Seminar

Grundlagen der Markup-Sprachen (des Internets)

Universität Dortmund
Wintersemester 1988/99

Betreuer:
Prof. Dr. Gisbert Dittrich
Dipl.-Inform. Jörg Westbomke

Teilnehmer:
Mustafa Baydar
Thorsten Bludau
Sabine Böhm
Lars Brauckmann
Georg Conrads
Rainer Dampmann
Markus Hövener
Frank Klein-Robbenhaar
Markus Rupp
Jens Schröder
Martin Stein
Anton Stoll
Achim Streit
Christoph Zobiegala

Vorwort

Das World Wide Web (WWW) ist mittlerweile zu einer Standardtechnologie im Kommunikationsbereich geworden. Die Inhalte des WWWs werden mit Hilfe der sogenannten Hypertext Markup Language (HTML) beschrieben. Diese Beschreibungssprache liegt gegenwärtig in der Version 4.0 vor und wird ständig weiterentwickelt.

Dieses Seminar sollte Studierenden im Hauptstudium des Fachbereichs Informatik einen Überblick über die grundlegenden Ansätze von deskriptiven Beschreibungsmitteln geben und die Kenntnisse in den speziellen Beschreibungsmitteln des WWWs vertiefen.

Dies führt zu folgenden Beiträgen:

Nach einer Einführung in formale Sprachen und die grundlegende Vorstellung von Markups wird in die für diesen Kontext grundlegende Grammatik SGML (Standard Generalized Markup Language) eingeführt. Als die für die Anwendungen derzeit am meisten verbreitete sogenannte Document Type Definition wird die mit Hilfe von SGML erzeugte DTD HTML vorgestellt. Der sich derzeit abzeichnenden Weiterentwicklung wird durch die Vorstellung von XML (Extensible Markup Language) Rechnung getragen. Alle diese Schwerpunkte werden in mehreren Vorträgen vorgestellt.

Um die Tragweite von HTML und XML besser abschätzen und sie besser gegeneinander abgrenzen zu können, werden für diese beiden jeweils Beispielanwendungen entwickelt und vorgestellt.

Dortmund, den 15. Februar 1999

Gisbert Dittrich Jörg Westbomke

Inhaltsverzeichnis

Formale Syntaxbeschreibungen

Sabine Böhm

FB Informatik, Uni-Dortmund

e-mail : sabineboehm@compuserve.com

Zusammenfassung:

In diesem Vortrag werden Grammatiken und die von ihnen erzeugten Sprachen behandelt.

Schwerpunktmäßig wird auf kontextfreie Grammatiken eingegangen, da diese für die Makup-Sprachen des Internets benötigt werden. In diesem Zusammenhang werden die BNF- und die eBNF-Notation, sowie Syntaxgraphen zur Erzeugung von Wörtern einer Sprache, sowie Automaten, Ableitungsbäume, Syntax-diagramme und LR(k)-Grammatiken zur Analyse einer Sprache vorgestellt.

Desweiteren wird kurz auf kontextsensitive Sprachen eingegangen und ein Exkurs über reguläre Grammatiken als Spezialfall der CFG's wird gemacht.

Einen weiteren Spezialfall der CFG's stellen die attributierten Grammatiken dar, die über Attribute semantische Informationen darstellen können.

Zum Schluß wird der Versuch unternommen, die informellen Nebenbedingungen der XML-Grammatik beispielhaft durch kontextsensitive Grammatiken zu formalisieren.

1 Einführung

1.1 Was sind Grammatiken ?

Grammatiken beschreiben den erlaubten Satzbau – also die Syntax - einer Sprache in kompakter Form.
Mit ihnen kann man genau festlegen, welche Sätze in einer Sprache enthalten sind, d.h. jeder Satz, der durch Anwendung der Regeln der erzeugenden Grammatik aus einem ausgezeichneten Startwort abgeleitet werden kann, ist auch in der Sprache enthalten.

Daraus sind schon die beiden Seiten von Grammatiken ersichtlich:
Grammatiken erzeugen einerseits eine Sprache und werden dafür z.B. durch BNF- / eBNF-Notation (s.u.) dargestellt, andererseits gibt es auch Analysewerkzeuge, wie z.B. Automaten und LR(k)-Grammatiken, die es ermöglichen - anhand der gegebenen Grammatik – einen Satz auf syntaktische Korrektheit zu überprüfen, d.h. zu testen, ob ein Wort in der Sprache liegt.

Semantische Prüfungen sind hingegen durch Grammatiken i.A. nicht möglich. Ausnahmen bilden z.B. die attributierten Grammatiken (s.u.), mit denen es möglich ist, semantische Informationen weiterzuleiten.

1.2 Wie sind Grammatiken definiert ?

Grammatiken sind formal gesehen 4-Tupel (N, T, S, P).

N bezeichnet die sogenannten Nicht-Terminalzeichen oder auch Bezeichner.
T sind die Terminalzeichen, also die Zeichen, die in der zu erzeugenden Sprache erlaubt sind.
S ist ein ausgezeichnetes Startsymbol, von dem aus jedes Wort der Sprache erzeugt werden kann. Es muß immer gelten : $S \in N$.
P bezeichnet die Produktionen oder Regeln.

N, T, S und P sind dabei endliche Mengen.

Anhand der Art der Regeln kann man 4 Grammatikarten unterscheiden, die in der sogenannten Chomsky-Hierarchie aufgeschlüsselt werden:

- Chomsky 0 Grammatiken sind hier nicht weiter von Interesse, daher sei auf Lehrbücher verwiesen.
- Chomsky 1 Grammatiken werden auch als kontextsensitive Grammatiken (CSG's) bezeichnet.
  Diese interessieren hier nur am Rande, daher wird auf sie später nur kurz eingegangen.
- Chomsky 2 Grammatiken bilden im Zusammenhang mit den Markup-Sprachen des Internets den wichtigsten Teil, daher werden diese sogenannten kontext-freien Grammatiken im nächsten Abschnitt ausführlich behandelt.
- Chomsky 3 Grammatiken sind insofern hier von Interesse, als daß sie eine echte Teilmege der CFG's bilden und somit alle ihre Eigenschaften ebenfalls auf kontextfreie Grammatiken übertragbar sind.
  Man bezeichnet sie als reguläre Grammatiken.

Es gilt $C3 \subset C2 \subset C1 \subset C0$.

2 Kontextfreie Grammatiken

Kontextfreie Grammatiken (CFG's) haben Produktionen, die folgendermaßen aussehen :

$A \rightarrow \beta$ wobei $A \in N$, $\beta \in (N \cup T)^*$.

Als Beispiel für eine kontextfreie Grammatik soll eine Grammatik betrachtet werden, die die Sprache $L(G1)= \{ a^n b^n \mid n \in N_0 \}$ beschreibt.

G1 = (N, T, S, P) mit $N = \{ S \}$, $T = \{ a, b \}$, S als Startsymbol und
$P = \{ S \rightarrow aSb ,$
$S \rightarrow \varepsilon \}$

Die erste Produktion stellt sicher, daß es gleichviele a's und b's gibt und daß alle a's links von den b's stehen.
Die zweite Produktion wird benötigt, damit das Verfahren abbricht.
ε heißt dabei das leere Wort .

Anhand dieses Beispiels soll gezeigt werden, wie man ein Wort der Sprache – durch Anwendung der Regeln der zugehörigen Grammatik - herleiten kann :

Es soll der Satz „aaabbb" der Sprache L(G1) erzeugt werden:

$S \rightarrow a\,S\,b \rightarrow a\,a\,S\,b\,b \rightarrow a\,a\,a\,S\,b\,b\,b \rightarrow a\,a\,a\,\varepsilon\,b\,b\,b$.

„ aaaεbbb“ ist gleichbedeutend mit „aaabbb“, da ε das leere Wort symbolisiert.

Zu beachten ist, daß das Zeichen „ → „ hier als Anwendung einer Regel interpretiert werden muß und nicht als Darstellung einer Regel, wie dies bisher der Fall war.

Formal betrachtet werden Ableitungen folgendermaßen spezifiziert :

Sei $G = (N, T, S, P)$ eine kontextfreie Grammatik und seien $v,w \in (N \cup T)^*$.

$w \Rightarrow v$
: $\Leftrightarrow (\exists\ \alpha,\ \beta \in (N \cup T)^*,\ A \in N, u \in (N \cup T)^*, p = A \rightarrow u \in P: w = \alpha A \beta, v = \alpha u \beta.)$
D.h. die Ableitung von v aus w mit der Grammatik G in nur einem Schritt ist möglich, genau dann, wenn w ein Non-Terminalzeichen A enthält , daß auf der linken Seite einer Produktion enthalten ist. Wenn nun A in w durch das auf der rechten Seite der Regel angegebene u ersetzt wird – durch Anwendung dieser Regel - , so erhält man v. Die Teilwörter α und β vor bzw. hinter A bleiben dabei unverändert.

Anmerkung:
Von nun an wird $w \Rightarrow v$ wieder einfach durch: $w \rightarrow v$ notiert.

$$w \rightarrow^* v : \Leftrightarrow \exists\ w_i\ \ 0 \le i \le l\ \ w = w_0 \rightarrow w_1 \rightarrow \ldots \rightarrow w_l = v$$

...beschreibt die Ableitung von v aus w in endlich vielen Schritten. Dies wird durch das Zeichen →* symbolisiert.

Die von G erzeugte Sprache hat also folgendes Aussehen :

$$L(G) := \{w \in T^* \mid S \rightarrow^* w\}$$, d.h.

alle Wörter w, die aus einer endlichen Zahl von Terminalzeichen bestehen und durch Anwendung endlich vieler Regeln aus dem Startwort abgeleitet werden können, sind in der Sprache enthalten.
Formal wird dies folgendermaßen notiert :

$$w \in L : \Leftrightarrow \{\ \exists\ w_i\ \ 0 \le i \le l \mid S = w_0 \rightarrow w_1 \rightarrow \ldots \rightarrow w_l = w\}$$

d.h. $w \in L \Leftrightarrow S \rightarrow^* w$

Das Aufzählen aller möglichen Produktionen kann aufwendig werden, wie an folgendem Beispiel gezeigt werden soll :

Eine Grammatik, die die Sprache $L(G2) = \{a^n b^m \mid n,m \in N_0\}$ erzeugt, habe folgendes Aussehen :

G2 = (N, T, S, P) mit N = { S , A , B } , T = { a, b } , S = Startsymbol
P = { S → AB ,
A → aA ,
A → ε ,
B → bB,
B → ε }

Mit einigen wenigen Hilfsmitteln kann man die Anzahl der Produktionen senken. Dazu wird nun die Backus-Naur-Form (BNF) vorgestellt :

2.1 Die BNF-Notation

In dieser Notation ist es möglich, verschiedene Alternativen von Ableitungen durch das Zeichen | (oder) darzustellen,

Non-Terminalzeichen werden hier in <...> gesetzt und

statt des Pfeiles, der eine Regel symbolisiert, wird in dieser Darstellungsform ::= benutzt. Dadurch wird auch die Doppeldeutigkeit des „→"- Symbols vermieden.

Die Grammatik G1 läßt sich durch diese Schreibweise nur wenig verkürzen:

$G1_{BNF}$: P = { <S> ::= a <S> b | ε }

Grammatik G2 läßt sich schon sehr übersichtlich darstellen :

$G2_{BNF}$: P = { <S> ::= <A> <B> ,
<A> ::= a <A> | ε ,
<B> ::= b <B> | ε }

Diese Darstellungsform ist also schon um einiges komfortabler, da Alternativen übersichtlich hintereinander aufgelistet werden können.

Um eine noch kompaktere Darstellung zu erhalten, kann man sich der eBNF- (enhanced BNF) Notation bedienen. Diese hat noch drei weitere Features zu bieten.

2.2 Die eBNF - Notation

Erstens kann man eine beliebige Anzahl an Wiederholungen eines Wortes durch ein nachgestelltes * - Symbol darstellen.
Dies ist im Fall G2 sehr von Vorteil, dagegen kann man G1 durch diese Neuerung nicht weiter verkürzen, da die Anzahl der a's und der b's voneinander abhängen, also eine beliebige Zahl von Wiederholungen nicht erlaubt ist.

G2 sähe nun folgendermaßen aus :

$G2_{eBNF}$: P = { <S> ::= a*b* }

Damit kann man beliebig viele a's, sowie b's erzeugen und die Reihenfolge ist ebenfalls festgelegt.
Ein zweiter Zusatz den die eBNF bietet sind optionale Elemente, die durch [...] dargestellt werden.
Anhand der bisher eingeführten Beispiele kann man keine Verwendung dafür angeben, daher wird an dieser Stelle ein weiteres Beispiel G3 eingeführt :

Die erzeugte Sprache soll hier die Sprache der ganzen Zahlen – also Z - sein:

Ein Wort dieser Sprache beginnt also mit einem „+" oder „-„ –Zeichen , bzw. ohne Vorzeichen.
Danach folgt eine endliche Anzahl von Ziffern.
Eine Grammatik, die genau diese Sprache erzeugt ist z.B. G3 :

$G3_{eBNF}$: P = { <S> ::= [+ | -] <Ziffer>* ,

<Ziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 }

Wenn man diese Sprache in der BNF-Notation beschreiben wollte, müßte man etwas mehr Aufwand betreiben :

$G3_{BNF}$: P = { <S> :: = + <Zahl> | - <Zahl> ,
<Zahl> ::= <Ziffer> <Zahl> | <Ziffer> ,
<Ziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 }

Da es in dieser Schreibweise nicht möglich ist, eine beliebige Anzahl von Wiederholungen darzustellen, muß ein neues Non-Terminalzeichen hinzugefügt werden, das rekursiv definiert wird („Zahl“).

Die dritte Vereinfachung, die die BNF-Notation bereitstellt , ist die Möglichkeit, das
| - Zeichen auch innerhalb einer Klammerung anzuwenden. Dies wurde schon ausgenutzt in dem Ausdruck [+ | -].
Des weiteren kann man $G3_{eBNF}$ noch weiter verkürzen :

$G3_{eBNF}$: P = { <S> ::= [+ | -] (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0)* }

Grammatiken in diesen 2 Darstellungsformen (BNF sowie eBNF) dienen also dazu Sprachen zu erzeugen.
Ebenso können dazu die nun folgenden Syntaxgraphen genutzt werden.

2.3 Syntaxgraphen

Durch die Definition der eBNF-Notation ergibt sich eine weitere Möglichkeit, um CFG’s darzustellen – die Syntaxgraphen (wegen ihres Aussehens auch oft Eisenbahndiagramme genannt).

Zur anschaulichen Erläuterung soll hier ein Beispiel vorgestellt werden :

Die Sprache L(G3) , die die ganzen Zahlen darstellt, hat folgenden Syntaxgraphen :

Zur Erinnerung nochmals $G3_{eBNF}$:

P = { < S > ::= [+ | -] < Ziffer >* ,
< Ziffer > ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 }

S :

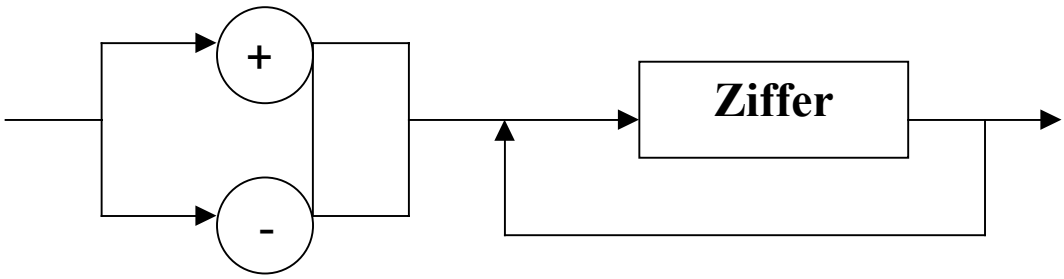


Abbildung 2-1 : Syntaxgraph zu S aus G3

Wie man dem Diagramm entnehmen kann, werden Bezeichner in rechteckigen Feldern und Terminalzeichen in Kreisen dargestellt. Der senkrechte Strich, der + und – verbindet, steht für die Optionalität des Elementes.
Nun muß nur noch der Syntaxgraph für <Ziffer> definiert werden. Da <Ziffer> nur zu Terminalzeichen abgeleitet wird, überrascht das Aussehen des Diagramms nicht :

Ziffer :

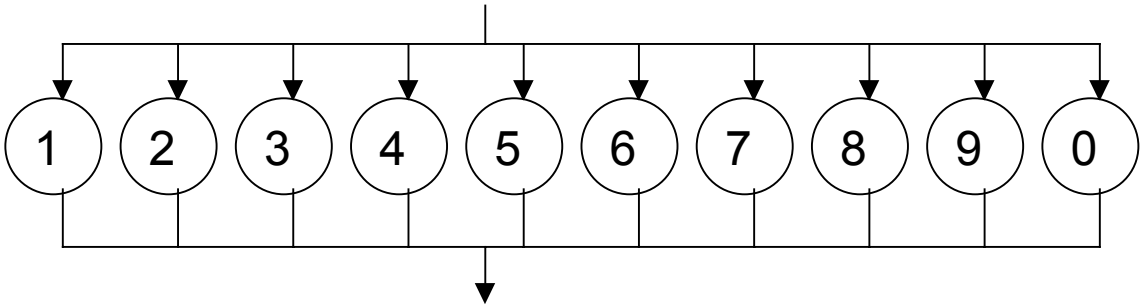


Abbildung 2-2: Syntaxgraph zu Ziffer aus G3

Um nun zu testen, ob ein Satz in der Sprache liegt, benötigt man Analyse-Tools, wie z.B. Automaten oder LR(k)-Grammatiken.

Bevor nun ein Automat für kontextfreie Grammatiken vorgestellt wird, soll noch kurz auf reguläre Grammatiken eingegangen werden, da diese einen Spezialfall der kontextfreien Sprachen darstellen.

3 Reguläre Grammatiken

Reguläre Grammatiken haben Produktionen der folgenden Art :

$A \rightarrow bB$ oder $A \rightarrow b$ oder $A \rightarrow \varepsilon$ mit $B \in N$, $b \in T$

Um beispielsweise die Sprache $L(G2) = \{ a^n b^m \mid n,m \in N0 \}$ darzustellen, benötigt man folgende Produktionen :

$G2_{reg}$: $P = \{ S \rightarrow aA$,
$S \rightarrow bB$
$S \rightarrow \varepsilon$,
$A \rightarrow aA$,
$A \rightarrow B$,
$A \rightarrow \varepsilon$,
$B \rightarrow bB$,
$B \rightarrow \varepsilon \}$

Um nun zu analysieren, ob ein Satz in der Sprache enthalten ist, bedient man sich der Automaten.
Im Falle der regulären Grammatiken benutzt man die sogenannten endlichen, nicht - deterministischen Automaten (NFA). Diese werden hier als bekannt vorausgesetzt.

4 Analyse-Tools für kontextfreie Grammatiken

4.1 Kellerautomaten (PDA)

Der Kellerautomat ist eine Erweiterung des endlichen Automaten.
Er enthält alle Elemente des NFA, aber zusätzlich noch einen Stack, den sogenannten Keller.
Ein PDA kann eine Eingabe akzeptieren, wenn der Keller leer ist , oder wenn der zugehörige Automat in einem akzeptierenden Zustand ist.
Hier wird die erste Variante gewählt, der Automat akzeptiert also, wenn kein Element auf dem Stack liegt.

Dadurch ist es möglich, Sprachen wie L(G1) = $\{ a^n b^n \mid n \in N_0 \}$ darzustellen.

Dies soll an einem Beispiel demonstriert werden:

Es soll geprüft werden, ob „aaabbb“ in der Sprache L(G1) liegt.

Der zugehörige Automat sieht folgendermaßen aus:

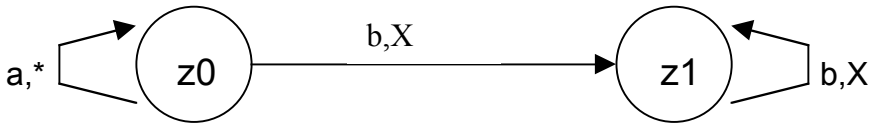


Abbildung 4-1: Kellerautomat zur Sprache L(G1)

Beschreibung des Ablaufes :

Wenn ein a eingelesen wird, soll ein Element X auf den Stack gelegt werden.
Wenn ein b eingelesen wird, soll ein Element X vom Stack entfernt werden.

Stack

Am Anfang befindet sich der Automat in Zustand z0.
Der Keller ist leer.

Nun wird das erste Zeichen der Eingabe gelesen: „a“.
Der Kellerinhalt darf also beliebig sein (durch das * symbolisiert).
Es wird ein X auf den Stack gelegt.

X

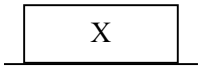
Ebenso wird für das 2. und 3. „a“ verfahren.
Der Automat bleibt in Zustand z0.

X
X
X

Wird nun das 1. „b“ eingelesen, so muß mindestens

X

ein X auf dem Stack vorhanden sein. Dies ist hier der Fall, also geht der Automat in Zustand z1 über und es wird ein X von dem Kellerstapel entfernt.



Analog wird für das 2. und 3. „b“ vorgegangen.
Der Keller ist anschließend wieder leer.

→ Der Satz ist in der Sprache L(G1) enthalten.

Es ist leicht einzusehen, daß der Automat nicht akzeptiert, wenn die Anzahl an a’s und b’s nicht gleich ist.

4.2 Ableitungsbäume

Eine andere Methode, um kontextfreie Grammatiken graphisch darzustellen, sind die Ableitungsbäume.

Zur anschaulichen Erläuterung ein Beispiel für einen Ableitungsbaum, der den Satz „aaaabbb“ ∈ L(G2) darstellt:

Zur Erinnerung : $G2_{BNF}$: P = { <S> ::= <A><B> ,
<A> ::= a <A> | ε ,
<B> ::= b <B> | ε }

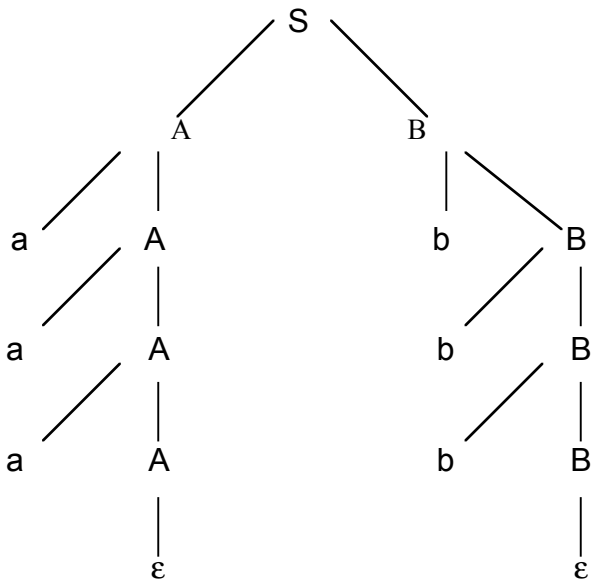


Abbildung 4-2: Ableitungsbaum zum Wort „aaabbb“ der Sprache L(G2)

An den Blättern kann nun der Satz abgelesen werden, also ist er auch in der Sprache enthalten.

Anhand der Ableitungsbäume läßt sich der Begriff der (inhärenten) Mehrdeutigkeit gut erläutern.

4.3 (Inhärente) Mehrdeutigkeit

Sei G kontextfreie Grammatik.

Ein Wort w∈ L(G) heißt eindeutig in G : ⇔
es gibt genau einen Ableitungsbaum (Linksableitung) für w in G.
Andernfalls heißt w mehrdeutig in G.

Die Grammatik G heißt eindeutig: ⇔
∀w ∈ L(G): w ist eindeutig in G.
Andernfalls heißt G mehrdeutig.

Eine kontextfreie Sprache L heißt inhärent mehrdeutig : ⇔
Jede kontextfreie Grammatik G mit L = L(G) ist mehrdeutig.

Aus obigem Diagramm ist ersichtlich, daß L(G2) nicht inhärent mehrdeutig ist, da es zu G2 für jedes Wort der Sprache einen eindeutigen Ableitungssbaum gibt.

Ist diese Sprache also eindeutig? Die Antwort lautet „nein“, da man sich eine Grammatik G4 mit L(G2) = L(G4) definieren kann, die mehrere Ableitungsbäume für ein Wort der Sprache erzeugt.

Diese Grammatik G4 hat z.B. folgende Produktionen :

G4 : P = { <S> ::= <A><B> | <A> | <B> | ε ,
<A> ::= <A><A> | a ,
<B> ::= <B><B> | b }

Die zugehörigen Ableitungsbäume haben somit für den Satz „aaaabbb“ folgendes Aussehen :

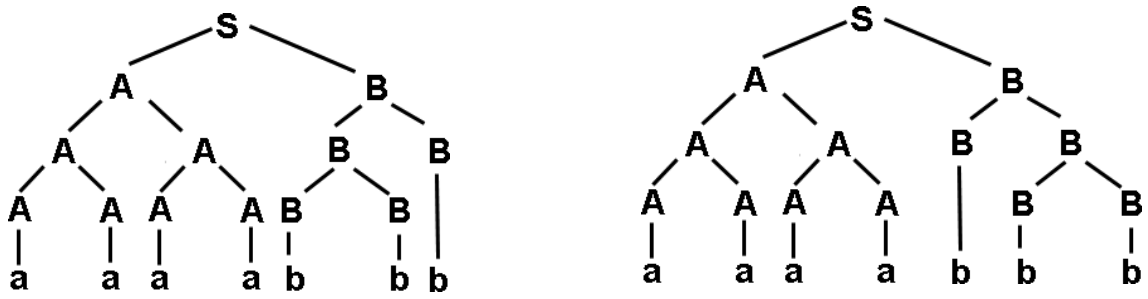


Abbildung 4-3: Ableitungsbäume der Grammatik G4

Diese Art der Darstellung ist das Ergebnis einer syntaktischen Analyse, wie sie z.B. zur Überprüfung der Klammersetzung gut einzusetzen ist.
Um allerdings einen Programmcode in Linearzeit zu übersetzen, benötigt man eine Sonderform der kontextfreien Grammatiken :

Die sogenannten LR(k)- Grammatiken.

5 LR(k) - Grammatiken

Bei den LR(k) – Grammatiken wird der Code einmal von links nach rechts durchgelesen und für jedes Zeichen steht spätestens nach k weiteren Schritten fest, wie es abgeleitet werden muß.
Als kleines Beispiel sei eine Addition von 2 Zahlen – die maximal 2 Stellen haben dürfen – erläutert:

1 4 + 2 3

k – auch der look-ahead – genannt sei hier als 2 gewählt, da spätestens nach dem Lesen von 2 weiteren Stellen festgestellt werden kann, ob es sich bei dem gelesenen Zeichen um eine Einer- oder Zehnerstelle handelt.
Ebenso kann in maximal 2 Schritten festgestellt werden, ob die Zahl tatsächlich nicht mehr als 2 Stellen hat.
Von weiterem Interesse – obwohl bei weitem nicht so wichtig wie CFG's – sind die kontextsensitiven Sprachen (CSG's), denen der nächste Abschnitt gewidmet werden soll.

6 Kontextsensitive Sprachen

Definition :

Eine Grammatik G = (N, T, S, P) heißt kontextsensitiv : $\Leftrightarrow$

$$P \subseteq \{ S \rightarrow \varepsilon \} \cup \{ w_1 B w_2 \rightarrow w_1 w_0 w_2 \mid w_1, w_2 \in (V \setminus \{S\})^*, B \in N, w_0 \in (V \setminus \{S\})^+ \}$$

mit $V := N \cup T$.

Dies bedeutet, daß die Produktionen, wie sie schon bei den kontextfreien Grammatiken gebraucht wurden ($B \rightarrow w_0$), von dem Kontext abhängen, der von w_1 und w_2 bestimmt wird, d.h. B darf nur zu w_0 abgeleitet werden, wenn es in dem beschriebenen Kontext steht.
Dabei dürfen w_1 und w_2 nicht das Startsymbol darstellen, ebenso wie w_0. w_0 muß zusätzlich die Bedingung erfüllen, daß es nicht leer sein darf , also $w_0 \neq \varepsilon$.
Damit wird sichergestellt, daß die rechte Seite der Ableitung niemals kürzer ist, als die linke Seite.
Zusätzlich ist noch die Produktion $S \rightarrow \varepsilon$ zugelassen.

Der Grund warum hier – wenn auch nur sehr kurz – auf diese Grammatikform eingegangen wurde, ist der, daß im letzten Kapitel versucht werden soll, ob - und wenn - wie es möglich ist, die informellen Notationen bei den Mark-Up–Sprachen durch kontextsensitive Grammatiken zu ersetzen.

Bevor aber darauf eingegangen wird, soll hier noch eine spezielle Grammatikform kurz vorgestellt werden - die attributierten Grammatiken :

7 Attributierte Grammatiken

Attributierte Grammatiken sind ein Spezialfall der kontextfreien Sprachen. Den Produktionen werden zusätzlich noch bestimmte Attribute zugewiesen.
So kann man z.B. bei einer Rechenaufgabe das Ergebnis an der Wurzel ablesen.
Dieses Beispiel soll hier nun einmal vorgeführt werden, zuvor aber noch zwei Definitionen :

Ein geerbtes Attribut ist eine Funktion, die ein N mit einem anderen N, das höher im Zerteilungsbaum steht, in Beziehung setzt, d.h.

der funktionale Wert eines N auf der rechten Seite = Funktion des N auf der linken Seite .

Ein synthetisiertes Attribut ist eine Funktion, die ein N auf der linken Seite mit den N's auf der rechten Seite in Beziehung bringt.
Sie leiten somit Informationen im Ableitungsbaum bis zur Wurzel hinauf, d.h die Werte werden von den unteren Informationen des Baumes synthetisiert.

Es werden also bei den attributierten Grammatiken semantische Elemente miteinbezogen.

Nun aber zu einem Beispiel:

Die Grammatik G6 habe also folgende Produktionen und (synthetisierte) Attribute:

E -> T | E+T	W(E) = W(T) | W(E) + W(T)
T -> P | T*P	W(T) = W(P) | W(T) * W(P)
P -> I | (E)	W(P) = I | W(E)

Die zu lösende Aufgabe sei : 2 + 4 * (1 + 2)

Der zugehörige Ableitungsbaum hat dann folgendes Aussehen :

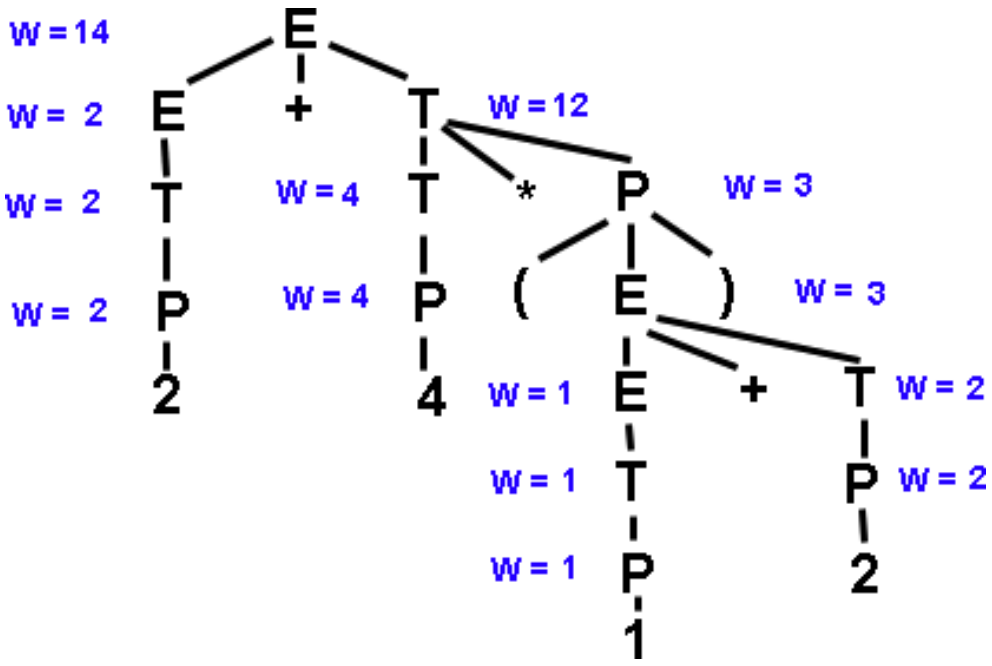


Abbildung 7-1: Ableitungsbaum der attributierten Grammatik G6

Nun soll an einem kleinen Beispiel versucht werden, ob sich die informellen Notationen der Mark-Up Sprachen durch kontextsensitive Grammatiken ersetzen lassen.

Anwendungsbeispiel

Dazu ein Ausschnitt einer CFG mit informellen Nebenbedingungen :

Ableitungsregeln :

[1] document ::= prolog element Misc*

[5]	Name	::=	(Letter | '_' | ' : ') (NameChar)*	
[22]	prolog	::=	XMLDecl? Misc* (doctypedecl Misc*)?	
[28']	doctypedecl	::=	'<!DOCTYPE' S Name ('[' rootdecl (markupdecl | PEReference | S)* ']' S?) '>'	[GKB : Wurzel-Elementtyp]
[28*]	rootdecl	::=	markupdecl	[GKB : Der Name muß identisch sein mit dem Namen in der Dokumenttyp - Deklaration]
[29]	markupdecl	::=	elementdecl | AttlistDecl | Entity-Decl | NotationDecl | PI | Comment	[GKB: Ordentliche Deklaration/PE – Verschachtelung] [WGB: PEs in interner Teilmenge]
[45]	elementdecl	::=	'<!ELEMENT' S Name S contentspec S? '>'	[GKB: Eindeutige Element- Deklarationen]
[46]	contentspec	::=	'EMPTY' | 'ANY' | Mixed | children	
[47]	children	::=	(choice | seq) ('?' | '*' | '+')?	
[50]	seq	::=	'(' S? cp (S? ',' S? cp)* S? ')'	[GKB: Ordentliche Gruppierung / PE-Verschachtelung]

7.1 Ableitung gemäß Grammatik

	document
[1]	prolog element Misc*
[22]	XMLDecl? Misc* (doctypedecl Misc*)? element Misc*
[]	doctypedecl element
[28'/5]	<!DOCTYPE Name [rootdecl markupdecl*]> element
[]	<!DOCTYPE Interview [rootdecl markupdecl markupdecl]>element
[]	<!DOCTYPE Interview [markupdecl markupdecl markupdecl]>element
[29]	<!DOCTYPE Interview [elementdecl elementdecl elementdecl]>element
[45/5]	<!DOCTYPE Interview [<!ELEMENT Name contentspec > elementdecl elementdecl]> element
[45/5]	<!DOCTYPE Interview [<!ELEMENT FAQ contentspec > ... → **Verletzung der GKB**

Anmerkungen :

- Die jeweils benutzten Regeln wurden in [ ] gestellt.
- Die Ableitungen wurden streng nach den kontextfreien Regeln durchgeführt, dabei wurden die informellen Nebenbedingungen nicht berücksichtigt.

Die Verletzung der Gültigkeitsbedingung tritt hier auf, da Name in der letzten Zeile zu FAQ abgeleitet wurde, aber der Wurzelelementtyp schon in Zeile 6 als Interview festgelegt wurde.

Nun soll versucht werden, ob sich dieser Fehler umgehen läßt, indem man eine kontextsensitive Grammatik entwirft, die diese Gültigkeitsbedingungen formalisiert.

Kontextfreie Grammatik mit Nebenbedingung (Ausschnitt) :

[28] doctypedecl ::= '<!DOCTYPE' S Name (markupdecl | PEReference)* ']>' [GKB: Wurzel – Elementtyp]

[29] markupdecl ::= elementdecl | AttlistDecl | EntityDecl [GKB: Ordentliche Deklaration/PE – Verschachtelung] [WGB: PEs in interner Teilmenge]

7.2 Kontextsensitive Grammatikregel

[28''] α_1 '<!DOCTYPE' N1 '['(markupdecl | PEReference)*']' β_1
→ α_1 '<!DOCTYPE' N1 [<!ELEMENT N1 contentspec '>' (markupdecl | PEReference)* ']>' β_1

mit $\alpha_1, \beta_1 \in (N \cup T)^*$; N1 ∈ <Name>

Durch die Festlegung von N1 auf der rechten Seite der Produktion wird verhindert, daß dort ein anderer Typ steht, als der des Wurzelelementes.

Jedoch: Damit ergibt sich keine kontextsensitive Grammatik, da dazu zunächst unendlich viele derartige Regeln nötig sind, weil jede mögliche Belegung von N1 berücksichtigt werden müßte.

8 Schlußfolgerungen / Zusammenfassung :

- Kontextfreie Grammatiken sind hier von größter Bedeutung, werden hier aber mit informellen Nebenbedingungen verwendet.
  Die Sprachen werden also nicht kontextfreie Sprachen !!
- Der Versuch, informelle Bedingungen kontextsensitiv zu formalisieren, war (noch) nicht erfolgreich.
- Es gibt zwei Seiten von Grammatiken :
  - Notationen, die die Konstruktion von Sprachen vereinfachen.
    (BNF- / eBNF - Notation ,Syntaxdiagramme)
  - Analyse-Tools, die Tests, ob ein Satz in einer Sprache enthalten ist , ermöglichen.
    (Automaten, Ableitungsbäume, LR(k) –Grammatiken)

9 Literatur

[Prat 98] T.Pratt / M. Zelkowitz : Programmiersprachen, Design und Implementierung. Prentice Hall 1998

[Wege 96] I.Wegener : Kompendium Theoretische Informatik – eine Ideensammlung B.G. Teubner Stuttgart 1996.

Der grundlegende Gedanke der Auszeichnungssprachen

Christoph Zobiegala

FB Informatik, Uni Dortmund

Zusammenfassung

Die kurze Abhandlung über den grundlegenden Gedanken der Auszeichnungssprachen, soll einen Einblick in den Grundgedanken aber auch die Geschichte, Motivation und nicht zuletzt über den Einsatz von Auszeichnungssprachen geben.

Der geschichtlicher Aspekt umfasst eine etwa 30-jährige Entwicklung sowohl der Ideen der Auszeichnung mit ihren unterschiedlichen Ausprägungen als auch den von ihr verursachten Veränderungen bei der Erstellung von Dokumenten beginnend bei der „primitiven“ Schreibmaschine mit ihren eher spärlichen Möglichkeiten der Auszeichnung über erste experimentelle computergestützte Systeme bishin zu den heutigen Textverarbeitungs und DTP Programmen.

Da Auszeichnungssprachen eng mit Dokumentenerstellug und deren Verarbeitung gekoppelt ist, wird ein Teil der Arbeit mit Dokumenten und das was sie ausmacht zu tun haben. Desweiteren werden Vorteile von Auszeichnungselementen in Dokumenten vorgestellt und deren Verwendung.

In dem Teil der Seminararbeit die sich mit der Motivation auseinnandersetzt, wird anhand von vorhandenen Bedürfnissen der Anwender, der Industrie oder gegenwärtig des Internets die Entwicklung der Ideen der Auszeichnungssprachen erklären.

1 Einleitung

Um den grundlegenden Gedanken der Auszeichnungssprachen zu erklären sollte zuerst geklärt werden was Auszeichnen überhaupt ist und worauf es angewand wird.

Beachtet man die Funktionalität der Auszeichnung nicht, so ist es eine Aufblähung vorhandenen Textes durch Auszeichnungselemente.

Es ist doch absurd. Einerseits ist man bestrebt Dinge zu vereinfachen, zu komprimieren anderseits durchsetzt man Text mit Auszeichnungselementen und macht ihn dadurch umfangreicher und un-

leserlicher. Um dieses scheinbare Paradoxon aufzuklären muss man sich den Grund des Auszeichnens vor Augen führen

2 Die geschichtliche Entwicklung von Markups

Die Geschichte von Auszeichnungselementen im Text ist bereits sehr alt und begann bei der Erstellung von Texten auf mechanischen Schreibmaschinen. Sie beschränkte sich da allerdings auf Unterstreichungen und g e s p e r r t e n Text.

Der eigentliche Beginn fand bei den Verlagen statt. Der Begriff des „markups“ stammt daher auch aus dem Verlagswesen, aus einer Zeit, als Begriffe wie Desktop-Publishing noch unbekannt waren. Nach inhaltlicher Überprüfung und Korrektur eines Manuskripts folgte die Bearbeitung durch den Layouter, der die Entscheidungen über Seitenformat, Zeichensätze und weitere typographische Festlegungen traf. Diese wurden in Form von handschriftlichen **Markierungen** und **Anweisungen** in das Manuskript eingefügt und anschliessend im Satz berücksichtigt.

Im September des Jahres 1967 machte WILLIAM TUNNICLIFFE, von der Graphic Communications Association (GCA) während einer Tagung des Canadien Gouvernment Printing Office, den Vorschlag, den Informationsgehalt eines Dokumentes von seiner äusseren Form zu trennen. Ebenfalls Ende der sechziger Jahre veröffentlichte STANLEY RICE, ein New Yorker Buch-Designer, seine Idee der *„editorial structure tags“*. NORMAN SCHARPF der Direktor der GCA erkannte die signifikanz dieser Entwicklung und gründete ein Komitee die sich mit den Ideen auseinandersetzen sollte. Daraus entstand später das Konzept des *„GenCode“* eine der grundlegenden Säulen von SGML (*Standard Generalized Markup Language*). Ebenfalls auf den Ideen von Tunnicliffe und Rice basierend, entwickelte CHARLES GOLDFARB, EDWARD MOSHER und RAYMOND LORIE bei IBM im Jahre 1969 die Generalized Markup Language (GML). GML enthielt erstmals das Konzept des formal definierten Dokumententyps mit einer verschachtelten Struktur. Nach der Fertigstellung von GML formulierte Goldfarb noch weitere Konzepte, die zwar nicht in GML, aber später in SGML Einzug hielten.

Parallele Entwicklungen in den Forschungsabteilungen der Universitäten und der Industrie beschäftigten sich mit der Vereinfachung der Handhabung von Dokumenten bei ihrer Erstellung,, Speicherung und Aufbereitung sei es für die Anzeige auf dem Bildschirm oder für den Druck.

Vannevar Bush	1945	Memex
Douglas Engelbart	1962	Augment
Ted Nelson	1965	Xanadu
Wiliam Tunnicliffe (GCA) Stanley Rice	1967	generic coding editorial structure tags
Norman Scharpf(Director GCA)		GenCode-Komitee
Goldfarb, Mosher, Lorie (IBM)	1969	GML
ANSI Charles Goldfarb	1978	
ISO	1986	SGML (ISO 8879)
Tim Berners-Lee (CERN)	1989	HTML
Netscape Microsoft	1994	HTML-Abweichungen
Dave Raggett (W3C) Hakon Lie (W3C)		HTML CSS
W3C (Jon Bosak (Sun), James Clark et. al.)	1997	XML

Abbildung 1-1: Die historische Entwicklung bis zur Extensible Markup Language

2.1 Die ersten Auszeichnungssysteme

Die computergestützte Erstellung von Dokumenten war zum Zeitpunkt der ersten Entwicklungen von Auszeichnungssprachen nicht weit verbreitet. Aber man erkannte die enormen Möglichkeiten die der Computer mit sich brachte.

Die ersten Softwaresysteme die Markups einsetzten waren der 1964 am MIT konzipierte Formatierer mit der Bezeichnung RUNOFF. Er war für ein Schreibmaschinen-Terminal entwickelt und besass nur sehr beschränkte Fähigkeiten; so kannte er nur 18 verschiedene Operationen auf einer sehr niedrigen Ebene, die dazu dienten, eine Seite des Dokumentes zu beschreiben.

Die Operationen galten für die Objekte

- **einzelne Zeile:** mit der Möglichkeit der Zeilentrennung, des Umbruches und des „Auszuges“, um Einzüge wettzumachen,
- **Folge von Zeilen:** es kann deren Länge, ihre Ausrichtung und ihr Einzug festgelegt werden,
- **Abstände zwischen Zeilen:** es war ein- und zweizeiliger Druck möglich, genauso wie ein definierte Raum zwischen Zeilen gelassen werden konnte,
- **Seitenaufbau:** der Beginn eine Seite, die Kopfzeile, die Seitenlänge und das Drucken der Seitennummern konnte bestimmt werden,
- **Dateibehandlung:** es konnte bestimmt werden, ob und mit welcher Datei ein Dokument fortgesetzt werden sollte.

CALL FOR PAPERS

The aim of this conferences is to survey the state of the art of computer aids for document preparetion

Papers are solicited on

- Picture editing
- Text processing
- Algorithems and software for documet preparation and other related topics

Detailed abstracts should not exceed five pages; they must be sendbefor October 31, 1972 to the Program Chairman. Selected authors will be notified by November 30.
Duration of one Presentation will be either 25 or 45 minutes.

Abbildung 1-2 : Ein Probe Text, der mit den Beispielen für die jeweiligen Formatierer erzeugt wird.

RUNOFF unterschied zwischen Befehlszeilen (eingeleitet mit einem Punkt“.“) und den Zeilen des Dokuments. Nachstehend ist aufgeführt, wie ein Dokument im RUNOFF-System ausgezeichnet wurde, um das in Abbildung 1.1 gezeigte Layout zu erzeugen.

```
.center
CALL FOR PAPERS
.space 2
the aim of this conference is to
survey the stste of the art
.nojust
.space 1
Papers are solicited on
.space 1
.undent 2
- Picture editing
.space 1
```

```
.undent 2
- Text processing
.space 1
.undent 2
- Algorithms and software for document
  preparation and other related topics
.indent 0
.space 1
.adjust
Detailed abstracts should not exceed five
pages: they must be send bevor October 31,
1972 to the Program Chairman. Selected
authors will be notified by November 30.
.space 1
Duration of one presentation will be of
either 25 or 45 minutes
```

Zu bemerken ist, dass es keine Möglichkeit gab, den automatischen Zeilenumbruch abzustellen, dass aber Block- und Flattersatz[1] möglich waren. Da es keinen ausdrücklichen Befehl für Leerraum in RUNOFF gab, wurde „.nojust“ vor der eingerückten Liste aufgerufen, um den automatischen Ausschluss abzuschalten.

