chris M.; Lou Burnard: *Guidelines for Electronic Text Encoding and Interchange.* Chicago, 1994

[THOM97] Thompson, Henry S.: *An Introduction to XSL.* 1997, http://www.ltg.ed.ac.uk/~ht/swindon.html

[TRWA95] Travis, Brian E.; Dale C. Waldt: *The SGML Implementation Guide.* Berlin, Springer-Verlag 1995

[UEBE92] Uebe, Götz; Joachim Fischer: *Macro-Econometric Models.* Avebury 1992

[UEBE95] Uebe, Götz: *World of Economic Models.* Avebury 1995

[VROM96] Vromans, Johan: *Perl 5.* Köln, O'Reilly 1996

[WIHE96] Wilhelm, Reinhard; Reinhold Heckmann: *Grundlagen der Dokumentenverarbeitung.* Bonn, Addison-Wesley 1996

[W3C96A] World Wide Web Consortium: *Cascading Style Sheets, level 1.* 1996, http://www.w3.org/TR/REC-CSS1

[W3C97A] World Wide Web Consortium: *CSS2 Specification Release.* 1997, http://www.w3.org/TR/WD-CSS2/

[W3C97B] Berners-Lee, Tim (World Wide Web Consortium): *W3C Data Formats.* 1997, http://www.w3.org/TR/NOTE-rdf-farch

[W3C97C] Guha, Ramanathan V.; Tim Bray (World Wide Web Consortium): *Meta Content Framework Using XML .* 1997, http://www.w3.org/TR/NOTE-MCF-XML-970624/

[W3C97D] World Wide Web Consortium: *Resource Description Framework.* 1997, http://www.w3.org/RDF/Overview.html

[W3C97E] Adler, Sharon; Anders Berglund, James Clark, Istvan Cseri, Paul Grosso, Jonathan Marsh, Gavin Nicol, Jean Paoli, David Schach, Henry Thompson, Chris Wilson (World Wide Web Consortium): *A Proposal for XSL.* 1997, http://www.w3.org/TR/NOTE-XSL.html

[W3C97F] Lassila, Ora; Ralph R. Swick (World Wide Web Consortium): *Resource Description Framework (RDF) Model and Syntax.* 1997, http://www.w3.org/Metadata/RDF/Group/WD-rdf-syntax

[W3C97G] Byrne, Steve (World Wide Web Consortium): *Document Object Model (Core) Level 1.* 1997, http://www.w3.org/WD-DOM/level-one-core

[W3C97H] Nicol, Gavin; Mike Champion (World Wide Web Consortium): *Document Object Model (XML) Level 1.* 1997, http://www.w3.org/TR/WD-DOM/level-one-xml

[W3C98A] World Wide Web Consortium: *Design Issues.* 1998, http://www.w3.org/DesignIssues/Overview.html

[W3C98B] World Wide Web Consortium: *Web Architecture: Extensible languages.* 1998, http://www.w3.org/DesignIssues/Extensible.html

[W3C98C] Layman, Andrew; Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, Steve DeRose (World Wide Web Consortium): *XML-Data.* 1998, http://www.w3.org/TR/1998/NOTE-XML-data

[W3C98D] Bray, Tim; Jean Paoli, C. M. Sperberg-McQueen (World Wide Web Consortium): *Extensible Markup Language (XML) 1.0.* 1998, http://www.w3.org/TR/REC-xml

[W3C98E] Maler, Eve; Steve DeRose (World Wide Web Consortium): *XML Linking Language (XLink).* 1998, http://www.w3.org/TR/1998/WD-xlink-19980303

[W3C98F] Maler, Eve; Steve DeRose (World Wide Web Consortium): *XML Pointer Language (XPointer).* 1998, http://www.w3.org/TR/1998/WD-xptr-19980303

[W3C98G] Maler, Eve; Steve DeRose (World Wide Web Consortium): *XML Linking Language (XLink) Design Principles.* 1998, http://www.w3.org/TR/1998/NOTE-xlink-principles

[W3C98H] Bray, Tim; Dave Hollander, Andrew Layman (World Wide Web Consortium): *Namespaces in XML.* 1998, http://www.w3.org/TR/1998/WD-xml-names

[W3C98I] Brickley, Dan; Ramanathan V. Guha, Andrew Layman (World Wide Web Consortium): *Resource Description Framework (RDF) Schemas.* 1998, http://www.w3.org/TR/1998/WD-rdf-schema-19980409

[W3C98J] Bray, Tim; Jean Paoli, C. M. Sperberg-McQueen (World Wide Web Consortium): *Extensible Markup Language (XML) 1.0.* 1998, http://www.mintert.com/xml/trans/REC-xml-19980210-de.html

[W3C98K] World Wide Web Consortium: *Extensible Style Language (XSL).* 1998, http://www.w3.org/TR/WD-xsl

[WHIT98] Whitehead, E. J.; M. Makoto: *The text/xml Media Type.* 1998, ftp://ftp.ietf.org/internet-drafts/draft-whitehead-mime-xml-02.txt