Das Hervorheben einer Textstelle geschah mangels anderer Möglichkeiten durch Unterstreichen, was aber durch den zugehörigen Editor vor der Formatierung zu erfolgen hatte.

Der Art der oben angegebenen Objekte und dem Beispiel ist zu entnehmen, dass RUNOFF allein eine physikalisch orientierte Auszeichnung zuliess. Erst die Entwicklung in seiner Nachfolge unterstützte die logische Auszeichnung.

Es wurden noch eine Reihe anderer experimentelle Systeme implementiert so zum Beispiel das System INTIME (INteractive Textual Information Managment Experiment) das von IBM am naturwissenschaftlichem Zentrum in Cambridge in Auftrag gegeben wurde.

3 Dokumente

Der Auszeichnungsgedanke ist, wie in der Einleitung bereits angeklungen, eng mit Text und somit auch mit Dokumenten verbunden. Wenn man von Dokumenten spricht meint man den darin enthaltenen Text als Informationsträger. Zunächst einmal wird man sagen, dass Dokumente etwas mit Information und der Übertragung von Information zu tun haben. Das ist aber nur teilweise richtig, den in einem Gespräch wird auch Information übertragen. Das wesentliche Merkmal von Dokumenten ist, dass Dokumente aufgezeichnete Information sind. Die visuelle Erscheinung wird stillschweigend vorausgesetzt da ein Dokument auf einem magnetischen Datenträger gespeichert für den menschlichen Betrachter wenig informatives zu bieten hat.

Die Frage die sich hiernach aufdrängt ist, ob der Text alleiniger Informationsträger eines Dokumentes ist oder gibt es darüber hinaus Bestandteile die ein Dokument ausmachen?

[1] Auf verschiedene Längen gesetzte Schriftzeilen, die nur auf einer Seite bündig abschliessen

3.1 Bestandteile von Dokumenten

Betrachtet man diese Seite so könnte man meinen, Dokumente bestünden aus Text oder seien Text. Dies ist aber nur ein Hauptbestandteil von Dokumenten zu den auch noch Bilder, Graphiken und Tabellen gehören. Insgesamt gibt es aber drei Hauptbestandteile, nämlich:

- Daten,
- Struktur und
- Format.

Die Daten stellen den Informationsgehalt des Dokumentes direkt dar. Man könnte auch sagen, dass die Daten die vom Dokument transportierte Information sind. Dazu gehören wie bereits erwähnt Text, Abbildungen usw.

Der zweite wichtige Bestandteil von Dokumenten ist die Struktur. Dazu gehört Gliederung, Aufzählungen, Listen, Verweise, Fussnoten, kurz gesagt alles, was uns auffinden und Verarbeiten der im Dokument enthaltenen Informationen erleichtert. Abstrakt: Die Struktur beschreibt die Beziehung der Datenelemente untereinander.

Der dritte Bestandteil schließlich ist die Formatierung, wobei unter Formatierung alles zu verstehen ist, was mit der sinnlich wahrnehmbaren Erscheinung des Dokuments zu tun hat. Die Formatierung macht die Struktur für den Leser erst wahrnehmbar. Eine Überschrift, die durch ihre Erscheinung im Text nicht irgendwie hervorgehoben wird, ist eigentlich keine Überschrift, sondern ein Satzfehler. Die Formatierung bedient sich der verschiedensten Mittel, um die einzelnen Strukturelemente eines Dokuments hervorzuheben und / oder gegeneinander abzugrenzen:

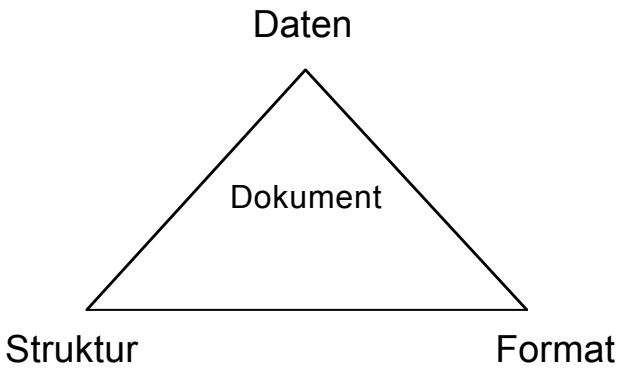
- Wechsel der Schriftart (Hervorhebung, Fettdruck),
- Absatzformat (Einrücken, Abstände, Wechsel von Block- und Flattersatz),
- besondere Seitenpositionen (Fussnote, Marginalie),
- graphische Elemente (Trennlinien in Tabellen).

So wie uns die Interpunktion hilft, die grammatikalische Struktur eines Satzes zu erfassen, hilft uns die Formatierung, die Struktur eines Dokumentes zu erfassen.

Demnach erleichtert die Strukturierung von Dokumenten das Erfassen des Informationsgehalts, die Formatierung wiederum erleichtert uns die Wahrnehmung der Struktur.

3.1.1 Das Problem und seine Teillösungen

Man kann die drei Bestandteile von Dokumenten als Ecken eines Dreiecks darstellen.



Herkömmliche, auf die Bearbeitung von Dokumenten ausgerichtete Anwendungen überdecken das aus *Daten, Struktur,* und *Format* bestehende Dreieck nur Teilweise. Um das zu verdeutlichen, un-

tersuchen wir drei Gattungen solcher Anwendungen, nämlich Editoren, Textverarbeitungsprogramme und DTP-Programme.

Editoren sind Anwendungen zum Erstellen von Texten, d.h. Daten. Struktur und Format werden nicht wiedergegeben. Das Dokument wird einfach als eine durch Trennzeichen (z. B. `CR-LF`) markierte Folge von Zeilen betrachtet, die Zeilen selbst sind Zeichenfolgen. Typischerweise werden Editoren heute bei der Erstellung von Programmen und ähnlichen Dokumenten verwendet, da diesen Dokumenten die Struktur aufgrund der Sprachdefinition implizit ist.

Textverarbeitungsprogramme geben Daten und Format eines Dokuments wieder. Sie geben dem Anwender die Möglichkeit, Teilen des Texte besondere Ausgabecharakteristika zuzuweisen, z.B. die Wiedergabe durch Fett- oder Kursivdruck oder bestimmte Schriften. Man kann den Text also *formatieren*. Die Formatbeschreibung ist dabei meist nicht portabel. Der Umbruch (d. h. die Verteilung des Textes auf Zeilen und Seiten) erfolgt häufig automatisch. Bekannte Beispiele sind Microsoft Word und WordPerfekt.

DTP-Programme geben ebenfalls Daten und Format wieder. Sie unterscheiden sich von Textverarbeitungsprogrammen (wobei der Übergang von TV- zu DTP-Systemen immer fliessender wird) durch die sehr genaue Kontrolle über Umbruch und die Anordnung von Texten und anderen Bestandteilen des Dokuments auf der Seite. Wenn eine (mehr oder minder standartisierte) Seitenbeschreibungssprache wie PostScript zugrunde liegt, so ist die Formatbeschreibung (mehr oder minder) portabel. Prominente Beispiele sind Aldus PageMaker und Quark Xpress.

Allen gennante Anwendungen ist gemeinsam, dass die Struktur als Teil des Dokumentes nicht wiedergegeben wird. Sie entsteht erst im Auge des Lesers.

Wie aber oben festgestellt wurde, ist die Struktur wesentlich für das Erfassen des Dokumentinhalts. Man kann jetzt argumentieren, dass, auch wenn keine abstrakte Strukturinformation vorhanden ist, die Struktur durch die Formatierung ja wiedergegeben wird. Das gilt aber nur für die konkrete Erscheinung.

Ein Beispiel: In Firma X wird ein Geschäftsbrief mit einer integrierten Bürolösung erstellt. Struktur ist (implizit) in der Datei vorhanden, teilweise vorgegeben durch die Software (denn deren Gliederungselemente wurden verwendet), teilweise aufgrund der Konfiguration (dem Brief liegt eine bestimmte Dokumentvorlage mit speziellen Absatzformaten für Anschrift etc. Zugrunde), teilweise informell durch die Gliederung des Textes in Absätze (entsprechend den Sitten und Gebräuchen beim schreiben von Geschäftsbriefen).

Der Brief wird Ausgedruckt und existiert dabei vorübergehend als PostScript-Datei. Zu diesem Zeitpunkt ist die ursprünglich vorhandene Struktur weitgehend verloren und die enthaltene Information nur noch eingeschränkt zugänglich, da es z.B. nicht mehr möglich ist, aus der Datei ohne weiteres den Adressaten zu ermitteln.

Schließlich wird der gedruckte Brief per Fax an Firma Y geschickt. An diesem Punkt ist der Struktur Verlust total. Die im empfangenen Fax enthaltene Information beschreibt ein Bitmuster, d. h. auch die Daten und das Format sind verlorengegangen.

Das hier ein Problem besteht, zeigt sich schon daran, dass teilweise die empfangenen Fax-Daten gespeichert werden und mit einem OCR-Verfahren (*Optical Character Recognition*) versucht wird, aus dem Bitmuster Text und Format zu rekonstruieren. Beachtet man den so insgesamt getriebenen Aufwand zur Übertragung eines Textes, so erscheint das Verfahren grotesk. Es wird aber notwendig, wenn beispielsweise Fax-Bestellungen automatisch oder halbautomatisch ausgewertet werden sollen.

Ein Problem besteht auch, wenn beispielsweise Postscript oder RTF-Dateien übertragen oder gespeichert werden. Das Dokument wird nämlich nur dann identisch reproduziert, wenn in der Zie-

lumgebung exakt dieselben Schriften wie in der Quellumgebung zur Verfügung stehen. Ansonsten werden zwar die Daten korrekt wiedergegeben, die Substitution nicht vorhandener Schriften durch Standardschriften verändert jedoch die Formatierung, wodurch das Erkennen der Dokumentstruktur schwierig oder unmöglich wird.

Auch wenn das Erscheinungsbild einer Seite überall identisch reproduziert werden kann gehen Informationen verloren. Das von Adobe Systems entwickelte *Portable Document Format* (*PDF*) zielt darauf ab, eine geräte- und systemunabhängige Darstellung beliebiger im PDF-Format vorliegende Dokumente zu ermöglichen. Text-, Bild-, und Graphikbestandteile werden dabei durch PostScript wiedergegeben. Lokal nicht vorhandene Schriften werden mit Hilfe des *Adobe Type Manager* durch zwei *Multiple Master Fonts* (Serif und Sans Serif) wiedergegeben, wobei zwar der Schriftschnitt verlorengeht, die Metrik aber erhalten bleibt. Als Metrik bezeichnet man die verschiedenen Grössenparameter eines Buchstabens, also Oberlänge, Unterlänge, Höhe usw. Mit der Metrik bleiben dann auch die Position der Buchstaben auf der Seite und die Abstände der Buchstaben erhalten. Spezielle Symbolschriften sind natürlich auch weiterhin nicht portabel.

Wo liegt das Problem, wenn Daten und Formatierung korrekt wiedergegeben werden, und damit auch die Struktur des Dokuments - soweit sie durch die Formatierung wiedergegeben wurde - erfasst sein sollte? Das Problem ist das Grundproblem der Seitenbeschreibungs- und Satzsprachen: die Struktur ist sehr wohl erfassbar, aber nur für den menschlichen Leser. Es wird beispielsweise nicht zwischen einer fett gesetzten Überschrift und einem gleichfalls fett gesetzten hervorgehobenen Text unterschieden. Der menschliche Leser erkennt den Unterschied aus dem Kontext, der dokumentenverarbeitenden Software fehlt dazu jedoch das entsprechende (Welt-) Wissen. Wo aber Differenzierungen verlorengehen, geht Information verloren.

Einige Textverarbeitungssysteme bieten eine sogenannte Gliederungsfunktion, d.h. Überschriften, Abschnitte usw. können als solche gekennzeichnet werden. Ein so erfasster Text hat eine durch die Gliederungselemente des betreffenden Programms vorgegebene Struktur. Doch auch das ist nicht ausreichend. Zum einen deshalb, weil zur Struktur mehr gehört als nur die Gliederung, zum anderen, und das ist wesentlich, sollen die durch die Struktur differenzierten Bestandteile eines Dokuments präzise und flexibel beschrieben werden. Durch die Vielfalt von Dokumentenstrukturen verbietet es sich von selbst eine Einheitsstruktur, auch eine noch so allgemeine zu standardisieren, sondern die Mittel zu Beschreibung von Dokumentenstrukturen bereitzustellen.

3.2 Auszeichnen von Dokumenten

Wir haben gesehen, dass das Auszeichnen von Dokumenten einen entscheidenden Beitrag zu deren Verständnis beiträgt. So ist es nicht verwunderlich , dass Auszeichnungselemente dazu gebraucht werden Text- und Strukturelemente kenntlich zu machen. Wir haben aber auch gesehen, dass es unterschiedlicher Arten des Auszeichnens gibt:

- die graphisch oder physikalisch orientierte Auszeichnung und
- die inhaltlich oder logisch orientierte Auszeichnung.

Bei der physikalisch orientierten Auszeichnung setzt man Satzkommandos die das Format der betreffenden Textstelle verändern sollen direkt vor die Textstelle. Man setzt Attribute wie Fettdruck, Schriftart oder Kursiv. Das ist für den Anwender einfach und auf den ersten Blick sinnvoll, da er auf dieser Weise bei der Fixierung des Textes genauso arbeitet, wie es der Setzer gemäss den Vorgaben des Layouters beim Satz eines Textes macht.

Bei näherer Betrachtung gibt es mehrere Gründe gegen diese Vorgehensweise:

1. Der Autor ist nicht genügend geschult. Die Konsequenz kann sein, dass er die guten Ideen des Textes durch ein schlechtes Layout dem Leser nur unvollständig zugänglich macht.
2. Die mangelhafte Erfahrung des Autors kann schnell zur Inkonsistenzen der Inhaltlichen Darstellung führen; dies kann dem Leser das Verständnis erheblich erschweren
3. Die Entwicklungsmethodik des Dokumentes ist eine andere. Während der Autor dem Lektor einen fertigen Text gibt, den dieser auszeichnet und der dann vom Setzer gesetzt wird, überschneiden sich die Phasen im Falle, dass der Autor sein eigener Lektor und Setzer ist. Das bedeutet üblicherweise, dass das Dokument schon ausgezeichnet werden muss, während es nochin der Entwicklung ist. Es kann aber sein, dass ein Teil eines Dokuments, das ursprünglich als Abschnitt in einem Kapitel gedacht war, nachträglich zu einem eigenen Kapitel wird, so dass alle Auszeichnungen dieses Teils einzeln von Hand umgesetzt werden müssen. Dieses Vorgehen ist extrem fehleranfällig und aufwendig.
4. Schließlich sollte man sich noch einen weiteren Aspekt vor Augen halten: Dokumente werden nicht unbedingt nur auf dem Papier veröffentlicht, sondern häufig auch in elektronischen Medien. In diesem Fall braucht es nicht unbedingt eine adäquate Darstellung der oben angeführten Formatangaben zu geben. Es muss eine vollständig andere Form des Layouts gewählt werden, die aus der graphischen Auszeichnung nicht direkt ableitbar ist.

Diese Gründe legen das Einführen einer logisch orientierten Auszeichnung nahe. Die logisch orientierte Auszeichnung legt im Gegensatz zur physikalisch orientierten nicht das „Wie“, sondern das „Was“ fest. Während also der Autor mit einer physikalisch orientierten Auszeichnung bestimmt, wie sein Text dargestellt werden soll, bestimmt er mit einer inhaltlich orientierten Auszeichnung z.B., dass ein Text eine Überschrift ist. Diese Auszeichnung wird erst in einem späteren Schritt ausgewertet, genauso, wie es mit dem Manuskript des Autors gemacht wird, das von einem Lektor und einem Setzer nachgearbeitet wird.

Wenn die inhaltlich orientierten Auszeichnung dazu noch durch eine genormte Auszeichnungssprache erfolgt, liegen ihre Vorteile auf der Hand, z.B. beim schreiben eines Buches:

1. Statt spezieller Erfassungs-Richtlinien einzelner Druckereien und Verlage gelten einheitliche Regeln, die von allen gleichermassen akzeptiert werden. Die dem Autor mit der Zeit geläufigen Leitlinien können immer wieder angewendet werden.
2. Die arbeitsteilige Erstellung von Manuskripten, die in Forschungs- und Entwiklungseinrichtungen gebräuchlich ist wird durch allgemeine Leitlinien erleichtert. In derartigen Einrichtungen werden Training, Anwender-Beratung und ggf. die Unterstützung durch Programme vereinfacht und erzielen damit einen höheren Wirkungsgrad. Aus dem Einsatz der Leitlinien folgt, dass sich Autoren, die an getrennten Orten unabhängig voneinander Manuskripte für ein gemeinsames Verlagsobjekt wie eine Buchreihe erstellen, sich nicht auf die Einzelheiten einer einheitlichen typographischen Text-Gestaltung zu einigen brauchen, sondern nur auf die Prinzipien der gewünschten inhaltlichen Text-Strukturierung.
3. Jede Bearbeitung des Textes beinhaltet Fehlerquellen. Da die inhaltsbezogen markierten Manuskripte im Verlag oder in der Setze-

rei nicht von Hand editiert werden müsse, um die Text-Befehle an die Erfordernisse anzupassen ist diese Fehlerquelle abgeschaltet.

4. Durch Anwendung dieser Leitlinien soll der Autor aber auch in die Lage versetzt werden, seine Manuskripte besser und konsistenter zu strukturieren. Die Verbesserung der Manuskript-Qualität in formaler Hinsicht ist eine wichtige Vorbedingung für die Handhabung der Publikationen in elektronischen Datenbanken.

4 Markup und seine Anwendung

Das Gedankengut des Auszeichnens hat sich in allen Dokumeterstellungssystemen in verschiedene Art und Weise niedergeschlagen. In den nachfolgenden Beispielen soll dies gezeigt werden.

Exemplarisch werden zwei Systeme vorgestellt. Das Eine LaTeX, ein System zum Setzen von Dokumenten, das in Natur- und Geisteswissenschaftlichen Bereich zum Standart geworden ist, das Andere Word, ein gehobenes Textverarbeitungsprogramm das nach dem WYSIWYG Prinzip arbeitet.

4.1 LaTex - Die Auszeichnungssprache zu TeX

Zunächst muss einmal bemerkt weden , dass LaTeX ein Makropaket ist und auf der Formatiersprache

TeX basiert. In diesem Zusammenhang sollte man die deklarative Auszeichnungssprache Scribe anführen die in den späten siebziger Jahren von B. Reid an der Carnegie-Mellon Universität entwikkelt wurde, von welcher LaTeX den Begriff der Umgebung übernommen hat und somit eine Reimplementation von Scribe darstellt.

TeX bietet dem Autor nur sehr geringe Möglichkeiten, sein Dokument zu strukturieren. In TeX sind etwa 300 sogenannte Primitive definiert, mit denen Anweisungen zum Setzen eines Textes formuliert werden können, d. h. mit TeX-Befehlen wird im wesentlichen die Layout-Struktur eines Dokumentes festgelegt. Die Aussnahme bilden die Befehle im Mathematik-Modus, bei denen die logische Struktur einer Formel der zugehörigen Layout-Struktur entspricht.

Erst der Einsatz von Makropaketen mit Makros, die die logische Struktur angeben, ermöglicht dem Autor eine Auszeichnung seines Dokuments in der beschriebenen Weise. Zwei der wichtigsten Makropakete, die hier zu nennen sind, sind das von L. Lamport entwickelte LaTeX und AMS-TeX, das unter Leitung der American Mathematical Society entstand. Letztered wurde insbesondere zum Satz von mathematischen Formeln entwickelt und übertrifft in dieser Beziehung die Fähigkeiten des im folgenden beschriebenen LaTeX bei weitem.

L. Lamport verbindetin LaTeX einerseit die hohe typographische Qualität von TeX mit einer relativ einfachen Auszeichnungssprache. Dazu stellt sie dem Autor verschiedene vordefinierte Dokumenttypenzur Verfügung; diese sind `book`, `report`, `article`, und `letter` und bestimmen die generisch logische und Layout-Struktur, wobei beide Strukturen noch durch Optionen modifiziert werden können. Es sind ohne weiteres aber auch eigene Dokumenttypen zu definieren.

Weiter stehen dem Autor diverse Befehle zur Verfügung, die sowohl das Layout als auch die logische Struktur beeinflussen, allerdings auch allein inhaltliche Bedeutung haben können. Beispiele für die Befehle sind

- `\hspace`, was horizontalen Leerraum, also ein Layout-Element erzeugt,
- `\footnote`, was eine Fussnote, also ein logisches Element erzeugt und

-\LaTeX, was den Text „LaTeX“ erzeugt.

Die wesentliche Möglichkeiten, die TeX dem Autor zur Strukturierung seiner Dokumente bietet, sind die Umgebungsbefehle und die Befehle zur Dokumentuntergliederung. Die Umgebung wird eingeleitet mit dem Kommando \begin{(*Umgebung)*}, sie wird beendet mit dem Kommando \end{*Umgebung*}. Der Parameter ((*Umgebung)*) legt fest, welcher Text und welche Kommandos in der Umgebung stehen dürfen und er bestimmt, gemäss welchem Layout die Umgebung formatiert wird. Die Befehle zur Dokumentuntergliederung schliesslich dienen dazu, ein Dokument in Kapitel, Abschnitte Unterabschnitte usw. zu unterteilen.

4.1.1 LaTeX-Umgebungen

LaTeX selbst definiert Umgebungen für

- den Mathematik-Modus,
- zentriertes, links- und rechtsbündiges Formatieren,
- das Formatieren diverser Listenarten,
- das Formatieren von Tabellen,
- das Formatieren einfacher Bilder,
- das Formatieren von Bibliographien und
- das Formatieren sogenannter „fluktuierender Elemente[2]“ (*floating bodies)*.

Weiterhin ist es erlaubt, sehr viele Layout-Befehlsnamen als Parameter von \begin{...} und \end{...} zu benutzen; schließlich bietet LaTeX dem benutzer auch noch die Möglichkeit, eigene Umgebungen zu definieren.

Die Syntax der Umgebungskommandos ist einfach: Eine Umgebung wird mit \begin{(*Umgebung*)} geöffnet und muss mit dem zugehörigen \end{(*Umgebung*)} geschlossen werden, zusätzlich dürfen Umgebungen geschachtelt werden. Darüberhinaus ist innerhalb der Listen-Umgebung der Befehl \item definiert, in der Bibliographie-Umgebung der Befehl \bibitem. Diese dienen der weiteren Strukturierung des Dokumentes im Sinne des folgenden Abschnittes.

4.1.2 LaTeX-Gliederungsbefehle

LaTeX kennt die folgenden Gliederungsbefehle (in absteigende Hierarchie):

1. \part,
2. \chapter,
3. \section,
4. \subsection,
5. \subsubsection,
6. paragraph und
7. \subparagraph.

Diese Gliedern das Dokument in natürlicher weise in Kapitel, Abschnitt usw.. Zusätzlich zu diesen Gliederungsbefehlen können die Befehle \item und \bibitem auftreten, die jeweils auf der nächstfolgenden Hierarchiestufe anzusiedeln sind.

Beim Formatieren wird die korrekte Hierarchie der Gliederungsbefehle innerhalb des Dokumentes nicht überprüft; es ist dem Autor überlassen, die Gliederungshierarchie zu beachten oder nicht. Au-

sserdem muss der Autor darauf achten, dass sich Gliederungsbefehle nicht mit Gruppierungsbefehlen überschneiden. Es ist durchaus möglich, ein Dokument mit LaTeX-Kommandos folgendermassen auszuzeichnen:

```
\begin{document}
  \paragraph{Ein Absatz}
   Inhalt des Absatzes
   \begin{quote}
     \subsection{Ein Unterabschnitt}
       Inhalt des Unterabschnittes
     \end{quote}
  \section{Ein Abschnitt}
     Inhalt des Abschnittes
\end{document}
```

Dies widerspricht natürlich jeder Logik, wird aber beim Formatieren nicht als fehlerhaft erkannt.

Es fällt auf, dass das Auszeichnungskommando für den Absatz fehlt. Absatz ist die kleinste logische Einheit, die der Autor definieren kann; dabei kann diese Einheit explizit mittels `\par` oder implizit durch die Eingabe einer Leerzeile definiert werden.

Als Abschluss dieser Betrachtung ein kleiner Nachteil dieses Systems. LaTeX ist ein kompilatives Satzsystem, so dass das Dokument erst nach dem Formatierungslauf, der sich der Eingabe anschliesst betrachtet werden kann. In manchen Fällen jedoch ist man während der Erstellungsphase am Layout der bisherigen Arbeit interessiert. Dies kann manchmal gar nicht oder aber durch den Formatierungslauf verursachten Verzögerung geschehen die vom Anwender nicht akzeptiert wird. Mit dem Fortschreiten der Prozessortaktraten sollte in naher Zukunft auch das kein Nachteil mehr sein.

4.2 Word - Ein gehobenes Textverarbeitungssystem

Ein gänzlich anders, nicht nur seiner äusseren Form sondern seines verwendeten Prinzips wegen, geartetes Programm ist Word. Es arbeitet nach dem WYSIWYG (*what you see is what you get*) Prinzip, was bedeutet, dass das Ergebniss dem Autor direkt bei der Eingabe angezeigt wird. Das Programm stellt dem Autor alle Hilfsmittel bereit, die zur Dokumentenerstellung benötigt werden: einen Editor zur Eingabe des Textes, einen Formatierer, der den Text gemäss den Layout-Einstellungen formatiert und schliesslich einen Gerätetreiber, der das Dokument für verschiedene Ausgabegeräte Aufbereitet. Ein weiteres wichtiges Kennzeichen dieses Systems ist, dass die Auszeichnung graphisch orientiert Erfolgt.

Dem Anwender erschließt sich Word über eine Vielzahl von Menüs die es Ihm ermöglichen fast alle Formateinstellungen vorzunehmen. Hier ein kurzer Auszug der Einstellungsmöglichkeiten:

Die Schriftgestaltung erfolgt durch:

- Schriftart und -grösse,
- Hervorhebung fett, kursiv, unterstrichen,
- Zeichen hoch- und tiefstellen,

Die Seitengestaltung erfolgt durch:

- einstellen der Seitenränder und des Papierformats,
- einfügen von Kopf- und Fusszeile,
- einfügen von Seitenzahlen,

- einfügen von Fussnoten.

Ein überaus nützliches Merkmal von Word ist die Tatsache , dass es viele integrierte Funktionen zur Eingabe und Bearbeitung von Text bereitstellt. Sie erstrecken sich über das Kopieren und Verschieben per Drag & Drop, automatische Silbentrennung, den Thesaurus, automatische Grammatikprüfung und viele weitere Funktionen. Die Bearbeitung oder Erstellung von gleichartigen Textpassagen kann über Formatvorlagen erfolgen, was eine nachträgliche Änderung der betreffenden Textstellen erleichtert. Auf den ersten Blick scheint dieses System dem kompilativen überlegen zu sein, den „man sieht sofort was man schreibt“. Dieser Vorteil sollte jedoch nicht überbewertet werden, den die Formatierung kann nur in dem Rahmen beeinflusst werden, die das System vorgibt. So ist man beim Satz eines Textes voll und ganz auf die Vorgaben (u. a. den oben angeführten Trennalgorithmus und das typographische Modell) angewiesen, die nicht unbedingt den Vorstellungen eines hochwertigen Satzes entsprechen. Wie bereits an anderer Stelle dargelegt, ist die Aufgabe des Autors, sich auf den logischen Aufbau und den Inhalt des Dokumentes zu konzentrieren, nicht aber auf dessen Darstellung. Es ist allerdings festzuhalten, dass Word, hier ein Vertreter der WYSIWYG basierten Programme, bei Dokumenten wo das Layout wichtiger ist als die logische Struktur durchaus die bessere Wahl sein kann.

5 Literatur

[Bad 87] Bader, G.: Desktop publishing: Setzen u. drucken in eigener Regie, 1. Aufl. - Würzburg: Vogel, 1987

[BeMi 98] Behme, H., Minert, S.: XML in der Praxis, S. 15-42, Addison-Wesley, 1998

[Gold 97] Goldfarb,Ch.: SGML History Niche, 1997, http://www.sgmlsource.com/history/index.htm [11.09.98]

[Lam 95] Lamport, L.: Das LaTeX-Handbuch. Bonn; Paris;Readind, Mass.[u. a.]: Addison-Wesley, 1995

SGML

Lars Brauckmann & Jens Schröder

FB Informatik, Uni Dortmund
brauck00@ls1.cs.uni-dortmund.de schroe01@marvin.cs.uni-dortmund.de

Zusammenfassung

SGML steht für Standard Generalized Markup Language, und ist im Jahre 1986 von der ISO als internationaler Standard zur Beschreibung von Dokumentstrukturen verabschiedet worden. Die drei wesentlichen Bestandteile eines SGML Dokuments sind SGML Declaration, Prolog und Document Instance Set. Während die SGML Declaration die lexikalische Struktur des Dokuments bestimmt, gestattet es der Prolog eine Document Type Definition zu definieren. Eine Document Type Definition kann als eine Strukturbeschreibung für eine Menge gleichartiger Dokumente angesehen werden. Das Document Instance Set stellt den eigentlichen Dokumentinhalt dar, der entsprechend der Document Type Definition mit Tags durchsetzt wird.

1 Einleitung

SGML steht für **S**tandard **G**eneralized **M**arkup **L**anguage und ist von der International Standard Organization (ISO) im Jahre 1986 als ISO 8879 als internationaler Standard zur Beschreibung von Dokumentstrukturen veröffentlicht worden.

SGML ist eine (Meta-)Sprache, welche es erlaubt Dokumentstrukturen zu beschreiben. Genauer gesagt bietet SGML die Möglichkeit, eine (Markup-)Sprache zu definieren, welche dann wiederum verwendet werden kann, um ein Dokument zu strukturieren.

Was unter der Struktur eines Dokumentes zu verstehen ist, soll nicht im Vordergrund dieser Arbeit stehen (näheres dazu siehe "SGML für die Praxis", Rieger 1995). Um dieser Ausarbeitung folgen zu können, sei nur soviel erwähnt, daß die Struktur eines Dokumentes

- durch seine Einzelbestandteile und
- durch Regeln, wie diese Bestandteile in Beziehung stehen,

bestimmt wird.

Die Bestandteile eines Buches sind bspw. :

- Kapitel
- Abschnitt
- Anhang

Regeln, welche die Beziehung der Bestandteile bestimmen, können bspw. sein :

- Ein Buch muß aus mindestens einem Kapitel bestehen.
- Ein Kapitel muß mindestens einen Abschnitt enthalten.
- Ein Abschnitt darf nur innerhalb eines Kapitels auftauchen.

2 Eigenschaften von SGML

2.1 Descriptive Markup

Um Bestandteile eines Dokumentes zu kennzeichnen, müssen zusätzliche Informationen in ein Dokument eingefügt werden. Diese zusätzlichen Informationen werden auch als MARKUPS bezeichnet. Beim Descriptive Markup hat der Anwender die Möglichkeit, Teile seines Dokuments zu kennzeichnen, und damit die Gesamtstruktur seines Dokuments festzulegen. Das Kennzeichnen von Dokumentteilen erfolgt dabei mit Hilfe von Markup-Codes.

Erwähnenswert ist an dieser Stelle, daß durch das Einfügen von Markups in einen Text nur die Struktur des Textes festgelegt wird, aber nichts darüber festgelegt wird, wie die einzelnen Strukturelemente angezeigt oder verarbeitet werden (d.h. Formatierung spielt keine Rolle).

```
<beginbook>
        <begintitle>
                All About SGML !
        <endtitle>
        <beginchapter>
                <beginchaptertitle>
                Einführung
                <endchaptertitle>
                <beginchapterbody>
                SGML steht für Standard Generalized Markup
                Language, ...
                <endchapterbody>
        <endchapter>
        <beginchapter>
                <beginchaptertitle>
                Merkmale von SGML
                <endchaptertitle>
                <beginchapterbody>
                ...
                <endchapterbody>
        <endchapter>
<endbook>
```

SGML unterstützt Descriptive Markup, indem es die Möglichkeit bietet Markup-Codes zu definieren. Diese Markup-Codes werden auch als TAGS bezeichnet. SGML gibt weiterhin vor, wie diese TAGS in den Inhalt von Dokumenten eingefügt werden müssen, um einzelne Bestandteile des Dokumentes auszuzeichnen.

2.2 Dokumentklassen und Dokumentinstanzen

Wie bereits erwähnt, ist SGML eine Meta-Sprache, welche die Definition einer Markup-Sprache ermöglicht. Im SGML-Sprachgebrauch ist jedoch nicht von der Definition einer Markup-Sprache, sondern von der Definition einer DOKUMENTKLASSE (Document Type Definition (DTD)) die Rede. Eine Dokumentklasse kann als Strukturbeschreibung einer Menge gleichartiger Dokumente verstan-

den werden. Ein einzelnes Dokument, was der Struktur der Dokumentklasse entspricht, wird dann als DOKUMENTINSTANZ bezeichnet. Eine Dokumentklasse legt somit fest, welche Bestandteile ein konkretes Dokument dieser Klasse besitzen kann bzw. muß, und nach welchen Regeln diese Bestandteile angeordnet werden können bzw. müssen.

Um die Bestandteile innerhalb einer Dokumentinstanz auszeichnen zu können, werden innerhalb der DTD die bereits erwähnten TAG's definiert. Die Zusammenfassung von erlaubten Bestandteilen und Regeln zur Anordnung dieser Bestandteile soll im folgenden als SYNTAKTISCHE STRUKTUR eines Dokumentes bzw. einer Dokumentklasse bezeichnet werden.

Durch die Bildung von Dokumentklassen bietet SGML somit die Möglichkeit, sämtliche Eigenschaften, die einer Menge von Dokumenten gemein sind, in einer Definition zusammenzufassen. Durch die Zugehörigkeit zu einer Dokumentklasse werden Dokumente somit typisiert, und es besteht die Möglichkeit, die syntaktische Korrektheit von Dokumenten unter Bezugnahme auf die DTD dieses Typs zu überprüfen. Des weiteren wird durch die Bildung von Dokumentklassen die einheitliche Verarbeitung von Dokumenten, die zu der gleichen Klasse gehören, vorangetrieben.

2.3 Datenunabhängigkeit

Ein wesentliches Ziel bei der Entwicklung von SGML war, daß Dokumente ohne Informationsverlust von einem Gerät bzw. System auf ein anderes übertragen werden können.

Probleme beim Austausch von Dokumenten zwischen zwei Rechnern treten immer dann auf, wenn diese beiden Rechner unterschiedliche Zeichensätze verwenden, und somit mit ein und demselben Dezimalcode unterschiedliche Zeichen verbinden (Bsp.: Der Dezimalcode 76 führt in ASCII zu "L" und in EBCDIC zu "<"). Um dieses Problem zu lösen, bietet SGML die Möglichkeit, den in einem Dokument verwendeten Zeichensatz auf den vom System verwendeten Zeichensatz abzubilden. Diese Lösung garantiert somit die Unabhängigkeit eines SGML-Dokuments von der Codierung eines speziellen Rechnersystems.

Ein weiteres Problem stellt der beschränkte Zeichenvorrat eines konkreten Zeichensatzes dar. In vielen Fällen sind die angebotenen Zeichen für einen Anwender nicht ausreichend oder besitzen eine Mehrdeutigkeit. Bspw. wird das Zeichen "-" (ASCII 45) sowohl als Bindestrich als auch als Minuszeichen verwendet. Um diese Mehrdeutigkeit aufzulösen bietet SGML die Möglichkeit, einem Zeichen oder einer Zeichenkette eine Rolle zuzuweisen, und somit die Mächtigkeit des zugrundeliegenden Zeichensatzes erweitern.

Auch Probleme hinsichtlich verschiedener SGML-Anwendungen sind lösbar. Um ein Dokument so portabel wie möglich zu gestalten, besteht die Möglichkeit FEATURES, die nicht von allen SGML-Anwendungen unterstützt werden müssen, zu verbieten.

3 Definition von SGML

3.1 Abstract Syntax und Concrete Syntax

Die von SGML definierte Sprache zur Definition einer Markup-Sprache wird indirekt definiert, d.h. die in der Grammatik verwendeten Zeichen und Symbole sind nicht an einen konkreten Zeichensatz gebunden. Diese indirekte Definition wird als "Abstract Syntax" bezeichnet. Diese "abstract syntax"

wird durch eine kontextfreie Grammatik **G = (N,T,R,S)** definiert, wobei

N die Menge der Nichterminal (syntactic variables)

T die Menge der Teminalzeichen (terminal variables, delimiter roles, terminal constants, syntactic literals)

R die Menge der syntaktischen Produktionen

S das Startsymbol („SGML Document“)

darstellt.

Die Terminalzeichen zerfallen dabei in die angegebenen vier Klassen terminal variables, delimiter roles, terminal constants und syntactic literals.

terminal constants Diese Klasse umfaßt die Menge der Zeichen/-klassen, die vom Standard fest vorgegeben sind, und nicht weiter verändert werden können. Ein Beispiel ist die Klasse LC Letter, die sämtliche Zeichen umfaßt, die als Kleinbuchstaben interpretiert werden.

terminal variables Diese Klasse umfaßt die Menge der Zeichen/-klassen, die vom Anwender spezifisch erweitert werden können. Als Beispiel sei hier die Klasse LCNMCHAR genannt, die sämtliche Zeichen umfaßt, die zusätzlich zu den Zeichen der Klasse LC Letter als Kleinbuchstaben interpretiert werden und innerhalb von SGML zur Bildung von Bezeichnern verwendet werden können.

syntactic literals Diese Klasse umfaßt sämtliche Schlüsselwörter von SGML. Auch die Schlüsselwörter können vom Anwender spezifisch abgeändert werden. Beispielsweise das Schlüsselwort DOCTYPE, welches dazu dient, eine Dokumenttypdefinition einzuleiten.

delimiter roles Unter Delimitern versteht man sämtliche Zeichen bzw. Zeichenketten, die zur Deklaration und Begrenzung von Markierungen verwendet werden. Innerhalb der "abstract syntax" werden keine speziellen Zeichenketten verwendet, sondern Symbole für die Delimiter angegeben. Als Beispiel sei der Delimiter mdo (Markup Declaration Open) erwähnt, der dazu dient, eine Tag-Deklaration einzuleiten.

Durch die Definition einer "concrete syntax" hat der Anwender die Möglichkeit, die Terminalzeichen auf Zeichenfolgen eines konkreten Zeichensatzes abzubilden. Um dies zu erreichen wird die "abstract syntax" folgendermaßen abgeändert: Es wird eine kontextfreie Grammatik G' definiert, mit **G'=(N',T',R',S)**, wobei

N' = N ∪ M={terminal variables, delimiter roles}

T' = {terminal constants,syntactic literals} ∪ K={Konstanten aus dem konkreten Zeichensatz}

R' = R ∪ {Menge der Produktionen, die M auf K abbilden}

S das Startsymbol („SGML document“)

darstellt.

Um den Anwendern das Arbeiten mit SGML zu erleichtern, und eine Standard "concrete syntax" festzulegen, definiert SGML vier "concrete syntaxes"

- Reference Concrete Syntax
- Core Concrete Syntax
- Multicode Basic Concrete Syntax
- Multicode Core Concrete Syntax,

wobei die Reference Concrete Syntax den quasi-Standard repräsentiert.

Durch die Reference Concrete Syntax wird beispielsweise festgelegt, daß der Delimiter mdo durch die Zeichenkette "<!" dargestellt wird, und daß die Klasse LCNMCHAR die Zeichen ".-" enthält.

Im folgenden aufgezeigte Beispiele werden unter Verwendung der Reference Concrete Syntax gebildet.

3.2 Notationsvereinbarungen

Die in der Grammatik verwendeten Produktionen haben folgenden Aufbau:

[Nr] syntactic variable = expression

[Nr]	:	Nummer der Produktion
syntactic variable	:	Name eines Nichtterminals
=	:	Zuweisungssymbol
expression	:	Ausdruck, zusammengesetzt aus *syntactic tokens* und *Metazeichen*

syntactic tokens können sein

- syntactic variables,
- syntactic literals (Schlüsselwörter),
- terminal variables (variable Zeichen/Zeichenklassen),
- terminal constants (konstante Zeichen/Zeichenklassen),

- delimiter roles (Symbole)

Metazeichen können sein :

- (), zur Klammerung von syntactic tokens
- occurrence delimiter, um die Häufigkeit des Auftretens eines syntactic token festzulegen

?	0 oder 1	(optional)
*	>= 0	(optional and repeatable)
+	>= 1	(required and repeatable)

- connector, um die Reihenfolge der syntactic token festzulegen

,	SEQ	(alle syntactic token in angegebener Reihenfolge)
|	XOR	(genau eines der syntactic token)
&	AND	(alle syntactic token in beliebiger Reihenfolge)

Beispiel (Auszug aus der "abstract syntax"):

[55] name =
name start character [53] ,
name character [52] *

[52] name character =
name start character [53] **|**
Digit **|**
LCNMCHAR **|**
UCNMCHAR **|**

[53] name start character =
LC Letter **|**
UC Letter **|**
LCNMSTRT **|**
UCNMSTRT

[91] comment declaration =
mdo ,
(comment [92] ,
(s [5] **|**
comment [92])*)?,
mdc

[174] base character set =
"BASESET" ,
ps [65] +,
public identifier [74]

Beispiel für

- syntactic variable — ***name, name start character, name character, comment declaration, comment, ps, public identifier***
- terminal constant — ***LC Letter, UC Letter***
- terminal variable — ***LCNMCHAR, UCNMCHAR, LCNMSTRT, UCNMSTRT***
- delimiter role — ***mdo, mdc***
- syntactic literal — ***BASESET***
- Metazeichen — ***"(", ")", "|", "*", "?", ",", "+"***

4 Aufbau eines SGML Dokuments

Durch Regel [1] der "abstract syntax" ergibt sich der Aufbau eines SGML-Dokuments.

[1] SGML document = SGML document entity [2],
(SGML subdocument entity [3] |
SGML text entity [4] |
character data entity [5.1] |
specific character data entity [5.2] |
non-SGML data entity [6])*

Der wichtigste Teil, der in dieser Produktion festgelegt wird, ist die ***SGML document entity***. Diese syntactic variable ist zwingend erforderlich, und definiert somit, welche Elemente innerhalb eines SGML Document auftauchen müssen.

Die notwendigen Elemente werden somit durch Regel [2] der "abstract syntax" definiert.

[2] SGML document entity = s [5] *,
SGML declaration [171] ,
prolog [7] ,
document instance set [10] ,
Ee

Die ***SGML declaration*** definiert die lexikalische Struktur des SGML-Dokuments, d.h. welche Zeichensätze dem Dokument zu Grunde liegen, welche Zeichen dieses Zeichensatzes innerhalb des restlichen Dokuments auftauchen dürfen, in welcher Art und Weise diese Zeichen zu Wörtern kombiniert werden dürfen und durch welche Zeichen die Nichtterminale der "abstract syntax" dargestellt werden.

Der ***prolog*** definiert die syntaktische Struktur der Dokumentinstanz. An dieser Stelle wird die eigene Markup-Sprache definiert und die Struktur für die Dokumentklasse festgelegt.

Das ***document instance set*** stellt den eigentlichen Inhalt des Dokuments, die sogenannte Dokumentinstanz dar. Hier wird der eigentliche Dokumenteninhalt mit den im prolog definierten Markups, entsprechend der im prolog festgelegten Syntax, durchsetzt, um den Dokumenteninhalt zu strukturieren.

5 Der Prolog

5.1 Ziele, Aufgaben und Funktionen

Bei einer deskriptiven Markup – Sprache muß ein Dokument in logische Strukturen und Bestandteile zerlegt werden. Im wesentlichen bedeutet dies, daß ähnlich wie bei einer Programmiersprache, eine Klasse eines bestimmten Dokumenttyps definiert werden muß, welche die einzelnen Bestandteile explizit benennt und somit definiert. Anschließend können dann die Definitionen der Dokumentklasse in der Instanz zur Auszeichnung eines Elementes verwendet werden. Diese Klassenbildung ermöglicht eine zentrale, einheitliche Definition von Dokumenten.

Im Prolog wird vom Entwickler die syntaktische Struktur eines Dokuments beschrieben. Diese logische Struktur muß in jedem Dokument definiert sein. Diese Strukturinformationen werden später vom Parser benötigt, um die Dokumentinstanz mit den Definitionen aus dem Deklarationsteil zu vergleichen und zu überprüfen, ob die Instanz mit der Deklaration „stimmig“ ist. In anderen Worten heißt dies, daß die Auszeichnungen definiert werden, welchen später auch in der Dokumentinstanz erlaubt sein sollen.

Formal besteht der Prolog aus den Elementen :

[7] prolog = **base document type declaration [9],** **(other prolog [8]*,** **document type decalration [110]** **)***	**[9] base document type declaration =** **document type declaration [110]**

Hier wird anhand der Produktionen der abstract Syntax definiert, daß ein Prolog aus mindestens einer oder mehreren Document Type Declarations besteht. Die erste der Deklarationen ist die Base Document Type Declaration. Sie ist zwingend in einem Dokument erforderlich, während die weiteren Deklarationen nur optional sind und somit (meistens) entfallen können. Auf diese Weise wird erzwungen, daß ein SGML-Dokument mindestens eine Dokument-Typ-Deklaration beinhaltet. Der Prolog erlaubt also eine Deklaration von **mehreren Dokumenttypen** innerhalb eines **SGML-Dokuments**. In der Dokumentinstanz kann nun angeben werden, auf welchen Dokumenttyp man sich gerade bezieht

< **(Brief)** Absender >

< **(Buch)** Kapitel >

Bei fehlender Angabe der Dokumentklasse wird immer die Basis-Dokumentklasse (*base document type declaration*) als Standardeinstellung gewählt.

5.2 Document Type Declaration

Eine Document Type Declaration beschreibt eine Klasse eines bestimmten Dokumenttyps. Die Document Type Declaration beinhaltet

1. Regeln, welche die syntaktische Struktur einer Dokumentklasse beschreiben
2. Eine Referenz auf diese Regeln
3. Sie besteht aus einer Kombination der ersten beiden Punkte

Diese Regeln zur Beschreibung der syntaktischen Struktur bezeichnet man auch als *„Document Type Definition“* . Da ein Dokument als Instanz von einer Dokumentklasse abgeleitet wird, besitzen alle Dokumente der selben Klasse die gleiche DTD. Aus diesem Grund müssen für die verschiedensten Dokumente eigene DTD’s erstellt werden. So existieren in Unternehmen z.B. eigene DTD’s für

1. Geschäftsberichte
2. Memoranden
3. Verträge

...

Da sich diese Regeln in der Document Type Declaration befinden, wollen wir diese einmal näher betrachten. Die Document Type Declaration wird aus der Regel [110] der abstract Syntax abgeleitet.

```
[110] document type declaration =
        mdo,
        „DOCTYPE“,
        ps+,
        document type name [111],
        (       ps+,
                external identifier [73],
                ps+
        )?,
        (       ps+,
                dso,
                document type declaration subset [112],
                dsc,
                ps+
        )?,
        ps+,
        mdc
```

Mit der obigen Produktion lassen sich die Bestandteile einer Document Type Declaration identifizieren :

1. Markup-Declaration-Open (mdo) == „<!“
2. Einem Markup-Declaration-Close (mdc) == „>“
3. Das Schlüsselwort „DOCTYPE“, welches die DTD einleitet
4. Einem Dokumentnamen
5. Einem externen Bezeichner
6. Einem Declaration-Subset-Open (dso) == „[„
7. Einem Declaration Subset-Close (dsc) == „]“
8. Und einer Document Type Declaration Subset

Das Markup Declaration Open weißt den Parser darauf hin, daß hier eine Deklaration eines Dokumentbestandteils beginnt. Es entspricht somit ungefähr dem C – Befehl **`typedef`**. Das Schlüsselwort *„DOCTYPE“* teilt dem Parser mit, daß nun die Document Type Declaration beginnt, welche die DTD beinhaltet. Der Name einer DTD läßt sich von der Regel [111] ableiten

[111] document type name = ***generic identifier [30]*** ***[30] generic identifier =*** ***name [55]***	***[55] name =*** ***name start character [53],*** ***name character [52]****

Es läßt sich aus dieser Produktion die Grobstruktur einer Dokumentdeklaration entnehmen. Am Beispiel würde eine solche Deklaration wie folgt aussehen :

```
<!DOCTYPE DTDName     SYSTEM „/usr/sgml/dtd/meine.dtd“
      [
      ...   Hier die Modifikationen von „meine.dtd“
      ]
      >
```

Der Name einer DTD ist ein sogenannter *„generic identifier“* . Über diesen generic identifier werden Auszeichnungselemente benannt und über die SGML Applikation voneinander unterschieden.

Ein weiterer Bestandteil der Produktion ist der externe Bezeichner. Dieser ist optional und darf somit fehlen. Ein externer Bezeichner legt fest, daß ein Element der Deklaration nicht explizit im Dokument angegeben wird, sondern extern gelagert ist und beispielsweise über eine Datei eingebunden wird.

[73] external identifier =	***[74] public identifier =***
(„SYSTEM“	***minimum literal [76]***
|	
(„PUBLIC“,	***[75] system identifier =***
ps+,	***lit,***
public identifier [74],	***system data [45],***
)	***lit***
),	
(ps+,	***[45] system data =***
system identifier [75]	***character data [47]***
)?	

Bei den externen Bezeichnern gibt es in zwei Varianten. Einen Systembezeichner und einen Public-Bezeichner. Ein Systembezeichner signalisiert der Applikation, daß die Daten sich in einer externen Datei befinden. Dem Schlüsselwort *„SYSTEM“* folgt ein optionaler Dateiname, in dem obigen Beispiel `„/usr/sgml/dtd/meine.dtd“`. Wenn diese explizite Angabe fehlt, sucht der Parser nach einer Datei, welche den gleichen Namen trägt wie die Document Declaration. Im obigen Beispiel würde der Parser also nach einer Datei mit dem Namen DTDName suchen.

Bei dem Public-Bezeichner handelt es sich um eine dem Parser bekannten Zeichenkette. Diese Zeichenkette wird vom Parser mit Hilfe interner Tabellen in einen Systemnamen übersetzt.

Über die DTD-Subset-Declaration hat man die Möglichkeit eine DTD neu zu erstellen oder eine über einen externen Bezeichner geladene DTD zu erweitern und zu modifizieren.

Die Document Type Declaration Subset wird von der Regel [112] abgeleitet :

[112] document type declaration subset =	***[113] entity set =***
(element set [114],	***(entity declaration [101]***
entity set [113],	***|***
short reference set [115])***	***ds [71])******
[114] element set =	***[115] short reference set =***
(element declaration [116]	***(entity declaration [101]***
|	***|***
attribute definition list declaration [141]	***short reference mapping declaration [150]***
|	***|***
notation declaration	***short reference use declaration [152]***
|	***|***
ds [71])***	***ds)******

An dieser Stelle der abstract Syntax gelangt man endlich an die eigentliche Dokumentmodellierung. Hier werden die wesentlichen Deklarationen zur Modellierung und somit zur Klassifizierung festgelegt. Die wesentlichen Bestandteile einer Document Type Definition sind somit

1. Elemente
2. Entitäten
3. Attribute
4. Notationen

Im folgenden werden diese wesentlichen Bestandteile einer Document Declaration vorgestellt.

5.3 Das Element

Die Elementdeklaration bildet wohl den wesentlichsten Bestandteil einer DTD und somit einer Dokumentmodellierung. Bei der Erarbeitung einer Dokumentstruktur wird das Hauptaugenmerk auf die logische Struktur eines Dokuments geworfen. Hier wird die Gliederung der Datenelemente sowie ihre Beziehungen untereinander beschrieben. Daten werden zu zusammengehörenden Informationspaketen zusammengefaßt und somit von anderen Daten abgegrenzt, die nicht so stark zu diesen Daten gehören.

Zu diesem Zweck muß es dem Entwickler möglich sein, Daten und Informationsobjekte zu kapseln und diese durch eindeutige, sinnvolle Bezeichnungen zu benennen. Auf diese Weise wird das Dokument in seine semantischen Strukturen zerlegt, da die gekapselten Datenobjekte durch beschreibende Namen ausgezeichnet werden. Formal wird eine Elementdeklaration in der abstract Syntax wie folgt definiert :

```
[116] element declaration =
        mdo,
        „ELEMENT“,
        ps+,
        element type [117],
        ps+,
        ommited tag minimization [122],
        ps+,
        (
                declared content [125]
                |
                content model [126]
        ),
        ps+,
        mdc
```

```
[117] element type =
        generic identifier [30]
        |
        name group [69]

[69] name group =
        grpo,
        ts [70]*,
        name [55],
        (       ts*,
                connector [131],
                ts*,
                name [55],
                ts*
        )
        grpc
```

5.3.1 Bestandteile und Aufbau einer Elementdeklaration

Aus der obigen Produktion lassen sich die folgenden wesentlichen Bestandteile einer Elementdeklaration ableiten :

1. Das Schlüsselwort „ELEMENT“
2. Der Name des Elements („element type“)
3. Minimierungsdeklaration („omitted tag minimization“)
4. Der Elementinhalt („declared content“ oder „content model“)

Eine Beispieldeklaration könnte somit folgendes Aussehen haben :

```
<!ELEMENT ElemName - - ( Inhalt ) >
```

Das Schlüsselwort „ELEMENT“ weist diese Deklaration als Elementdeklaration aus und der Parser nimmt den Elementnamen in die Liste der für die Instanz gültigen Auszeichnungen auf. Der Name kann allerdings auch aus einer sog. „Model Group“ bestehen. Eine ModelGroup bezeichnet zwei Elemente mit verschiedenen Namen, aber dem völlig identischen Inhaltsmodell.

<!ELEMENT (Absender, Empfaenger) – – (Content) >

Die Minimierungsdeklaration (die zwei Minuszeichen zwischen dem Elementnamen und dem Inhalt) dient der Bestimmung, ob in der Dokumentinstanz vor und nach dem Element die Start- und Endmarkierung vorhanden sein müssen. Der Elementinhalt umfaßt das eigentliche Element. Hier werden die Daten und Informationsobjekte zusammengefaßt und definiert. Der Elementinhalt kann in zwei verschiedenen Variationen auftreten.

[125] declared content = ***„CDATA“*** | ***„RCDATA“*** | ***„EMPTY“***	***[126] content model =*** ***model group [127],*** ***exceptions [138]****

5.3.2 Declared Content

Der deklarierte Inhalt (=declared content) beinhaltet lediglich drei verschiedene vordefinierte Inhaltsmodelle.

CDATA

Der erste der drei vordefinierten Inhaltstypen ist CDATA. Dies steht für **c**haracter-**data**. Der Text innerhalb dieser Auszeichnung wird bis auf die zugehörige Endmarkierung vom Parser vollkommen ignoriert.. Sämtliche Markup-Auszeichnungen als auch →Entitäten werden vom Parser weder identifiziert noch interpretiert.

Diese Art von Text eignet sich für bestimmte Textpassagen, welche Auszeichnungsähnliche Elemente enthalten die nicht interpretiert werden sollen. Als Beispiel soll hier ein Programmtext betrachtet werden, der ein C-Programm beinhaltet.

```
<!ELEMENT Code - - CDATA >
```

Diese Element könnte dann in der Instanz das Programm enthalten und es somit von anderen Daten kapseln und isolieren.

```
//1   <Code>
//2   for ( i=0 ; i<x ; i++)
//3   {
//4         printf(´´Month : %s\n´´, MonthName(i) );
//5   }
//6   </Code>
```

In Zeile 2 könnte der Parser die Zeichenfolge „i<x“ als Elementbeginn interpretieren. Da aber in einem CDATA Inhalt keine Auszeichnungen identifiziert werden, wird der Parser erst wieder nach der Endmarkierung </Code> mit der Identifikation von Elementen beginnen.

RCDATA

Der zweite vordefinierte Inhaltstyp ist RCDATA und steht für ***r***eplaceable-***c***haracter-***data***. Der Inhalt dieses Elements wird zwar ebenfalls nicht auf Markups überprüft (abgesehen von der eigenen Endmarkierung), wohl aber werden →generische Entitäten.

```
<!ELEMENT formel - - RCDATA >
```

In der zugehörigen Instanz würde das Formelelement dann folgend instanziiert:

<formel> Result &equal; <X&sub2;>

Der Parser ersetzt die Entities, ignoriert aber die eckige Klammer die auch eine Startmarkierung darstellen könnte, und liefert das Ergebnis :

Result = $<X_2>$

EMPTY

Es kann unter Umständen auch nützlich sein Elemente ohne irgendeinen Inhalt zu deklarieren. Ein solches Element enthält zwar keinen Text, könnte aber beispielsweise vom Parser durch eine externe Grafik ersetzt werden. Das EMPTY-Element dient im Grunde nur als qualifizierbarer Platzhalter, da dem Parser noch über Elementattribute noch einige Informationen übergeben werden können.

5.3.3 Inhaltsmodell

Eine andere Möglichkeit einen Elementinhalt zu deklarieren besteht darin, anstatt eines deklarierten Inhalts ein Inhaltsmodell zu definieren. Ein Inhaltsmodell läßt sich von den folgenden Produktionen ableiten :

[126] content model = ***model group [127]*** ***(ps+,*** ***exceptions [138]*** ***)?*** ***[127] model group =*** ***grpo,*** ***(ts [70]*,*** ***content token [128],*** ***(ts*,*** ***connector [131],*** ***ts*,*** ***content token)*,*** ***)**** ***grpc,*** ***occurrence indicator [132]?***	***[128] content token =*** ***primitive content token [129]*** ***|*** ***model group*** ***[129] primitive content token =*** ***(rni, „PCDATA“,)*** ***|*** ***element token [130]*** ***[130] element token =*** ***generic identifier [30],*** ***occurrence indicator [132]?*** ***[131] connector =*** ***and |*** ***or |*** ***seq*** ***[132] occurrence indicator =*** ***opt |*** ***rep |*** ***plus***

In Produktion [126] wird ein Inhaltsmodell als eine Sequenz einer Modellgruppe (= model group) und einer optionalen →Exception. Eine Modellgruppe beinhaltet ein oder mehrere Content-Tokens, welche durch Konnektoren miteinander verbunden sind. Die Konnektoren können folgender Art sein :

Konnektoren und Indikatoren

Sequenzkonnektor (,)

Der Sequenzkonnekor legt die Reihenfolge fest, in der die Subelemente innerhalb eines Inhaltsmodells auftreten müssen.

```
Kapitel = ( Anfang, Mitte, Schluß )
```

Bedeutet, daß in einem Kapitel zuerst der Anfang, danach der Mittelteil und am Ende der Schlußteil auftreten muß.

AND-Connector (&)

Hier werden Elemente verbunden, welche auftreten müssen, deren Reihenfolge unerheblich ist.

```
Anschrift = ( tel & fax )
```

OR-Connector (|)

Der OR-Connector verbindet Elemente, bei denen entweder das eine oder das andere Element auftreten sollen, aber nicht beide zusammen.

```
Urteil = ( schuldspruch | freispruch )
```

Gruppierungskonnektor ("(" und ")")

Konnektoren sind binäre bzw. sogar n-äre Operatoren. Sie beziehen sich also immer auf die Elemente zwischen denen sie platziert sind. Das es aber unter den Konnektoren keinen Vorrang gibt, können sich Konnektoren nur auf Elemente oder auf Elementgruppen beziehen. Eine solche Gruppe läßt sich mittels der Zeichen "(" und ")" bestimmen. Die Deklaration

```
Anschrift = ( tel & fax | email )
```

würde vom Parser als fehlerhaft markiert, da er nicht entscheiden kann welchen der beiden Konnektoren er zuerst auswerten soll. Bei der folgenden Modellierung

```
Anschrift = ( tel & ( fax | email ) )
```

Ist die logische Auswertung eindeutig festgelegt. Der Parser wertet hier zuerst den geklammerten Ausdruck aus und verbindet diesen dann mit dem AND-Konnektor mit dem übrigen Element.

Ein Element kann nun ebenfalls wieder eine Modellgruppe enthalten oder einen sogenannten „primitive content“. Die Produktion [129] legt diesen Inhalt durch das „primitive content token fest“. Ein primitive Content besteht aus einem Element-Token gefolgt von einem optionalen Häufigkeitsindikator (= occurrence indicator), d.h. hier gelangt man an die terminalen Produktionen des Ableitungsbaumes der Dokumentstruktur. Ein Element-Token ist eine eindeutige Bezeichnung eines Elements und der Häufigkeitsindikator legt die Anzahl der in der Instanz auftretenden Elemente fest.

Indikator	**Bedeutung**
+	Element kann beliebig oft, muß allerdings mindestens einmal auftreten
?	Element darf höchstens einmal auftreten, darf allerdings auch fehlen
*	Element darf beliebig oft auftreten und unterliegt keinen Einschränkungen

Indikatoren sind unäre Operatoren und beziehen sich immer auf die direkt vorangestellte Gruppe oder Element.

BEISPIEL

```
<!ELEMENT Drama - - (Einleitung?, Akt )+ >
...
```

Ein weiterer Inhaltstyp wird durch das Schlüsselwort *PCDATA* eingeleitet. Um diesen Inhalt von den vom Benutzer modellierten Inhalten zu trennen, wird ihm ein reserved name indicator (rni = "#") vorangestellt. Dieser reserved name indicator teilt dem Parser mit, daß es sich bei dem Inhalt *PCDATA* nicht um einen frei definierten Inhalt, sonder um einen von SGML festgelegten Inhalt handelt. Hinter *PCDATA* verbirgt sich folgendes Datenelement :

PCDATA (= parsable character data)

Inhalte dieses Elementtyps bestehen aus Zeichenketten, deren Inhalt komplett auf →Entitäten und auf weitere Markups, und somit auf weitere Subelemente untersucht wird. Der Parser würde also im folgenden Beispiel das Subelement "Fußnote" erkennen

```
<!ELEMENT  Text - - #PCDATA >
...
<text>...  <Fußnote>...</Fußnote> ... </text>
```

5.3.4 Exceptions

Als letzter Bestandteil des Inhaltsmodells soll nun auf die Exception eingegangen werden. Es ist manchmal notwendig, innerhalb eines Elements ein Subelement zu deklarieren, welches „beweglich“ in dem Oberelement deklariert ist, d.h. es kann an jeder Stelle innerhalb des Oberelements auftauchen. Um solche Elemente modellieren zu können bietet SGML die Möglichkeit Elemente über Exceptions in ein Inhaltsmodell einbinden zu können. Exceptions werden in der Regel [138] der abstract Syntax behandelt

[138] exceptions = ***(exclusions [140],*** ***ps+,*** ***inclusions [139]?)*** | ***inclusions [139]***	***[139] inclusions =*** ***plus,*** ***name group*** ***[140] exclusions =*** ***minus,*** ***name group***

Exceptions können in zwei Variationen auftreten, als

- Inklusion
- oder als Exklusion

Die Exklusion hat die Aufgabe die Elemente aus der exkludierten Elementgruppe aus dem Inhaltsmodell des Elements und seiner Subelemente zu streichen. Bei einer Exklusion wird der Elementgruppe ein Minuszeichen direkt vorangestellt wodurch sich folgender Aufbau ergibt

```
<!ELEMENT Kapitel     - -   ( Titel, Zeile+) -( Fußnote )>
```

Nun darf das Kapitel, sowie der Titel und die Zeile keine Fußnoten mehr enthalten. Dieser Mechanismus, daß sich eine Exception auch auf seine Subelemente überträgt nennt man ***Vererbung***.

Um ein Element exkludieren zu dürfen muß allerdings mindestens eine der folgenden Eigenschaften erfüllt sein :

1. Das Element ist optional → (?)
2. Das Element ist vorher durch eine Inklusion in das Inhaltsmodell eingebunden worden

Sollte ein nicht-optionales Element Element exkludiert werden, so verstößt es in gegen das Inhaltsmodell des Elements :

```
<!ELEMENT  Kapitel    - -   ( Titel, Zeile+ ) -( Zeile ) >
```

Die andere Variante der Exception ist die Inklusion. Mittels der Inklusion kann ein Element in das Inhaltsmodell eines Elementes eingebunden werden. Der Elementgruppe wird ein Plus ("+") vorangestellt, so daß sich folgender Aufbau ergibt :

```
<!ELEMENT Kapitel     - -   ( Titel, Zeile+) +( Fußnote ) >
```

Nun darf die Fußnote an jeder beliebigen Stelle innerhalb eines Kapitels auftreten. Durch den Mechanismus der Vererbung auch innerhalb eines Titels oder einer Zeile auftreten. Sogar innerhalb einer Fußnote dürfen wiederum Fußnoten auftreten, da eine Fußnote jetzt zum Inhaltsmodell des Kapitels zählt, und die Vererbung sich auf alle Elemente des Inhaltsmodells bezieht. Um Fußnoten nicht wieder in Fußnoten zuzulassen, kann die Fußnote wieder aus dem Inhaltsmodell der Fußnote exkludieren

```
<!ELEMENT Fußnote     - -   ( Text )    -( Fußnote ) >
```

Als letztes ist noch zu sagen, daß eine Exklusion immer eine Inklusion schlägt, d.h. ein mittels einer Inklusion inkludiertes Element kann mittels einer Exklusion wieder aus dem Inhaltsmodell entfernt werden :

```
<!ELEMENT Absatz - -   ( Zeile+ ) -( Fußnote) +( Fußnote ) >
```

In diesem Fall wird die Fußnote aus dem Inhaltsmodell des Absatz exkludiert, da inkludierte Elemente exkludiert werden dürfen.

5.3.5 Element / Data / Mixed – Content

Man unterscheidet je nach Inhaltsmodell der Elementdeklaration drei verschiedene Inhaltstypen :

- Element-Content
- Data-Content
- Mixed-Content

Bei einem Element-Content beinhaltet ein Inhaltsmodell weitere (Sub-)Elemente, welche in der Dokument-Deklaration vom Entwickler definiert worden sind.

```
<!ELEMENT Absatz - -   ( Titel, Zeile+ ) >
<!ELEMENT Titel  - -   ( RCDATA ) >
<!ELEMENT Zeile  - -   ( RCDATA ) >
```

Hier wäre das Inhaltsmodell des Absatzes ein Element-Content, da es nur aus Elementen besteht.

Ein anderer Inhaltstyp wäre der Data-Content. Hier besteht ein Element nur aus deklarierten Datenelementen wie *CDATA*, *RCDATA* oder sogar *EMPTY*. Ein Beispiel für einen solchen Inhaltstyp wären die im obigen Beispiel deklarierten Elemente Titel und Zeile.

```
<!ELEMENT Titel  - -   ( RCDATA ) >
<!ELEMENT Zeile  - -   ( RCDATA ) >
```

Der letzte Inhaltstyp wäre dann der Mixed-Content. Hier besteht ein Element aus Element- und Data-Content. Dies geschieht wenn die entsprechenden Teile im Inhaltsmodell deklariert werden, oder der Inhalt als *PCDATA* deklariert wird.

```
<!ELEMENT  para  - O   ( title, #PCDATA ) >
<!ELEMENT  title - O   ( #PCDATA ) >
```

In der Instanz würde das Element `para` dann wie folgt instanziert :

```
<para>   ◄──────────────── angenommener Dateninhalt
<title>Getting Started
The first step in the analysis is choosing the appropriate ...
```

Der SGML-Standard warnt allerdings vor der Verwendung von Mixed-Content, da hier oft ungewollt Fehler auftreten können. Er betont, *PCDATA*-Elemente nur zu definieren, wenn sie das einzige Element im Content sind, oder wenn der OR-Konnektor der einzige erlaubte Konnektor der Modellgruppe ist.

Diese Bedingung wurde gestellt, weil Separator Characters (TABS, SPACES,..), welche im Content auftreten, nun vom Parser als Datenelemente im Mixed-Content interpretiert werden. Im obigen Beispiel würde bereits das RETURN (= " ") hinter `<para>` ausreichen und der Parser würde den Zeilenwechsel bereits als Zeicheninhalt deuten. Als Folge dessen, werden drei im ersten Moment für den Benutzer völlig verwirrende Fehlermeldungen auftreten :

1. Das START-Tag des `title`-Elements wurde impliziert
2. Ein `title`-Tag trat außerhalb des definierten Kontexts auf
3. Das END-Tag des `title`-Elements wurde impliziert

Der Parser faßt also das RETURN als Dateninhalt auf und ordnet es dem *PCDATA*-Element in dem Inhaltsmodell des `para`-Elements zu. Der Parser impliziert nun ein `title`-Element, da dieses im `para`-Element vor dem Dateninhalt deklariert ist. Das anschließende `title`-Tag wird im *PCDATA* Element identifiziert, paßt aber nun laut Inhaltsmodell nicht in den Dateninhalt hinein. Daher rührt die zweite Fehlermeldung. Die letzte Fehlermeldung folgt von der Implikation des End-Tag des `title`-Elements.

Um das Mixed-Content Problem zu beheben gibt es drei Möglichkeiten :

1. Sicherstellen, daß zwischen dem `para`- und dem `title`-Tag keine Trennzeichen enthalten sind

2. Modifikation des Inhaltsmodells, so daß eine wiederholbare OR-Gruppe entsteht

```
      <!ELEMENT  para  - O   ( title, #PCDATA ) >
⇒     <!ELEMENT para   - O   ( title | #PCDATA )+ >
```

3. Ersetzung aller Mixed-Contents durch „Zwillinge“

```
      <!ELEMENT  para  - O   ( title, #PCDATA ) >
⇒     <!ELEMENT  para  - O   ( title, text ) >
      <!ELEMENT  title - O   ( #PCDATA ) >
      <!ELEMENT  text  O O   ( #PCDATA ) >
```

Die erste Möglichkeit ist mehr als fragwürdig, da hier die Dokumentinstanz einer sehr harten Einschränkung unterliegt, welche nur schwer durchzusetzen ist. Die zweite Alternative löst zwar das Mixed-Content Problem, läßt aber nun einen Inhalt zu, der aus Titeln und Daten in beliebiger Reihenfolge besteht. Diese Lösung ist also auch unzufriedenstellend. Die letzte Alternative definiert einen neuen Level von Elementen. Die „Terminalen“ des Ableitungsbaumes liegen somit auf einer Ebene und die „Nichtterminalen“ einer Ableitung liegen niemals zusammen mit einer „Terminalen“ in einer Ableitung. Dieses Inhaltsmodell löst das Mixed-Content Problem, ohne die Struktur des Inhalts zu verändern.

5.3.6 Minimierung

Bei der Auszeichnung der Elemente in der Instanz müssen im „Normalfall“ alle Elemente mit einer expliziten Start- und einer Endemarkierung bezeichnet werden. Ein Element sieht u.U. in der Instanz folgendermaßen aus

```
<Anschrift>
      <Straße>Otto-Hahn-Straße 16</Straße>
      <PLZ>44277</PLZ><Ort>Dortmund</Dortmund>
</Anschrift>
```

Dies kann sehr schnell ausarten, wenn sich einmal vorstellt wie nun eine etwas größere Tabelle in dieser Art und Weise der Darstellung aussehen würde. Die Darstellung einer Tabelle könnte somit mehr Zeichen für die Auszeichnung, als für die eigentlichen Daten enthalten.

Nun bietet SGML die Möglichkeit, daß Auszeichnungselemente unter gewissen Umständen vom Parser impliziert werden können, d.h. sie müssen in der Dokumentinstanz nicht mehr explizit angegeben werden. Damit eine solche Minimierung in der Instanz vom Parser akzeptiert wird, muß in der Elementdeklaration angegeben werden, daß eine Start- bzw. Endmarkierung weggelassen werden darf. Dies geschieht über die „ommitted tag minimization".

[122] ommitted tag minimization = ***start-tag minimization [123],*** ***ps+,*** ***end-tag minimization [124]***	***[123] start-tag minimization =*** ***„O"*** | ***minus*** ***[124] end-tag minimization =*** ***„O"*** | ***minus***

In der Dokumentinstanz hat eine Minimierung folgende Erscheinung

```
<!Element  Zeile        - -          ( RCDATA ) >
```

Endemarkierung

Startmarkierung

Die Variablen für die Start- und die Endemarkierung können zwei verschiedene Werte annehmen

Variablenbelegung	Bedeutung
–	Eine Minimierung der Markierung ist nicht zulässig und muß in der Instanz zwingend vorhanden sein
O	Eine Minimierung der Markierung ist zulässig. Die Auszeichnungsmarkierung eines Elements darf aber muß nicht zwingend vorhanden sein

Das obige Beispiel mit der Anschrift würde unter End-Tag Minimierung wie folgt aussehen:

```
<Anschrift>
     <Straße>Otto-Hahn-Straße 16
     <PLZ>44277<Ort>Dortmund
</Anschrift>
```

Bei der End-Tag Minimierung ist allerdings zu beachten, daß sichergestellt sein muß, daß der Parser die fehlende Markierung an der richtigen Stelle impliziert. Es gibt vier Möglichkeiten, an denen der Parser das Ende eines Elements annimmt :

1. Die Endemarkierung eines Elements wird gefunden. An dieser Stelle braucht der Parser nichts zu implizieren, da ihm eine explizite Markierung vorliegt
2. Das Dokument ist zu Ende. Da das Dokument zu Ende ist und nun keine Elemente mehr folgen können, kann der Parser davon ausgehen, daß auch die noch nicht explizit geschlossenen Elemente nun geschlossen werden sollen.
3. Es wird eine Endemarkierung eines Elements gefunden, zu dessen Inhalt das betreffende Element gehört. Der Parser kann die Endemarkierung implizieren, da eine Endemarkierung gefunden wurde, die zu einem Element in einer höheren Rekursionsstufe gehört.

   <Anschrift>

   <Straße>Otto-Hahn-Straße 16

   <PLZ>44227

   <Ort>Dortmund **</Ort>** ← impliziert

   </Anschrift>
4. Es wird eine Startmarkierung eines Elements gefunden, die mit dem Inhaltsmodell des aktuellen Elements nicht vereinbar ist.

   Beispiel:

   <Kapitel>Erstes Kapitel **</Kapitel>** ← impliziert

   <Kapitel>Zweites Kapitel

   ...

Auch die Startmarkierung kann minimiert werden, wenn die Strukturdefinition ihr Vorhandensein vorschreibt. Allerdings kann es hier ein Problem geben

```
<!ELEMENT  text  - -   ( a? ) >
<!ELEMENT  a     O O   ( b ) >
<!ELEMENT  b     - -   ( #PCDATA ) >
```

Normalerweise kann ein b-Element nur in einem a-Element auftreten. Also sollte der Parser ein Startmarkierung für a implizieren können. Der Parser meldet allerdings einen Fehler. Warum ?

Der Parser müßte „vorausschauen“ können um bisher noch nicht ausgezeichnete Sektionen implizierend auszeichnen zu können. Da ein Parser immer möglichst einfach gehalten wird werden sie nicht mit einer solchen LOOK-AHEAD Funktion ausgestattet und deshalb muß folgende Bedingung für eine Minimierung eines Startelements gelten :

„Eine Startmarkierung kann nur dann minimiert werden, wenn das betreffende Element nicht optional ist und alle anderen an dieser Stelle möglichen Elemente optional sind."

```
<!ELEMENT  Kapitel          - -   ( Titel, Absatz+ ) >
<!ELEMENT  Titel            O O   ( Text ) >
<!ELEMENT  Absatz           - O   ( Text ) >
```

Da zu Beginn eines Kapitel ein Titel kommen muß kann der Parser an dieser Stelle die Startmarkierung des Titel-Elements implizieren.

5.3.7 Mehrdeutigkeiten (= Ambiguity)

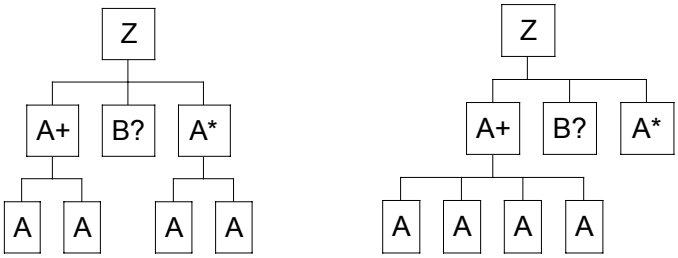
Als letztes soll noch die Mehrdeutigkeit besprochen werden. Eine Beschränkung für ein korrektes Inhaltsmodell ist, daß es sinnvoll und möglichst eindeutig auswertbar ist. Ein arithmetischer Ausdruck mit ganzen Zahlen kann schematisch eindeutig ausgewertet werden. Ähnliches muß für das Muster gelten. Der Auswertung entspricht hier der Vergleich der in der Dokumentinstanz gefundenen Elemente mit den Inhaltsmodellen. Dieser Vergleich wird im allgemeinen vom Parser durchgeführt. Dieser Parser hat im wesentlichen drei Funktionen :

1. Vergleich des Namens des Basiselements mit dem Namen der DTD
2. Der Parser liest die Dokumentinstanz. Die Dokumentinstanz als Ganzes muß mit dem Inhaltsmodell des Basiselements übereinstimmen.
3. Falls fehlerfrei, Erzeugung von Ausgabedaten

Ein Beispiel für eine Mehrdeutigkeit ist folgendes einfache Beispiel

```
<!ELEMENT Z      - -   ( A+, B?, A*) >
```

Auf den ersten Blick wirkt dieses Inhaltsmodell völlig korrekt. Wenn der Parser in der Dokumentinstanz aber die Abfolge <A>...</A><A>...</A><A>...</A><A>...</A> vorfindet, muß er entscheiden, welche A-Elemente zu der Gruppe vor und welche zu der Gruppe nach dem gehören.



Hier hat der Parser in der Tat ein schweres Problem, da dieses Problem über das Entscheidungspr o-blem der Mehrdeutigkeit in kontextfreien Grammatiken hinausgeht. Es gehört somit zur Klasse der NP-vollständigen Probleme (*vgl. Wegener, „Theoretische Informatik" S.163f*). Aus diesem Grund gelten mehrdeutige Inhaltsmodelle im SGML-Standard als fehlerhaft, der Parser braucht diese allerdings nicht zu erkennen, bzw. diese aufzulösen. Die Vermeidung solcher Fehler liegt hier also beim Modellierer.

5.4 Entitäten

Eine weitere wichtige Funktion in der Dokumentmodellierung stellt die Entität dar. Sie definiert einen Platzhalter, der durch Objekte ersetzt werden kann. Eine solches Ersetzungsobjekt kann vom einzelnen Zeichen bis hin zur kompletten Datei alles enthalten. Entitäten definieren somit einen Ersetungsmechanismus, wobei der Parser bei jedem vorkommen des Entitätsnamens den Namen der Entität durch ein definiertes Ersetzungsobjekt substituiert. Entitäten werden in der abstract Syntax von Regel [101] abgeleitet :

[101] entity declaration = ***mdo,*** ***„ENTITY“,*** ***ps+,*** ***entity name [102],*** ***ps+,*** ***entity text [105],*** ***ps+,*** ***mdc***	***[102] entity name =*** ***general entity name [103]*** ***|*** ***parameter entity name [104]*** ***[103] general entity name =*** ***name [55]*** ***|*** ***rni,*** ***„DEFAULT“*** ***[104] parameter entity name =*** ***pero, (= "%")*** ***ps+,*** ***name [55]***

Es werden neben den üblichen Bestandteilen wie Markup-Declaration-Opens usw. drei wesentliche Bestandteile bei der Deklaration einer Entität

1. Das Schlüsselwort ENTITY, welches dem Parser mitteilt, daß es sich bei der folgenden Deklaration um die Deklaration einer Entität und somit um einen Ersetzungsmechanismus handelt.
2. Der Entitätsname. Über diesen Namen wird der Ersetzungstext referenziert. Sobald der Parser auf eine Entität mit diesem Namen trifft, ersetzt er diesen „Platzhalter“ durch das Ersetzungsobjekt. An dieser Stelle wird bereits in Produktion [102] festgelegt, ob es sich um eine generische oder um eine parameter Entität handelt.
3. Der Ersetzungstext. Hier wird das Ersetzungsobjekt deklariert.

Der Entitätsname wird in der Regel [102] der abstract Syntax definiert. Hier lassen sich die zwei Varianten der Entitätsform unterscheiden. Bei diesen beiden Formen handelt es sich im

- eine general entity
- eine parameter entity

5.4.1 Generische Entitäten (general entity)

Die "general entity declaration" verbindet einen Entitätsnamen mit einem Ersetzungsobjekt. General Entities werden ausschließlich in der Dokumentinstanz verwendet, etwa um bestimmte Sonderzeichen in das Dokument einzufügen. Folglich müssen sie vorher in der Dokumentinstanz deklariert werden. Eine Deklaration einer generische Entität kann im Prolog folgenden Aufbau haben

```
<!ENTITY txt     "Dies ist ein Ersetzungstext" >
```

Der Name einer generischen Entität ist ein Name (im obigen Beispiel das Wort "txt"). Dieser Name darf 8 Zeichen lang sein und muß mit einem SGML-Character beginnen. Auf diese Weise wird der Name mit dem Ersetzungsobjekt verbunden. Eine weitere Möglichkeit ist die Angabe eines Schlüsselwortes. Diesem Schlüsselwort muß ein "reserved name indicator" (= "#") vorangestellt werden, damit der Parser diesen Namen als Schlüsselwort erkennt. Das einzig gültige Schlüsselwort in der Namensdeklaration einer Entität ist das Wort „DEFAULT“. Auf diese Weise kann man eine Default-Entity deklarieren. Diese Default-Entity wird immer dann eingesetzt, wenn in der Dokumentinstanz eine Entität referenziert wird, zu der es im Prolog keine Deklaration gibt.

```
<!ENTITY #DEFAULT "Entity declaration not found ! " >
```

Wenn nun in der Instanz eine Entität referenziert wird, welche in der DTD nicht existiert, wird sie automatisch mit der DEFAULT Entität verbunden.

Die Ersetzungsobjekte einer Entität können verschiedenster Art sein und werden in der Regel [105] der abstract Syntax beschrieben

[105] entity text =
paramter literal [66]
|
data text [106]
|
external entity specification [108]

Die drei wesentlichen Ersetzungsobjekte sind somit

1. parameter literals
2. data text
3. external entity specification

Parameter Literals

Solche Parameter Literale lassen sich von der Produktion [66] ableiten und enthalten folgende Elemente

[66] parameter literals = ***lit,*** ***replaceable parameter data [67]*** ***lit*** **|** ***lita,*** ***replaceable parameter data [67]*** ***lita***	***[67] replaceable parameter data =*** ***data character [48]*** **|** ***character reference [62]*** **|** ***parameter entity reference [60]*** ***[62] character reference =*** ***cro, (= "&#")*** ***character number,***

Ein parameter literal läßt sich also auf data character ableiten. Diese Form der Entität wird benötigt, wenn der Ersetzungsmechanismus einer Entität eine Zeichenkette zuordnet die nur aus Zeichen besteht, die in SGML zugelassen sind. Ein Beispiel hierfür wäre die übliche Grußformel

```
<!ENTITY   myadress   "10 Downing Street" >
```

Weiterhin läßt sich das parameter literal auf die Produktion [62] character reference ableiten. Hier lassen sich Sonderzeichen und Umlaute, welche nicht im SGML-Zeichensatz vorhanden sind zusätzlich in das Dokument einfügen. In der Deklaration des Ersetzungstextes wird ein Character Reference Open (cro) vorangestellt. Auf diese Weise weiß der Parser, daß er die dem Character Reference Open Delimiter folgende Ziffernsequenz als Zahl interpretieren soll, welche einem Buchstaben zugeordnet ist.

```
<!ENTITY   hyphen     "&#126" >
```

Dem Namen des Ersetzungsmechanismus wird das ASCII-Zeichen ~ zugeordnet. Auf diese Weise können auch non-SGML Zeichen in dem Dokument deklariert werden.

Data Text

Eine Weitere Form der Ersetzungsobjekte bietet der Data Text. Hier wird dem Parameter Literal eines Schlüsselworte *SDATA* oder *CDATA* vorangestellt.

[106] data text =
(***„CDATA“***
|
„SDATA“
)
ps+,
parameter literal

Die Deklarationen im Prolog haben dadurch etwa die folgende Form

```
<!ENTITY   hyphen     SDATA "&#45" >
<!ENTITY   term       CDATA "a<b ∧ b<c ⇒ a<c" >
```

Daten, deren Gültigkeit lokal ist und die bei einem Transfer auf ein anderes System konvertiert werden müssen, werden als systemabhängig bezeichnet. Wenn der Zeichensatz einer Entität vom lokalen Zeichensatz abhängt, ist es sinnvoll diese Daten der SGML-Applikation kenntlich zu machen. Dies geschieht über das Schlüsselwort *SDATA*.

Ein anderes Problem ergibt sich, wenn Entitäten Daten enthalten, die von der Anwendung falsch interpretiert werden könnten. Im obigen Beispiel könnte der Parser fälschlicherweise das "a<b" als eine Startmarkierung für ein b-Element interpretieren. Damit der Inhalt der Entität später nicht mehr interpretiert wird, bietet SGML die Möglichkeit, den Ersetzungstext als *CDATA* zu deklarieren. Jetzt wird der Text nicht mehr überprüft oder interpretiert (→ CDATA bei Declared Content).

External Entity Specification (externe Entitäten)

Kleine Dokumentbausteine, wie etwa eine Grußformel, kann man den Ersetzungstext direkt angeben. Umfangreichere Bausteine, wie zum Beispiel ein Kapitel eines Buches, wird man als externe Entitäten deklarieren, d.h. jedes Kapitel eines Buches kann jeweils in einer Datei abgelegt werden. Externe Entitäten werden von der Produktion [108] abgeleitet :

[108] external entity specification =	*[109] entity type =*
external identifier [73], *(ps+,* *entity type [109]* *)?*	*„SUBDOC“ |* *((„NDATA“ |* *„CDATA“ |* *„SDATA“) ,* *notation name [41])*

Genau wie bei einer generischen Entität können auch externe Entitäten jeweils *SDATA* und *CDATA* Datenelemente enthalten. Diese werden nur jetzt spezifisch in einer Datei abgelegt. Im Gegensatz zur generischen Entität wird bei der externen Entität der Typ des Ersetzungsobjektes erst **nach** dem externen Bezeichner angegeben.

Der Aufbau eines Buches mit Hilfe von externen Entitäten könnte nun wie folgt lauten

```
<!ENTITY   kap1  SYSTEM "/usr/author/mybook/kap1.sgm" CDATA  >
<!ENTITY   kap2  SYSTEM "/usr/author/mybook/kap2.sgm" CDATA  >
...
```

In der Instanz würde das Buch dann nur noch aus den Verweisen auf die externen Entitäten bestehen

```
<Book>
&kap1;
&kap2;
...
```

Die externe Entität liefert noch zwei weitere Datentypen, nämlich *NDATA* und *SUBDOC*. Bei *NDATA* darf der Ersetzungstext Zeichen enthalten, welche keine zulässigen SGML-Zeichen sind. Wenn z.B. eine 10 MB große Datei mit Binärdaten referenziert werden soll, wäre es wenig sinnvoll alle Zeichen durch Zeichenreferenzen zu ersetzen. Man deklariert die externe Entität also als *NDATA* und bindet somit die Daten ein. Dies geschieht aber über →Notationen und soll daher auf ein späteres Kapitel verschoben werden.

Als letztes soll noch das Subdokument angesprochen werden. Bei der Deklaration einer externen Entität als Subdokument wird der Entität noch das Schlüsselwort „SUBDOC“ hinten angestellt. Dadurch wird bewirkt, daß die externe Entität als Subdokument behandelt wird. Der unterschied eines Subdokuments zu „normalen“ externen Entität ist, daß Subdokumente eine eigene DTD besitzen dürfen, d.h. die obige Deklaration als Subdokumente

```
<!ENTITY   kap1  SYSTEM "/usr/author/mybook/kap1.sgm" SUBDOC >
<!ENTITY   kap2  SYSTEM "/usr/author/mybook/kap2.sgm" SUBDOC >
...
```

würde bedeuten, daß jedes Kapitel wieder eine eigene DTD und somit wieder einen eigene Struktur und Aufbau besitzt.

5.4.2 Parameter Entitäten

Eine Spezielle Form der Entitäten sind die Parameter Entitäten. Sie unterscheiden sich hauptsächlich dadurch von den generischen Entitäten, daß sie in der DTD eines Dokuments verwendet werden. Paramter Entitäten werden bereits in den Produktionen [102] und [67] definiert. Der Name einer Parameter Entität wird durch ein vorangestelltes "%" (parameter entity reference open = pero) eingeleitet. Dadurch wird dem Parser signalisiert, daß es sich um keine generische sondern um eine Parameter Entität handelt.

Der Inhalt einer Parameter Entität ist eine Zeichenkette, welche den Aufbau einer Name Group besitzt. Hier werden Elemente bestimmt, welche anstelle der Parameter Entität in das Inhaltsmodell eingefügt werden können.

```
<!ENTITY % floats      "( list | footnote | appendix )" >
```

Die Zeichenkette wird vom Parser durchlaufen und als Inhaltsmodell aufgefaßt. Auf diese Weise können Inhaltsmodelle in der DTD über die Parameter Entitäten sehr schnell modifiziert werden. Als Beispiel für den Gebrauch einer Parameter Entität können die Exceptions dienen. Typischerweise werden alle Elemente, welche inkludiert werden sollen in der Inklusionsliste direkt angegeben. Wenn nun die Inklusionsliste an mehreren Stellen innerhalb der DTD erscheint, muß der Entwickler jedes mal die Inklusionsliste explizit angeben. An dieser Stelle entfalten die Parameter Entitäten ihre volle Leistung. Das oben deklariert Element `floats` könnte nun problemlos als Parameter Entität in der Inklusionsliste referenziert werden

```
<!ELEMENT  chap  - - ( title, para+)  +( %floats; ) >
```

Die Referenz einer Parameter Entität ist in der abstract Syntax wie folgt festgelegt

[60] parameter entity reference = ***pero,*** ***document type specification [28],*** ***name [55],*** ***reference end [61]***	***[28] document type specification =*** ***name group [69] ?*** ***[61] reference end =*** ***refc (= ";")*** ***|*** ***RE***

Eine weitere Funktion der Parameter Entitäten ist die DTD-Modularisierung. Bei der Softwareentwicklung werden die Funktionen einer Anwendung meist in Gruppen zusammengefaßt. Diese Gruppen werden als Module bezeichnet. Ebenso können Elemente und Entitäten einer DTD in Gruppen gegliedert werden. Das erhöht die Übersichtlichkeit und erlaubt die Wiederverwendung solcher DTD-Module. Da Dokumentstrukturen nicht total unterschiedlich sind, können immer wieder in verschiedenen DTD's auftretende Elementdeklarationen in Module zusammengefaßt werden und ähnlich wie bei Funktionsbibliotheken bei der Softwareentwicklung immer wieder verwendet werden.

5.5 Notationen

Bisher konnte der Elementinhalt lediglich als CDATA, RCDATA oder PCDATA qualifiziert werden. Dies bedeutet, daß Daten in den Elementinhalten sich ausschließlich auf Textdaten bezogen haben. Um beispielsweise eine Formel wie

$$\cos(x) = \sqrt{1 - \sin^2(x)}$$

in einen Text einzubinden, bietet SGML keine Möglichkeit, da hier keine Formatierungen berücksichtigt werden. In einem Textsystem wie z.B. TeX könnte eine solche Formel durch den folgenden Befehl dargestellt werden :

```
\cos ( x ) = \sqrt{ 1-sin^2( x ) }
```

In einem SGML Dokument könnte die obige Formel in einer Element-Umgebung wie folgt ausgezeichnet werden :

```
<formel>\cos ( x ) = \sqrt{ 1 - sin^2( x ) }</formel>
```

Dieses Element könnte nun natürlich über →Attribute weiter qualifiziert werden, allerdings „weiß“ die Applikation nun noch immer nicht, wie diese Daten zu interpretieren sind. Um nun diese Daten an einen externen Prozeß zu binden und somit ihre Formatierung zu gewährleisten, werden Notationen deklariert, welche Daten eines SGML-Dokuments an einen externen Prozeß binden. Die obige Formel könnte über einen Notation mit einem TeX-Compiler verbunden werden, welcher die Daten auswertet.

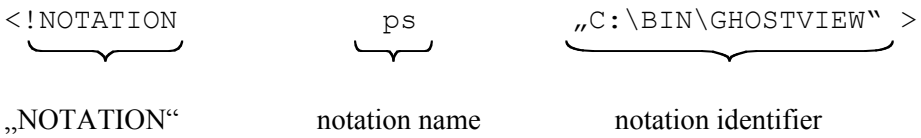
In der abstract Syntax werden Notationen von der Produktion [148] abgeleitet.

[148] notation declaration = ***mdo,*** ***„NOTATION“,*** ***ps+,*** ***notation name [41],*** ***ps+,*** ***notation identifier [149],*** ***ps*,*** ***mdc***	***[41] notation name =*** ***name [55]*** ***[149] notation identifier =*** ***external identifier [73]***

Eine Notation besteht also aus den folgenden wesentlichen Elementen :

1. Markup-Declaration-Open und Markup-Declaration-Close
2. Schlüsselwort „NOTATION“
3. Notationsname
4. Notation identifier

Die Markup-Declarations weisen auf das Ende bzw. auf den Anfang einer Deklaration hin. Das Schlüsselwort „NOTATION“ weist die Applikation darauf hin, daß nun eine Deklaration einer Notation folgt. Der Notationsname gibt den Namen an, mit dem die Notation später in der Dokumentinstanz referenziert werden kann. Der „notation identifier“ gibt letztendlich das Programm an, mit dem die Daten bearbeitet werden sollen



Wenn nun in der Dokumentinstanz eine externe Entität vom *NDATA* (NON-SGML-DATA) deklariert wird kann ihre Daten über eine Notation mit dem zugehörigen Prozeß verbunden werden.

```
<!NOTATION tex   SYSTEM     „C:\BIN\LATEX“ >
<!ENTITY   file  SYSTEM     „C:\DOCS\DOKU.TEX“ NDATA tex >
```

Die externe Entität `file` wird also über die Notation `tex` mit dem Programm LATEX verbunden.

5.6 Attribute

5.6.1 Aufgabe und Funktion

Als letzter Punkt des Prologs sollen noch Attribute besprochen werden. Ein Attribut dient in einer Sprache dazu, einem Gegenstand ein Merkmal, oder eine gewisse Eigenschaft zuzuordnen. Bisher wurden nur die syntaktische Struktur betrachtet und ein Dokument in seine logischen Bestandteile zerlegt. Die Frage, wie man ein bestimmtes Element zusätzlich qualifizieren kann blieb bisher unbeantwortet.

5.6.2 Syntax einer Attributdeklaration

Eine Attributdeklaration läßt sich von der Produktion [141] ableiten

[141] attribute definition list declaration = ***mdo,*** ***„ATTLIST“,*** ***ps+,*** ***(*** ***associated element type [72]*** ***|*** ***associated notation name [149.1]*** ***)*** ***ps+,*** ***attribute definition list [142]*** ***ps*,*** ***mdc***	***[142] attribute definition list =*** ***attribute definition [143],*** ***(*** ***ps+,*** ***attribute definition [143]*** ***)?*** ***[143] attribute definition =*** ***attribute name [144],*** ***ps+,*** ***declared value [145],*** ***ps+,*** ***default value [147]***

Die Deklaration einer Attributliste besteht aus einem Markup-Declaration-Open gefolgt von dem Schlüsselwort „ATTLIST“, wodurch die Deklaration einer Attributliste eingeleitet wird. Nun folgen der Name des assoziierten Elements oder die Bezeichnung einer Notation, auf welche sich das Attribut bezieht. Als letztes folgt dann noch die Deklaration der Attribute mittels der „attribute definition list“. Hier werden die Namen der Attribute, sowie ihre Wertebereiche und Voreinstellungen festgelegt.

Beispiel:

```
<!ATTLIST  ELEMENTName
     ATTRIBUT   ( Wertebereich )      DEFAULT >
```

5.6.3 Declared Values

Der eigentliche Baustein ist die Attributliste (`attribute definition list`). Hier werden die Wertebereiche der Attribute festgelegt. Wertebereiche von Attributen werden von der abstract Syntax festgelegt und können folgende Bereiche annehmen:

[145] declared values =	***[68] name token group =***
CDATA |	***grpo,***
ID |	***name token,***
IDREF |	***(***
NAME |	***ts*,***
NAMES |	***connector,***
NUMBER |	***ts*,***
NUMBERS |	***name token,***
NMTOKEN |	***ts****
NMTOKENS |	***)?,***
NUTOKEN |	***grpc***
NUTOKENS |	
name token group[68] |	
...	

Wertebereiche werden unterschieden in festgelegte Wertebereiche und benutzerdefinierte Wertebereiche. Die benutzerdefinierten Wertebereiche werden von der „name token group“ abgeleitet. Der Benutzer gibt hier explizit die Werte an, die das Attribut annehmen darf z.B.

```
<!ATTLIST  CHAPTER
      adjust     ( center | right | left | block )     left >
```

Hier wird einem Kapitel eine Textjustierung beigefügt, welche die Werte `left, right, center` und `block` annehmen kann. Als Voreinstellung wird hier der Wert left angenommen.

Die andere Form der Wertebereiche sind die festgelegten Wertebereiche. Hier werden den Wertebereichen ähnlich wie bei einer Variablen von SGML vordefinierte Wertebereiche zugeordnet. Die Attribute können nun einen Wert aus dieser Wertemenge annehmen. Ein Auszug aus diesen Wertebereichen bildet die folgende Tabelle

Wertebereich	*Beschreibung*	*Beispiel*
CDATA	Der Wertebereich CDATA umfaßt Zeichenketten von einer Länge bis maximal 8 Zeichen.	<chapter author="Jens" >
ID	Das Attribut wird als „Anker" eines Verweises aufgefaßt. Daher darf hier jeder Wert innerhalb der Dokumentinstanz nur einmal auftreten	<appendix ID=1002>
IDREF	Das Attribut wird als Referenz auf einen Anker eines Verweises interpretiert. Dieser Anker muß vorher mittels eines Attributs als ID in der Dokumentinstanz festgelegt worden sein	<xref reference=1002>
NAME	Der Wert dieses Attributs darf ein SGML-Name sein, also maximal 8 Zeichen und nur aus SGML-Zeichen	<figure name=graphic>
NAMES	Der Wert eines solchen Attributs wird durch eine Sequenz von Namen bestimmt, welche durch Leerzeichen untereinander getrennt sind	
NUMBER	Der Wertebereich beschränkt sich auf eine Ziffernfolge. Die Ziffern werden allerdings nicht als Zahlen interpretiert. „01" und „001" sind also verschiedene Werte	<chapter date=040199>
NUMBERS	Analog zu NAMES	
NUTOKEN	Die Zeichenkette beginnt mit einer Ziffer und darf anschließend eine beliebige Folge aus Ziffern und Zeichen enthalten.	<graphic dim="3D" >
NUTOKENS	Analog zu NAMES	
NMTOKEN	Die Zeichenkette darf nun Buchastaben und Ziffern in beliebiger Reihenfolge enthalten. Das erste Zeichen darf hier ebenfalls beliebig gewählt werden	<command arg="-r" >
NMTOKENS	Analog zu NAMES	

Eine Attributdefinition mit vordefinierten Wertebereichen könnte also wie folgt lauten :

```
<!ATTLIST  Kapitel
      author          CDATA       „Robert L. Stevenson"
      index           NUMBER      3
      verweis         ID          „X125"
>
```

5.6.4 Voreinstellungen

Bei Attributen hat man die Möglichkeit, neben den Wertebereichen auch eine Standardeinstellung vorzunehmen. Dieser Standardwert wird z. B. bei einer nicht expliziten Angabe eines Attributwertes von der Applikation für diese Attribut eingesetzt.

```
<!NOTATION table SYSTEM „c:\windows\excel.exe“ >
<!ATTLIST  table
      rows  NUMBER      3
      cols  NUMBER      5 >
```

Hier wird bei fehlender Angabe der Attribute `rows` oder `cols` die Standardwerte 3 für `rows` und 5 für `cols` eingesetzt. Wichtig ist, daß der Standardwert im Wertebereich des Attributs vorhanden ist. Die Deklaration :

```
<!ATTLIST  table
      rows  NUMBER      „Joho, und ´ne Buddel voll Rum!“ >
```

wäre zwar laut abstract Syntax korrekt, der Parser würde allerdings einen Fehler melden, da sich eine Zeichenkette nicht im Wertebereich von **NUMBER** befindet.

Voreinstellungen werden in der abstract Syntax von der Produktion [147] abgeleitet.

[147] default value =
(
(
rni,
„FIXED“,
ps+
)?,
attribute value specification [33]
)
|
(
(
rni,
(
„REQUIRED“ **|**
„IMPLIED“ **|**
„CURRENT“ **|**
„CONREF“
)
)

Neben der Auswahl eines speziellen Elements aus dem Wertebereich können noch weitere Informationen zur Attributierung übergeben werden. Diese Schlüsselwörter charakterisieren das Auftreten eines Attributs und seiner Interpretation. Es lassen sich die 5 verschiedenen Typen unterscheiden :

1. **FIXED**
2. **REQUIRED**
3. **IMPLIED**
4. **CURRENT**
5. **CONREF** (hierauf wird in dieser Ausarbeitung nicht weiter erläutert)

FIXED

Eine Möglichkeit ein Element, aus einer vom Benutzer aufgezählten Menge von Attributwerten als Standard zu bezeichnen geschieht durch die Angabe dieses Elements direkt nach der Definition der möglichen Werte.

```
<!ATTLIST  Kapitel
      format      ( left | center | right | block )     left >
```

Hier wird left als Standardelement definiert und wird bei fehlender Attributangabe in der Instanz, als Standardwert eingesetzt. Wenn nun dem Standardelement das Schlüsselwort ***#FIXED*** vorangestellt wird, so kann der Attributwert nicht mehr geändert werden. Eine Deklaration eines Attributwertes mit dem Schlüsselwort ***#FIXED*** entspricht einer Konstantendeklaration in einer Programmiersprache. Bei der Deklaration

```
<!ATTLIST  table
      rows  NUMBER      #FIXED      3
      cols  NUMBER      #FIXED      3 >
```

könnte der Attributwert für `rows` und `cols` in der Dokumentinstanz nicht mehr verändert werden. Aus diesem Grund darf die Angabe der Attribute in der Instanz auch fehlen.

```
<table>    .... </table>
```

REQUIRED

Bei der Voreinstellung ***#REQUIRED*** darf in der Deklaration kein Wert aus dem Wertebereich angegeben werden, da hier die Angabe eines Attributs in der Dokumentinstanz unbedingt erforderlich ist. Die Deklaration

```
<!ATTLIST  Kapitel
      author      CDATA       #REQUIRED >
```

bedeutet, daß in der Dokumentinstanz jedes Start-Tag mit der Angabe eines Attributs versehen sein muß

```
<kapitel author="Robert L. Stevenson">
```

IMPLIED

Bei der Voreinstellung ***#IMPLIED*** ist die Angabe eines Attributwertes in der Dokumentinstanz optional. Der Attributwert darf fehlen und kann von der Applikation impliziert werden.

BEISPIEL:

```
<!ATTLIST  table
      rows  NUMBER      #IMPLIED
      cols  NUMBER      #IMPLIED >
```

In der Dokumentinstanz darf die Angabe eines Wertes fehlen oder vorhanden sein.

```
<table rows=12    cols=5 >  ...   </rows>
```

oder

```
<table>    ...   </table>
```

Wobei im zweiten Fall die Applikation „entscheidet“, welchen Wert die Attribute annehmen.

CURRENT

Bei der Deklaration eines Attributvoreinstellung als ***#CURRENT*** muß beim ersten Auftreten des Elements in der Dokumentinstanz das Attribut angegeben werden. Dieser Wert gilt nun für alle weiteren Elemente bis der ein neuer Wert angegeben und der alte Wert damit überschrieben wird. Dies erinnert ein wenig an einen Variablenbelegung in einer Programmiersprache. Der Wert muß am Anfang initiiert werden und bleibt bis zur nächsten Modifikation unverändert

BEISPIEL:

```
<!ATTLIST  Kapitel
      author      CDATA       #CURRENT >
```

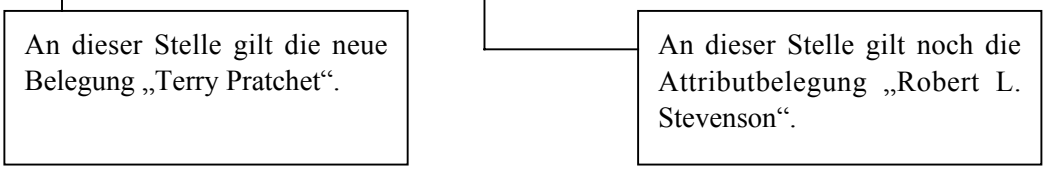
...

<Kapitel author="Robert L. Stevenson"> ... </Kapitel>

<Kapitel> ... </Kapitel>

<Kapitel author="Terry Pratchet"> ... </Kapitel>

<Kapitel> ... </Kapitel>



6 Document Instance

6.1 Aufbau der Document Instance

Die Dokumentinstanz beinhaltet den eigentlichen Inhalt des Dokuments. Dieser wird mit den im Prolog definierten Elementen durchsetzt. Der Standard kann allerdings keine syntaktische Korrektheit bzgl. der in der DTD definierten syntaktischen Struktur gewährleisten. Der Standard schreibt eine Syntax vor, wie die in der DTD deklarierten Elemente in den Dokumentinhalt einzufügen sind. Die syntaktische Korrektheit bzgl. der in der DTD definierten Syntax wird dann von einem SGML-Parser überprüft.

Das bedeutet, daß es möglich ist, eine Dokumentinstanz zu verfassen, die im Sinne des SGML-Standards syntaktisch korrekt ist, aber die bezüglich der in der DTD definierten Syntax falsch ist.

Der syntaktische Aufbau der Dokumentinstanz ergibt sich aus Regel[10] der abstract syntax.

[10] document instance set =
base document element [11] ,
other prolog [8] *

[11] base document element =
document element [12]

[12] document element =
element [13]

[13] element =
start-tag [14] ?,
content [24] ,
end- tag [19] ?

Die syntactic variable "other prolog" gestattet es, die Dokumentinstanz mit einem Kommentar(comment declaration) oder einer Verarbeitungsanweisung zu beenden (processing instruction).

Im wesentlichen besteht die Dokumentinstanz somit aus einem Element entsprechend Regel[13] der abstract syntax. Dieses Element wird auch das BASISELEMENT genannt. Das Basiselement ist das Element, dessen Name der Name des Dokumenttyps ist.

Während über die syntactic variables start-tag und end-tag die Syntax für die Angabe eines Tags bestimmt wird, beschreibt die syntactic variable "content", wie der Inhalt dieses Elements gebildet werden kann.

Dadurch, daß sowohl das start-tag als auch das end-tag optional sind, wird dem Omit-Tag-Feature Rechnung getragen. Dieses Feature besagt, daß ein erforderliches Tag weggelassen werden kann, wenn dessen Notwendigkeit impliziert werden kann.

6.2 Start-Tags

Ein Start-Tag wird angegeben, um den Anfang eines Strukturbestandteils zu signalisieren. Dieser Strukturbestandteil ist innerhalb des Prologs über eine ELEMENT-Deklaration definiert worden. Wird innerhalb eines Start-Tags ein Generic Identifier bezeichnet, der nicht innerhalb des Prologs deklariert wurde, oder wird dieses Element an einer Stelle innerhalb der Document Instance angegeben, an der es laut der in der DTD definierten Syntax nicht angegeben werden darf, so wird der SGML-Parser einen Fehler melden.

Die Syntax für die Angabe eines Start-Tags ergibt sich aus Regel[14].

[14] start-tag =
(stago ,
document type specification [28] ,
generic identifier specification [29] ,
attribute specification list [31] ,
s [5] *,
tagc) |
minimized start-tag [15]

Beispiel :

<(Book)Book Author="Lars Brauckmann" Date="10.12.1998">

<	Zeichenabbildung der „reference concrete syntax“ für stago (Start Tag Open)
(Book)	Document Type Specification
Book	Generic Identifier Specification
Author=“Lars Brauckmann“ Date=“10.12.1998“	Attribute Specification List
>	Zeichenabbildung der „reference concrete syntax“ für tagc (Tag Close)

6.2.1 Document Type Specification

Die Document type specification ist optional und ist nur notwendig, wenn konkurrente Dokumentstrukturen spezifiziert wurden. In diesem Fall muß die DTD, in welcher der folgende Generic identifier deklariert wurde, angegeben werden.

[28] document type specification =
name group [69] ?

[69] name group =
grpo ,
ts [70] *,
name [55] ,
(ts [70] *,
connector [131] ,
ts [70] *,
name [55])*,
ts [70] *,
grpc

Der Name der spezifizierten DTD wird dabei in Form einer "name group" angegeben, wobei die Namen der angegebenen DTDs durch einen beliebigen Konnektor getrennt werden (der Konnektor dient hier also als Trennzeichen und hat keine besondere Bedeutung).

Inwieweit es sinnvoll ist, einem Generic Identifier eine Menge von DTD-Bezeichnern voran zu stellen, kann an dieser Stelle nicht mit Sicherheit beantwortet werden. Denkbar ist, daß es somit möglich

wäre, einen Strukturbestandteil, der in zwei DTDs deklariert wurde, mit nur einem Start-Tag sowohl die Rolle des einen, als auch des anderen Strukturbestandteiles zuzuweisen.

Bem.: Die syntactic variable ts steht für "token seperator" und umfaßt die "whitespace character", die in Regel[5] definiert sind, sowie das Ee-Signal.

[5] s =
SPACE **|**
RE **|**
RS **|**
SEPCHAR

6.2.2 Generic Identifier Specification

Die Generic Identifier specification identifiziert ein Element innerhalb des Prologs, und stellt somit den Bezeichner für das Strukturelement dar.

[29] generic identifier specification =
generic identifier [30] **|**
rank stem [120]

[30] generic identifier =
name [55]

[120] rank stem =
name [55]

Hier gilt es, einen gültigen Namen entsprechend Regel[55] anzugeben. Der Parser überprüft anschließend, ob dieser Name innerhalb des Prologs deklariert wurde, und ob dieses Tag an dieser Stelle eingefügt werden durfte (entsprechend der in der DTD spezifizierten Syntax). Für Elemente gültige Namen werden somit durch Regel [55] spezifiziert.

[55] name =
name start character [53] ,
name character [52] *

[52] name character =
name start character [53] **|**
Digit **|**
LCNMCHAR **|**
UCNMCHAR **|**
Bem.: Digit ist ein terminal constant und umfaßt die Ziffern 0 bis 9
LCNMCHAR und UCNMCHAR sind terminal variables und umfassen die Groß- bzw. Kleinbuchstaben, die zur Bildung von Namen benutzt werden können.

[53] name start character =
LC Letter **|**
UC Letter **|**
LCNMSTRT **|**
UCNMSTRT
Bem.: LC Letter und UC Letter sind terminal constants und umfassen sämtliche Klein- bzw. Großbuchstaben

LCNMSTRT und UCNMSTRT sind terminal variables und umfassen die Groß- bzw. Kleinbuchstaben, mit denen ein Name beginnen darf (zusätzlich zu LC Letter,UC Letter)

Beispiele für syntaktisch korrekte Elementbezeichner :

Book

Chapter1

List-Start (die Reference Concrete Syntax definiert die Klassen LCNMCHAR und UCNMCHAR mit den Zeichen „–„ und „.“. Der Bindestrich ist daher zur Bildung von Namen zulässig)

Beispiel für syntaktisch falsche Elementbezeichner :

_Book

1.Chapter

6.2.3 Attribute Specification List

Attribute werden innerhalb von SGML zur Qualifizierung von Elementen verwendet. Somit besteht die Möglichkeit, zu einem Element zusätzliche Information zu verwalten. Während Elemente ein Dokument beschreiben, dienen Attribute dazu, ein Element zu beschreiben.

Die Attribute specification list dient dazu, Attribute des Elements mit Werten zu belegen.

[31] attribute specification list =
attribute specification [32] *

[32] attribute specification =
s [5] *,
(name [55] ,
s [5] *,
vi , =
s [5] *)?,
attribute value specification [33]

Um einem Attribut einen Wert zuzuweisen, muß zunächst der Name des Attributs angegeben werden (wiederum entsprechend Regel [55]). Der Parser muß dann sicherstellen, daß dieses Attribut innerhalb der DTD für das entsprechende Element deklariert wurde. Der Name eines Attributs kann weggelassen werden, wenn durch die Angabe des Attributwertes deutlich wird, welchem Attribut dieser Wert zugewiesen werden soll.

Beispiel:

```
<!ATTLIST   Book   type     (novel | report)      #REQUIRED
                   status   (draft | final)       #REQUIRED>
...
<Book  novel draft>
```

Das Weglassen von Attributbezeichnern ist aber nur dann erlaubt, wenn bei der Attributdeklaration die zulässigen Attributwerte aufgelistet werden. Da SGML es nicht gestattet, daß ein Attributwert für zwei Attribute innerhalb eines Elements verwendet wird, ist eine eindeutige Zuordnung des Attributwerts zu einem Attribut möglich. Werden die Attributwerte nicht in einer Liste vorgegeben, so kann der Attributname nicht weggelassen werden.

Beispiel:

```
<!ATTLIST   Book   type     CDATA      #REQUIRED
                   status   CDATA      #REQUIRED>
...
<Book  type=novel status=draft>
```

6.2.4 Attribute Value Specification

Ein Attributwert wird entsprechend Regel[33] angegeben. Dabei wird vom Parser überprüft, daß der angegebene Attributwert tatsächlich dem in der DTD vorgegebenen Wertebereich entspricht.

[33] attribute value specification =
attribute value [35] |
attribute value literal [34]

[34] attribute value literal =
(lit
replaceable character data [46] *,
lit) |
(lita ,
replaceable character data [46] *,
lita)

[35] attribute value =
character data [47] |
general entity name [103] |
general entity name list [35.1] |
id value [36] |
id reference value [38] |
id reference list [37] |
name [55] |
name list [39] |
name token [57] |
name token list [40] |
notation name [41] |
number [56] |
number list [42] |
number token [58] |
number token list [43]

Attribute Value Literal	Nur wenn Wertebereich CDATA entspricht. ***[46] replaceable character data =*** ***(data character [48]*** | ***character reference [62]*** | ***general entity reference [59]*** | ***Ee)**** Erlaubt es, innerhalb des angegebenen Werts Referenzen zu benutzen. Die Angabe erfolgt in Hochkomma . Bsp.: <book author =“Klaus M&ue;ller“>
Character Data	Nur wenn Wertebereich CDATA entspricht ***[47] character data =*** ***data character [48] **** ***[48] data character =*** ***SGML character [50]*** Erlaubt es, eine beliebige Zeichenkette als Attributwert zuzuweisen, bestehend aus gültigen SGML Characters. Die Verwendung von Referenzen ist in diesem Fall nicht möglich. Bsp.:<book type = novel>
General Entity Name	Nur wenn Wertebereich ENTITY entspricht ***[103] general entity name =***

	name [55] | ***(rni ,*** ***"DEFAULT")*** Erlaubt es, eine Entität als Attributwert zuzuweisen.
General Entity Name List	nur wenn Wertebereich ENTITIES entspricht ***[35.1] general entity name list =*** ***name list [39]*** ***[39] name list =*** ***name [55] ,*** ***(SPACE ,*** ***name [55])**** Erlaubt es, eine Menge von Entitäten als Attributwert zuzuweisen
Id Value	nur wenn Wertebereich ID entspricht ***[36] id value =*** ***name [55]*** Erlaubt es, einem Attribut einen eindeutigen ID-Bezeichner zuzuweisen. Der Bezeichner kann dann verwendet werden, um diesen Dokumentteil zu referenzieren. Bsp.: <figure id = Abb1>
Id Reference Value	nur wenn Wertebereich IDREF entspricht ***[38] id reference value =*** ***name [55]*** Erlaubt es, einem Attribut einen existierenden ID-Bezeichner zuzuweisen, und somit Teile innerhalb eines Dokuments zu referenzieren. Bsp.: Wie in Abbildung <figureRef idref = Abb1> ersichtlich ... *Bem.: Die Konstruktion von Verweisen mittels ID und IDREF ist nur innerhalb eines Dokuments möglich. Eine über IDREF referenzierte Id muß innerhalb des Dokuments existieren.*
Id Reference List	nur wenn Wertebereich IDREFS entspricht ***[37] id reference list =*** ***name list [39]*** Erlaubt es, eine Menge von ID-Bezeichnern zu referenzieren.
Name	nur wenn Wertebereich NAME entspricht ***[55] name =*** ***name start character [53] ,*** ***name character [52] **** Erlaubt es, nur gültige Namen als Attributwert anzugeben.
Name List	nur wenn Wertebereich NAMES entspricht ***[39] name list =*** ***name [55] ,*** ***(SPACE ,*** ***name [55])****

	Erlaubt es, eine Liste von gültigen Namen als Attributwert anzugeben
Name Token	nur wenn Wertebereich NMTOKEN entspricht ***[57] name token =*** ***name character [52] +*** ***[52] name character =*** ***name start character [53] |*** ***Digit |*** ***LCNMCHAR |*** ***UCNMCHAR |*** Erlaubt es, Zeichenketten, die aus beliebigen in Namen gültigen Zeichen zusammengesetzt sind, als Attributwert zuzuweisen.
Name Token List	nur wenn Wertebereich NMTOKENS entspricht [***40] name token list =*** ***name token [57] ,*** ***(SPACE ,*** ***name token [57])**** Erlaubt es, eine Liste von Name Token als Attributwerte zuzuweisen.
Notation Name	nur wenn Wertebereich NOTATION entspricht ***[41] notation name =*** ***name [55]***
Number	nur wenn Wertebereich NUMBER entspricht ***[56] number =*** ***Digit + 0-9*** Erlaubt es, eine Ziffer bzw. eine Ziffernfolge als Attributwert anzugeben. *Bem.: Ziffernfolgen werden nicht normiert. Die folgenden Ziffernfolgen stellen alle die gleiche Zahl dar, werden jedoch als unterschiedliche Attributwerte interpretiert : „1“ „01“ „001“*
Number List	nur wenn Wertebereich NUMBERS entspricht ***[42] number list =*** ***number [56] ,*** ***(SPACE ,*** ***number [56])**** Erlaubt es, eine Liste von Ziffernfolgen als gültige Attributwerte anzugeben.
Number Token	nur wenn Wertebereich NUTOKEN entspricht ***[58] number token =*** ***Digit ,*** ***name character [52] **** Erlaubt es, eine Zeichenfolge, die mit einer Ziffer beginnt als gültigen Attributwert zuzuweisen. Bsp.: 1.5cm
Number Token List	nur wenn Wertebereich NUTOKENS entspricht

	[43] number token list = ***number token [58] ,*** ***(SPACE ,*** ***number token [58])**** Erlaubt es, eine Menge von Number Tokens als gültige Attributwerte zuzuweisen.

Zu beachten ist, daß Attributwerte im allgemeinen normiert werden (mit Ausnahme der Attribute vom Typ NUMBER bzw. NUMBERS). Das bedeutet, daß Zwischenraumzeichen am Anfang bzw. am Ende eliminiert werden, und daß eine Folge von Zwischenraumzeichen innerhalb des Attributwerts zu einem Zwischenraumzeichen zusammengezogen wird.

6.3 End-Tags

Ein End-Tag wird angegeben, um das Ende eines Strukturbestandteils zu signalisieren. Dieser Strukturbestandteil ist innerhalb des Prologs über eine ELEMENT-Deklaration definiert worden. Wird innerhalb eines End-Tags ein Generic Identifier bezeichnet, der nicht innerhalb des Prologs deklariert wurde, oder wurde das End-Tag an einer Stelle angegeben, wo es laut DTD-Syntax nicht angegeben werden darf, so wird der SGML-Parser einen Fehler melden. Ein End-Tag ist entsprechend Regel[19] anzugeben.

[19] end-tag =
(etago ,
document type specification [28] ,
generic identifier specification [29] ,
s [5] *,
tagc) |
minimized end-tag [20]

Beispiel:

</(Book)Book>

</	Zeichenabbildung der „reference concrete syntax“ für etago (Start Tag Open)
(Book)	Document Type Specification
Book	Generic Identifier Specification
>	Zeichenabbildung der „reference concrete syntax“ für tagc (Tag Close)

Dieses Beispiel signalisiert, daß das Ende des Strukturelements Book, welches innerhalb der DTD Book deklariert wurde, erreicht ist.

Auch hier ist die Angabe einer Document type specification nur notwendig, wenn konkurrente Dokumentstrukturen spezifiziert wurden.

6.4 Short-Tag-Minimization

Während die Omit-Tag-Minimization es erlaubt, Tags komplett auszulassen, wenn deren Notwendigkeit implizit vorhanden ist, gestattet es die Short-Tag-Minimization, daß Tags verkürzt dargestellt werden können.

Über Regel[15] der "abstract syntax" wird spezifiziert, welche Minimierungsmöglichkeiten für Start-Tags bestehen.

[15] minimized start-tag =
empty start-tag [16] **|**
unclosed start-tag [17] **|**
net-enabling start-tag [18]

Aus Regel[20] wird ersichtlich, welche Minimierungsmöglichkeiten für End-Tags erlaubt sind.

[20] minimized end-tag =
empty end-tag [21] **|**
unclosed end-tag [22] **|**
null end-tag [23]

6.4.1 Empty Start-Tag

Empty Start-Tags werden durch Regel [16] spezifiziert.

[16] empty start-tag =
stago ,
tagc

Beispiel:

<List>

<LItem> Item1

<> Item2

<> Item3

</List>

Die leere Startmarkierung stellt eine Wiederholung des zuletzt geöffneten Elements dar. In diesem Fall also LItem. Zu beachten ist, daß die zuletzt in der Instanz angegebene Startmarkierung nicht notwendigerweise die Startmarkierung des zuletzt geöffneten Elements ist. Es ist möglich, daß die Startmarkierung des letzten geöffneten Elements impliziert worden ist.

Beispiel:

<List> Item1

<> Item2

<> Item3

</List>

Bei folgender Deklaration hätte dieses Beispiel den gleichen Effekt wie oben, da das Start-Tag für LItem impliziert wird:

<!ELEMENT List - - (LItem+) >

<!ELEMENT LItem O O (#PCDATA) >

6.4.2 Empty End-Tag

Empty-End-Tags werden durch Regel [21] spezifiziert.

[21] empty end-tag =
etago ,
tagc

Leere Endemarkierungen terminieren das zuletzt geöffnete Element.

Beispiel:

<List>

<LItem> Item1

</> <!-- Schließt LItem -->

<LItem> Item2

</> <!-- Schlisset LItem -->

</> <!-- Schlisset List -->

6.4.3 Unclosed Start-Tag / Unclosed End-Tag

Unclosed Start-Tags / End-Tags ermöglichen eine gewisse Lässigkeit im Umgang mit Tags, natürlich mit dem Ziel unnötige Schreibarbeit zu vermeiden.

Unclosed Start-Tags werden durch Regel [17] spezifiziert.

[17] unclosed start-tag =
stago ,
document type specification [28] ,
generic identifier specification [29] ,
attribute specification list [31] ,
s [5] *

Unclosed End-Tags werden durch Regel [22] spezifiziert.

[22] unclosed end-tag =
etago ,
document type specification [28] ,
generic identifier specification [29] ,
s [5] *

Die Angabe eines unclosed Tags entspricht syntaktisch also der Angabe eines kompletten Tags, mit der Ausnahme, daß der tagc-Delimiter weggelassen werden kann.

Unclosed Start-Tags können immer dann verwendet werden, wenn auf ein Start-Tag direkt wieder ein Start-Tag folgt.

Unclosed End-Tags können immer dann verwendet werden ,wenn auf ein End-Tag direkt wieder ein End-Tag folgt.

Beispiel :

<List<LItem> Item1 </LItem>

<LItem> Item2 </Litem</List>

6.4.4 NET-Enabling Start-Tag / Null End-Tag

Net-Enabling Start-Tags erlauben ein Maximum an Minimierung. Das Net-Enabling Start-Tag wird durch Regel[18] vorgegeben.

[18] net-enabling start-tag =
stago ,
generic identifier specification [29] ,
attribute specification list [31] ,
s [5] *,
net

Das Net-Enabling Start-Tag entspricht einem unclosed Start-Tag, mit der Ausnahme, daß das net-Zeichen angehängt wird (in der reference concrete syntax "/"). Durch dieses Zeichen wird signalisiert, daß ein Null-End-Tag (NET) erlaubt ist.

Ein Null-End-Tag wird durch Regel [23] bestimmt.

[23] null end-tag =
net

Beispiel :

<List>

<LItem/Item1/

<LItem/Item2/

</List>

Diese Art der Minimierung erlaubt es also, Endemarkierungen nahezu vollständig abzukürzen. Sinnvoll ist der Einsatz insbesondere bei inkludierten Elementen, da deren Tags meist nicht implizierbar sind.

6.5 Content

Content stellt, ausgehend von Regel [10], zunächst einmal den Inhalt des Basiselements dar. Es wird sich aber herausstellen, daß ein Content wiederum Elemente und somit auch weitere Contents enthalten kann. An dieser Stelle besteht nur die Möglichkeit, einen bzgl. der abstract syntax syntaktisch korrekten Content zu beschreiben. Ob ein angegebener Content bzgl. der in der DTD angegebenen Syntax für die Dokumentklasse korrekt ist hängt von dem Content Model des gerade aktiven Elements ab. Für jedes Element wird innerhalb der DTD ein Content Model definiert, so daß der Parser die syntaktische Korrektheit des Dokument überprüfen kann.

Der Aufbau des "content" ergibt sich aus Regel[24].

[24] content =
mixed content [25] **|**
element content [26] **|**
replaceable character data [46] **|**
character data [47]

Die syntactic variables "replaceable character data" und "character data" erlauben es, Daten in den Dokumentinhalt einzufügen.

[46] replaceable character data =
(data character [48] |
character reference [62] |
general entity reference [59] |
Ee)*

[47] character data =
data character [48] *

[48] data character =
SGML character [50]

[50] SGML character =
markup character [51] |
DATACHAR

[51] markup character =
name character [52] |
function character [54] |
DELMCHAR
Bem.: DELMCHAR umfassen sämtliche Zeichen, die innerhalb der SGML Declaration zur Abbildung der delimiter roles bestimmt wurden

[52] name character =
name start character [53] |
Digit |
LCNMCHAR |
UCNMCHAR |
Bem.: Digit ist ein terminal constant und umfaßt die Ziffern 0 bis 9

LCNMCHAR und UCNMCHAR sind terminal variables und umfassen die Groß- bzw. Kleinbuchstaben, die zur Bildung von Namen benutzt werden können.

[53] name start character =
LC Letter |
UC Letter |
LCNMSTRT |
UCNMSTRT
Bem.: LC Letter und UC Letter sind terminal constants und umfassen sämtliche Klein- bzw. Großbuchstaben LCNMSTRT und UCNMSTRT sind terminal variables und umfassen die Groß- bzw. Kleinbuchstaben, mit denen ein Name beginnen darf (zusätzlich zu LC Letter,UC Letter
und Special)

[54] function character =
RE |
RS |
SPACE |
SEPCHAR |
MSOCHAR |
MSICHAR |
MSSCHAR |
FUNCHAR
Bem.: RE bezeichnet das Zeilenendezeichen (terminal variable)
RS bezeichnet das Zeilenstartzeichen (terminal variable)
SPACE bezeichnet das Zwischenraumzeichen (terminal variable)
SEPCHAR bezeichnet die Klasse der Trennzeichen(terminal variable)
MSOCHAR, MSICHAR, MSSCHAR, FUNCHAR siehe SGML Declaration

6.5.1 Mixed Content

Der mixed content erlaubt es, Daten und weiter Elemente innerhalb eines Elementes zu kombinieren. Diese Elemente sind dann wiederum Elemente entsprechend Regel[13] der abstract syntax und können somit auch wieder einen content enthalten. Dadurch, daß ein Content wiederum einen Content enthalten kann, besteht die Möglichkeit, eine hierarchische Dokumentstruktur zu realisieren.

[25] mixed content =
(data character [48] |
element [13] |
other content [27])*

Beispiel für einen Mixed Content:

```
<Book>
        All About SGML
        <chapter>
                Einleitung
        </chapter>
</Book>
```

In obigem Beispiel beinhaltet das Element Book einen Mixed Content. Einerseits sind innerhalb des Elements Daten aufgeführt ("All About SGML"), andererseits ist aber auch ein Element enthalten (chapter).

6.5.2 Element Content

Der element content gestattet es, weitere Elemente innerhalb eines Elements anzugeben. Diese Elemente sind dann wiederum Elemente entsprechend Regel[13] der abstract syntax und können somit auch wieder einen content enthalten. Auch hier besteht die Möglichkeit, eine hierarchische Dokumentstruktur zu realisieren.

Beispiel für einen Element content:

```
<Book>
        <title>
                All About SGML
        </title>
        <chapter>
                Einleitung
        </chapter>
</Book>
```

In diesem Fall beinhaltet das Element Book keine Daten, sondern besteht nur aus weiteren Elementen (title und chapter).

6.5.3 Other Content

Sowohl mixed content, als auch element content können other content beinhalten. Other content wird durch Regel[27] spezifiziert.

[27] other content =
comment declaration [91] |
short reference use declaration [152] |
link set use declaration [169] |
processing instruction [44] |
shortref |
character reference [62] |
general entity reference [59] |
marked section declaration [93] |
Ee

Die syntactic variable "other content" ermöglicht es somit, neben Daten und Elementen, weitere Inhalte innerhalb der Dokumentinstanz anzugeben.

Ein "other content" kann entsprechend der Syntax nur innerhalb von Daten, oder nach der Angabe eines Elements angegeben werden. Es ist somit nicht möglich "other content" innerhalb von Tags anzugeben.

Comment Declaration

Die Comment Declaration ermöglicht das Einbringen von Kommentartexten in die Dokumentinstanz.

[91] comment declaration =
mdo ,
(comment [92] ,
(s [5] |
comment [92])*)?,
mdc

[92] comment =
com ,
SGML character [50] *,
com

Beispiel für einen Kommentar:

<!-- Dies ist ein Kommentar -->

<!	Zeichenabbildung der „reference concrete syntax“ für mdo (Markup Delimiter Open)
--	Document Type Specification
Dies ist ein Kommentar	Text bestehend aus SGML Character
>	Zeichenabbildung der „reference concrete syntax“ für mdc (Markup Delimiter Close)

Short Reference Use Declaration / Shortref

Die Verwendung von Kurzreferenzen ermöglicht es, (bestimmten) Zeichenketten Entitäten zuzuordnen. Der SGML-Parser ersetzt dann diese Zeichenkette mit dem Inhalt der zugeordneten Entität. In-

nerhalb der Dokumentinstanz lassen sich keine neuen Kurzreferenzen-Abbildungen (short reference maps) deklarieren. Die Angabe von short reference maps kann nur innerhalb der DTD erfolgen.

Innerhalb der Dokumentinstanz hat der Anwender jedoch die Möglichkeit, eine short reference map zu aktivieren bzw. zu deaktivieren. Die Syntax dazu ergibt sich aus Regel[152].

[152] short reference use declaration =
mdo ,
"USEMAP" ,
ps [65] +,
map specification [153] ,
(ps [65] +,
associated element type [72])?,
ps [65] *,
mdc

[153] map specification =
map name [151] |
(rni ,
"EMPTY")

[151] map name =
name [55]

Wird dem Schlüsselwort USEMAP der Bezeichner für eine in der DTD deklarierte short reference map eingegeben, so wird diese Abbildung innerhalb der Dokumentinstanz aktiviert. Wird statt des Bezeichners die Zeichenkette #EMPTY eingegeben, so wird die derzeit aktive Abbildung deaktiviert.

Beispiel:

```
<!ELEMENT q          -- (#PCDATA) >
<!ENTITY startq "<q>" >
<!SHORTREF qmap
        "~"     startq >
<!USEMAP qmap q >
...
<q> Dies ist Text</q>
```

wäre auch darstellbar durch

~ Dies ist Text </q>

In diesem Fall wird das Zeichen "~" als shortref erkannt und der in startq angegebene Text wird eingesetzt.

Probleme treten dann auf, wenn das Zeichen "~" nicht als shortref interpretiert werden soll, sondern selbst als Text erscheinen soll. In diesem Fall besteht die Möglichkeit die short reference map zu deaktivieren.

~ Dies ist Text mit<USEMAP #EMPTY> ~ <USEMAP qmap> </q>

Dieses Verfahren ist jedoch kein sehr elegantes Verfahren, da der Verfasser des Textes den Bezeichner der short reference map kennen muß.

Als Zeichenketten für shortrefs dürfen nicht beliebige Zeichen gewählt werden, sondern nur die Zeichen, die innerhalb der SGML Declaration dafür deklariert wurden.

Des weiteren muß das Feature SHORTREF innerhalb der SGML Declaration aktiviert sein.

Link Set Use Declaration

Siehe *Weitere Möglichkeiten von SGML - LINK-Attribute.*

Processing Instruction

Verarbeitungsanweisungen dienen dazu, Befehle direkt an eine SGML-Anwendung zu schicken, und bieten somit die Möglichkeit systemspezifische Informationen in ein Dokument zu integrieren. Diese Information wird vom Parser nicht beachtet, und kann somit sämtliche erlaubten Zeichen enthalten (außer die Zeichendarstellung für mdo).

Die Syntax für die Angabe einer Processing Instruction wird durch Regel[44] bestimmt.

[44] processing instruction =
pio ,
system data [45] ,
pic

[45] system data =
character data [47]

Von der Verwendung von Processing Instruction ist nach Möglichkeit abzusehen, da durch deren Verwendung die Portabilität eines SGML-Dokuments nahezu unmöglich wird.

Character Reference

Character References ermöglichen es, den Zeichencode für ein Zeichen zu referenzieren. Somit besteht die Möglichkeit, Zeichen, die mit dem aktuellen Zeichensatz nicht dargestellt werden können, oder deren Darstellung nicht erwünscht ist, in den Inhalt eines Dokuments mit einzubeziehen.

[62] character reference =
cro ,
(function name [63] |
character number [64]),
reference end [61]

[61] reference end =
(refc |
RE)

Es gibt zwei unterschiedliche Möglichkeiten einen Zeichencode zu referenzieren :

i. Der Zeichencode für das Zeichen wird direkt angegeben
ii. Der Name eines Function Characters wird angegeben (Diesem Name muß innerhalb der SGML Declaration ein Zeichencode zugewiesen worden sein)

Möglichkeit i. ist sinnvoll, wenn das gewünschte Zeichen mit dem Zeichensatz des Texteditors nicht darstellbar ist.

Möglichkeit ii. wird aus Gründen der Portabilität eingesetzt. Ist der Zeichencode auf einem anderen System für das gewünschte Zeichen ein anderer, so muß bei der Portierung nur die Zuweisung innerhalb der SGML Declaration angepaßt werden.

Wichtig ist, daß die Referenzierung immer durch ein reference end abgeschlossen wird, also mit einem ";" (reference concrete syntax für refc) oder einem chr(13) (reference concrete syntax für RE).

Beispiel:

Der Dezimalcode 67 entspricht Zeichen C

oder

Der Dezimalcode 32 entspricht Zeichen &#SPACE;

General Entity Reference

Ähnlich wie Zeichencodes referenziert werden können, können auch Entitäten, die innerhalb der DTD deklariert wurden, referenziert werden. Wird eine Entität referenziert, so ersetzt der SGML-Parser die Referenz durch den Inhalt, der dieser Entität zugewiesen wurde.

[59] general entity reference =
ero ,
document type specification [28] ,
name [55] ,
reference end [61]

[61] reference end =
(refc |
RE) ?

Ähnlich wie bei der Angabe eines Generic Identifiers kann eine document type specification angegeben werden, um die DTD zu bestimmen, in der die Entität deklariert wurde.

Eine Entitätsreferenz wird mit dem Zeichen "&" (reference concrete syntax für ero) eingeleitet. Anschließend wird der Entitätsbezeichner angegeben. Die Entitätsreferenz ist wiederum über ein reference end abzuschließen.

Beispiel :

<!ENTITY T "Referenzen" >

...

Über Entitäten sind &T; möglich

Marked Section Declaration

Ein markierter Bereich ist ein Teil der Dokumentinstanz mit besonderem Status.

[93] marked section declaration =
marked section start [94] ,
status keyword specification [97] ,
dso,
marked section [96] ,
marked section end [95]

[94] marked section start =
mdo ,
dso

[95] marked section end =
msc ,
mdc

[96] marked section =
SGML character [50] *

[97] status keyword specification =
(ps [65] +,
(status keyword [100] |
"TEMP"))*,
ps [65] *

[100] status keyword =
"CDATA" |
"IGNORE" |
"INCLUDE" |
"RCDATA"

Der Status des markierten Bereichs wird durch eines der folgenden Schlüsselwörter bestimmt :

CDATA	der markierte Bereich enthält nur Zeichen. Elementmarkierungen und Entitätsreferenzen werden nicht verarbeitet
RCDATA	der markierte Bereich enthält Zeichen und Entitätsreferenzen. Elementmarkierungen werden nicht verarbeitet
IGNORE	der markierte Bereich wird behandelt, als wäre er nicht vorhanden
INCLUDE	der markierte Bereich wird behandelt, als ob die Markierung nicht vorhanden wäre
TEMP	der markierte Bereich ist ein temporärer Dokumentteil

Wie aus der Grammatik ersichtlich wird, können mehrere Stati spezifiziert werden. In diesem Fall gilt der Status mit höchster Priorität. Die Prioritätsreihenfolge ist IGNORE, CDATA, RCDATA, INCLUDE.

Die Verwendung markierter Bereiche läßt sich am besten an einem Beispiel verdeutlichen. Um bspw. in einem Buch über SGML Beispiele anzugeben, läßt sich ein markierter Bereich vom Status CDATA angeben :

...

Das folgende Beispiel zeigt eine ELEMENT-Deklaration :

<![CDATA [

<! ELEMENT Book - - (Title,Chapter+) >

]]>

...

<!	Reference Concrete Syntax für mdo (Markup Delimiter Open)
[	Reference Concrete Syntax für dso (Declaration Subset Open)
CDATA	Status Keyword
<! ELEMENT Book - - (Title,Chapter+) >	Text zusammengesetzt aus SGML Character
]]	Reference Concrete Syntax für msc (Marked Section Close)
>	Reference Concrete Syntax für mdc (Markup Delimiter Close)

Obiges Beispiel hat zur Folge, daß der innerhalb der Marked Section angegebene Text nicht auf Markierungen oder Referenzen untersucht wird.

Die Stati IGNORE und ICNLUDE eignen sich besonders gut, um mehrere Versionen eines Dokuments zu verwalten. In folgendem Beispiel soll ein Mathematikbuch geschrieben werden, wobei eine Version für Schüler (nur Aufgaben) und eine Version für Lehrer (Aufgaben und Ergebnisse) verwaltet werden soll.

...

<! ENTITY % Teacher „IGNORE“ >

<! ENTITY % Student „INCLUDE“ >

...

<excercise> 1 + 1 = <![%Teacher; [2]]]> </excercise>

...

Dieses Beispiel hätte zur Folge, daß die Ergebnisse der Aufgaben nicht ausgedruckt werden. Um nun die Lehererversion des Buches zu drucken, müßte nur das Wort IGNORE innerhalb der ENTITY-Deklaration von Teacher mit INCLUDE ausgetauscht werden.

7 SGML Declaration

Die SGML Declaration bietet dem Anwender die Möglichkeit, die lexikalische Struktur seines SGML-Dokuments zu definieren, d.h. wie die einzelnen Komponenten seines Dokumentes dargestellt werden können bzw. müssen. Für den Anwender ist es nicht zwingend erforderlich eine SGML-Declaration anzugeben. In diesem Fall werden Standardeinstellungen von der SGML-Software angenommen, die für die meisten Anwendungen ausreichend sind. Gründe für die Angabe einer SGML Declaration sind häufig folgende :

- Die von der verwendeten "concrete syntax" vorgegebenen delimiter strings sollen individuell angepaßt werden
- Die maximale Länge von Bezeichnern (bspw. für ELEMENT-Deklarationen soll individuell angepaßt werden
- Markup Minimization soll nicht erlaubt sein
- Der im Dokument verwendete Zeichensatz wird vom System nicht unterstützt, und muß deshalb auf den vom System verwendeten Zeichensatz abgebildet werden.

Soll eine dieser Anpassungen vorgenommen werden, so ist die Angabe einer SGML Declaration erforderlich

7.1 Aufbau der SGML Declaration

Die einzelnen Bestandteile einer SGML Declaration ergeben sich aus Regel[171] der "abstract syntax".

[171] SGML declaration =
mdo ,
"SGML" ,
ps [65] +,
minimum literal [76] ,
ps [65] +,
document character set [172] ,
ps [65] +,
capacity set [180] ,
ps [65] +
concrete syntax scope [181]
ps [65] +,
concrete syntax [182] ,
ps [65] +,

feature use [195] ,
ps [65] +,
application-specific information [199] ,
ps [65] *,
mdc

Das hinter dem Schlüsselwort "SGML" anzugebende minimum literal dient dazu, die verwendete SGML Version anzugeben. Derzeit ist nur eine SGML Version von der ISO standardisiert, so daß nur der Wert "ISO 8879:1986" angegeben werden kann.

Die restlichen Bestandteile der SGML Declaration sollen nun etwas genauer untersucht werden.

7.2 Die Spezifikation von Zeichensätzen

Die Spezifikation von Zeichensätzen ist immer dann notwendig, wenn ein Dokument mittels eines Zeichensatzes kodiert wurde, der auf dem aktuellen System nicht zur Verfügung steht. Würde keine Anpassung des Zeichensatzes vorgenommen, so würde dies dazu führen, daß Zeichen innerhalb des SGML-Dokuments nicht korrekt interpretiert werden können.

Wie die Zeichensatzspezifikation anzugeben ist, ergibt sich aus Regel [172] und [173] der "abstract syntax".

[172] document character set =
"CHARSET" ,
ps [65] +,
character set description [173]

[173] character set description =
base character set [174] ,
ps [65] +,
described character set portion [175] ,
(ps [65] +
base character set [174] ,
ps [65] +,
described character set portion [175])*

Die Zeichensatzspezifikation wird durch das Schlüsselwort "CHARSET" eingeleitet. Im Anschluß daran wird die eigentliche Spezifikation vorgenommen. Zunächst muß ein Basiszeichensatz angegeben werden, der auf dem aktuellen System zur Verfügung steht, und der SGML Anwendung bekannt ist.

[174] base character set =
"BASESET" ,
ps [65] +,
public identifier [74]

Bem.: Ein "public identifier" ist ein "minimum literal", d.h. eine minimale Zeichenkette. Minimal bedeutet, daß nur die allerwichtigsten Zeichen erlaubt sind, d.h. Buchstaben und Ziffern, sowie

Zeilenanfangs- und Zeilenendezeichen sowie die Spezialzeichen : "'()+,-./=? Sonderzeichen, wie z.B. das Tabulatorzeichen, sind nicht erlaubt. Diese Zeichen sind durch Regel [78] der "abstract syntax" fest gelegt.

[78] minimum data character =

***RS* |**

***RE* |**

***SPACE* |**

***LC Letter* |**

***UC Letter* |**

***Digit* |**

Special

Durch die Einschränkung auf das Allernötigste ist es möglich, diese Zeichen in nahezu jedem Systemzeichensatz darzustellen.

Der angegebene "public identifier" dient somit dazu, den Basiszeichensatz der SGML Anwendung zu identifizieren (Es sei erwähnt, daß der "public identifier" zur Identifizierung eines Zeichensatzes nach einer bestimmten Syntax aufgebaut sein muß, die durch Regel[79] der "abstract syntax" festgelegt wird. Auf diese Syntax soll allerdings nicht weiter eingegangen werden.).

Im Anschluß an die Angabe eines Basiszeichensatzes erfolgt die Abbildung der im Dokument verwendeten Zeichencodes auf die Zeichencodes des Basiszeichensatzes.

[175] described character set portion =

"DESCSET" ,

(ps [65] +,

character description [176])+

[176] character description =

described set character number [177] ,

ps [65] +,

number of characters [179] ,

ps [65] +,

(base set character number [178] |

minimum literal [76] |

"UNUSED")

Anschaulich gesehen erfolgt nach der Angabe des Schlüsselworts "DESCSET" die Definition einer Tabelle, die die Abbildung von Zeichencodes des Dokuments auf Zeichencodes des Basiszeichensatzes festlegt.

Beispiel zur Zeichensatzspezifikation :

CHARSET

BASESET "ISO 646-1983//CHARSET

International Reference Version (IRV)//ESC 2/5 4/0"

DESCSET

0	**9**	**UNUSED**	
9	**2**	**9**	
11	**2**	**UNUSED**	
13	**1**	**13**	
14	**18**	**UNUSED**	
32	**33**	**32**	
65	**26**	**97**	**-- Upcase to Lowcase --**
91	**6**	**91**	
97	**26**	**65**	**-- Lowcase to Upcase --**
123	**4**	**123**	
127	**1**	**UNUSED**	

Dieses Beispiel stellt eine sehr einfache Variante einer Zeichensatzspezifikation dar. Diese Spezifikation kann verwendet werden, um den Großbuchstaben eines Dokuments die Rolle der Kleinbuchstaben zuzuordnen und umgekehrt, unter der Voraussetzung, daß das Dokument mit dem ISO 646 Zeichensatz codiert wurde (Wahrscheinlich nicht sehr sinnvoll, aber illustrativ).

Als Basiszeichensatz wird die ASCII-Variante der ISO, der Standard 646 gewählt. Zunächst wird der SGML-Anwendung mitgeteilt, daß die neun Zeichencodes 0 bis 8 innerhalb eines SGML-Dokuments nicht verwendet werden dürfen. Die beiden im Dokument verwendeten Zeichencodes 9 und 10 werden auf die Zeichencodes 9 und 10 des ISO 646 Standards abgebildet (HT und LF). Die Zeichencodes 11 und 12 dürfen wiederum nicht im Dokument verwendet werden. Die 33 Zeichencodes 32 bis 64 werden auf die Spezialzeichen und die Ziffern des ISO 646 Zeichensatzes abgebildet. Die 26 Zeichencodes 65 bis 90 werden auf die Zeichencodes 97 bis 122 abgebildet. Die 26 Zeichencodes 97 bis 122 werden auf die Zeichencodes 65 bis 90 abgebildet. Erwähnenswert ist, daß die im Dokument verwendeten Zeichencode nicht ausgetauscht werden, sondern lediglich von der SGML-Anwendung anders interpretiert werden. Die ursprüngliche Codierung des SGML-Dokuments bleibt also erhalten.

Werden in dem SGML-Dokument auch die Zeichencodes 128 bis 255 verwendet (weil das SGML-Dokument bspw. mit Hilfe des ISO Latin 1 Zeichensatzes codiert wurde), so gelten diese Zeichen bei der obigen Zeichensatzspezifikation als ungültige Zeichencodes (nicht aufgeführte Zeichencodes gelten grundsätzlich als unerlaubte Zeichencodes). Um auch diesen Zeichencodes eine Bedeutung zu geben, könnte die obige Spezifikation folgendermaßen erweitert werden :

CHARSET

BASESET "ISO 646-1983//CHARSET

International Reference Version (IRV)//ESC 2/5 4/0"

DESCSET

0	**9**	**UNUSED**
9	**2**	**9**
11	**2**	**UNUSED**
13	**1**	**13**

14 18 UNUSED
32 33 32
65 26 97 -- Upcase to Lowcase --
91 6 91
97 26 65 -- Lowcase to Upcase --
123 4 123
127 1 UNUSED

BASESET "ISO Registration Number 100//CHARSET ECMA-94 Rigth Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"

DESCSET
128 32 UNUSED
160 96 160

Durch die Angabe eines weiteren Basiszeichensatzes wird der SGML-Anwendung der ISO Latin 1 Zeichensatz zur Verfügung gestellt. Die 32 Zeichencodes 128 bis 159 werden als ungültige Zeichen markiert. Die Codes 160 bis 255 werden eins-zu-eins auf die Zeichencodes des ISO Latin 1 Zeichensatzes abgebildet.

7.3 Die Spezifikation von Kapazitäten

Die Kapazitätsspezifikation definiert die Menge der notwendigen Ressourcen, die zur Verarbeitung eines Dokuments notwendig sind. Hinsichtlich der Portabilität von einem System auf ein anderes ist es sinnvoll, die Kapazitätsbeschränkungen zur Verarbeitung eines SGML-Dokuments anpassen zu können. Wie eine Kapazitätsspezifikation anzugeben ist, wird durch Regel[180] der "abstract syntax" definiert.

[180] capacity set =
"CAPACITY" ,
ps [65] +,
(("PUBLIC" ,
ps [65] +,
public identifier [74]) |
("SGMLREF" ,
(ps [65] +,
name [55] ,
ps [65] +,
number [56])+))

Die Kapazitätsspezifikation wird durch das Schlüsselwort "CAPACITY" eingeleitet. Anschließend hat man die Wahl, über einen "public identifier" eine Standardspezifikation zu wählen, oder eigene Kapa-

zitätsspezifikationen durchzuführen (Für den public identifier gilt die gleiche Bemerkung wie im vorherigen Abschnitt).

Für folgende Werte lassen sich Kapazitätsbeschränkungen festlegen :

Bezeichnung	Standardwert	Beschreibung
TOTALCAP	35000	Gesamtsumme der Kapazitätspunkte
ELEMCAP*	35000	Elemente
ENTCAP	35000	definierte Entitäten
IDREFCAP*	35000	IDREF-Attribute
ENTCHCAP*	35000	Zeichen im Ersetzungstext für Entitäten
EXGRCAP*	35000	Exklusions- oder Inklusionsgruppen
EXNMCAP*	35000	Namen in Exklusions- oder Inklusionsgruppen
ATTCHCAP	35000	Gesamtlänge der voreingestellten Attributwerte
GRPCAP*	35000	Bestandteile von Inhaltsmodellen
IDCAP*	35000	ID-Attribute
ATTCAP*	35000	Attribute
LKNMCAP*	35000	Elemente in Linkdeklarationen
LKSETCAP*	35000	Linkdeklarationen
AVGRPCAP*	35000	Namen in der Namensliste von Attributdeklarationen
MAPCAP*	35000	Kurzreferenzen
NOTCAP*	35000	Notationen
NOTCHCAP	35000	Zeichen in Notationsnamen

Bei den mit * versehenen Kapazitäten ist der angegebene Wert durch die vorgegebene NAMELEN-Quantität zu dividieren (Voreinstellung für NAMELEN ist 8).

Das bedeutet, daß bei einem Wert von ATTCAP 72 innerhalb der DTD 9 Attribute deklariert werden dürfen.

Der Vorgabewert 35000 erlaubt es somit, innerhalb einer DTD bis zu 4375 Bestandteile zu deklarieren.

Beispiel zur Kapazitätsspezifikation :

CAPACITY
PUBLIC "ISO 8879:1986//CAPACITY Reference//EN"

In diesem Fall werden die vom Standard vorgeschlagenen Kapazitätswerte übernommen. Als Obergrenze gibt der Standard für alle Kapazitätswerte 35000 vor. Sind diese Kapazitätswerte für eine Anwendung nicht ausreichend, so könnte eine Kapazitätsspezifikation folgendermaßen aussehen :

CAPACITY SGMLREF
TOTALCAP 50000
ELEMCAP 40000
ENTCAP 40000

Dies führt dazu, daß für die angegebenen Kapazitätswerte nicht der Standardwert 35000, sondern angegebene Wert eingesetzt wird.

7.4 Die Spezifikation einer Concrete Syntax

Wie bereits zu Beginn dieser Ausarbeitung angesprochen wurde, wird durch den SGML Standard keineswegs festgelegt, wie die Terminalzeichen der Grammatik auf konkrete Zeichen abgebildet werden. Zur Definition von SGML wurde eine "abstract syntax" definiert, wobei die Terminalzeichen als Platzhalter angesehen werden, die dann durch die Angabe einer "concrete syntax" auf konkrete Zeichen eines Zeichensatzes abgebildet werden können.

Durch die "abstract syntax" ist bspw. nicht festgelegt, daß eine Elementdeklaration mit den Zeichen "<!" beginnen muß. Die "abstract syntax" verwendet zu diesem Zweck das Symbol mdo (markup delimiter open). Die Abbildung des Symbols mdo auf die Zeichenkette "<!" wird erst durch die "reference concrete syntax" festgelegt.

Die bisher SGML-Syntax ist also nur einer spezielle Syntax unter einer nahezu unendlichen Anzahl denkbarer Syntaxmöglichkeiten. Die bisher verwendete "reference concrete syntax" ist allerdings dadurch ausgezeichnet, daß sie die durch den Standard definierte "concrete syntax" ist, und daher als quasi-Standard einer "concrete syntax" angesehen werden kann.

Über die Regel[181] wird festgelegt, wie der Gültigkeitsbereich der definierten "concrete syntax" anzugeben ist.

[181] concrete syntax scope =
"SCOPE" ,
ps [65] +,
("DOCUMENT" |
"INSTANCE")

Der Anwender hat die Möglichkeit, die "concrete syntax" auf das gesamte Dokument (also Prolog und Instanz) oder nur auf die Instanz anzuwenden. Findet die "concrete syntax" ihre Anwendung nur in der Instanz, so wird für den Prolog die "reference concrete syntax" verwendet. Wie eine "concrete syntax" zu definieren ist, wird durch die Regel[182] der "abstract syntax" festgelegt.

[182] concrete syntax =
"SYNTAX" ,
ps [65] +,

(public concrete syntax [183] |
(shunned character number identification [184] ,
ps [65] +,
syntax-reference character set [185] ,
ps [65] +,
function character identification [186] ,
ps [65] +
naming rules [189] ,
ps [65] +,
delimiter set [190] ,
ps [65] +,
reserved name use [193] ,
ps [65] +,
quantity set [194]))

Bei der Definition einer "concrete syntax" kann der Anwender entweder eine bereits existierende "concrete syntax" auswählen (im folgenden als Public Syntax bezeichnet), oder aber eine eigene "concrete syntax" definieren. Näheres zum Einbinden einer Public Syntax siehe Abschnitt „Einbinden einer Public syntax“

7.4.1 Shunned Characters

Zunächst sind die "shunned characters" entsprechend Regel[184] festzulegen.

[184] shunned character number identification =
"SHUNCHAR" ,
ps [65] +,
("NONE" |
(("CONTROLS" |
character number [64]),
(ps [65] +,
character number [64])*))

"shunned characters" sind Zeichencodes, die vom System als Kontrollcodes verwendet werden, und nicht innerhalb eines SGML-Dokuments verwendet werden dürfen, da deren Verarbeitung innerhalb eines SGML-Dokuments zu Schwierigkeiten führen könnte. Wird bspw. ein Kontrollcode zur Dateiendemarkierung eingesetzt, so sollte dieser Code nicht innerhalb eines SGML Dokuments verwendet werden. Durch die Angabe des Schlüsselworts "CONTROLS" können sämtliche Zeichencodes, die innerhalb des verwendeten Dokumentzeichensatzes als Kontrollzeichen dienen, als "shunned characters" deklariert werden. Des weiteren kann durch aufzählen sämtlicher Zeichencodes, von denen angenommen wird, daß sie als Kontrollzeichen verwendet werden, die Menge der "shunned characters" erweitert werden. Sollen keine "shunned characters" spezifiziert werden ,so wird dies durch das

Schlüsselwort "NONE" gekennzeichnet. Sämtliche als "shunned characters" deklarierten Zeichencodes müssen innerhalb der Zeichensatzspezifikation als "UNUSED" gekennzeichnet werden.

Auch in diesem Fall bestätigt die Ausnahme die Regel. Es gibt "shunned characters", die zur Bildung von Bezeichner benutzt werden können (bspw. SPACE). Diese Zeichen werden im Teil function character identification deklariert, und müssen innerhalb der Dokumentzeichensatzes nicht als "UNUSED" gekennzeichnet sein.

Beispiel zur Definition von "shunned characters" :

SHUNCHAR CONTROLS 255

Diese Deklaration hat zur Folge, daß sämtliche Kontrollzeichen, sowie der Zeichencode 255 als "shunned characters" festgelegt werden.

7.4.2 Syntax Reference Character-Set

Das "syntax reference character set" ist entsprechend Regel[185] der "abstract syntax" festzulegen.

[185] syntax-reference character set =

character set description [173]

Wie bereits ersichtlich wird, erfolgt die Definition analog zu der Definition eines Dokumentzeichensatzes.

Allerdings ist die Bedeutung eine grundsätzlich andere. Der Syntaxzeichensatz bestimmt den Zeichensatz, der der Syntaxspezifikation zu Grunde liegt. Sämtliche in der Syntaxspezifikation verwendeten Zeichencodes beziehen sich auf den hier angegebenen Zeichensatz.

7.4.3 Function Characters

"function characters" sind Zeichen, die eine besondere Bedeutung, und somit eine besondere Funktion, innerhalb von SGML Dokumenten haben.

Die Definition von "function characters" erfolgt entsprechend Regel[186].

[186] function character identification =

"FUNCTION" ,

ps [65] +,

"RE" ,

ps [65] +,

character number [64] ,

ps [65] +,

"RS" ,

ps [65] +,

character number [64] ,

ps [65] +,
"SPACE" ,
ps [65] +,
character number [64] ,
(ps [65] +,
added function [187] ,
ps [65] +,
function class [188] ,
ps [65] +,
character number [64])*

Zwingend angegeben werden müssen die Zeichencodes für die "function characters" RE, RS, SPACE. RE bezeichnet das Zeilenendezeichen, RS das Zeilenstartzeichen und SPACE das Zwischenraumzeichen. Darüber hinaus können weitere "function characters" spezifiziert werden. Jedem weiteren "function character" muß dabei eine Funktionsklasse zugewiesen werden.

Funktionsklassen werden durch Regel[188] festgelegt.

[188] function class =
"FUNCHAR" |
"MSICHAR" |
"MSOCHAR" |
"MSSCHAR" |
"SEPCHAR"

Die Bedeutung der Funktionsklassen ist folgende :

SEPCHAR — Zeichen, die als Trennzeichen erlaubt sind, und wie ein Leerzeichen interpretiert werden. RE, RS und SPACE sind solche Zeichen. Ein weiters Zeichen, welches häufig als SEPCHAR deklariert wird ist das TAB-Zeichen.

MSSCHAR — Markup supression characters. Diese Zeichen dienen dazu den Parser anzuweisen, das direkt folgende Zeichen nicht als Markierung zu erkennen. Somit besteht die Möglichkeit, Zeichen die sonst vom Parser als Markierungszeichen erkannt werden als Text in ein SGML-Dokument einzubauen. Wird bspw. der Backslash durch

BACKSL MSSCHAR 92

als MSSCHAR deklariert, so führt das Parsen der Textpassage "\&" dazu, daß das "&"-Zeichen nicht als Konnektor, sondern als normales Textzeichen aufgefaßt wird.

MSICHAR MSICHAR und MSOCHAR sind ebenfalls Markup supression

MSOCHAR characters. Findet der Parser ein MSOCHAR so werden alle diesem Zeichen folgenden Zeichen nicht als Markierungen erkannt, bis ein MSICHAR gefunden wird. Wird ein MSOCHAR definiert, so muß mindestens ein MSICHAR angegeben werden.

FUNCHAR Ein FUNCHAR ist ein Zeichencode, welcher einen Namen ,aber keine besondere Funktion zugewiesen bekommt. Ein Zeichencode kann dann innerhalb eines Dokuments über diesen Name referenziert werden. Wird bspw. das Delete-Zeichen durch

DEL FUNCHAR 127

als FUNCHAR deklariert, so kann der Zeichencode innerhalb des SGML-Dokuments über

&#DEL

referenziert werden.

7.4.4 Naming Rules

Durch die Spezifikation von Naming Rules kann festgelegt werden, welche Zeichen in Namen verwendet werden können, und ob Groß- und Kleinschreibung signifikant ist (Mit Namen sind in diesem Fall Element-Namen, Entity-Namen, Attribut-Namen , etc. gemeint. Also alles, was in SGML bezeichnet werden kann bzw. muß.).

Ein SGML-Name wird nach Regel [55] der abstract syntax gebildet.

[55] name =
name start character [53] ,
name character [52] *

[53] name start character =
LC Letter |
UC Letter |
LCNMSTRT |
UCNMSTRT

[52] name character =
name start character [53] |
Digit |
LCNMCHAR |
UCNMCHAR |

Um eine spezifische Anpassung der Naming Rules vorzunehmen hat der Anwender die Möglichkeit, die von SGML vorgegebenen Zeichenklassen zu erweitern. Von SGML werden bereits Zeichenklassen vorgegeben, die vom Anwender nicht mehr geändert werden können. Diese Zeichenklassen werden unter dem Begriff "terminal constants" zusammengefaßt und umfassen folgende Klassen:

Digit 0 1 2 3 4 5 6 7 8 9 (ASCII 48...57)

Ee The Entity End is a signal, and is never treated as data.

LC Letter a b c d e f g h i j k l m n o p q r s t u v w x y z
(ASCII 97...122)

Special ' ( ) + , - . / : = ? (ASCII 39 40 41 43 44 45 46 47 58 61 63)

UC Letter A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
(ASCII 65...90)

Erweiterbare Klassen sind folgende:

LCNMSTRT Diese Klasse umfaßt alle Zeichen, die als Kleinbuchstaben interpretiert werden und am Beginn eines Namens stehen dürfen.

UCNMSTRT Diese Klasse umfaßt alle Zeichen, die als Großbuchstaben interpretiert werden und am Beginn eines Namens stehen dürfen.

LCNMCHAR Diese Klasse umfaßt alle Zeichen, die als Kleinbuchstaben interpretiert werden und innerhalb eines Namens verwendet werden dürfen.

UCNMCHAR Diese Klasse umfaßt alle Zeichen, die als Großbuchstaben interpretiert werden und innerhalb eines Namens verwendet werden dürfen.

Die Spezifikation von Naming Rules erfolgt entsprechend Regel[189].

```
[189] naming rules =
   "NAMING" ,
   ps [65] +,
   "LCNMSTRT" ,
   ( ps [65] +,
   extended naming value [189.1] )+,
   ps [65] +,
   "UCNMSTRT" ,
   ( ps [65] +,
   extended naming value [189.1] )+,
   ( ps [65] +,
   "NAMESTRT" ,
   ( ps [65] +,
   extended naming value [189.1] )+ ,
   ps [65] + )?,
   "LCNMCHAR" ,
   ( ps [65] +,
   extended naming value [189.1] )+,
   ps [65] +,
   "UCNMCHAR" ,
   ( ps [65] +,
   extended naming value [189.1] )+,
   ( ps [65] +,
   "NAMECHAR" ,
   ( ps [65] +,
   extended naming value [189.1] )+ ,
   ps [65] + )?,
   "NAMECASE" ,
   ps [65] +,
   "GENERAL" ,
   ps [65] +,
   ( "NO" |
   "YES" ),
   ps [65] +,
   "ENTITY" ,
   ps [65] +,
   ( "NO" |
   "YES" )
```

Nach dem Schlüsselwort NAMECASE wird festgelegt, ob Groß-/Kleinschreibung signifikant ist. Unterschieden wird dabei nach Entitätsname bzw. Entitätsreferenzen und allen anderen Name, die in einem SGML-Dokument auftauchen können. Die Angabe eines YES bedeutet, daß ein Kleinbuchstabe durch einen Großbuchstaben ersetzt wird. Die Angabe eines NO bedeutet, daß dies nicht geschieht. Bei NO ist Groß-/Kleinschreibung somit signifikant.

Eine Spezifikation für Naming Rules könnte bspw. folgendermaßen aussehen:

```
NAMING
LCNMSTRT ""
UCNMSTRT ""
LCNMCHAR ".-_"
UCNMCHAR ".-_"
NAMECASE   GENERAL     YES
           ENTITY      NO
```

Durch diese Spezifikation wird festgelegt, daß innerhalb von Namen auch die Zeichen ".", "-" und "_" vorkommen können, und daß Groß-/Kleinschreibung nur für Entitäten signifikant ist.

An dieser Stelle soll noch auf eine wichtige Besonderheit hingewiesen werden. Die Reihenfolge, in der die Zeichen in die Zeichenklassen eingefügt werden, ist keinesfalls beliebig. Es besteht ein Zusammenhang zwischen den Klassen LCNMSTRT und UCNMSTRT, sowie den Klassen LCNMCHAR und UCNMCHAR. Die Großbuchstabenentsprechung des ersten Zeichens der Klasse LCNMSTRT ist das erste Zeichen der Klasse UCNMSTRT, die Großbuchstabenentsprechung des zweiten Zeichens in LCNMSTART ist das zweite Zeichen der Klasse UCNMSTRT etc.

Gleiches gilt für die Klassen LCNMCHAR und UCNMCHAR. In obigem Beispiel ist die Großbuchstaben Entsprechung des Zeichens "." also wieder das Zeichen ".".

7.4.5 Delimiter Set

Die Spezifikation eines Delimiter Sets dient dazu, die delimiter roles der abstract syntax auf konkrete Zeichen oder Zeichenketten abzubilden. Zu unterscheiden ist zwischen den "General Delimiters" und den "Short Reference Delimiters". Unter General Delimitern werden die in der abstract syntax verwendeten Symbole, die zur Deklaration und Begrenzung von Markierungen verwendet werden, zusammengefaßt. Vom SGML Standard werden folgende Abbildungen der General Delimiter vorgeschlagen:

delimiter role	Zeichen	Beschreibung
AND	"&"	AND Connector
COM	"--"	Comment start or end
CRO	"&#"	Character reference open
DSC	"]"	Declaration subset close
DSO	"["	Declaration subset open
DTGC	"]"	Data tag group close
DTGO	"["	Data tag group open
ERO	"&"	Entity reference open
ETAGO	"</"	End tag open
GRPC	")"	Group close
GRPO	"("	Group open
LIT	""	Literal start or end
LITA	""	Literal start or end (alternative)
MDC	">"	Markup declaration close
MDO	"<!"	Markup declaration open
MINUS	"-"	Exclusion
MSC	"]]"	Marked section close
NET	"/"	Null end-tag
OPT	"?"	Optional occurence indicator
OR	"|"	Or connector

PERO	"%"	Parameter entity reference open
PIC	">"	Processing instruction close
PIO	"<?"	Processing instruction open
PLUS	"+"	Required and repeatable, inclusion
REFC	";"	Reference close
REP	"*"	Optional and repeatable
RNI	"#"	Reserved name indicator
SEQ	","	Sequence connector
STAGO	"<"	Start tag open
TAGC	">"	Tag close
VI	"="	Value indicator

Unter short reference delimitern versteht man sämtliche Zeichen, die zur Bildung von Short references benutzt werden können. Vom SGML Standard werden folgende Zeichen vorgeschlagen :

" # % ' ( ) * + , - -- : ; = [ ] ^ { | } ~

Darüber hinaus sieht der Standard eine Reihe von Funktionszeichen als gültige Zeichen vor (dabei steht das B für mindestens ein Zwischenraumzeichen(SEPCHAR), und das BB für mindestens zwei Zwischenraumzeichen):

&#TAB
&#RE
&#RS
&#RS;B
&#RS;&#RE
&#RS;B&#RE
B&#RE
&#SPACE
BB

Um eine spezifische Anpassung der Delimiter vorzunehmen, geht man entsprechend Regel [190] der abstract syntax vor.

[190] delimiter set =
"DELIM" ,
ps [65] +,
general delimiters [191] ,
ps [65] +,
short reference delimiters [192]

[191] general delimiters =
"GENERAL" ,
ps [65] +,
"SGMLREF" ,
(ps [65] +,
name [55] ,
ps [65] +,
*parameter literal [66])**

[192] short reference delimiters =
"SHORTREF" ,
ps [65] +,
("SGMLREF" |
"NONE"),
(ps [65] +,
*parameter literal [66])**

Beispiel für eine Delimiter-Spezifikation :

DELIM
GENERAL SGMLREF
MDO "[:"
SHORTREF NONE
"("
")"

Diese Spezifikation hat zur Folge, daß zunächst alle vom Standard vorgeschlagenen Zeichen für die Delimiter gewählt werden (signalisiert über das Schlüsselwort SGMLREF). Anschließend wird für den MDO-Delimiter spezifiziert, daß er durch "[:" statt "<!" dargestellt wird. Das Schlüsselwort NONE hinter SHORTREF signalisiert, daß keine Zeichen zur Bildung von Kurzreferenzen erlaubt sind, außer den in der folgenden Liste angegebenen Zeichen.

Wäre das Schlüsselwort SGMLREF statt NONE angegeben worden, so wären alle vom Standard vorgeschlagenen Zeichen zusätzlich zu den angeführten Zeichen zur Bildung von Kurzreferenzen erlaubt gewesen.

7.4.6 Reserved Names

Der Abschnitt "Reserved Name Use" erlaubt es, eine Teilmenge der von SGML verwendeten Schlüsselwörter (syntactic literals) umzubenennen. Es besteht die Möglichkeit, genau die Schlüsselwörter umzubennen, die innerhalb des "prologs" und der "document instance" verwendet werden können.

Die Definition von Schlüsselwörtern erfolgt nach Regel[193].

[193] reserved name use =
"NAMES" ,
ps [65] +,
"SGMLREF" ,
(ps [65] +,
name [55] ,
ps [65] +,
name [55])*

Folgende Schlüsselwörter können umbenannt werden:

ANY	ATTLIST	CDATA	CONREF
CURRENT	DEFAULT	DOCTYPE	ELEMENT
EMPTY	ENDTAG	ENTITIES	ENTITY
FIXED	ID	IDLINK	IDREF
IDREFS	IGNORE	IMPLIED	INCLUDE
INITIAL	LINK	LINKTYPE	MS
NAME	NAMES	NDATA	NMTOKEN
NMTOKENS	MD	NOTATION	NUMBER
NUMBERS	NUTOKEN	NUTOKENS	O
PCDATA	PI	POSTLINK	PUBLIC
RCDATA	REQUIRED	RESTORE	SDATA
SHORTREF	SIMPLE	STARTTAG	SUBDOC
SYSTEM	TEMP	USELINK	USEMAP

Beispiel für eine Umbenennung :

NAMES SGMLREF
DOCTYPE DTD

Diese Umdefinition legt fest, daß der "prolog" nicht mehr mit "<!DOCTPYE" sondern mit "<!DTD" eingeleitet wird.

Es ist darauf zu achten, daß die Namen entsprechend den Naming Rules der "reference concrete syntax" gebildet werden. Ein Name darf nicht zweimal als Schlüsselwort definiert werden.

7.4.7 Quantity Set

Die Spezifikation eines Quantity Sets bietet die Möglichkeit Restriktionen bezüglich der in SGML definierbaren Objekte festzulegen. Ein Quantity Set wird entsprechend Regel[194] angegeben.

Beispiel :

```
QUANTITY SGMLREF
        NAMELEN         32
```

Dieses Beispiel legt den Wert für die maximale Länge eines Namens auf 32 Zeichen fest (Standard 8). Alle anderen Werte werden vom Standard übernommen. Weitere änderbare Werte sind :

Name	Standardwert	Beschreibung
ATTCNT	40	Namen in der Namensliste einer Attribut-Deklaration
ATTSPLEN	960	Normalisierte Länge der Attributspezifikation in einer Startmarkierung
BSEQLEN	960	Länge einer Folge von Zwischenraumzeichen in einer Kurzreferenz
DTAGLEN	16	Länge einer Datenmarkierung
DTEMPLEN	16	Länge eines Datenmarkierungsmusters
ENTLVL	16	Schachtelungstiefe für eingebundene Entitäten
GRPCNT	32	Bestandteile der Gruppe eines Inhaltsmodells
GRPGTCNT	96	Bestandteile eines Inhaltsmodells
GRPLVL	16	Schachtelungstiefe für Gruppen in Inhaltsmodellen
LITLEN	240	Länge eines Literals oder Attributwerts
NAMELEN	8	Länge von Namen und ähnlichen Objekten
NORMSEP	2	Parameter bei der Berechnung der normalisierten Länge eines Attributwertes
PILEN	240	Länge einer Verarbeitungsanweisung
TAGLEN	960	Länge einer Startmarkierung
TAGLVL	24	Schachtelungstiefe für Elemente

7.5 Einbinden einer Public Syntax

Soll eine Public Syntax eingebunden werden, so muß dies entsprechend Regel[183] erfolgen.

[183] public concrete syntax =

"PUBLIC" ,

ps [65] +,

public identifier [74] ,

(ps [65] +,

"SWITCHES" ,

(ps [65] +,

character number [64] ,

ps [65] +,

character number [64])+)?

Beispiel für das Einbinden einer Public Syntax:

SYNTAX
PUBLIC "ISO 8879:1986//SYNTAX Reference//EN"
SWITCHES 60 123 123 60 62 125 125 62

Über den public identifier wird die "reference concrete syntax" ausgewählt. Mit der SWITCHES-Anweisung werden dann spezifische Anpassungen vorgenommen.

In diesem Fall werden die Zeichen "<" und ">" mit den Zeichen "{" und "}" vertauscht, was zur Folge hat, das bspw. der mdo nicht mehr durch "<!" sondern durch "{!" dargestellt wird.

Die Angabe einer SWITCHES-Anweisung ist optional.

7.6 Die Spezifikation von Features

Der Teil "Feature Use" dient dazu, einer SGML-Anwendung mitzuteilen, welche Features innerhalb eines SGML-Dokuments verwendet werden dürfen. Dadurch besteht die Möglichkeit, Features, die von einer SGML-Anwendung nicht unterstützt werden, zu verbieten.

Die "Feature Use"-Deklaration erfolgt nach Regel[195].

[195] feature use =

"FEATURES" ,

ps [65] +,

markup minimization features [196] ,

ps [65] +,

link type features [197] ,

ps [65] +,

other features [198]

Aus dieser Regel ist zu erkennen, daß die Features in drei Gruppen zerfallen.
Die "Minimization Features" sind Regel[196] zu entnehmen.

[196] markup minimization features =
"MINIMIZE" ,
ps [65] +,
"DATATAG" ,
ps [65] +,
("NO" |
"YES"),
ps [65] +,
"OMITTAG" ,
ps [65] +,
("NO" |
"YES"),
ps [65] +,
"RANK" ,
ps [65] +,
("NO" |
"YES"),
ps [65] +,
"SHORTTAG" ,
ps [65] +,
("NO" |
"YES")

Die Bedeutung der einzelnen Features sind :

DATATAG — Ähnlich zu der Nutzung von short references.
In frühen Entwicklungsphasen der Sprache entworfen, und kaum noch unterstützt.

OMITTAG — Dieses Feature zeigt an, ob Start- und Endemarkierungen innerhalb der Dokumentinstanz weggelassen werden können, obwohl ihr vorhanden sein von der DTD gefordert wird.

SHORTTAG — Während bei der Omit-Tag-Minimization Start- und Endemarkierungen komplett weggelassen werden können, erlaubt das Feature SHORTTAG, daß bei der Angabe von Tags Abkürzungen erlaubt sind.
Short-Tag-Minimization manifestiert sich in

- unclosed short tag
- empty start tag
- empty end tag
- attribute minimization

RANK — In frühen Entwicklungsphasen der Sprache entworfen, und

kaum noch unterstützt.

Die "Link Type Features" entsprechen Regel[197].

```
[197] link type features =
    "LINK" ,
    ps [65] +,
    "SIMPLE" ,
    ps [65] +,
    ( "NO" |
    "YES" ,
    ps [65] +,
    number [56] ),
    ps [65] +,
    "IMPLICIT" ,
    ps [65] +,
    ( "NO" |
    "YES" ),
    ps [65] +,
    "EXPLICIT" ,
    ps [65] +,
    ( "NO" |
    ( "YES" ,
    ps [65] +,
    number [56] ))
```

Auf die Bedeutung der einzelnen Features soll an dieser Stelle nicht weiter eingegangen werden. Es sei nur soviel erwähnt, daß sich diese Features auf LINK-Attribute beziehen, die es ermöglichen, Elemente eines Dokuments mit Formatierungseigenschaften zu verknüpfen.

Die Gruppe "Other Features" umfaßt Features entsprechend Regel[198].

```
[198] other features =
    "OTHER" ,
    ps [65] +,
    "CONCUR" ,
    ps [65] +,
    ( "NO" |
    ( "YES" ,
    ps [65] +,
    number [56] )),
    ps [65] +,
    "SUBDOC" ,
    ps [65] +,
    ( "NO" |
    ( "YES" ,
    ps [65] +,
    number [56] )),
    ps [65] +,
    "FORMAL" ,
    ps [65] +,
    ( "NO" |
    "YES" )
```

Die Bedeutung der einzelnen Features sind :

CONCUR	Dieses Feature legt fest, ob konkurrente Dokumentstrukturen innerhalb eines Dokuments unterstützt werden.

Innerhalb des Dokuments spiegeln sich konkurrente Strukturen durch die Definition mehrerer DTDs wider.
Durch konkurrente Strukturen können sich unterschiedliche Sichtweisen auf ein und dasselbe Dokument definiert werden.
Die innerhalb der Dokumentinstanz verwendeten Tags und Referenzen müssen dann allerdings der jeweiligen DTD, in der sie definiert sind, zugeordnet werde.
Dies geschieht dadurch, daß der Name der DTD dem Tag voran-gestellt wird.

SUBDOC — Über dieses Feature wird signalisiert, ob ein SGML-Dokument aus mehreren Subdokumenten bestehen kann. Ein Subdokument ist dabei wiederum ein eigenständiges SGML-Dokument, d.h. besitzt eine eigene DTD und einen eigenen Inhalt.
Die Subdokumente besitzen keine eigene SGML-Declaration.
Die SGML-Declaration des Hauptdokuments gilt auch für alle Subdokumente.
Wird das Feature SUBDOC auf YES gesetzt, so gibt die folgende Zahl die maximale Anzahl von Subdokumenten an.

FORMAL — Dieses Feature legt fest, ob Public identifier formale Public identifier sein müssen.
Der Aufbau eines "formal public identifier" wird durch Regel[79] der abstract syntax festgelegt.

7.7 Beispiel

Das folgende Beispiel stellt eine übliche SGML Declaration dar. In den meisten SGML Anwendungen wird bei fehlender SGML Declaration in einem Dokument, eine solche Declaration als Standard angenommen.

```
<! SGML "ISO 8879:1986
        CHARSET
                BASESET "ISO 646-1983//CHARSET International Reference
                                Version (IRV)//ESC 2/5 4/0"
                DESCSET
                        0       9       UNUSED
                        9       2       9
```

```
        11      2       UNUSED
        13      1       13
        14      18      UNUSED
        32      95      32
        127     1       UNUSED

CAPACITY
        SGMLREF
        TOTALCAP 35000

SCOPE
        DOCUMENT

-- hier beginnt die reference concrete syntax --

SYNTAX
        SHUNCHAR
                CONTROLS
                0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
                18 19 20 21 22 23 24 25 26 27 28 29 30 31
                127 255
        BASESET "ISO 646-1983//CHARSET International Reference
                        Version (IRV)//ESC 2/5 4/0"
        DESCSET
                0 128 0
        FUNCTION
                RE      13
                RS      10
                SPACE 32
                TAB     SEPCHAR      9
        NAMING
                LCNMSTRT   ""
                UCNMSTRT   ""
                LCNMCHAR  "-."
                UCNMCHAR  "-."

                NAMECASE
```

```
                    GENERAL YES
                    ENTITY      NO
          DELIM
                GENERAL   SGMLREF
                SHORTREF  SGMLREF
                NAMES           SGMLREF

          QUANTITY        SGMLREF

-- hier endet die reference concrete syntax --

FEATURES
          MINIMIZE
                DATATAG         NO
                OMITTAG         YES
                RANK      NO
                SHORTTAG  YES
          LINK
                SIMPLE          NO
                IMPLICIT  NO
                EXPLICIT  NO
          OTHER
                CONCUR          NO
                SUBDOC          NO
                FORMAL          YES
                APPINFO         NONE

>
```

8 Weitere Möglichkeiten von SGML

8.1 Konkurrente Dokumentstrukturen

Entsprechende Regel[7] der abstract syntax gestattet es SGML, daß mehrere DTDs angegeben werden können.

[7] prolog =
other prolog [8] *,
base document type declaration [9] ,
(document type declaration [110] **|**
other prolog [8])*,
(link type declaration [154] **|**
other prolog [8])*

Werden tatsächlich mehrere DTDs spezifiziert, so spricht man von konkurrenten Dokumentstrukturen. Die Verwendung konkurrenter Dokumentstrukturen ist allerdings nur dann möglich, wenn innerhalb der SGML Declaration das Feature CONCUR mit YES belegt wurde.

Konkurrente Strukturen erlauben es, verschiedene Sichtweisen in einem Dokument zu codieren. Wird bspw. ein Bericht verfaßt, der sowohl als wissenschaftlicher Report innerhalb eines Magazins, als auch als Kapitel eines Buches erscheinen soll, so ist es sinnvoll, diesen Text entsprechend zweier Dokumentklassen zu strukturieren.

Beispiel:

```
<! DOCTYPE Report [
<! ELEMENT Report  - - (Title,Data) >
<! ELEMENT Title - - (#PCDATA) >
<! ELEMENT Data - - (#PCDATA) >
]>
<! DOCTYPE Book [
<! ELEMENT Book    - - (Chapter,Section+) >
<! ELEMENT Chapter - - (#PCDATA) >
<! ELEMENT Section - - (#PCDATA) >
]>
<(Report) Report> <(Book) Chapter)>
        <(Report) Title><(Book) Chapter>
        Einführung in SGML
```

</(Report) Title></(Book) Chapter>

<(Report) Data>

<(Book) Section> blablablabla ... </(Book) Section>

<(Book) Section> blablablabla ... </(Book) Section>

<(Book) Section> blablablabla ... </(Book) Section>

<(Report) Data>

</(Report) Report> </(Book) Chapter)>

In obigem Beispiel ist zu erkennen, daß der Inhalt des Reports nicht weiter unterteilt wird. Wird dieses Dokument jedoch als Buch verarbeitet, so wird der Inhalt noch in weitere Sections aufgeteilt.

8.2 LINK-Attribute

Die Verwendung von LINK-Attributen erlaubt es, Elementen Formatierungseigenschaften zuzuweisen, und widerspricht somit eigentlich dem Ansatz des Descriptive Markup. Nach Angabe einer (oder mehrerer) DTD besteht die Möglichkeit eine Link Type Declaration anzugeben.

[7] prolog =

other prolog [8] *,

base document type declaration [9] ,

(document type declaration [110] |

other prolog [8])*,

(link type declaration [154] |

other prolog [8])*

Innerhalb der Link Type Declaration wird eine Menge von LINK-Attributen definiert, denen die Werte der Formatierungseigenschaften zugewiesen werden können. Die Menge der Attribute stellt somit eine Formatvorlage für eine konkrete Formatierung dar. Durch die Definition von Link-Sets können einer Formatvorlage konkrete Werte zugewiesen werden, und somit ein konkreter Stil vorgeben werden. Dieser Stil kann dann auf Elemente der DTD angewandt werden.

Eine Link Type Declaration hat folgende Syntax :

[154] link type declaration =

mdo , <!

"LINKTYPE" ,

ps [65] +,

link type name [155] ,

ps [65] +,

(simple link specification [156] |
implicit link specification [157] |
explicit link specification [158]),
(ps [65] +,
external identifier [73])?,
(ps [65] +,
dso , [
link type declaration subset [161] ,
dsc)?,]
ps [65] *,
mdc >

Nach dem Schlüsselwort LINKTYPE folgt zunächst der Name, mit dem die Formatvorlage bezeichnet werden soll (ein Name ist wiederum entsprechend Regel[55] anzugeben).

Das Link Type Declaration Subset gestattet es, konkrete Stile zu definieren.

[161] link type declaration subset =
(link attribute set [162] |
link set declaration [163])*,
ID link set declaration [168.1] ?,
(link attribute set [162] |
link set declaration [163])*

[162] link attribute set =
(attribute definition list declaration [141] |
entity set [113])*

[163] link set declaration =
mdo , <!
"LINK" ,
ps [65] +,
link set name [164] ,
(ps [65] +,
link rule [163.1]),
ps [65] *,
mdc >

Einer Formatvorlagenspezifikation können mehrere konkrete Stile folgen, wobei jeder Stil durch einen Namen identifiziert wird.

Beispiel:

```
<!LINKTYPE myformat text #IMPLIED [
<! ATTLIST    (chapter | section)
        leftSpace CDATA
        rightSpace CDATA
>
<! LINK #INITIAL
```

```
        chapter [leftSpace = 5 rightSpace =5]
        section [leftSpace = 7 rigthSpace = 7]
>
<!LINK format1
        chapter [leftSpace = 3 rightSpace = 3]
>
<!LINK format2
        section [leftSpace = 9 rightSpace = 9]
>
]
```

Dieses Beispiel definiert eine Formatvorlage mit dem Name „myformat“, die innerhalb der DTD „text“ auf die Elemente „chapter“ und „section“ angewandt werden soll. Das Attribut „leftspace“ soll dabei den Abstand zum linken Rand, und das Attribut „rightSpace“ den Abstand zum rechten Rand darstellen. Über das folgende Link-Set wird die Formatvorlage mit konkreten Werten gefüllt. (Das Schlüsselwort INITIAL signalisiert, daß dies die Voreinstellungen sein sollen.)

Innerhalb der Dokumentinstanz kann allerdings auch einer der anderen definierten Stile verwendet werden.

Beispiel:

```
...
<chapter> <!USELINK format2 myformat>
<section>
        blablabla
</section>
<section>
        blablabla
</section>
</chapter>
```

Die USELINK-Deklaration hat zur Folge, daß die folgenden Sections den Stil format2 zugewiesen bekommen, als nicht die Voreinstellung leftSpace = 7 und rightSpace = 7, sondern die Werte leftSpace = 9 und rightSpace = 9.

Eine USELINK-Deklaration gilt bis zum Ende des aktuellen Elements (in diesem Fall also bis zum Tag </chapter>. Alle nach diesem Tag folgenden Sections werden wieder mit der Voreinstellung verbunden.

Die Angabe einer USELINK-Declaration erfolgt entsprechend Regel [169].

[169] link set use declaration =

mdo ,

"USELINK" ,

ps [65] +,

link set specification [170] ,

ps [65] +,

link type name [155] ,

ps [65] *,

mdc

Eine USELINK-Deklaration kann in der Dokumentinstanz innerhalb eines Other Content angegeben werden (siehe *Document Instance Set Content Other Content*).

9 Literaturangaben

B.Travis, D.Waldt : “The SGML Implementation Guide”, Springer, 1996
W.Rieger : “SGML für die Praxis”, Springer, 1995
http://www.geocities.com/Athens/2694/sgml.html [13.01.1999]
http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG [13.01.1999]
http://www.tiac.net/users/bingham/sgmlsyn/index.htm [13.01.1999]
http://www.oasis-open.org/cover/sgml-xml.html [13.01.1999]
http://www.omnimark.com/white/ [13.01.1999]
http://www.cm.spyglass.com/doc/#sgml [13.01.1999]
http://www.pineapplesoft.com/reports/sgml/index.html [13.01.1999]
http://www.oreilly.com/people/staff/crism/sgmldefs.html [13.01.1999]

„HTML I –Beschreibung des Sprachumfangs anhand von HTML 2.0“

Anton Stoll

FB Informatik, Uni Dortmund

astoll00@marvin.informatik.uni-dortmund.de

Inhalt:

1 Einführung

Seit 1993, nachdem zwei Physiker Tim Berners-Lee und Don Connoly eine Formatiersprache und ein Verteilungssystem für die Erzeugung von integrierten elektronischen Multimedia-Dokumenten veröffentlicht haben, hat sich HTML-HyperText Markup Language sehr schnell verbreitet und ist heute eine von W3-Konsortium standardisierte Sprachen des WWW. Diese leicht zu erlernende Sprache beschreibt den Aufbau eines Dokuments mit Abschnitten, Überschriften, Auszeichnungen (z.B. Fett, Kursiv), Verweisen zu anderen Dokumenten usw. HTML-Dokumente enthalten nur ASCII-Zeichen und lassen sich rechner- und plattformunabhängig bearbeiten und anzeigen, was es im mehrsprachigen Internet sehr wichtig ist.

HTML definiert die Syntax und Anordnung von besonderen, im Text verankerten Anweisungen (Marken), die vom Browser nicht dargestellt, sondern zur Darstellung des Dokumentinhalts mit Text, Bildern und anderen Hilfsmedien verwendet werden.

2 Das HTML-Skelett

HTML ist eine eingebettete Sprache- man fügt die Sprachanweisungen oder Marken in das selbe Dokumenten ein. Der Browser entscheidet anhand der Marken, wie er den Inhalt eines HTML-Dokuments darstellen oder behandeln soll. Die meisten Marken definieren und betreffen nur einen bestimmten Bereich des HTML-Dokuments. Der Bereich beginnt dort , wo die Marke und ihre Attribute zum ersten Mal im Quelldokument erscheinen (wird auch als Anfangsmarke bezeichnet) und endet bei der entsprechenden Endmarke. Anfangsmarke wird im eckigen Klammern angeschlossen- <*Marke*> und Endmarke ergibt sich aus der Anfangsmarke mit einem Schrägstrich vor dem Markennamen- </*Marke*>. Es gibt auch solche Marken, die keine Endmarke brauchen. Man kann auch solche Endmarken weglassen, die aus Konzept hervorgehen, was es aber nicht bei allen Browsern funktioniert.

Jedes HTML-Dokument fängt mit einer <*html*>-Marke und endet mit einer </*html*>-Marke. Obwohl HTML-Standard verlangt diese Marken, können die meisten Browser HTML-Code auch ohne solche finden und darstellen.

Innerhalb dieser Marken haben alle HTML-Dokumente 2 Grundstrukturen: einen Header und einen Dokumentenkörper oder Body.

<*html*>
    <*head*>
        .....
    </*head*>
<*body*>
.....
</*body*>
</*html*>

In den Header kommen Informationen über das Dokument, und das, was im Browserfenster erscheinen soll, kommt in den Body-Block. Im Header-Block werden der Dokumenttitel sowie verschiedene Verwaltungsinformationen abgelegt. Hier lassen sich Marken wie *base, isindex, link, meta* und *title* einfügen(diese Marken werden später ausführlich beschrieben). Alle anderen Marken kommen nur im Body-Block vor.Bei den meisten Dokumenten ist jedoch das wichtigste Element des Headers der Titel. Er wird zwischen den <*title*> und </*title*> eingefügt und normalerweise am oberen Rand des Browserfensters dargestellt.

Wie bei allen Programmiersprachen hat auch HTML Kommentare. Kein Teil des Kommentars, auch nicht der Inhalt, der sich zwischen der speziellen Anfangsmarke *"<!--"* und der Endmarke *"-->"* befindet, wird im Browserfenster dargestellt.

Man kann in HTML alle Zeichen des ASCII-Zeichensatzes über eine besondere Codierung (Character Entity) in den Text einfügen. Diese Zeichenbeschreibung beginnt mit & und endet mit ; . Z.B. Größer-Zeichen wird so dargestellt- *&gt*;

Ein Umlaut- *¨* Buchstabe A- *A*

2.1 Beispiel

```
<html>
<head>
   <title>Das erste HTML-Dokument</title>
</head>
<body>

<!--Hier fängt der Body-Block an-->
 Hier kann beliebiger  Text stehen, der im Browser Fenster dargestellt
 wird. Auch &gt; und Umlaute k&ouml;nnen im Text enthalten sein.

</body>
</html>
```

3 Textpositionierung und Schriftdarstellung

Um den Textfluß zu steuern und ihn in gewünschte Blocke zu teilen, gibt es in HTML eine Reihe von inhaltsorientierten und darstellungsorientierten Formatmarken.

Ein langer Textfluß, der nicht durch Titel und andere Überschriften unterbrochen wird, kann nur schwer gelesen werden, deshalb gibt es in HTML 6 Überschriftsebenen, mit denen sich ein Dokument gut strukturieren läßt.

Die 6 Überschriftenmarken (*h1,..,h6*) weisen einer Überschrift im Dokument eine Hierarchie von der höchsten h1 bis zur niedrigsten h6 zu . Der Browser stellt eine Überschrift vielleicht zentriert, fett, vergrößert oder in einer anderen Farbe dar. Normalerweise wird <*h1*>-Marke für Dokumenttitel, <*h2*> für Abschnittsüberschriften usw. verwendet.

Die <*p*>-Marke signalisiert den Beginn eines neuen Absatzes (</*p*> wird normalerweise weggelassen). Trifft ein Browser auf eine <*p*>-Marke, so fügt er üblicherweise eine Leerzeile und etwas zusätzlichen Durchschuß ein und beginnt dann den neuen Absatz. Man braucht sich beim Schreiben eines HTML-Dokuments nicht um Zeilenlänge, Wortumbruch und Zeilenumbruch kümmern. Der Browser nimmt jede beliebige Wort- und Bildfolge und baut daraus einen Absatz.

Normalerweise wird HTML-Dokument ohne konkrete Formatierungsvorgaben wiedergeben. Die Marken weisen die Struktur und Semantik eines Dokuments aus. Zeilenumbrüche, Zeichenabstand, Zeilendurchschuß usw. bleiben dem Browser überlassen. Manchmal aber wünscht man sich da mehr Flexibilität. Es gibt auch dafür HTML-Marken.

Die <*br*>-Marke unterbricht das normale Zeilenfüllungs- und Absatzumbruchmuster. Sie hat keine Endmarke und bezeichnet eigentlich nur einen Punkt im Textfluß, an dem eine Zeile beginnen muß. Dieser Effekt ist nützlich, wenn man Addresse, Liedtexte oder Gedichte mit festen Zeilenumbrüchen formatiert.

Die <*pre*>-Marke und ihre Endmarke </*pre*> definieren ein Textsegment, in dem der Browser den Text mit genau den Zeichenmerkmalen und Zeilenumbrüchen darstellt, die im HTML-Quelldokument festgelegt sind. Der Browser stellt den Text zwischen <*pre*> und </*pre*> in einer Nichtproportionalschrift dar. Andere Marken können im <*pre*>-Block genau so wie in jedem anderen Teil HTML-Dokuments eingefügt werden. Die <*pre*>-Marke hat ein optionales Attribut namens *width*, das die Anzahl an Zeichen pro Zeile innerhalb des <*pre*>-Blocks bestimmt. Der Browser kann anhand dieses Wertes eine Schriftart oder -größe wählen, bei der die angegebene Zeichenzahl in eine Zeile des <*pre*>-Blocks paßt.

Die <*plaintext*>-Marke erlaubt es, den Text nach dieser Marke so darzustellen, wie er geschrieben wurde, wobei keine weiteren Marken erlaubt sind- auch keine </*plaintext*>-Marke. Der Text wird in einer Nichtproportionalschrift dargestellt.

Die <*blockquote*>-Marke- definiert ein Zitat. Der Text innerhalb einer <*blockquote*>-Marke und ihrer Endmarke </*blockquote*> hebt sich vom normalen Dokumenttext meistens durch einen leicht eingerückten linken und rechten Rand und manchmal auch durch kursive Schrift ab.

Auch für dir Darstellung von Adressen gibt es eine eigene <*adress*>- Marke. Das mag extravagant erscheinen- Adressen haben eigentlich keine besonderen Formatierungsmerkmale. Der Browser weiß aber dann, daß er es mit einer Adresse zu tun hat und kann sie entsprechend benutzen, z.B. ein Online-Adreßverzeichnis erstellen. Man sollte immer die Adresse eingeben, um Lesern die Möglichkeit zu geben, Kommentare und Anmerkungen zu machen.

Inhaltsorientierte Formatmarken ändern das Erscheinungsbild von Text gemäß der Bedeutung oder Verwendung des jeweiligen Textes. Es ist dann leichter Dokumente automatisch zu durchsuchen.

Die <*cite*>-Marke weist normalerweise auf eine bibliographische Angabe wie einen Buch- oder Zeitschriftentitel hin. Im allgemeinen wird der enthaltene Text kursiv dargestellt.

Die <*code*>-Marke stellt den enthaltenen Text in einer schreibmaschinenähnlichen Nichtproportionalschrift dar. Man sollte die <*code*>-Marke nur für Computer-Quellcode verwenden, weil es die Browser der Zukunft diesen Quellcode vielleicht auf eine andere Weise benutzen werden.

Die <*em*>-Marke läßt den Browser den in ihr enthaltenen Text betont darstellen. Bei fast allen Browsern bedeutet das kursiv. Um es wirklich etwas zu betonen, muß man möglichst sparsam mit <*em*>- Marke umzugehen. Auch diese Marke könnte in Zukunft wahrscheinlich anders dargestellt werden.

Die <*kbd*>-Marke benutzt man, um auf Text hinzuweisen, der auf einer Tastatur eingegebin wird. Die <*kbd*>-Marke wird am häufigsten in Computer-Dokumentationen und -Handbüchern verwendet.

Die <*samp*>-Marke zeigt eine Folge von Druckbuchtaben an, die keine andere Bedeutung für den Benutzer haben soll. Diese Marke wird am häufigsten verwendet, wenn eine Zeichenfolge aus dem normalen Kontext herausgenommen wird oder bei der Darstellung von Beispielen.

Wie die <*em*>-Marke sorgt auch die <*strong*>-Marke für eine Hervorhebung des entsprechenden Texts, nur wirkt sie stärker. Ein Browser stellt die <*strong*>-Marke anders dar als die <*em*>-Marke, normalerweise die eine fett und die andere kursiv.

Die <*var*>-Marke zeigt einen Variablennamen oder einen Benutzerwert an. Die Marke wird häufig zusammen mit der <*code*>-Marke und der <*pre*>-Marke für spezielle Abschnitte von Programm-Codes und ähnlichem verwendet. Text, der mit der <*var*>-Marke ausgezeichnet ist, wird normalerweise in Nichtproportionalschrift dargestellt. Wie bei den anderen Marken, die mit Computer-Programmen und -Dokumenten zu tun haben, erleichtert die <*var*>-Marke dem Benutzer nicht nur das Verständnis und die Durchsicht einer Dokumentation, sondern wird eventuell irgendwann einmal von einem automatisierten System für die Zusammenstellung von bestimmten Informationen und nützlichen Parametern verwendet. Wieder gilt, je mehr semantische Informationen dem Browser zur Verfügung gestellt werden, desto besser kann der Browser dem Benutzer die Dokumentinhalte präsentieren.

HTML 2.0 sieht 3 darstellungsorientierte Formate vor, und zwar für fetten, kursiven und nichtproportionalen Text. Alle darstellungsorientierten Formatmarken benötigen zwingend ihre jeweiligen Endmarken.

Die <*b*>-Marke ist das darstellungsorientierte Äquivalent zur inhaltsorientierten -Marke, nur ohne die erweiterte Bedeutung. Die <*b*>-Marke schreibt für die zwischen ihr und der -Endmarke enthaltenen Zeichen den Fettdruck vor.

Die <*i*>-Marke entspricht der inhaltsorientierten <*em*>-Marke. Sie und ihre und ihre unbedingt erforderliche Endmarke </*i*> weisen den Browser dazu an, den entsprechenden Text kursiv oder schräg darzustellen.

Wie die <*code*>-Marke und die <*kbd*>-Marke weisen die <*tt*>-Marke und ihre Endmarke </*tt*> den Browser an, den Text in Nichtproportionalschrift darzustellen. Bei den Browsern, die sowieso schon eine solche Schrift verwenden, hat diese Marke bei der Darstellung vielleicht keine sichtbar Wirkung.

3.1 Beispiel

<*html*>

<*head*>

<*title*>Textpositionierung und Schriftendarstellung- Beispiel 1**</***title*>

</*head*>

<*body*>

<*h2*>Textpositionierung und Schriftendarstellung</*h2*>

<!-- Marken (h1,..h6) -->

Marken h1 bis h6-für Überschritdarstellung:<*br*>
<*h1*>So sieht Überschriftsebene 1 aus</*h1*>
<*h2*>So sieht Überschriftsebene 2 aus</*h2*>
<*h3*>So sieht Überschriftsebene 3 aus</*h3*>
<*h4*>So sieht Überschriftsebene 4 aus</*h4*>
<*h5*>So sieht Überschriftsebene 5 aus</*h5*>
<*h6*>So sieht Überschriftsebene 6 aus</*h6*>

<!-- p-Marke -->

<*p*>
Betrachten wir jetzt eine p-Marke, die uns erlaubt Absäte zu machen. Dies ist schon der zweite Absatz nach den Überschriftsebenen, wie wir sehen.
<*p*>
Die Marke signalisiert den Anfang des dritten Absatzes.

<!-- br-Marke -->

<*p*>Die br-Marke fügt einen Zeilenumbruch<*br*>in einen Textfluß ein.
<*br*>33333 Münsterland
Kappuze Str.<*br*>Mustermann

<!-- pre-Marke -->

<*p*> pre-Marke stellt einen Textblock ohne Formatierung dar:
<*pre*>so wie
es ge-
schrieben ist.

Cola-20
Kaffe-35
Wasser-10</*pre*>

<!-- blockquote-Marke -->

<*p*>Die blockquote-Marke definiert ein Zitat, wie folgt:<*blockquote*>Ich bin ein Berliner</*blockquote*>Wie wir sehen, wird das Zitat durch Einrückung der rechten und linken Ränder hervorgehoben.

<!-- adress-Marke -->

<*p*>Obwohl eine adress-Marke fast immer am Ende eines Dokuments steht, benutzen wir sie an dieser Stelle: <*adress*><*br*>Dortmund<*br*>Universität<*br*></*adress*>

<!-- Inhaltsorientierte Marken -->

<*h2*>Inhaltsorientierte Marken</*h2*>

<!-- cite-Marke -->

<*p*>cite-Marke wird normalerweise bei der <*cite*>Literaturangaben</*cite*> benutzt

<!-- code-Marke -->

<*p*>Die nächste (inhaltsorientierte) code-Marke ermöglicht es , den Computer-Quellcode darzustellen:

<*br*><*code*> <*pre*> main()

{

int a,b;

cin<< a<< b;

if (a>b) cout>>"hello ,world"

}

</*pre*></*code*>

<!-- em-Marke -->

<*p*>em-Marke <*em*>betont den Text</*em*>

<!-- kbd-Marke -->

<*p*>Mit der nächsten Marke wird der Text , der auf der Tastatur angegeben werden soll, dargestellt. Z.b.: Drücken Sie <*kbd*>Enter</*kbd*> Taste.</*br*> Wie wir alle richtig vermutet haben, ist es die kbd-Marke.

<!-- samp-Marke -->

<*p*>samp-Marke benutzt man selten. Sie wird bei der <*samp*>Darstellung von Beispielen</*samp*> oder nichts weiter für den Benutzer bedeutenden <*samp*>Zeichenfolgen</*samp*> benutzt.

<!-- strong-Marke -->

<*p*>strong-Marke, wie der Name schon sagt, sorgt für die noch stärkere als bei der em-Marke <*strong*>Hervorhebung</*strong*> des beinhalteten Textes.

<!-- var-Marke -->

<p>Die letzte inhaltsorientierte Marke: var-Marke zeigt einen <var>Variblennamen</var> an, wie im Befehl:

cp <var>quelldatei</var> <var>zeildatei</var>

<!--Darstellungsorientierte Marken -->

<h2>Darstellungsorientierte Marken</h2>

<p>Die nächsten 3 Marken werden genauso dargestellt, wie auch entsprechende inhaltsorientierte Marken. Sie haben aber keine erweiterte Bedeutung.

<!-- b-Marke -->

<p>Wie auch strong-Marke stellt die b-Marke den Text in Fettschrift dar.

<!-- i-Marke -->

<p>Die i-Marke entspricht der em-Marke. Man benutzt sie, um den Text normalerweise kursiv hervorzuheben.

<!-- tt-Marke -->

<p>Und die letzte in diesem Beispiel tt-Marke (genauso wie auch code oder kbd-Marke) zeigt den Text in <tt>Nichtproportionalschrift</tt> an.

</body>

</html>

4 Hyperlinks

Eine Grundeigenschaft von Hypertext ist, daß Dokumente durch Hyperlinks verknüpft werden können; eine solche Verknüpfung kann auf eine andere Stelle im aktuellen Dokument oder auf ein anderes Dokument im Internet verweisen. Aus den Dokumenten wird so ein kompliziert verwobenes Netz (auf englisch Web, daher World Wide Web)von Informationen. Durch einen Mausklick auf dem Browser-Bildschirm wählen die Leser einen interessanten Eintrag an und springen gleichzeitig dorthin, egal, ob sich das Ziel im aktuellen Dokument oder in einem Dokument auf der anderen Seite der Erde befindet.

Jedes Dokument im World Wide Web hat eine eigene Adresse, die auch als Uniform Resource Locator (URL) bezeichnet wird. Da durch URLs fast jede Ressource im Internet angesprochen werden kann, gibt es verschiedene Arten vom URLs. Im Prinzip verfügen alle über die gleiche Syntax:

Schema: schema-spezifischer_teil

Das Schema beschreibt die Art von Objekt, auf die sich der URL bezieht. Der schema-spezifischer_teil hängt vom jeweiligen Schema ab. Hier sind ein paar Beispiele: ***Fehler! Hyperlink-Referenz ungültig.*** ; ***Fehler! Hyperlink-Referenz ungültig.***

Der URL ist entweder absolut, wobei jeder Bestandteil des URLs direkt vom Autor beigesteuert wird, oder relativ, wobei gewisse Elemente des URLs ausgelassen und vom Browser beigesteuert werden.

Die HTML-Marke <*a*> wird dazu verwendet, Verbindungen zu anderen Dokumenten herzustellen sowie die Anker für Fragment-Bezeichner innerhalb von Dokumenten zu erzeugen. Diese Marke wird meistens mit dem Attribut *href* benutzt, um einen Hyperlink zu erstellen, der auf eine andere Stelle des Dokuments oder auf ein anderes Dokument verweist. In diesen Fällen ist das aktuelle Dokument die Quelle der Verbindung; der Wert des *href*-Attributs, ein URL, ist das Ziel. Wenn der Benutzer den Inhalt der <*a*>-Marke anwählt, lädt der Browser das Dokument, das vom URL des *href*-Attributes angegeben wird, und stellt es dar.

Das *methods*-Attribut weist den Browser an, das Dokument, auf das durch *href*-Attribut verwiesen wird, auf besondere Weise zu bearbeiten. Der zu *methods* gehörige Wert ist ein Name, der eine bestimmte Art von Dokumentenverarbeitung darstellt(meist der Name eines Programms). Da solche Namen browsersabhängig sind, wird das *methods*-Attribut selten verwendet.

Um in einem Dokument einen Fragment-Bezeichner zu erstellen, braucht man das *name*-Attribut. Ab der Erstellung wird dieser zum potentiellen Ziel eines Hyperlinks. Der Wert des *name*-Attributs kann eine beliebige, in Anführungszeichen eingeschlossene Zeichenkette sein. Der Inhalt der <*a*>-Marke wird bei der Verwendung des *name*-Attributs nicht auf besondere Weise angezeigt.

Durch das *title*-Attribut kann man für das Dokument, auf das verwiesen wird, einen Titel angeben. Der Browser kann bei der Darstellung des Hyperlinks den Titel anzeigen, wenn die Maus über den Hyperlink bewegt wird. Ein großer Nutzen des *title*-Attributs besteht im Verweis auf eine Ressource, die keinen eigenen Namen hat, wie beispielweise ein Bild oder ein Nicht-HTML-Dokument.

Die Syntax und Semantik des URN (Universal Resource Name) wurden noch nicht eindeutig definiert, deshalb ist das *urn*-Attribut ehe für zukünftige HTML-Versionen vorgesehen.

Die *rel*- und *rev*-Attribute können benutzt werden, um Hyperlinks zu dokumentieren, indem man die Art der Verbindungen angibt. Das *rel*-Attribut gibt die Beziehung vom Quelldokument zum Ziel an, das *rev*-Attribut die umgekehrte Beziehung. Die Benennung der Beziehungen und deren Bedeutung sind nicht im HTML-Standard festgelegt.

Ein Dokument, das durchsucht werden kann, wird durch die <*isindex*>-Marke gekennzeichnet. Die Marke muß man in den *head*-Block setzen. Trifft der Browser auf die <*isindex*>-Marke, wird dem Dokument eine Suchzeile zugefügt, in der Schlüsselwörter eingeben werden können. Nachdem die Eingabetaste gedrückt wird, übergibt der Browser die Informationen zur weiteren Verarbeitung an den Server.

Zusätzlich zum *rel*-Attribut der <*a*>-Marke können noch 2 Marken benutzt werden-<*base*>- und <*link*>-Marke, um die Position und Beziehung eines Dokuments innerhalb einer Dokumentfamilie weiter zu verdeutlichen.

Diese Marken werden innerhalb der <*head*>-Marke untergebracht.

Normalerweise füllt der Browser die Leerstellen eines relativen URLs durch Kopieren der fehlenden Teile vom URL des aktuellen Dokuments aus. Dies kann man durch die <*base*>-Marke ändern. Der Browser verwendet dann den angegebenen Basis-URL und nicht den URL des aktuellen Dokuments,

und vervollständigt alle relativen URLs, die in Marken vorkommen (z.B. <*a*>,<*img*>). Dies ist dann hilfreich, wenn sich die Lage des Dokuments geändert hat.

Die <*base*>-Marke verfügt über ein benötigtes Attribut: *href*, dessen Wert ein gültiger URL sein muß.

Durch die <*link*>-Marke definiert man die Beziehung zwischen dem aktuellen Dokument und einem anderen Dokument einer Web-Sammlung. Die Attribute der <*link*>-Marke sind identisch mit denen der <*a*>-Marke (es fehlt nur *name*-Attribut) , allerdings dienen sie lediglich zur Dokumentation der Beziehung zwischen Dokumenten. Die <*link*>-Marke hat keinen Inhalt und keine abschließende Marke.

Die nächsten 2 Marken, die auch im Kopfteil stehen können, sind hauptsächlich dazu gedacht, die automatische Erstellung von Dokumenten zu unterstützen.

Die <*meta*>-Marke hat keinen Inhalt. Sie stellt zusätzliche Informationen über ein Dokument bereit.

Das *name*-Attribut definiert einen Bezeichner für den in *content*-Attribut, das die eigentliche Information enthält, angegebenen Wert. Die HTML-Norm schreibt keine bestimmten Bezeichner vor. Das *http-equiv*-Attribut bindet das Element an den Browser geschickten HTTP-Header. Damit läßt sich ein Server-Kopfelement simulieren.

Die<*nextid*>-Marke erscheint in HTML-Norm nur noch aus historischen Gründen und sollte nicht mehr verwendet werden. Diese Marke zusammen mit ihrem *n*-Attribut definiert den nächsten Dokument-Bezeichner.

4.1 Beispiel

Hyperlinks_bsp2.html :

<*html*>
<*head*>
<*title*>Hyperlinks - Beispiel 2</*title*>
<*link href*="Hyperlinks_bsp3.html" *rel*=next *rev*=prev>
</*head*>
<*body*>

<*h2*>Hyperlinks(Teil 1)</*h2*>

Machen wir jetzt einen Hyperlink<*a href*="#hier"> zum ***.</*a*>

<*p*>Mit #hier haben wir einen lokalen Hyperlink erstellt.

<p>Und auf dieser Stelle wird
zu der anderen Seite verwiesen.Die Beziehungen zwischen Quell- und Zieldokument haben
wir mit den rel- und rev-Attributen dokumentiert.

<p>Wenn jetzt der Verweis auf dasselbe
Dokument mit dem title-Attribut gemacht wird, dann müßten wir eigentlich beim Fahren
der Maus über den Link dessen title sehen. Ist leider nicht der Fall :-?

<p><a name="hier">***Wenn der Hyperlink oben angeklickt war, müssen wir hier landen.
Mit dem name-Attribut wurde das Ziel festgelegt.

</body>
</html>

Hyperlinks_bsp3.html :

<html>
<head>
<isindex>
<title> Hyperlinks - Beipiel 3 </title>
</head>
<body>

<h2><a name="da">Hyperlinks (Teil 2)</h2>
<p>Bei der Benutzung von der isindex-Marke, ist es möglich Suchschlüssel
an den Server zu übersenden.

</body>
</html>

5 Linien und Bilder

Zwar besteht der Hauptteil eines HTML-Dokuments aus Text, aber durch Benutzung der Linien und Bilder wird das Dokument viel attraktiver. Mit Querlinien beispielsweise kann man Abschnitte eines Dokuments optisch voneinander abheben.

Die <*hr*>-Marke weist den Browser dazu an, eine waagerechte Linie über das Darstellungsfenster zu ziehen. Wie <*br*>-Marke bewirkt auch die <*hr*>-Marke einen Zeilenumbruch. Der Browser plaziert die Linie direkt in der nächsten Zeile und fährt in der darunterliegenden Zeile mit dem normalen Textfluß fort.

Eines der Merkmale von HTML ist auch die Möglichkeit, dem Dokumententext Bilder hinzufügen, sei es als fester Bestandteil des Dokuments (Inline-Bilder) oder als eigene Dokumente. Das Hinzufügen von Bildern, Symbolen, Illustrationen, Zeichnungen usw. dient der Attraktivität und dem professionellen Aussehen der Dokumente.

Mit der <*img*>-Marke kann man im HTML-Dokument auf eine Grafik verweisen und Bilder in den aktuellen Textfluß integrieren. Die Definition sieht keine </*img*>-Marke vor. Vor und nach der <*img*>-Marke wird nicht automatisch ein Zeilen- oder Absatzumbruch durchgeführt, wodurch Bilder richtig im Text und in anderen Inhalten "mitfließen" können. Das Bildformat selbst ist im HTML-Standard nicht festgelegt, die gängigsten grafikorientierten Browser unterstützen in erster Linie GIF- und JPEG-Bilder. Die Bilddarstellung ist im allgemeinen sehr browserspezifisch. Da Bilder relativ zum Text viel mehr Speicher verbrauchen, muß man mit der Benutzung von Bildern in HTML-Dokumenten "sparsam" umgehen, um möglichst vielen Benutzern den Zugriff auf die Seite zu ermöglichen.

Das *src*-Attribut für die <*img*>-Marke ist unbedingt erforderlich; sein Wert ist der URL der Bilddatei, der entweder absolut oder relativ zu der das Bild enthaltenden HTML-Datei angegeben werden kann. Um die Dokumentverwaltung übersichtlicher zu machen, bewahren viele HTML-Autoren ihre Bilder in einem eigenen Ordner auf, den sie oft "Picts", "Images" oder "Bilder" nennen.

Mit dem *alt*-Attribut kennzeichnet man Text, den der Browser anzeigt, wenn ein Bild nicht dargestellt werden kann oder soll. Es ist nicht unbedingt erforderlich, aber sinnvoll, einen Alternativtext für jedes der Bilder zu schreiben. So hat der Benutzer wenigstens eine Ahnung davon, was er versäumt, wenn ein Bild nicht verfügbar ist. Der Attributwert für alt ist eine Zeichenkette von bis 1024 Zeichen, die in Anführungszeichen eingeschlossen werden muß, wenn sie Leerzeichen oder Satzzeichen beinhaltet. Grafikorientierte Browser ignorieren das *alt*-Attribut, wenn das Bild verfügbar ist und der Benutzer es laden will. Textbasierte Browser fügen den *alt*-Text wie jedes andere Element direkt in den Textfluß ein.

Der HTML-Standard sieht für Bilder keine Standardausrichtung vor. HTML-Bilder erscheinen normalerweise auf einer bestimmten Textzeile. Glücklicherweise hat ein HTML-Dokumententwickler mit dem *align*-Attribut der <*img*>-Marke die Möglichkeit, den Textfluß um ein Bild herum etwas zu steuern. Es gibt 3 Ausrichtungsattribute: *top* (oben), *middle* (zentriert) und *bottom* (unten).

Das *ismap*-Attribut teilt dem Browser mit, daß das Bild ein besonderes, mit der Maus anwählbares Abbild eines oder mehrerer Hyperlinks ist, das man als Imagemap bezeichnet. Es kann nur in einen Hyperlink mit <*a*>-Marke eingefügt werden. Der Browser schickt automatisch die x/y-Position der Maus (relativ zur oberen linken Bildecke) an den Server, wenn der Benutzer irgendwo in das *ismap*-Bild klickt. Landenkarten eingnen sich hervorragend als Beispiel. Man klickt beispielweise an seine Stadt und bekommt dann lokale Nachrichten.

5.1 Beispiel

<*html*>

<*head*>

<*title*>Linien und Bilder- Beispiel 4</*title*>

</*head*>

<*body*>

<*h2*>Linien und Bilder</*h2*>

Jetzt beschäftigen wir uns mit den Linien und Bildern.<*hr*>

Dieser Text liegt zwischen zwei Linien. Dazu haben wir 2 Mal hr-Marke angewendet.<*hr*>

Das Bild wird so: <*img src*="bild1.jpg" *alt*="Torte"> in den Textfluß eingefügt.

<*p*>Wenn die Seite mit dem textorientierten Browser betrachtet wird, ist es mit dem alt-Attirbut möglich,anstelle vom Bild einen Text zu zeigen,in diesem Falle: Torte.

<*hr*>

<*p*>Das nächste Attribut hilft uns, das Bild wie gewünscht zu plazieren:

<*img src*="bild1.jpg" *align*=top>Mit *align*=top wird der obere Bildrand am oberen Rand der aktuellen Textzeile ausgerichtet.

<*hr*>

<*p*>Bei *align*=middle wird die Bildmitte <*img src*="bild1.jpg" *align*=middle>

an der Grundlinie der Textzeile ausgerichtet.

<*hr*>

<*p*>Und mit *align*=bottom setzen wir das Bild so,

<*img src*="bild1.jpg" *align*=bottom> daß der untere Bildrand an der Grundlinie der Zeile ausgerichtet wird.Bie den meisten Browsern ist auch Standardausrichtung.

<*hr*>

<*p*>Das letzte *ismap*-Argument ermöglicht uns, dieses Bild als Hyperlink zu benutzen:

<*a href*="Hyperlinks_bsp3.html#da">

<*img src*="bild1.jpg" *ismap*>

</*a*>

</*body*>

</*html*>

6 Listen

HTML bietet nicht nur spezifische Marken zur Hervorhebung von Text, sondern auch Hilfsmittel zur Organisation von Inhalten in formatierten Listen.

Eine ungeordnete Liste, wie eine Einkaufsliste, ist eine Sammlung zusammengehöriger Objekte, die nicht speziell geordnet sind.

Die <*ul*>-Marke signalisiert dem Browser, daß der nachfolgende Inhalt bis zur Marke </*ul*> eine ungeordnete Liste von Objekten ist. Jedes Objekt innerhalb der Liste wird durch eine vorangestellte

<*li*>-Marke identifiziert. Normalerweise wird jeder Eintrag vom Browser vorn mit einem Bullet-Zeichen versehen und dann in eine eigene Zeile gesetzt.

Durch das *compact*-Attribut der <*ul*>-Marke wird eine ungeordnete Liste in einem kleineren, kompakteren Block zusammengefaßt.

Geordnete Listen, wie der Name schon sagt, werden dann verwendet, wenn die Reihenfolge der aufgelisteten Elemente wichtig ist. Normalerweise formatiert der Browser den Inhalt einer geordneten Liste genau wie eine ungeordnete Liste, anstelle des Bullet-Zeichens wird allerdings eine Numerierung eingefügt.

Die <*ol*>-Marke definiert eine geordnete Liste. Sie hat auch ein *compact*-Attribut. Da das Ende eines Listeneintrags immer aus der umgebenden Dokumentstruktur erkannt werden kann, kann man die </*li*>-Marke weglassen.

Mit der <*dir*>-Marke definiert man eine Verzeichnisliste, die aber nicht mit jedem Browser bei der Verwendung vom *compact*-Attribut kleiner gemacht werden kann.

Die Menü-Liste ist eine weitere Spezialform der ungeordneten Liste.

Die <*menu*>-Marke präsentiert dem Leser eine Liste mit kurzen Einträgen.

Auch diese Marke hat ein *compact*-Attribut, das aber bei den wenigsten Browsern benutzt wird. Netscape unterscheidet nicht zwischen einer Menüliste und ungeordneten Liste.

Schließlich bietet HTML ein Listenformat, das sich von den bisher besprochenen Listen ziemlich unterscheidet- die Definitionsliste. Mit ihr läßt sich beispielsweise ein Glossar sehr gut präsentieren.

Die Definitionsliste wird in die Marken <*dl*> und </*dl*> eingeschlossen. Innerhalb dieser Marken besteht jedes Element einer Definitionsliste aus zwei Teilen: einem Begriff und seiner Definition oder Erklärung. Anstelle von <*li*> wird jeder Begriff durch <*dt*> eingeleitet, gefolgt von der durch <*dd*> markierten Definition oder Erklärung. Für die <*dl*>-Marke gibt es auch ein *compact*-Attribut.

6.1 Beispiel

<*html*>

<*head*>

<*title*> Listen - Beipiel 5</*title*>

</*head*>

<*body*>

<*h2*>Listen</*h2*>

<*p*>In diesem Beispiel betrachten wir folgende Arten von Listen:

<*ul*>

<*li*>Ungeordnete Liste

<*ul*>

<*li*>Verzeichnisliste

<*li*>Menüliste
</*ul*>
<*li*>Geordnete Liste
<*li*>Definitionsliste
</*ul*>
<*p*>So sieht also eine ungeordnete Liste mit Verschachtelung.

<*h3*>Verzeichnisliste</*h3*>
<*p*>Beispiel einer Verzeichnisliste:
<*dir*>
<*li*>Bsp1.html
<*li*>Bsp2.html
<*li*>HTML_Dokument.doc
</*dir*>

<*h3*>Menüliste</*h3*>
<*menu*>
<*li*>Eintrag 1
<*li*>Eintrag 2
<*li*>Eintrag 3
</*menu*>

<*h3*>Geordnete Liste</*h3*>
<*ol*>
<*li*>8-10 Mathe
<*li*>10-12 RvS
<*li*>12-14 Physik
</*ol*>
<*p*>Diese Liste wird also anderes dargestellt.

<*h3*>Definitionsliste</*h3*>
Und anschließend eine Definitionsliste:
<*dl*>
<*dt*>*dd*-Marke
<*dd*>Die *dd*-Marke kennzeichnet in einer Definitionsliste den Beginn der Erklärung
eines Begriffs
<*dt*>*dl*-Marke
<*dd*>Definition einer Definitionsliste

<*dt*>*dt*-Marke

<*dd*>Durch *dt* wird der Term oder Begriff eines Definitionslisten-Elements markiert.

</*dl*>

</*body*>

</*html*>

7 Formulare

Erst Formulare geben dem Benutzer die Möglichkeit richtig interaktiv im HTML zu arbeiten. Nach dem Ausfüllen können sie an den Server geschickt werden, wo ein Programm diese Formulare bearbeitet und dann eine kurze Antwort an den Browser-Client zurücksendet.

Die Formularbestandteile werden von der <*form*> und </*form*>-Marke innerhalb des HTML-Dokumentkörpers eingeschlossen. Es ist auch möglich ganz normaler Text- und Bildinhalt verwendet werden. Da es keine Layout-Regeln für Formularelemente gibt, muß man innerhalb des Textflusses beispielsweise <*br*> und <*p*>- Marken benuzten.

Das erforderliche *action*-Attribut innerhalb einer <*form*>-Marke gibt den URL der Anwendung an, welche die Formulardaten empfangen und verarbeiten soll.

Der Browser kodiert die Formulardaten vor dem Absenden mit der Standardverschlüsselung. Diese Verschlüsselung kann man mit dem *enctype*-Attribut auf das einzige derzeit unterstützte Alternativformat "multipart/form-data" ändern.

Das zweite erforderliche Attribut-*method*-Attribut gibt an, auf welche Art und Weise der Browser die Formulardaten zur Bearbeitung an den Server sendet.

Es gibt zwei Möglichkeiten: die Post-Methode und Get-Methode.

Bei der Post-Methode sendet der Browser die Daten in zwei Schritten: es wird zuerst Kontakt zum Server hergestellt und anschließend die Daten im zweiten Schritt übertragen. Die get-Methode hingegen führt die Kontaktierung des Servers sowie die Übertragung der Formulardaten in einem einzigen Schritt durch.

Über die <*input*>-Marke definiert man beliebig viele Formularelemente.

Den Typ des Elements, das im Formular eingefügt werden soll, wählt man über das erforderliche *type*-Attribut. Der Namen des Feldes (der dann bei der Bearbeitung des Formulars von Server benötigt wird) wird über das *name*-Attribut vergeben.

Der HTML-Standard läßt in Fomularen den Einsatz von drei Typen von Texteingabefeldern zu: das normale Texteingabefeld, ein maskiertes Feld für die Eingabe vertraulicher Daten sowie ein Feld zur Angabe einer Datei. Die beiden ersten Möglichkeiten sind in allen Browsern verfügbar und enthalten als Attribute *size*, *maxlength* (maximale Länge) sowie *value* (Vorgabewert). Das Dateiauswahlfeld akzeptiert nur die Attribute *size* sowie *maxlength* und wird nur von Netscape unterstützt.

Der vordefinierte Wert von *type*-Attribut ist Text, man kann aber zwischen Password, Checkbox, Radio, Image (wird mit *align* und *src*-Attributen benutzt), Submit und Reset wählen.

Sowohl normale als auch verdeckte Texteingabefelder beschränken die Benutzereingaben auf eine einzelne Zeile. Die Formularmarke <*textarea*> beseitigt diese Einschränkung- sie erzeugt im Browserfenster des Benutzers ein mehrzeiliges Texteingabefeld, dessen Größe mit Hilfe der Attribute rows und cols festgelegt werden kann.

Durch Plazierung einer Liste von Optionselementen innerhalb der <*select*>-Marke mit dem erforderlichen *name*-Attribut erstellt man ein Pulldown-Menü mit mehreren Auswahlmöglichkeiten in einem Formular. Um gleichzeitig mehrere Optionen auswählen zu können, muß man in der <*select*>-Marke lediglich das Attribut *multiple* einfügen. Und über das *size*-Attribut wird festgelegt, wie viele Optionen für den Benutzer gleichzeitig sichtbar sind.

Die <*option*>-Marke definiert die verfügbaren Optionen innerhalb eines Auswahlmenüs. Über das *value*-Attribut kann man für jede Option einen Wert festlegen, den der Browser dann an den Server sendet, wenn diese Option ausgewählt wurde. Normalerweise ist keine der Optionen angewählt. Durch Angabe des Attributs *selected* kann ein Wert als angewählt dargestellt werden.

7.1 Beispiel

<*title*>Formulare-Beipiel6</*title*>

</*head*>

<*body*>

<*h2*>Formulare</*h2*>

<*p*>So sieht ein Standard-Formular aus, das *type*=Text hat:

<*p*><*form*>

<*input type*=text *name*=test1>

<*p*>Jetzt legen wir die Größe auf 10 und die maximale Länge auf 5 Buchstaben:

<*p*><*input type*=text *name*=test2 *size*=10 *maxlength*=5>

<*p*>Mit value wird das Formularfeld mit einem Vorgabewert initialisiert:

<*p*><*input type*=text *name*=test3 *size*=6 *maxlength*=4 *value*="199">

<*p*>Mit *type*=password ermöglicht das nächste Formular die Paßworteingabe:

<*p*><*input type*=password *name*=test4>

<*p*>Mit *type*=checkbox erstellen wir eine Auswahlfeld-Gruppe:

<*p*><*input type*=checkbox *name*=test5 *value*="Hund">Hund

<*p*><*input type*=checkbox *name*=test5 *value*="Katze">Katze

<*p*><*input type*=checkbox *name*=test5 *checked value*="Fisch">Fisch (wird mit checked aktiviert)

<p>Noch mal dasselbe, aber jetzt mit Radiobuttons:
<p><input type=radio name=test6 value="Hund">Hund
<p><input type=radio name=test6 value="Katze">Katze
<p><input type=radio name=test6 checked value="Fisch">Fisch (wird mit checked aktiviert)

<p>Beim Klicken auf die reset-Taste werden alle Eingaben in Formularfeldern gelöscht:
<p><input type=reset name=test7 value="Vorgaben löschen">

<p>Wenn das Formular ausgefüllt wird, kann man es mit der Submit-Taste abschicken:
<p><input type=submit name=test8 value="Abschicken">

<p>Das nächste Formular ist mehrzeilig. Es hat größe 4x40:
<p><textarea name=test9 cols=40 rows=4>
AAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEE
</textarea>

<p>Um längere Formulare übersichtlicher zu machen,benutzt man Pulldown-Menüs,
die mit der select-Marke erstellt werden:
<p><select name=test10 size=3 multiple>
<option>Hund
<option>Katze
<option>Fisch
<option selected>Schlange
<option value=Kh>Kuh
</select>
</form>
</body>
</html>

8 Abschließendes Beispiel

9 HTML- eine SGML Documet Type Definition (DTD)

Beschäftigen wir uns nun mit der "theoretischen" Basis von HTML.

Die HTML 2.0-Norm ist formell als eine der vielen möglichen SGML -Dokumenttyp-Definition (DTD) festgelegt, mit dem Ziel eine einfache und platzsparende Hypersprache zu beschreiben. In einer DTD sind die Regeln zur Auszeichnung (markup) einer Klasse von Dokumenten formal beschrieben.

Vorweg eine kurze Wiederholung, was Enities, Elemente und Attribute

bedeuten.

Entities kann man als eine Art Abkürzung von beliebigen Texten. Sie sind sehr nützlich, wenn es um Texte geht, die entweder zu schwer angegeben werden können oder sehr oft benutzt werden.

Elemente- die wichtigsten Komponenten der DTD, definieren meistens Strukturen oder das gewünschte Verhalten. Elemente werden gelegentlich auch als "Container" bezeichnet, da sie als "Behälter" für Text dienen.

Elemente können ein oder mehrere optionale oder zwingende Attribute besitzen.

Attribute enthalten zusätzliche Informationen über Elemente.

Jetzt lassen wir uns nur einige ausgewählte Fragmente der HTML-DTD betrachten, die erläutern sollen, wie HTML-DTD aufgebaut ist.

Kommentare in der DTD werden in -- -- eingefügt, z. B.

<!-- <*i*> Italic Text -->

HTML-DTD fängt mit einer Serie von Entities-Definitionen an, die als Makros innerhalb DTD benutzt werden. Eine Entity-Definition beginnt mit

<!ENTITY % gefolgt mit Entity-Name und endet mit >. Folgendes Beispiel definiert einen String, der beim Vorkommen von %font bzw. %phrase im Text an seiner Stelle geschrieben wird:

<! ENTITY %font " TT | B | I" >

<!ENTITY %phrase " EM | STRONG | CODE | SAMP | KBD | VAR | CITE ">

In Enities können auch andere Entities benutzt werden:

<!ENTITY %text "#PCDATA | A | IMG | BR | %phrase | %font" >

Die Element-Definition beginnt mit <!ELEMENT und endet mit >. Zwischen ihnen stehen der Element-Name, (-- oder -O oder OO) und möglicher Element-Inhalt (content) . Elemente können entweder "gehaltvoll" oder leer sein. Zwei Bindestriche nach dem Element-Namen bedeuten, daß der Element eine Start- und eine End-Marke hat. Ein Bindestrich mit O zeigen das Fehlen einer End-Marke ,und bei zwei O kann man beide Marken weglassen. Z.b.:

<!ELEMENT (%font; | %phrase) -- (%text)* > -Das ist ein Element mit Start- und End-Marke, das "gehaltvoll" ist- enhtält content- (%text)*, der entweder kein oder mehrere Male vorkommen darf.

<!ELEMENT *img* -O EMPTY>- Ein Element ohne End-Marke. EMPTY bedeutet, daß ELEMENT keinen Inhalt haben kann.

<!ELEMENT *body* OO %body.content >- Ein Element ohne Marken.

Ein Element kann auch Attribute haben, die mit <!ATTLIST agefangen, von Element-Name gefolgt und mit > geschlossen werden. Die Definition ist ein Tripel: Attribut-Name, Typ des Attributs oder explizierte Werte, #Required -notwendiges Attribut oder #Implied- nicht notwendiges Attribut:

<!ATTLIST *textarea*

name CDATA #REQUIERED
rows NUMBER #REQUIERED
cols NUMBER #REQUIERED
.... >

Alle obengenannten Attribute von *textarea* sind notwendig, die letzten zwei müssen Zahlen sein.

<!ATTLIST *input*

....
align (top | middle | bottom) #IMPLIED

Das war das Beispiel eines Attibuts mit explizierten Werten.

Da die komplete Version von HTML DTD sehr umfangreich ist, werden wir an dieser Stelle nur ausgewählte Abschnitte betrachten: DTD "Macros" , Character mnemonic entities, Text Markup, Paragraphs und Headings, Titles, Sections. Hier sieht man, wie oben behandelten Marken und Strukturen in der HTML DTD definiert werden.

```
<!--==================DTD  "Macros" ===============-->

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

<!ENTITY % list " UL | OL | DIR | MENU " >


<!--================Character mnemonic entities =========-->

<!ENTITY % ISOlat1 PUBLIC
  "-//IETF//ENTITIES Added Latin 1 for HTML//EN">
%ISOlat1;

<!ENTITY amp CDATA "&#38;"    -- ampersand         -->
<!ENTITY gt CDATA "&#62;"     -- greater than      -->
<!ENTITY lt CDATA "&#60;"     -- less than         -->
<!ENTITY quot CDATA "&#34;"   -- double quote      -->


<!---------- Text Markup ======================-->
```

```
<![ %HTML.Highlighting [

<!ENTITY % font " TT | B | I ">

<!ENTITY % phrase "EM | STRONG | CODE | SAMP | KBD | VAR | CITE ">

<!ENTITY % text "#PCDATA | A | IMG | BR | %phrase | %font">

<!ELEMENT (%font;|%phrase) - - (%text)*>
<!ATTLIST ( TT | CODE | SAMP | KBD | VAR )
      %SDAFORM; "Lit"
      >
<!ATTLIST ( B | STRONG )
      %SDAFORM; "B"         >

<!ATTLIST ( I | EM | CITE )
      %SDAFORM; "It"
      >

<!-- <TT>      Typewriter text                  -->
<!-- <B>       Bold text                        -->
<!-- <I>       Italic text                      -->

<!-- <EM>      Emphasized phrase                -->
<!-- <STRONG>  Strong emphasis                  -->
<!-- <CODE>    Source code phrase               -->
<!-- <SAMP>    Sample text or characters        -->
<!-- <KBD>     Keyboard phrase, e.g. user input -->
<!-- <VAR>     Variable phrase or substitutable -->
<!-- <CITE>    Name or title of cited work      -->

<!ENTITY % pre.content "#PCDATA | A | HR | BR | %font | %phrase">

]]>

<!ENTITY % text "#PCDATA | A | IMG | BR">

<!ELEMENT BR    - O EMPTY>
<!ATTLIST BR
      %SDAPREF; "&#RE;"
      >

<!-- <BR>      Line break     -->
```

```
<!--========== Paragraphs=======================-->

<!ELEMENT P     - O (%text)*>
<!ATTLIST P
      %SDAFORM; "Para"
      >
<!-- <P>       Paragraph      -->


<!--========== Headings, Titles, Sections ===============-->

<!ELEMENT HR    - O EMPTY>
<!ATTLIST HR
      %SDAPREF; "&#RE;&#RE;"
      >

<!-- <HR>      Horizontal rule -->

<!ELEMENT ( %heading )  - -  (%text;)*>
<!ATTLIST H1
      %SDAFORM; "H1"
      >
<!ATTLIST H2
      %SDAFORM; "H2"
      >
<!ATTLIST H3
      %SDAFORM; "H3"
      >
<!ATTLIST H4
      %SDAFORM; "H4"
      >
<!ATTLIST H5
      %SDAFORM; "H5"
      >
<!ATTLIST H6
      %SDAFORM; "H6"
      >

<!-- <H1>      Heading, level 1 -->
<!-- <H2>      Heading, level 2 -->
```

<!-- <H3> Heading, level 3 -->
<!-- <H4> Heading, level 4 -->
<!-- <H5> Heading, level 5 -->
<!-- <H6> Heading, level 6 -->

9 Zusammenfassung

HTML ist die meistverbreitete im Internet Sprache, die sehr leicht zu erlernend und zu benutzen ist. Mit Hilfe von im Text verankerten Anweisungen-Marken wird der Textfluß wie gewünscht dargestellt und die benötigten Strukturen zu Verfügung gestellt. Man braucht sich nicht um die Einzelheiten kümmern- das ist die Aufgabe von Browsern.

Der HTML 2.0 -Standard sieht folgende Möglichkeiten vor: Text- und Bilddarstellung, Verweise auf andere Dokumente, Benutzung von Listen und Formularen.

Es fehlen zwar viele "extras", die in nächsten Versionen implementiert sind, aber mit diesem Grundgerüst lassen sich schon anspruchsvolle Dokumente erstellen.

10 Literatur

Musciano, C. ,Kennedy, B.: HTML Das umfassende Referenzwerk , O'Reilly 1997

Connolly, D.:http://www.w3.ogr/hypertext/WWW/MarkUp/MarkUp.htm [Stand: 21.09.1995]

HTML II – Beschreibung der Änderungen des Sprachumfangs der Versionen 3.2-4

Georg Conrads

FB Informatik, Uni Dortmund
Georg.Conrads@ruhr-uni-bochum.de

Zusammenfassung

Dieses Referat beschreibt die Änderungen des Sprachumfangs der HyperText Markup Language (HTML) von der Versionen 2.0 bis zur Version 4.0. Es baut auf das Vorgängerreferat „HTML –Beschreibung des Sprachumfangs anhand von HTML 2.0“ auf, so daß hier lediglich die Änderungen gegenüber den Vorgängerversionen aufgeführt werden.

1 Grundstruktur von HTML-Dokumenten

Ein HTML 4.0 Dokument besteht aus 3 Teilen:

- Einer Dokumenttypdeklaration
- Einer Header-Sektion
- Einer Body-Sektion, die den eigentlich Inhalt des Dokuments umfaßt. Die Body-Sektion kann mit Hilfe das BODY-Elements oder des FRAMESET-Elements implementiert werden.

1.1 Dokumenttypdeklaration

Die Dokumenttypdeklaration bestimmt die Document Type Definition (DTD), die dem Dokument zugrunde liegt.

In HTML 4.0 gibt es folgende drei DTDs. Sie unterscheiden sich in den Elementen, die durch sie unterstützt werden:

- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">`

  Die HTML 4.0 Strict DTD beinhaltet alle Elemente, die nicht veraltet sind und nicht in Frameset-Dokumenten auftauchen.

- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" http://www.w3.org/TR/REC-html40/loose.dtd">`

  Die HTML 4.0 Transitional DTD beinhaltet die komplette Strict DTD und zusätzlich alle veralteten Elemente und Attribute.

- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" http://www.w3.org/TR/REC-html40/frameset.dtd">`

  Die HTML 4.0 Frameset DTD beinhaltet die komplette Transitional DTD und unterstützt zusätzlich Frames.

(Mit Hilfe des URI in den jeweiligen Dokumenttypdeklarationen können Internetbrowser die benötigten DTDs finden.)

1.2 Die Header-Sektion

Die Header-Sektion enthält eine Sammlung von ungeordneten Informationen über das Dokument.

Die Header Sektion wird wie gewohnt durch das **HEAD**-Element umschlossen. Neu ist, daß mit dem **profile**-Attribut ein oder mehrere Meta-Daten-Profile angegeben werden können. In einem solchen Profil werden die Bedeutungen und erlaubten Werte der Meta-Daten definiert.

Die Meta-Daten selbst werden mit Hilfe des **META**-Elements angegeben. Neu hierbei ist das **scheme**-Attribut, welches Informationen über die korrekte Interpretation der Meta-Daten liefert.

Beispiel: Wenn ein Datum als "10-9-97" angegeben wird, dann ist es unklar, ob der 10. September 1997 oder der 9. Oktober 1997 gemeint ist. Das scheme-Attribut mit dem Wert "Month-Date-Year" würde die Zweideutigkeit aufheben.

Das META-Element kann im übrigen in HTML 4.0 dazu verwendet werden, die Standard-Skriptsprache [S. 168], die Standard-Stylesheet-Sprache [S. 159] und die Standard-Charakter-Kodierung festzulegen.

1.3 Die Body-Sektion

Alle Attribute, die die visuelle Gestaltung eines HTML-Dokuments betreffen (wie z.B. die Textfarbe und die Hintergrundfarbe) sind veraltet. Anstelle dieser Attribute wird nun die Verwendung von Style Sheets [S. 159]) empfohlen.

In Dokumenten, die Framesets enthalten, wird das **BODY**-Element durch das **FRAMESET**-Element ersetzt.

1.3.1 id und class Attribute

Eine Neuheit sind die **id-** und **class**-Attribute. Mit ihnen können Elemente identifiziert werden. Wird ein Element mit einem id-Attribut versehen, so muß der Name eindeutig sein. Mehrere Elemente dürfen jedoch denselben Klassennamen haben, so wie auch ein einzelnes Element mehreren Klassen zugehörig sein kann.

Das id-Attribut dient verschiedenen Zwecken. Es kann beispielsweise

- als Style Sheet Selektor,
- als Zielanker für Hyperlinks,

- als Erkennungsschlüssel, um auf ein bestimmtes Element von einem Skript aus zu verweisen,

verwendet werden.

Mit dem class-Attribut kann eine Klassenzugehörigkeit von Elementen festgelegt werden, die ebenfalls als Style Sheet Selektor oder für sonstige Zwecke benutzt werden kann.

1.3.2 title Attribut

Nahezu jedes Element kann mit einem title-Attribut versehen werden. Im Gegensatz zu dem TITLE-*Element*, welches Informationen über das gesamte Dokument liefert, können nun einzelne Elemente selbst mit Informationen versehen werden. Wie diese Informationen dargestellt werden, hängt vom verwendeten Internetbrowser ab. Beispielsweise kann der Text als Popup-Fenster erscheinen, wenn sich der Mauszeiger eine Weile über dem entsprechenden Element befindet.

1.4 DIV und SPAN Elemente

Mit diesen Elementen läßt sich Struktur in ein Dokument zu bringen. Diese Elemente kennzeichnen ihren Inhalt als *inline (SPAN)* oder *block-level (DIV)*. Die Interpretation dieser Begriffe hängt vom Kontext ab, in dem sie verwendet werden. Beispielsweise können block-level Elemente anders formatiert werden als inline Elemente.

2 Internationalisierung

2.1 lang Attribut

Mit Hilfe des lang-Attributs kann die Sprache des Inhalts eines Elements bestimmt werden. Wie diese Information genutzt werden kann, hängt vom verwendeten Internetbrowser ab. In den folgenden Beispielen könnte eine solche Angabe sinnvoll sein:

- Unterstützung vom Suchmaschinen
- Unterstützung vom Sprachsynthesizern
- Unterstützung der optischen Gestaltung des Textes (Sonderzeichen, Anführungszeichen...)
- Unterstützung von Grammatik- und Rechtschreibungsassistenten

Der Wert des lang-Attributs besteht aus einem *language code*, der aus einem *primären* und einer (möglichen) Reihe von *Subcodes* gebildet wird.

Beispiele:

- "de": Deutsch
- "en": Englisch
- "en-US": U.S. Version von Englisch
- "en-cockney": Cockney Version von Englisch
- "x-klingon": der Primärcode "x" kennzeichnet eine experimentale Sprache

2.2 dir Attribut

Mit dem **dir**-Attribut läßt sich die Textrichtung bestimmen. Mögliche Werte sind:

- LTR: Links-nach-rechts
- RTL: Rechts-nach-links

Beispiel:

```
<Q lang="he" dir="rtl">... ein hebräisches Zitat...</Q>
```

Die **UNICODE**–Spezifikation, auf der HTML Dokumente basieren, besitzt bereits Informationen über die Textrichtung für jedes einzelne Zeichen und definiert einen Algorithmus (*bidirectional algorithm*), um die Richtung eines Textes automatisch zu bestimmen. Im allgemeinen ist es jedoch übersichtlicher, das dir-Attribut zu verwenden.

In einigen Fällen ist es wünschenswert, den *bidirectional algorithm* auszuschalten – beispielsweise, wenn mit einem geeigneten Editor das oben angesprochene hebräische Zitat schon von *rechts nach links* eingegeben wurde. Für diesen Fall ist das **BDO**-Element gedacht:

```
<BDO dir="LTR">deutsch1 2ehcsiärbeh deutsch3</BDO>
```

Auf diese Weise wird dem *bidirectional algorithm* explizit deutlich gemacht, den Text von *links nach rechts* auszugeben.

3 Text

3.1 Phrase Elemente

In HTML 4.0 gibt es zwei neue Phrase Elemente:

- ABBR: Kennzeichnet eine abgekürzte Form (z.B. WWW, HTTP, URI, usw.)
- ACRONYM: Kennzeichnet ein Akronym (z.B. "GmbH", "NATO", usw.)

Die Verwendung dieser Elemente bietet eine hilfreiche Unterstützung für Sprachsynthesizer und Übersetzungssysteme.

3.2 BLOCKQUOTE und Q Elemente

Diese Elemente kennzeichnen Zitate. **BLOCKQUOTE** ist für lange *Zitate (block-level)* und **Q** für kurze Zitate *(inline)* gedacht. Neu ist das **cite**-Attribut, mit dem sich ein URI angeben läßt, um auf die Quelle des Zitates zu verweisen.

3.3 SUB und SUP Elemente

Mit diesen beiden Elementen läßt sich eine Textpassage hoch- oder tiefstellen.

Beispiel:

```
E = mc<SUP>2</SUP>
```

3.4 INS und DEL Elemente

Diese Elemente werden verwendet, um Bereiche in einem Text zu markieren, die eingefügt oder entfernt wurden.

Beispiel:

```
<P>
CD-Rohlinge kosten nur noch <DEL>3,50</DEL><INS>2,90</INS> DM
</P>
```

4 Listen

4.1 Ungeordnete Listen (UL), geordnete Listen (OL) und List Items (LI)

Die Attribute **type**, **start**, **value** und **compact** sind veraltet. An Stelle dieser wird die Verwendung von *Style Sheets* [S.159] empfohlen.

Im folgenden Beispiel wird mit Hilfe von *Cascading Style Sheets* festgelegt, daß die Listenelemente mit römischen Kleinbuchstaben durchnumeriert werden sollen:

```
<STYLE type="text/css">
OL.withroman { list-style-type: lower-roman }
</STYLE>
<BODY>
<OL class="withroman">
<LI> Step one ...
<LI> Step two ...
</OL>
</BODY>
```

4.2 DIR und MENU Elemente

Die DIR- und das MENU-Element sind veraltet.

5 Tabellen

In HTML 4.0 lassen sich recht komplexe Tabellen erstellen. Eine Besonderheit im Vergleich zu früheren Versionen ist die, daß die Tabellen schon während des Downloads dargestellt werden können und nicht erst komplett übertragen werden müssen.

5.1 TABLE Element

Eine Tabelle wird mit Hilfe des **TABLE**-Elements definiert.

Attribute:

`summary = text`
Mit diesem Attribut läßt sich eine Zusammenfassung des Tabelleninhalts angeben (beispielsweise für Sprachsynthesizer).

`align = left | center | right`
Veraltet. Dieses Attribut bestimmt die Position der Tabelle bezogen auf das Dokument.

`width = length`
Dieses Attribut bestimmt die gewünschte Weite der gesamten Tabelle.

`frame = void | above | below ...`
Dieses Attribut bestimmt, welche Seiten des Tabellenrahmens sichtbar sind.

`rules  = none | groups | rows | cols | all`
Dieses Attribut bestimmt, welche Linien zwischen den Zellen gezeigt werden sollen.

`border = pixels`
Dieses Attribut bestimmt die Dicke des Rahmens.

`cellspacing = length`
Dieses Attribut bestimmt den Abstand der Zellen zueinander.

`cellpadding = length`
Dieses Attribut bestimmt den Abstand des Zellrandes zum Zellinhalt.

5.2 CAPTION Element

Mit dem **CAPTION**-Element läßt sich eine Überschrift der Tabelle angeben. Dieses Element muß direkt hinter dem TABLE *Start Tag* stehen und jede Tabelle darf nur ein **CAPTION**-Element besitzen.

5.3 THEAD, TFOOT und TBODY Elemente

Die Zeilen einer Tabelle können in *Table Head*, *Table Foot* und in eine oder mehrere *Table Body* Sektionen gruppiert werden. Auf diese Weise können Sektionen der Tabelle unabhängig von den Kopf- und Fußzeilen gescrollt werden. Wenn die Tabelle ausgedruckt wird, können die Kopf- und Fußzeilen auf jeder Seite wiederholt werden.

Das **TFOOT**-Element muß *vor* den **TBODY**-Elementen stehen, damit Internetbrowser die Fußzeile darstellen können, bevor alle Table Body Sektionen empfangen wurden.

5.4 COLGROUP und COL Elemente

Mit diesen beiden Elementen lassen sich die Spalten einer Tabelle gruppieren. Dabei bewirkt das **COLGROUP**-Element eine strukturelle Unterteilung der Tabelle (welche von einem Internetbrowser entsprechend visualisiert werden kann), wohingegen das **COL**-Element lediglich dazu dient, Attribute auf mehrere Spalten zu übertragen.

Attribute:

`span = number`
Dieses Attribut bestimmt die Anzahl der Spalten einer Gruppierung.

`width = multi-length`
Dieses Attribut bestimmt die Standardbreite für jede Spalte in der Gruppierung.

5.5 TR Element

Das TR-Element definiert eine Tabellenzeile und beinhaltet Tabellenzellen.

5.6 TH und TD Elemente

Mit diesen Elementen werden die Tabellenzellen definiert. Hierbei gibt es zwei Arten: **Header**-Zellen (**TH**) und **Data**-Zellen (**TD**).

Zum Abschluß eines kleines Beispiel einer Tabelle:

```
<TABLE border="1">
<CAPTION>Cups of coffee consumed</CAPTION>
<TR><TH>Name<TH>Cups<TH>Type of Coffee<TH>Sugar?
<TR><TD>T. Sexton<TD>10<TD>Espresso<TD>No
<TR><TD>J. Dinnen<TD>5<TD>Decaf<TD>Yes
<TR><TD>A. Soria<TD colspan="3">Not available
</TABLE>
```

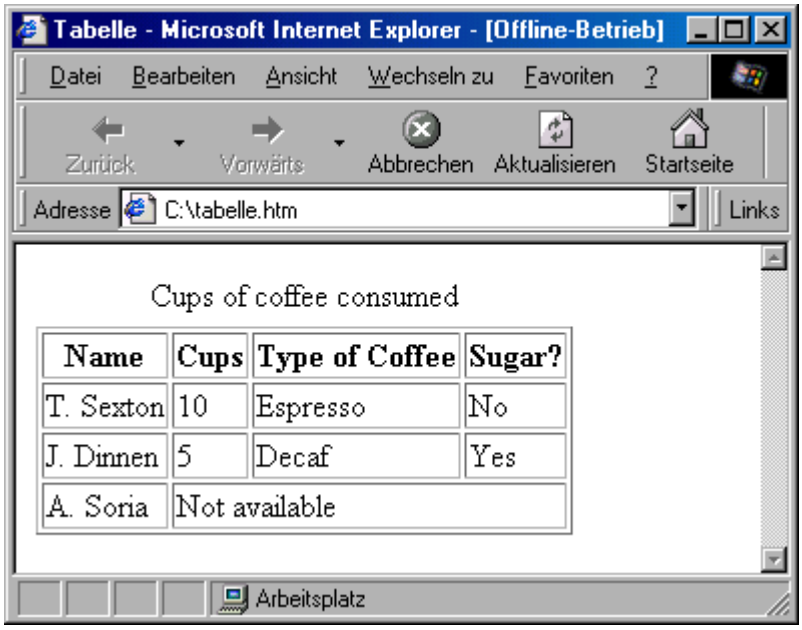


Abbildung 5-1: Tabelle

6 Links

In HTML 4.0 kann nun jedes Element innerhalb eines HTML-Dokumentes mit dem **id**-Attribut zu einem Hyperlink-Anker gemacht werden.

Beispiel:

```
Weitere Informationen finden Sie in <A href="#kapitel2"Kapitel 2</A>
... später im Dokument
<H2 id="kapitel2">Kapitel 2</H2>
```

Dieses Beispiel hat den gleichen Effekt wie:

```
Weitere Informationen finden Sie in <A href="#kapitel2"Kapitel 2</A>
... später im Dokument
<H2><A name="kapitel2">Kapitel 2</H2>
```

6.1 A und LINK Elemente

Neu bei diesen Elementen sind die Attribute **hreflang** und **charset**, die die Sprache bzw. den Zeichensatz des Zieldokumentes angeben. Mit dem Attribut **target** kann das Zielframe [S.163] bestimmt werden. (Dieses Attribut kann ebenfalls für das **BASE**-Element gesetzt werden.)

7 Objects, Images und Applets

In früheren HTML-Versionen konnten Bilder mit Hilfe des **IMG**-Elements und Applets (Programme, die automatisch heruntergeladen und auf dem Computer ausgeführt werden) mit Hilfe des **APPLET**-Elements in HTML-Dokumente eingebunden werden. In HTML 4.0 erledigt nun das **OBJECT**-Element alle diese Funktionen. (Aus Kompatibilitätsgründen ist die Verwendung der IMG- und APPLET-Elemente weiterhin möglich, jedoch nicht mehr empfohlen!)

Mit dem Object-Element können dann auch recht einfach zukünftige Multimedia-Objekte (*Plug-Ins*...) eingebunden werden.

7.1 OBJECT Element

Attribute:

`classid = uri`
Dieses Attribut bestimmt den Ort der Implementation des Objektes. Es kann zusammen mit oder als Alternative zu dem **data**-Attribut verwendet werden (abhängig vom Objekttyp).

`codebase = uri`
Dieses Attribut bestimmt die Basisadresse für relative URIs der **classid**-, **data**- und **archive**-Attribute.

`codetype = content-type`
Dieses Attribut bestimmt den Typ eines Objektes, das mit dem **classid**-Attribut angegeben wurde.

`data = uri`
Dieses Attribut bestimmt den Ort, an dem sich die eigentlichen Objektdaten befinden (beispielsweise eine Bilddatei für ein Bild).

`type = content-type`
Dieses Attribut bestimmt den Typ eines Objektes, das mit dem **data**-Attribut angegeben wurde.

`archive = uri list`
Mit diesem Attribut läßt sich eine Liste von URIs angeben, die auf Archive von benötigten Ressourcen für das durch **classid** oder **data** bestimmte Objekt verweist.

`declare`
Ist dieses boolesche Attribut gesetzt, dann ist die gegenwärtige Objekt-Definition nur eine Deklaration. Das Objekt kann dann später im Dokument durch einen Verweis auf diese Deklaration instantiiert werden.

`standby = text`

Dieses Attribut bestimmt einen Text, der angezeigt werden kann, während das Objekt geladen wird.

In der Regel muß man drei Arten von Informationen angeben:

- Die Implementation des eingefügten Objekts. Wenn das Objekt beispielsweise ein Applet ist, welches eine Uhr darstellt, muß der Ort des ausführbaren Codes angegeben werden.

- Die Daten, die verarbeitet werden sollen. Ist das Objekt z.B. ein Programm, das Zeichensatzdaten visualisiert, müssen diese Daten angegeben werden.

- Zusätzliche Werte, die das Objekt während der Laufzeit benötigt. Z.B. benötigen einige Programme Initialisationswerte für ihre Parameter. Diese Werte werden mit Hilfe des **PARAM**-Elements (siehe unten!) übergeben.

Nicht alle diese Informationen müssen immer angegeben werden. Beispielsweise brauchen nicht alle Objekte zusätzliche Implementations-Informationen, da Internetbrowser bereits wissen, wie sie die Daten darstellen müssen (z.B. GIF-Bilder).

OBJECT-Elemente dürfen auch in der Header-Sektion eines Dokumentes auftreten.

Kann ein Objekt nicht dargestellt werden (aufgrund ungenügender Ressourcen, falscher Architektur usw.), wird der Inhalt des OBJECT-Elements ausgegeben. (Objekte, die in der Header-Sektion auftauchen, dürfen jedoch keinen Inhalt besitzen, da Internetbrowser in der Regel Elemente der Header-Sektion nicht darstellen!)

Wenn im folgenden Beispiel der Internetbrowser nicht das Applet starten kann, welches eine analoge Uhr darstellt, wird der Text "Eine animierte Uhr." angezeigt.

```
<OBJECT classid="http://www.zeitistgeld.com/analogclock.py">
Eine animierte Uhr.
</OBJECT>
```

OBJECT-Elemente können geschachtelt werden. Auf diese Weise lassen sich alternative Objekttypen angeben, die dargestellt werden, wenn die äußeren Objekte nicht angezeigt werden können.

7.2 PARAM Element

Mit dem **PARAM**-Element können innerhalb des **OBJECT**-Elements Parameter an das Objekt übergeben werden.

Attribute:

`name = cdata`
Dieses Attribut bestimmt den Namen eines Parameters, der dem eingefügten Objekt bekannt sein muß.

`value = cdata`
Dieses Attribut gibt den Wert des Parameters an, der durch **name** bestimmt wurde.

`valuetype = data | ref | object`
Dieses Attribut bestimmt, ob der angegebene Wert des **value**-Attributs direkt dem Objekt übergeben werden soll (**data**), der angegebene Wert ein Verweis auf eine Ressource ist, in der die eigentlichen zu übergebenen Daten gespeichert sind (**ref**), oder der angegebene Wert der *Identifier* eines anderen Objektes innerhalb desselben Dokument ist (**object**).

`type = content-type`
Dieses Attribut wird nur gebraucht, wenn das **valuetype**-Attribut den Wert "ref" hat. In diesem Fall gibt es den Typ des Wertes an, der unter der angegebenen Adresse zu finden ist.

7.3 Objekt-Deklarationen und -Instanzen

Wie bereits oben erwähnt, lassen sich Objekte mit Hilfe des **declare**-Attributs deklarieren und später im Dokument instantiieren. Dies ist z.B. von Vorteil, wenn dasselbe Objekt gleich mehrmals im Dokument auftritt. In diesem Fall müssen die Objektdaten nur ein einziges Mal übertragen werden.

Instantiiert werden solche Objekte einfach durch einen Verweis, wie das folgende Beispiel zeigt.

```
<P><OBJECT declare
      id="earth.declaration"
      data="TheEarth.mpeg"
      type="application/mpeg">
Die Erde.
</OBJECT>
...später im Dokument...
<P>Eine <A href="#earth.declaration"> Animation der Erde!</A>
```

Sobald auf den Link geklickt wird, erscheint die Animation der Erde.

7.4 Image Maps

Mit Hilfe von Image Maps kann ein Bild in verschiedene Bereiche unterteilt werden, die mit einem Link versehen werden können. Wenn der Anwender auf einen solchen Bereich klickt, wird der entsprechende Link ausgeführt.

Es gibt zwei Arten von Image Maps:

- **Client-side**. Wenn der Anwender auf eine Region klickt, werden die Koordinaten vom Internetbrowser ausgewertet und der entsprechende Link ausgeführt.
- **Server-side**. In diesem Fall werden die Koordination zum Server übertragen und dort ausgewertet.

Im allgemeinen ist die Verwendung von Client-side Image Maps vorzuziehen, da sie u.a. schneller sind, weil die Koordinaten nicht erst übergeben werden müssen.

7.5 Client-side Image Maps: MAP und AREA Elemente

Eine Image Map wird mit Hilfe des **MAP**-Elements definiert.

Innerhalb des **MAP**-Elements werden die Bereiche mit dem **AREA**-Element oder dem **A**-Element bestimmt.

Das **AREA**-Element besitzt folgende Attribute:

`shape = default | rect | circle | poly`
Dieses Attribut bestimmt den Umriß der Region (**default** steht für das gesamte Bild).

`coords = `*`coordinates`*
Dieses Attribut bestimmt die Position einer Region. Die Anzahl und Reihenfolge der Werte hängen vom **shape**-Attribut ab:

- **rect**: links-x, oben-y, rechts-x, unten-y
- **circle**: mittel-x, mittel-y, Radius
- **poly**: x1, y1, x2, y2, ..., xN, yN.

Alle Koordinaten beziehen sich auf die linke obere Ecke.

`nohref`
Dieses boolesche Attribut bestimmt, daß die Region keinen Link besitzt.
Eine Image Map wird mit Hilfe des **usemap**-Attributs mit einem Bild verknüpft.

Beispiel:

```
<OBJECT data="navbar1.gif" type="image/gif" usemap="#map1">
   <P>This is a navigation bar.
</OBJECT>

<MAP name="map1">
  <AREA href="guide.html"
        alt="Access Guide"
        shape="rect"
        coords="0,0,118,28">
  <AREA href="search.html"
        alt="Search"
        shape="rect"
        coords="184,0,276,28">
</MAP>
```

Die gleiche Image Map könnte mit dem **A**-Element wie folgt definiert werden:

```
<MAP name="map1">
  <P>Navigate the site:
  <A href="guide.html" shape="rect" coords="0,0,118,28">
    Access Guide</A>
  <A href="search.html" shape "rect" coords="184,0,276,28">
    Search</A>
</MAP>
```

Zu beachten ist die Verwendung der **alt**-Attribute bei den **AREA**-Elementen, um eine Navigation für textbasierte Browser zu ermöglichen.

8 Style Sheets

Style Sheets stellen ein Mittel dar, die Präsentation von HTML-Seiten reichhaltiger gestalten zu können als es mit konventionellen HTML-Elementen möglich ist.

In HTML 4.0 Dokumenten können Style Sheets über externe Dateien eingebunden werden oder direkt im HTML-Quelltext definiert werden. Dabei können die Style Sheets in verschiedenen Sprachen programmiert werden. In den folgenden Beispielen wird jedoch ausschließlich die Sprache Cascading Style Sheets (CSS) verwendet. Auf die genaue Syntax der Sprache wird an dieser Stelle nicht genauer eingegangen; es sei auf das Referat "Cascading Style Sheets" verwiesen.

Die Standard Style Sheet Sprache (in diesem Fall CSS) wird in der Header-Sektion eines HTML-Dokuments wie folgt festgelegt:

```
<META http-equiv="Content-Style-Type" content="text/css">
```

Die Standardsprache kann ebenfalls im HTTP-Header gesetzt werden. Die obige META-Deklaration ist äquivalent zu dem folgenden HTTP-Header:

```
Content-Style-Type: text/css
```

8.1 Inline Styles

Über das **style**-Attribut können Style-Informationen für ein einzelnes Element bestimmt werden. Die Sprache eines solchen Inline Styles wird durch die Standard Style Sheet Sprache festgelegt.

Im folgenden Beispiel wird die Textfarbe und Zeichengröße eines Paragraphen gesetzt:

```
<P style="font-size: 12pt; color: blue">Schönes Wetter heute!"
```

In CSS haben Eigenschaftsdeklarationen die Syntax "name: value" und werden durch Semikolons getrennt.

8.2 STYLE Element

Attribute:

`type = content-type`

Dieses Attribut bestimmt die Style Sheet Sprache des Elementinhalts.

`media = media-descriptors`

Dieses Attribut bestimmt das Zielmedium [S.160] der Style Information. Es können mehrere Werte angegeben werden. Der Standardwert ist "`screen`".

Mit dem STYLE-Element werden Style Sheet Regeln in der Header-Sektion eines Dokumentes definiert. Es dürfen beliebig viele STYLE-Elemente in der Header-Sektion vorkommen.

In CSS können Regeln mit einem STYLE-Element Regeln für

- alle Instanzen eines bestimmten HTML-Elements (z.B. alle **P**-Elemente, alle **H1**-Elemente, usw.),
- alle Instanzen eines bestimmten HTML-Elementes, die einer speziellen Klasse zugehörig sind (entsprechend dem **class**-Attribut) oder
- einzelne Instanzen eines HTML-Elements (entsprechend dem **id**-Attribut)

gesetzt werden.

Im folgenden Beispiel erhält jedes H1-Element, das der Klasse "myclass" zugehörig ist, einen Rahmen:

```
<HEAD>
  <STYLE type="text/css">
    H1.myclass {border-width: 1; border: solid}
  </STYLE>
</HEAD>
<BODY>
  <H1 class="myclass"> Ich habe einen Rahmen </H1>
  <H1> Ich habe keinen Rahmen </H1>
</BODY>
```

8.3 Media Types

Mit dem **media**-Attribut kann festgelegt werden, für welches Zielmedium das Style Sheet gedacht ist. Auf diese Weise können Internetbrowser selektiert Style Sheets laden und anwenden.

Im folgenden Beispiel werden alle H1-Elemente blau angezeigt, sofern sie mit einem Projektor (bei einem Vortrag beispielsweise) dargestellt werden. Bei einem Ausdruck werden sie zentriert ausgegeben:

```
<STYLE type="text/css" media="projection">
  H1 { color: blue}
</STYLE>

<STYLE type="text/css" media="print">
  H1 { text-align: center }
</STYLE>
```

8.4 Externe Style Sheets

Externe Style Sheets bieten eine Reihe von Vorteilen:

- Es können die gleichen Style Sheets für mehrere Dokumente verwendet werden.
- Es können Style Sheets geändert werden, ohne das HTML-Dokument ändern zu müssen.
- Internetbrowser können Zeit sparen, indem sie nur die benötigten Style Sheets laden (entsprechend dem **media**-Attribut).

Externe Style Sheets werden entweder mit dem HTTP-Link Header oder dem LINK-Attribut eingebunden.

Cascading Style Sheets können, wie der Name schon sagt, geschachtelt werden. Die Style Regeln werden in der Reihenfolge geschachtelt, wie sie in der Header-Sektion auftauchen.

Beispiel:

```
<LINK rel="stylesheet" media="screen" href="screen.css"
type="text/css">
<LINK rel="stylesheet" media="print" href="print.css"
type="text/css">
<LINK rel="stylesheet" href="techreport.css" type="text/css">
```

(In diesem Beispiel gilt das "techreport" Style Sheet für alle Medientypen.)

Weiterhin ist es möglich, mit dem **rel**-Attribut alternative Style Sheets anzugeben (`rel="alternate stylesheet"`) . In diesem Fall hätte der Betrachter dann die Möglichkeit, sich aus einer Liste von Style Sheets das gewünschte herauszusuchen.

9 Frames

Mit Frames kann der Inhalt eines HTML-Dokumentes in verschiedenen unabhängigen Fenstern dargestellt werden. Ein sogenanntes *Frameset Dokument* ist im Vergleich zu Standard-Dokumenten anders aufgebaut. An die Stelle des **BODY**-Elements tritt hier das **FRAMESET**-Element.

9.1 FRAMESET Element

Attribute:

`rows = multi-length-list`
Dieses Attribut ist eine Liste, welche die Größen der horizontalen Frames bestimmt.

`cols = mulit-length-list`
Dieses Attribut ist eine Liste, welche die Größen der vertikalen Frames bestimmt.

Mit dem FRAMESET-Element wird die Unterteilung des Hauptfensters in Form von rechteckigen Feldern *(sogenannten Frames)* bestimmt. Diese Felder können beliebig weiter unterteilt werden, indem mehrere FRAMESET-Elemente geschachtelt werden.

Die einzelnen Frames werden nun von links nach rechts in der ersten Reihe, von links nach rechts in der zweiten Reihe usw. mit Hilfe des FRAME-Elements aufgebaut.

9.2 FRAME Element

Attribute:

`name = cdata`
Dieses Attribut legt einen Namen für das Frame fest.

`longdesc = uri`
Dieses Attribut bestimmt einen Link zu einer längeren Beschreibung des Frames (geeignet für nichtvisuelle Browser).

`src = uri`
Dieses Attribut bestimmt den Ort des ursprünglichen Inhalts, der in dem Frame dargestellt werden soll.

`noresize`

Dieses boolesche Attribut legt fest, daß die Größe des Frames nicht geändert werden darf.

`scrolling = auto | yes | no`

Dieses Attribut bestimmt, ob Scroll-Mechanismen zur Verfügung gestellt werden sollen, sofern es nötig ist (auto). Bei "yes" werden immer Scroll-Mechanismen zur Verfügung gestellt, bei "no" niemals.

`frameborder = 1 | 0`

Dieses Attribut bestimmt, ob ein Rand gezeigt (1) oder nicht gezeigt (0) werden soll.

`marginwidth = pixels, marginheigth = pixels`

Mit diesen Attributen läßt sich bestimmen, wieviel Platz zwischen dem Frame-Inhalt und den Rändern gelassen werden soll.

Beispiel für ein Frameset-Dokument:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
   "_THE_LATEST_VERSION_/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset dokument</TITLE>
</HEAD>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
    <FRAME src="contents_of_frame1.html">
    <FRAME src="contents_of_frame2.gif">
  </FRAMESET>
  <FRAME src="contents_of_frame3.html">
  <FRAME src="contents_of_frame4.html">
</FRAMESET>
</HTML>
```

9.3 target Attribut

Mit dem **target**-Attribut läßt sich der Name eines Frames (enstsprechend dem **name**-Attribut) angeben, in dem ein Dokument geöffnet werden soll. Das **target**-Attribut läßt sich für Elemente setzen, die Links (A, LINK), Image Maps (AREA) oder Forms (FORM) kreieren.

Es ist auch möglich, daß **target**-Attribut innerhalb des **BASE**-Elements zu setzen, wodurch dann ein Standard-Zielframe für alle Links des Dokumentes festgelegt wird.

9.4 NOFRAMES Element

Mit dem **NOFRAMES**-Element läßt sich der Dokumenteninhalt bestimmen, der angezeigt werden soll, wenn ein Internetbrowser keine Frames unterstützt (oder konfiguriert wurde, keine Frames anzuzeigen).

9.5 IFRAME Element

Mit dem **IFRAME**-Element lassen sich *Inline Frames* innerhalb eines Textblocks eines HTML-Dokuments einfügen. Das Einfügen eines *Inline Frames* wird ähnlich gehandhabt wie das Einfügen eines *Objekts* [S. 154]. Der Inhalt des Frames wird wieder durch das **src**-Attribut bestimmt. Der Inhalt des **IFRAME**-Elements darf jedoch nur angezeigt werden, wenn ein Browser keine Frames unterstützt.

10 Formulare

Für die aus HTML 2.0 bekannten Elemente zur Definition von Eingabefeldern (*Forms*) gibt es in HTML 4.0 ein Reihe von neuen Attributen:

10.1 tabindex Attribut

Dieses Attribut bestimmt die Position der *Tabbing Order* eines Eingabe-Elements. Um einen Wert eingeben zu können, muß der Benutzer entweder das Element mit der Maus anklicken, oder er drückt solange die Tab-Taste, bis das gewünschte Eingabefeld aktiviert ist. Die Reihenfolge, in der die einzelnen Eingabefelder aktiviert werden, hängt von der *Tabbing Order* ab, wobei jeweils das Element mit dem niedrigsten **tabindex**-Wert als nächstes an der Reihe ist.

10.2 accesskey Attribut

Ein Eingabe-Element läßt sich ebenfalls mit einem *Access-Key* aktivieren. Diese Taste läßt sich mit dem **accesskey**-Attribut bestimmen.

10.3 disabled Attribut

Mit diesem Attribut läßt sich ein Eingabefeld unwirksam machen. Beispielsweise soll ein "Submit"-Button erst aktiviert werden können, wenn alle Eingaben getätigt wurden. Die einzige Möglichkeit, ein solches Element wieder aktivieren zu können, ist die Anwendung eines *Skripts*.

10.4 readonly Attribut

Ist dieses boolesche Attribut gesetzt, so kann der Eingabewert nicht verändert werden. Auch diese Eigenschaft kann nur durch ein *Skript* aufgehoben werden.

10.5 OPTGROUP Element

Mit diesem neuen Element lassen sich Menüoptionen in einem SELECT-Element gruppieren. Dabei läßt sich der Name einer Menügruppe durch das **label**-Attribut festlegen. Somit läßt sich eine gewisse Struktur in ein Menü bringen, was die Bedienerfreundlichkeit erhöht.

10.6 LABEL Element

Einige Eingabeelemente werden automatisch mit *Labels* versehen (Buttons), die meisten jedoch nicht (Textfelder, Checkboxen, etc.).

Mit dem LABEL-Element können nun explizit Labels für solche Eingabefelder bestimmt werden.

Beispiel:

```
<LABEL for="vname">Vorname</LABEL>
<INPUT type="text" name="vorname" id="vname">
```

Mit dem **for**-Attribut wird das Element bestimmt, für das das Label gelten soll (es bezieht sich auf das **id**-Attribut).

10.7 FIELDSET und LEGEND Elemente

Mit dem FIELDSET-Element können ähnlich wie mit dem OPTGROUP-Element thematisch zusammengehörige Eingabefelder gruppiert werden. Das LEGEND-Element gibt hierbei eine kurze Information über eine Gruppe an (speziell gedacht für nicht-visuelle Browser). Insbesondere läßt sich die visuelle Gestaltung einer Eingabegruppe verbessern, wenn FIELDSET-Elemente mit Style Sheet Regeln versehen werden.

Beispiel:

```
<FIELDSET>
  <LEGEND>Personal Information</LEGEND>
  Last Name: <INPUT name="lastname" type="text">
  First Name: <INPUT name="firstname" type="text">
</FIELDSET>
<FIELDSET>
  <LEGEND>Medical History</LEGEND>
```

```
  <INPUT name="history_illness"
         type="checkbox"
         value="Smallpox"> Smallpox
  <INPUT name="history_illness"
         type="checkbox"
         value="Mumps"> Mumps
</FIELDSET>
```

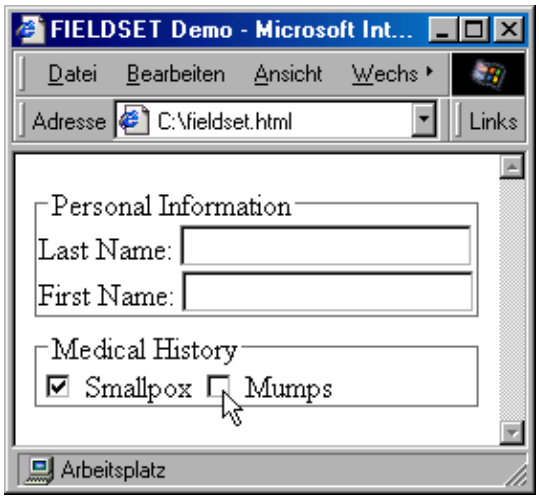


Abbildung 10-1: Fieldset

Eine weitere Neuerung in HTML 4.0 für Eingabefelder ist eine Reihe von Attributen, die in Zusammenghang mit *Skripten* verwendet werden können, um beispielsweise die Eingabedaten automatisch überprüfen zu können.

11 Skripte

Skripte sind Programme, die auf dem Rechner des Internetbrowsers ausgeführt werden, beispielsweise wenn ein Dokument geladen oder ein Link aktiviert wird. Die Skript-Unterstützung von HTML 4.0 ist unabhängig von der Skriptsprache.

Skripte lassen sich für verschiedene Aufgaben einsetzen:

- Skripte können den Inhalt eines Dokuments dynamisch verändern.
- Skripte können dazu verwendet werden, den Inhalt von Eingabefeldern zu überprüfen.
- Skripte können durch bestimmte Ereignisse gestartet werden (Mausbewegungen, etc.).

Es gibt zwei Typen von Skripten, die an ein HTML-Dokument angefügt werden können:

- Solche, die ein einziges Mal ausgeführt werden, wenn das Dokument geladen wird. Diese Skripte werden durch das **SCRIPT**-Element in ein HTML-Dokument eingefügt.
- Solche, die immer dann ausgeführt werden, wenn ein bestimmtes Ereignis eintritt. Diese Skripte können mit einer Reihe von Elementen mit Hilfe der **intrinsic event** Attribute verknüpft werden.

11.1 SCRIPT Element

Attribute:

`src = uri`
Dieses Attribut bestimmt den Ort eines externen Skripts.

`type = content-type`
Dieses Attribut bestimmt die Skriptsprache.

`defer`
Dieses boolesche Attribut gibt einen Hinweis darauf, daß das Skript keinen Dokumentinhalt generieren wird (z.B. kein "document.write" in Javascript). Somit kann der Browser fortfahren, das übrige Dokument darzustellen.

Das SCRIPT-Element darf beliebig oft in der Header- oder Body-Sektion eines HTML-Dokuments auftauchen. Das Skript selbst wird entweder im **SCRIPT**-Element direkt oder in einer externen Datei (mit Hilfe des **src**-Attributs) definiert.

Eine Standardskriptsprache für alle Skripte eines Dokuments kann mit der folgenden **META**-Deklaration im **HEAD**-Element festgelegt werden:

```
<META http-equiv="Content-Script-Type" content="type">
```

Hierbei bezeichnet `"type"` die Skriptsprache (z.B. "text/tcl", "text/javascript", etc.).

Alternativ kann die Deklaration im HTTP-Header stattfinden:

```
Content-Script-Type: type
```

11.2 Intrinsic Events

Eine andere Möglichkeit, Skripte in HTML-Dokumente einzubinden, besteht darin, sie mit einzelnen Elementen des Dokumentes zu verknüpfen. Dies geschieht mit Hilfe der **Intrinsic Events** Attribute, die für verschiedene Elemente gesetzt werden können. Der Bereich der Intrinsic Events wird laufend weiterentwickelt von der *W3C Document Object Model Working Group* (weitere Informationen sind unter http://www.w3.org/ zu finden).

Hier eine kleine Auswahl:

`onclick`
Dieses Ereignis tritt ein, wenn auf ein Element geklickt wird.

`ondblclick`
Dieses Ereignis tritt ein, wenn ein Doppelklick auf ein Element stattgefunden hat.

`onmouseover`
Dieses Ereignis tritt ein, wenn der Mauszeiger über ein Element bewegt wird.

`onsubmit`
Dieses Ereignis tritt ein, wenn die Eingabe eines Eingabefeldes abgeschickt wurde. Dieses Attribut findet nur beim FORM-Element Anwendung.

usw.

Beispiel:

```
<A href="http://www.somesite.com/index2.html"
 onmouseover="window.document.picture.src=´enter_bright.gif´"
 onmouseout="window.document.picture.src=´enter_dark.gif´">
 <IMG name=picture src="enter_dark.gif">
</A>
```

In diesem Beispiel werden Ereignisse des **A**-Elements abgefragt. Zunächst wird das Bild "enter_dark.gif" dargestellt (z.B. ein grafischer "Enter"-Button, um auf die eigentliche Seite zu gelangen). Wird der Mauszeiger über das Bild bewegt, erscheint er heller. (Das Bild wird durch "enter_bright.gif" ausgetauscht.) Verläßt der Mauszeiger wieder den Bereich des Bildes, dann tritt das Ereignis **onmouseout** ein und das ursprüngliche Bild wird wieder angezeigt.

11.3 NOSCRIPT Element

Der Inhalt des **NOSCRIPT**-Elements wird dargestellt, wenn der Internetbrowser keine Skripts unterstützt (oder konfiguriert wurde, keine Skripts auszuführen).

Beispiel:

```
<SCRIPT type="text/tcl">
  ... ein TCL Skript um Daten einzufügen ...
</SCRIPT>
<NOSCRIPT>
  <P>Access the <A href="http://someplace.com/data">data.</A>
</NOSCRIPTY>
```

12 Document Type Definition

HTML ist eine SGML-Applikation und wird durch die Document Type Definition (DTD) definiert.

Hier ein Auszug der DTD, in dem die HTML-Elemente definiert werden, die der Erstellung von Frameset-Dokumenten dienen:

```
<!--================== Document Frames ===================================-->

<!--  The content model for HTML documents depends on whether the HEAD is
  followed by a FRAMESET or BODY element. The widespread omission of
  the BODY start tag makes it impractical to define the content model
  without the use of a marked section.
-->

<!-- Feature Switch for frameset documents -->
<!ENTITY % HTML.Frameset "IGNORE">

<![ %HTML.Frameset; [
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?) -- window subdivision-->
<!ATTLIST FRAMESET
  %coreattrs;                           -- id, class, style, title --
  rows        %MultiLengths; #IMPLIED   -- list of lengths,
                                           default: 100% (1 row) --
  cols        %MultiLengths; #IMPLIED   -- list of lengths,
                                           default: 100% (1 col) --
  onload      %Script;       #IMPLIED   -- all the frames have been loaded  --
  onunload    %Script;       #IMPLIED   -- all the frames have been removed --
  >
]]>

<![ %HTML.Frameset; [
<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY               -- subwindow --><!ATTLIST FRAME
  %coreattrs;                           -- id, class, style, title --
  longdesc    %URI;          #IMPLIED   -- link to long description
                                           (complements title) --
  name        CDATA          #IMPLIED   -- name of frame for targetting --
  src         %URI;          #IMPLIED   -- source of frame content --
  frameborder (1|0)          1          -- request frame borders? --
  marginwidth %Pixels;       #IMPLIED   -- margin widths in pixels --
  marginheight %Pixels;      #IMPLIED   -- margin height in pixels --
  noresize    (noresize)     #IMPLIED   -- allow users to resize frames? --
  scrolling   (yes|no|auto)  auto       -- scrollbar or none --  >
]]>

<!ELEMENT IFRAME - - (%flow;)*          -- inline subwindow --><!ATTLIST IFRAME
  %coreattrs;                           -- id, class, style, title --
```

```
  longdesc    %URI;          #IMPLIED  -- link to long description
                                          (complements title) --
  name        CDATA          #IMPLIED  -- name of frame for targetting --
  src         %URI;          #IMPLIED  -- source of frame content --
  frameborder (1|0)          1         -- request frame borders? --
  marginwidth %Pixels;       #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels;      #IMPLIED  -- margin height in pixels --
  scrolling   (yes|no|auto)  auto      -- scrollbar or none --
  align       %IAlign;       #IMPLIED  -- vertical or horizontal alignment --
  height      %Length;       #IMPLIED  -- frame height --
  width       %Length;       #IMPLIED  -- frame width --
  >

<![ %HTML.Frameset; [
<!ENTITY % noframes.content "(BODY) -(NOFRAMES)">
]]>

<!ENTITY % noframes.content "(%flow;)*">

<!ELEMENT NOFRAMES - - %noframes.content;
 -- alternate content container for non frame-based rendering -->
<!ATTLIST NOFRAMES
  %attrs;                              -- %coreattrs, %i18n, %events --
>
```

13 Literatur

[BeCo95] Berners-Lee, T. - Conolly, D.: Hypertext Markup Language – 2.0, Internet Draft http://www.w3.org, 22. September 1995

[RLHJ97] Raggett, D. - Le Hors, A. - Jacobs, I.: HTML 4.0 Specification, W3C Recommendation, http://www.w3.org, 18. Dezember 1997

HTML – Eine Beispielanwendung

Thorsten Bludau

FB Informatik, Uni Dortmund
tbludau@ispro.de

Zusammenfassung

Dieser Vortrag ist Bestandteil des Seminars "Grundlagen der Markup-Sprachen des Internets" an der Universität Dortmund im Wintersemester 1998/99 und beschäftigt sich mit einer konkreten Anwendung der Markup-Sprache HTML. Als Beispiel hierfür wurde das Seminar selbst implementiert. Im Folgenden sollen die Probleme, die sich bei der Realisierung diese Projektes ergeben haben, mit entsprechenden Lösungsstrategien vorgestellt werden. Dabei ist zu beachten, daß der hier dargestellte Lösungsweg nur einer von vielen möglichen ist.

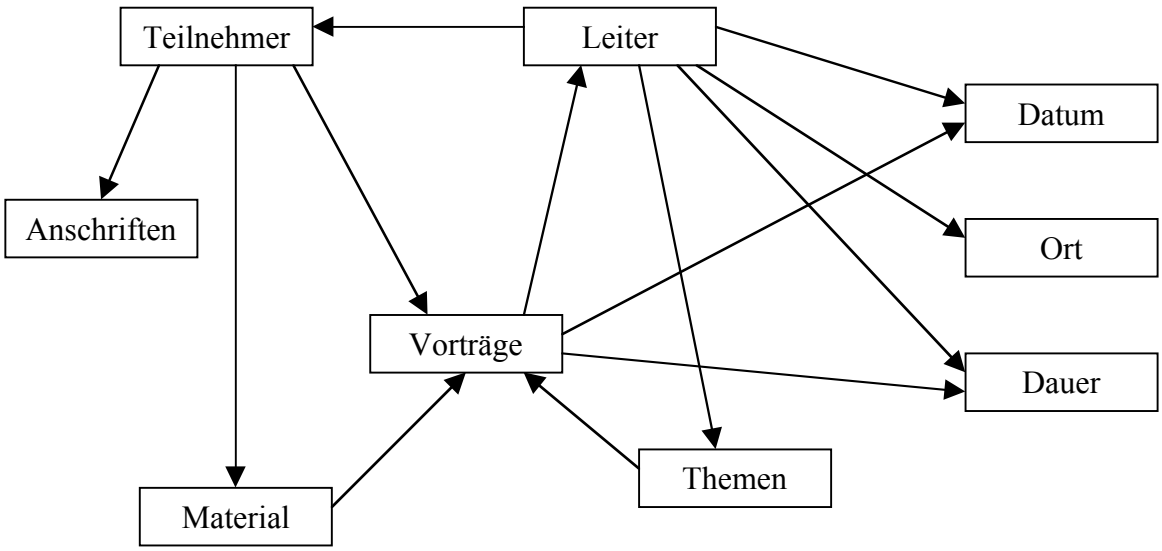
Kapitel 1 beschäftigt sich mit der theoretischen Seite des gegebenen Projektes und zeigt zu den gewünschten Anforderungen Lösungsansätze, die später dann auch mit HTML-Befehlen realisiert werden können. Anschließend werden die theoretisch erarbeiteten Ergebnisse in Kapitel 2 Schritt für Schritt umgesetzt, bis dann in Kapitel 3 das Ergebnis präsentiert werden kann. Den Abschluß dieses Vortrages bildet Kapitel 4, in dem eine kurze Übersicht über die bei der Erstellung verwendeten Tools (Editor, Grafikprogramme, etc.) erfolgt.

1 Die Planung des Projektes

1.1 Was macht ein Seminar aus ?

Bevor man anfängt sich Gedanken über die Implementierung des Seminars als HTML-Anwendung zu machen, ist es zunächst einmal sinnvoll zu definieren, was ein Seminar eigentlich ausmacht.

Ein Seminar besteht aus vielen kleinen Einzelkomponenten, die untereinander in Wechselbeziehungen stehen. Diese Aufgliederung kann man grafisch wie folgt darstellen:



Die Pfeile weisen dabei auf die Wechselbeziehungen hin. So bestimmt zum Beispiel der Leiter die Teilnehmer, die wiederum Material sichten, um daraus Seminarvorträge zu erstellen und diese anschließend dem Leiter zu präsentieren.

Es sei an dieser Stelle darauf hingewiesen, daß diese Unterteilung keinesfalls die einzig mögliche ist. Die Anzahl der Komponenten, sowie die Genauigkeit der Informationen (z.B. Straße, PLZ, Ort anstatt Anschriften) hängt ganz vom jeweiligen Betrachter ab und dient an dieser Stelle lediglich dem Zweck, einen groben Überblick über die gegebenen Grundinformationen zu erhalten.

1.2 Sichten der Grunddaten

Nachdem das Seminar in seine Bestandteile aufgegliedert ist, kann man feststellen, daß hier drei Arten von Informationen vorliegen. Es gibt allgemeine, seminar-spezifische und personelle Informationen. Diesen Informationstypen werden in der folgenden Abbildung die in Kapitel 1.1 ermittelten Grundinformationen zugeordnet:

Allgemeine	**Seminar-Spezifische**	**Personelle**
• Leiter • Ort • Datum • Dauer	• Vorträge • Themen • Material	• Teilnehmer • Anschriften

Auch hier gilt, daß diese Zuordnung nicht **DIE** Zuordnung ist. Es wäre ebenso denkbar, daß zum Beispiel die Angaben bezüglich des Leitenden auch unter die Rubrik "Personelle Informationen" fallen können.

1.3 Bewertung der Informationen

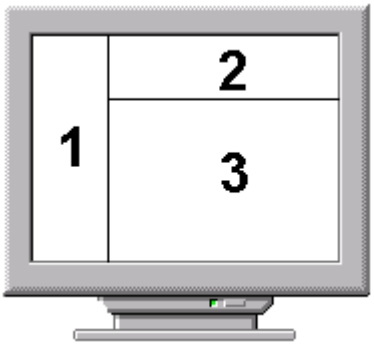
Desto umfangreicher ein Projekt ist, desto mehr Gedanken muß man sich über dessen Struktur und die darin enthaltenden Informationen machen. Diese Gedanken bilden den Grundstein für eine koordinierte und effiziente Implementierung. Es nützt dem Entwickler wenig, wenn er einfach anfängt zu programmieren und plötzlich feststellt, daß das Projekt mit seinen Vorstellungen oder Möglichkeiten nicht zu realisieren ist. In diesem Fall können wir aus den bisher erhaltenen Informationen folgende Rückschlüsse für die Implementierung ziehen:

Die Fülle der Daten läßt das Erstellen einer einzigen HTML-Seite nicht zu, da der Anwender gezwungen wäre, den gesamten Seiteninhalt, der sich durchaus über mehrere Bildschirme erstrecken kann, zu lesen, um seine gewünschten Informationen zu finden. Deshalb ist es notwendig, mehrere HTML-Seiten zu erstellen, die jeweils inhaltlich verwandte Daten enthalten. Um dem Anwender den Zugriff auf diese Seiten zu ermöglichen, ist es ratsam eine Menüstruktur zu erarbeiten, die Verweise auf die verschiedenen Seiten beinhaltet und außerdem kontinuierlich sichtbar ist. Die Verweise selbst werden hierbei später grafisch dargestellt. Es ist jedoch darauf zu achten, daß die Anzahl der Verweise eine realistische Größe darstellt, um den Anwender zum Beispiel durch eine zu große Anzahl von Menüpunkten nicht zu irritieren. Für das Seminar werden wir die folgenden sieben Menüpunkte verwenden, deren Inhalt zu einem späteren Zeitpunkt genauer erläutert wird:

- Allgemein
- Themen
- Referenten
- Download
- Links
- Quellen
- E-Mail

1.4 Das Menü

Wie schon erwähnt, soll das Menü permanent sichtbar sein. Da wir jedoch gleichzeitig auch die Seminarinformationen darstellen wollen, müssen wir den Bildschirm in verschiedene Bereiche aufteilen, denen wir anschließend die gewünschten Informationen zuweisen. Diese Bereiche werden später dann durch die von HTML zur Verfügung gestellte Frame-Struktur realisiert, die eine solche Aufteilung des Bildschirms in sogenannte Frames ermöglicht. Die nachstehende Abbildung zeigt die für das Projekt verwendete Aufteilung:



Frame-Nr.	Inhalt
1	Menü
2	Allgemeine Informationen (Überschriften, Seminartitel, o.ä.)
3	Spezielle, auf die Menüauswahl des Anwenders bezogene Informationen

Abbildung 1-1: Die Bildschirmaufteilung

Hierbei besitzen die einzelnen Frames bestimmte "Eigenschaften", die im Folgenden näher erläutert werden sollen.

1.4.1 Frame Nr.1 – Das Menü

Er enthält das komplette Menü, über das der Anwender sich durch die HTML-Beispielanwendung bewegt. Der Anwender wählt hierbei per Mausklick einen der Menüpunkte aus, woraufhin in Frame 3 die gewünschten Informationen angezeigt werden. Der Inhalt des Menü-Frames darf sich dabei nicht verändern, da das Menü die ganze Zeit sichtbar und verfügbar sein muß.
Hier nun eine kurze Übersicht über die Menüpunkte und den damit verbundenen Informationen:

Nr.	Titel	späterer Inhalt
1	Allgemein	Informationen über "Wann und wo fand das Seminar statt ?" oder "Wer hat das Seminar geleitet ?" finden sich hier
2	Themen	Das Seminar gliedert sich in 15 Sub-Themen auf, die hier mit einer kurzen Beschreibung des Inhalts vorgestellt werden sollen
3	Referenten	Um sich eine grobe Vorstellung darüber machen zu können, wer die Beiträge zu den einzelnen Themen geliefert hat, werden hier die Referenten kurz vorgestellt. Außerdem sollen hier die entsprechenden E-Mail-Adressen erscheinen, damit der Anwender im Falle einer Frage oder eines Kommentars sich direkt an die Verantwortlichen wenden kann
4	Download	Interessierte Anwender können hier die Seminar-Beiträge als Word-Dokument auf ihrem eigenen Rechner als Download speichern
5	Links	Hier finden sich Verweise auf andere Web-Seiten, die sich mit dem Seminar-Thema oder ähnlichen Themen befassen

6	Quellen	Diese Seite enthält eine Auflistung aller für das Seminar verwendeten Quellen (Bücher, Zeitschriften, etc.)
7	E-Mail	Schickt eine E-Mail an den Leiter des Seminars, um eventuelle Fragen, Kritiken oder ähnliches zu äußern

1.4.2 Frame Nr. 2 – Allgemeine Informationen

Hier soll lediglich der Titel des Seminars erscheinen.

1.4.3 Frame Nr. 3 – Spezielle Informationen

In diesem Frame werden später die vom Anwender gewünschten Informationen dargestellt. Dabei kann der Inhalt eine der oben genannten Seiten sein, die im Menü ausgewählt werden können. Aufgrund der Tatsache, daß hier die Seminar-Spezifischen Informationen dargestellt werden sollen, wird ihm der größte Teil des Bildschirms zugewiesen.

2 Die Implementierung

2.1 Der Zeitplan

Es wäre sicherlich falsch, sich nur Gedanken über die Inhalte der verschiedenen Frames und Seiten zu machen. Ebenso wichtig ist ein gut durchdachter Ablauf der Implementierung. Das Erstellen von HTML-Anwendungen bietet die angenehme Möglichkeit, sich schon während der Implementierung zu beliebigen Zeitpunkten einen Eindruck vom bisherigen Ergebnis zu machen. Wir werden die Seiten jedoch nicht in einer beliebigen Reihenfolge erstellen, sondern folgendem Ablaufschema folgen: Erst die Unterteilung des Bildschirms in Frames, dann das Menü und anschließend die zu dem Menü gehörigen Seiten. Das Ganze sieht dann etwa wie folgt aus:

Erstellen der Seite	Für Frame Nr.	Kommentar
index.html	1 / 2 / 3	Im Normalfall gibt der Anwender lediglich eine Adresse wie zum Beispiel "www.uni-dortmund.de" an. Dabei wird standardmäßig die Datei "index.html" geöffnet, so daß lediglich die oben genannte Adresse bekannt sein muß, aber nicht der explizite Name der Start-Datei (was bei der Anzahl von möglichen Namensgebungen auch ziemlich ungeschickt wäre). Inhaltlich wird hier die Aufteilung des Bildschirms in drei Frames, sowie die Zuordnung von Namen für diese Frames vorgenommen
menu.html	1	Hier wird das Menü eingebunden. Die frühzeitige Einbindung dieser Datei hat den Vorteil, daß der Entwickler schon während der Implementierung die Möglichkeit hat, seine Zwischenresultate zu betrachten und gegebenenfalls zu ändern
titel.html	2	Wie schon am Anfang beschrieben, besteht hier die Möglichkeit diverse Informationen anzuzeigen. Dies wird in unserem Fall der Titel des Seminars sein, damit der Anwender die ganze Zeit hindurch weiß, in welcher

		Umgebung er sich befindet
allgemein.html themen.html referenten.html download.html links.html quellen.html	3	Hinter diesen Seiten verbergen sich die in Kapitel 1.4.1 beschriebenen Inhalte. Hat der Anwender sich per Mausklick für den entsprechenden Link im Menü entschieden, so wird der aktuelle Inhalt des dritten Frames mit dem Inhalt der gewünschten Seite überschrieben. Wird die Beispielanwendung gestartet, so wird in Frame 3 zunächst einmal immer die Seite "allgemein.html" dargestellt

2.2 Allgemeines

Die folgenden Seiten beschäftigen sich nun mit der Implementierung des Seminars als HTML-Anwendung. Zuvor sollte man sich jedoch einige grundlegende Dinge kurz vor Augen halten:

Die Vorgehensweise für diese HTML-Anwendung ist nur eine beispielhafte und schließt andere (ebenso "richtige") Methoden nicht aus

Der für die Anwendung verwendete Code umfaßt nicht den kompletten HTML-Sprachumfang, da es nicht Aufgabe dieses Vortrages ist, eine Einführung in diese Sprache zu geben

Es werden keine weiteren Hilfsmittel wie z.B. Java, Java-Script, DHTML, o.ä. verwendet

Insgesamt werden zwei wichtige Teilaspekte von HTML-Anwendungen behandelt:

Erstellen einer "Standard"-Seite, wie sie oft im Internet gefunden werden kann

Arbeiten mit komplexeren Strukturen (Frames, Tabellen und Verweisen/Links)

Bevor wir uns mit der konkreten Implementierung der einzelnen Seiten beschäftigen, sei nochmal kurz an die grundlegende Struktur einer HMTL-Seite erinnert. Diese besteht aus einem HTML-Tag, der das gesamte Dokument umschließt, einem Head-Abschnitt, in dem zum Beispiel mittels des TITLE-Tags eine Information in den Fensterrahmen oben links geschrieben werden kann und einen Body-Abschnitt, der später den eigentlichen Seiteninhalt enthält.

Das Ganze sieht dann wie folgt aus:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Bei der Verwendung von Frames ist darauf zu achten, daß anstelle des BODY-Tags der FRAMESET-Tag verwendet werden muß

2.3 Die Aufteilung in Frames

Unsere gewünschte Aufteilung in drei unabhängige Bildschirmbereiche nehmen wir, wie schon erwähnt, in der Datei "index.html" vor. Dabei teilen wir das Browser-Fenster zunächst vertikal in zwei Frames. Der linken Seite weisen wir eine relative Breite von 20% der insgesamt zur Verfügung stehenden Breite zu und geben ihr den Namen "menu". Die rechte Seite erhält die restlichen 80% und

wird als "rechts" bezeichnet. Der entsprechende HTML-Code sieht dann wie folgt aus (die drei Punkte weisen hier und im Weiteren auf Quell-Code hin, der an dieser Stelle nicht relevant ist):

```
...
<frameset cols="20%,80%" frameborder="no">
<frame name="menu" frameborder="no" src="menu.html" scrolling="yes">
<frame name="rechts" frameborder="no" src="menu.html" scrolling="yes">
</frameset>
...
```

Die Attribute des FRAME-Tags haben hierbei folgenden Effekt:

- *name* weist dem Frame eine Bezeichnung zu, mit derer Hilfe später einem Link mitgeteilt wird, in welchem Frame er geöffnet werden soll
- *frameborder* gibt an, ob der Frame einen Rahmen erhalten soll, oder nicht
- *src* hier steht der Name der Seite, die in dem Frame dargestellt werden soll
- *scrolling* schaltet die Scroll-Leisten im Frame an oder aus

Um die von uns gewünschte Aufteilung in drei Bereiche zu erhalten, muß abschließend der Frame mit der Bezeichnung "rechts" nochmals horizontal aufgeteilt werden. Dies geschieht durch eine geschachtelte FRAMESET-Struktur, bei der der rechte Frame durch zwei Frames mit den Bezeichnungen "titel" und "allgemein" ersetzt wird. Diesen wird eine relative Höhe von 12% ("titel") und 88% ("allgemein") der insgesamt zur Verfügung stehenden Höhe zugewiesen. Die fertig gestellte "index.html" hat dann folgende Gestalt:

```
...
<frameset cols="20%,80%" frameborder="0">
<frame name="menu" frameborder="no" src="menu.html" scrolling="yes">
<frameset rows="12%,88%">
<frame name="titel" frameborder="no" src="titel.html"  scrolling="no">
<frame name="inhalt" frameborder="no" src="allgemein.html" scrolling="yes">
</frameset>
</frameset>...
...
```

Bevor wir uns dem Menü zuwenden, möchte ich kurz auf ein Problem eingehen, welches zuvor noch nicht behandelt wurde.

Wann immer man mit Tabellen, Frames oder ähnlichen Gebilden arbeitet, bieten sich für die entsprechenden Größenangaben zwei Möglichkeiten.

Zum Einen kann der Entwickler zum Beispiel die Breite einer Tabelle in absoluten Werten (in Pixeln) angeben. Diese Methode bietet jedoch ausschließlich Nachteile, da nicht jeder Anwender mit derselben Bildschirmauflösung arbeitet. Werden zum Beispiel für eine Tabellenbreite 700 Pixel vorgegeben und der Anwender zu Hause hat nur eine Bildschirmauflösung von 640x480 Pixel eingestellt, schneidet der Browser die überstehenden 60 Pixel einfach ab. Deshalb sollte man ausschließlich die zweite Variante verwenden, bei der solche Größen in relativen Werten (in Prozent) angegeben werden. Dabei beziehen sich diese Werte immer auf die Größe des Frames, in dem zum Beispiel die Tabelle erscheinen soll. Mittlerweile bieten jedoch zum Beispiel Java oder Java-Script die Möglichkeit, Parameter wie Bildschirmauflösung, Farbtiefe oder Browserversion abzufragen, und dann Seiten zu öffnen, die entsprechend für bestimmte Bildschirmauflösungen oder verschiedene Internet-Browser geschrieben wurden. Da wir uns hier jedoch nur mit HTML-Befehlen beschäftigen, wählen wir im Folgenden ausschließlich den Weg der relativen Größenangaben. Desweiteren wird diese Beispielanwendung so implementiert, daß man die besten Ergebnisse bei einer Auflösung von 800x600 Bildpunkten erreicht.

2.4 Die Datei "menu.html"

Nachdem die Grundstruktur der HTML-Anwendung steht, ist es nun an der Zeit, die einzelnen Seiten, die in den Frames erscheinen sollen, zu implementieren.

Als erstes wird dabei die Datei "menu.html" erstellt, in welcher das Menü mit den Verweisen auf die einzelnen Seiten realisiert wird. Diese Verweise müssen in den BODY-Teil der Grundstruktur aus Kapitel 2.2 eingefügt werden. Die Verweise werden in dieser Beispielanwendung als Grafiken in Form von ovalen Schaltflächen dargestellt.

Zunächst erhält das Menü jedoch ein Hintergrundbild, indem der BODY-Tag um das Attribut *background*="hintergrund.jpg" erweitert wird. In diesem Fall sieht der Hintergrund wie folgt aus:

Abbildung 2-1: Der Menühintergrund

Dabei fällt auf, daß der linke Teil der Grafik unter Umständen verhindert, daß Objekte, die in der Seite eingefügt werden, sichtbar sind, weil die Hintergrundfarbe der Objektfarbe entspricht. So könnte zum Beispiel das Einfügen eines Links, der durch eine Grafik dargestellt werden soll, folgenden Effekt verursachen:

Abbildung 2-2: Das Problem

Eine Lösung dieses Problems besteht zum Beispiel im Einfügen einer gewissen Anzahl von Leerzeichen, was allerdings dazu führen kann, daß die Objekte nicht genau untereinander stehen. Da es auch keine HTML-Befehle zum Ansprechen von Bildschirmpositionen und auch keine Tabulatoren gibt, ist der Entwickler gezwungen, auf einen kleinen Trick zurückzugreifen. Dieser Trick besteht darin, eine Tabelle mit zwei Spalten zu erstellen. Dabei umfaßt die erste Spalte genau den kritischen Bereich der Hintergrundgrafik und enthält lediglich ein Leerzeichen. Die zweite Spalte hingegen umfaßt den restlichen Bereich, in dem der Entwickler seine Objekte (in diesem Fall die durch Grafiken dargestellten Verweise) anordnen will. Wir haben also folgende Situation geschaffen:

Abbildung 2-3: Die Lösung

Jetzt ist es kein Problem mehr, die Grafiken zum Beispiel über das TABLE-Attribut *align* in der rechten Tabellenspalte untereinander auszurichten. Das Einfügen der Grafiken geschieht mit der Befehlszeile

<img src="allgemein.gif" name="Allgemein" alt="Allgemein" border="0">

Die hier verwendeten Attribute sind:

- *src* gibt den Namen (evtl. inklusive Verzeichnisangabe) der Grafikdatei an
- *name* stellt den hier angegebenen Wert anstelle der Grafik dar, falls der Browser keine Grafiken anzeigen kann, oder aber falls der Anwender diese Option explizit ausgeschaltet hat
- *alt* gibt den angegebenen Wert in einem kleinen Kästchen über dem Mauszeiger aus, falls dieser auf die Grafik geführt wird
- *border* weist der Grafik einen Rahmen bestimmter Stärke zu, wobei border="0" gleichbedeutend mit "kein Rahmen" ist

Nachdem das Menü soweit fertig gestellt ist, müssen die soeben eingefügten Grafiken noch als Verweise gekennzeichnet werden. Dies geschieht mittels des A-Tags, auf den wir aber später noch genauer eingehen. Ansonsten steht dem Abschluß des Menüs nur noch ein letztes "optisches" Problem im Wege.

Schaut man sich den aktuellen Stand der Beispielanwendung an (hierzu muß die Datei "index.html" im Browser geöffnet werden), so stellt man fest, daß die Grafiken zwar horizontal ausgerichtet sind, daß der vertikale Abstand jedoch zu gering ist, um sie gleichmäßig über die gesamte Höhe des Menüframes zu verteilen. Auch hier gibt es eine geschickte Möglichkeit, zwischen den Grafiken Leerräume zu schaffen, so daß der zur Verfügung stehende Platz im Menüframe optimal aufgeteilt wird. Dazu erstellt man eine 1x1 Pixel große GIF89a-Grafik mit als transparent gekennzeichnetem Hintergrund und erweitert die oben angegebene Befehlszeile folgendermaßen:

```
<img src="allgemein.gif" name="Allgemein" alt="Allgemein" border="0"><br>
<img src="leer.gif" height="8" border="0"><br>
```

Aufgrund der Tatsache, daß Grafiken mit den Attributen *height* und *width* skaliert werden können, ist es nicht notwendig, für jede Situation in der man auf diese Art und Weise Objekte ausrichten will, eine eigene "leere" Grafik in der entsprechenden Größe anzulegen. Desweiteren ist die Zeit zum Laden der Grafik minimal ist, da sie nur wenige Byte groß ist. Zum Abschluß stelle ich das soeben verwendete Prinzip nochmal kurz grafisch dar:

"leere" Grafik mit skalierter Höhe (hier: 8 Punkte)

Schaltfläche

"leere" Grafik mit skalierter Höhe (hier: 8 Punkte)

Schaltfläche

"leere" Grafik mit skalierter Höhe (hier: 8 Punkte)

2.5 Die Dateien "titel.html" und "allgemein.html"

Die Datei "allgemein.html" enthält allgemeine Informationen, wie zum Beispiel "Leiter des Seminars", "Ort", "Datum", etc. Sie beinhaltet keine komplexen Strukturen und dient gleichzeitig als kleines Beispiel für eine einfache HTML-Seite, wie sie sehr oft im Internet zu finden ist:

Zeile	Befehl
1	<html>
2	<head>
3	<title>Inhalt</title>
4	</head>
5	<body background="bilder/hintergrundo.jpg">
6	
7	<u>Universität Dortmund</u>
8	<img src="bilder/anilogo.gif" border="0">
9	<hr width="50%">
10	<h3>Prof. Dr. Gisbert Dittrich
11	Dipl.-Inf. Jörg Westbomke
12	Wintersemester 1998/1999 </h3>
13	
14	</body>
15	</html>

Die gesamte Seite wird von den Tags <html> und </html> in den Zeilen 1 und 15 eingeschlossen und gliedert sich dort wiederum in die beiden Teile <head>...</head> (Zeile 2-4) und <body>...</body> (Zeile 5-14) auf. Im HEAD-Abschnitt wird lediglich die Überschrift der Seite mittels des TITLE-Tags (Zeile 3) festgelegt. In diesem Fall wird in die linke obere Ecke des Browsers die Überschrift "Inhalt" geschrieben. Anschließend folgt der eigentliche im Browser darzustellende Inhalt. Er wird vom BODY-Abschnitt repräsentiert. Der Seiteninhalt besteht dann aus den Zeilen 6 bis 13 und wird aufgrund des CENTER-Tags (... in Zeile 6, bzw. Zeile 13) zentriert im Fenster ausgegeben. Dabei haben die Zeilen 7 bis 12 folgende Bedeutung:

7	Gibt eine Überschrift der Größe 2 (<h2>) aus und unterstreicht diese (<u>)
8	Fügt die Grafik "anilogo.gif" aus dem Verzeichnis "bilder" ohne einen Rahmen (border="0") ein und führt einen Zeilenumbruch ( ) durch
9	Fügt eine horizontale Linie ein, die eine Breite von 50% (width="50%") der Gesamtbreite des Frames einnimmt
10-12	Gibt die Informationen "Prof. Dr. Gisbert Dittrich", "Dipl.-Inf. Jörg Westbomke" und "Wintersemester 1998/1999" in der Überschriftengröße 3 (<h3>) an und führt nach jeder dieser Informationen zwei Zeilenumbrüche ( ) durch

Die Datei "titel.html", welche den Inhalt des Frames 2 wiedergibt, enthält nur ein Hintergrundbild, sowie das Seminarthema in der Überschriftengröße 2 (<h2>). Sie kann somit sehr schnell aus der Datei "allgemein.html" hergeleitet werden.

2.6 Die Datei "referenten.html"

Tabellen sind eine der komplexeren HTML-Strukturen und bedürfen einer genaueren Betrachtung. Das Ziel der Datei "referenten.html" ist es, eine Tabelle mit den Namen, Vornamen und E-Mail-Adressen der Referenten zu erstellen. Diese soll folgendes "Layout" besitzen:

Thema Nr.	Name	Vorname	E-Mail

Es ist also eine Tabelle mit vier Spalten gefordert. In der ersten sollen die Themen numerisch und zentriert aufgelistet werden. Die Spalten zwei bis vier enthalten jeweils den Namen, Vornamen und die E-Mail-Adresse des Referenten und sind linksbündig auszurichten. Abschließend ist noch zu bemerken, daß die Spalten im Bezug auf die Gesamtbreite von Frame 3 folgende relative Breitenwerte erhalten sollen:

- Thema-Nr., Name, Vorname je 20%
- E-Mail 40%

Zu Anfang ist es eine Überlegung wert, ob für jeden Referenten genau eine Tabellenzeile notwendig ist, oder ob alle Referenten in ein und derselben Zeile untereinander aufgeführt werden. Wir entscheiden uns hier für die letzte Variante, da sie zum einen weniger Aufwand verursacht und zum anderen denselben Effekt wie die erste Variante hat.

Da eine Tabelle zum eigentlichen Seiteninhalt zählt, muß auch sie im BODY-Abschnitt eingefügt werden und wird dort durch den TABLE-Tag eingeleitet. Weiterhin soll der Rahmen der Tabelle nicht sichtbar sein, so daß wir wir die Rahmenstärke auf 0 setzen (border="0"). Eine Tabellenzeile wird vom TR-Tag umrahmt und ist wie folgt aufgebaut:

```
<tr>
    <td width="20%" align="middle"><font size="2"><u><b>Thema Nr.</b></u></font></td>
    <td width="20%" align="left"><font size="2"><u><b>Name</b></u></font></td>
    <td width="20%" align="left"><font size="2"><u><b>Vorname</b></u></font></td>
    <td width="40%" align="left"><font size="2"><u><b>E-Mail</b></u></font></td>
</tr>
```

Dabei haben die Attribute folgende Bedeutung:

- *width* gibt die Zellenbreite an (in Prozent oder Pixeln)
- *align* legt die Ausrichtung in der Zelle fest (right, left oder middle)
- *size* gibt hier im Zusammenhang mit dem FONT-Tag die Textgröße der Überschriften an

Die Überschriften selbst werden fett (<b>) und unterstrichen (<u>) ausgegeben. Jetzt ist eine gleichsam aufgebaute Zeile einzufügen, in der die Überschriften jeweils durch die entsprechenden Werte (Name, Vorname, E-Mail) ersetzt und durch einfache Zeilenumbrüche (<br>) getrennt untereinander aufgeführt werden. Damit ist auch die "Referenten"-Seite abgeschlossen (Ergebnis s. Kapitel 3) und kann im Internet präsentiert werden.

2.7 "Interne" und "Externe" Verweise

Bevor wir uns mit der eigentlichen Implementierung der "Themen"-Seite zuwenden, möchte ich gerne auf einen zentralen Punkt der Markup-Sprache HTML etwas näher eingehen.

HTML bietet die Möglichkeit, Texte oder Grafiken als Verweise/Links zu kennzeichnen. Klickt der Anwender mit der Maus auf einen solchen Verweis, so wird eine referenzierte Datenquelle geöffnet. Dies können zum Beispiel andere HTML-Seiten, Bilder oder auch E-Mail-Adressen sein. Der Browser öffnet diese Quelle wahlweise entweder im aktuellen, oder in einem neuen Fenster.

Allgemein kann man Verweise in "interne" und "externe" Verweise aufteilen (im Folgenden sprechen wir dann nur noch von "internen" und "externen" Links):

- "Interne" Links:

  Sie verweisen auf eine andere Textstelle innerhalb desselben Dokuments, die durch eine Art Label gekennzeichnet ist.

- "Externe" Links:

  Mit ihnen bietet sich die Möglichkeit, Referenzen auf andere HTML-Dokumente, Bilder, o.ä. zu erstellen

Die "externen" Links sind hier besonders für die mit Frames realisierte Menüstruktur, sowie für die "Links"-Seite wichtig. Links werden mit dem A-Tag gekennzeichnet und verwenden das Attribut *href*, um die zu referenzierende Quelle anzugeben.

Hier nun ein Beispiel für zwei "externe" Links:

- Hier geht's weiter
- Zur Seite allgemein

Das erste Beispiel öffnet im Browser die Internetseiten der Universität Dortmund, sobald der Anwender auf den als Link gekennzeichneten Satz "Hier geht's weiter" klickt. Das zweite Beispiel hingegen würde zur Seite "allgemein.html" verzweigen. In beiden Fällen ist es jedoch wichtig zu wissen, daß die zu öffnenden Seiten in dem Fenster geöffnet werden, in dem auch der entsprechende Link steht. Die Seite mit dem Link würde also zum Beispiel im ersten Fall durch die Startseite der Universität Dortmund ersetzt. Für unser Menü hätte eine solche Konstruktion sicherlich fatale Folgen, da die Menüseite ja nicht durch eine andere Seite ersetzt werden soll. Durch die Erweiterung des A-Tags mit dem Attribut *target* ist es jedoch möglich, als Zielfenster, bzw. Zielframe, zum Beispiel den Frame "inhalt" anzugeben, so daß der in Frame 1 gewählte Link in Frame 3 geöffnet wird. Der entsprechende Quell-Code für die Links im Menü sieht dann wie folgt aus:

Download

Wichtig hierbei ist, daß in der FRAMESET-Struktur das name-Attribut für die einzelnen Frames vergeben wurde.
Die "internen" Links schließlich werden im folgenden Abschnitt aufgeführt und erläutert.

2.8 Die Datei "themen.html"

Da die Beschreibung der Seminarthemen sicherlich über mehr als eine Bildschirmseite geht, wäre es durchaus hilfreich, wenn der Anwender nicht gezwungen ist, die ganze Seite "durchzuscrollen". Dieses Problem läßt sich mit "internen" Links lösen.

Zunächst führen wir die Themenüberschriften als Links auf und verweisen dabei genau auf die Stelle im Text, an der das Thema beschrieben wird. Diese Stelle wir durch eine Art "Label" gekennzeichnet. Umgekehrt kann man anschließend nach jeder Themenbeschreibung einen weiteren "internen" Link setzen, der auf ein "Label" am Seitenanfang verweist. So ist es dem Anwender möglich, blitzschnell die gewünschte Themenbeschreibung zu finden und auch sofort wieder zum Seitenanfang zurückzukehren. Der entsprechende HTML-Code sieht dann wie folgt aus:

```
    <a href="#thema01">
(1)     <font size="2">01 – Formale Syntax-Beschreibungen</font>
    </a>
    ...
    <a name="thema01">
(2)     <font size="2">Hier folgt in Kürze eine Erläuterung zum Thema Nr.1</font>
    </a>
```

Teil (1) kennzeichnet den Link und teilt ihm mit, daß, wenn der Anwender das Thema "01 – Formale Syntax-Beschreibungen" anklickt, zum "Label" "thema01" verzweigt werden muß. Das Zeichen "#" weist hierbei auf einen "internen" Link hin.

Teil (2) kennzeichnet das "Label" selbst. Dieser Stelle wurde der Name "thema01" zugewiesen, so daß der oben aufgeführte Link seine Zielstelle im Dokument hier findet.

Das weitere Vorgehen

Bisher haben wir sechs der neun geforderten Seiten fertiggestellt. Die verbleibenden Seiten "links.html", "quellen.html" und "download.html" werden im Folgenden nicht mehr betrachtet, da sie mit bereits behandelten Methoden erstellt werden und keine neuen Strukturen enthalten. Es sei an dieser Stelle nur kurz erwähnt, daß in diesen Dateien die Links als externe Links realisiert werden und auch der Menüpunkt "E-Mail" wird als "externer" Link durch folgende Befehlszeile realisiert:

Hier können Sie eine E-Mail verschicken

Die drei Punkte müssen dabei durch die E-Mail-Adresse des Empfängers ersetzt werden. Wählt der Anwender den Satz "Hier können Sie eine E-Mail verschicken" an, so öffnet sich automatisch das E-Mail-Programm des Anwenders, um eine E-Mail an die angegebene Adresse zu schicken.

3 Das Ergebnis

Da das Ergebnis dieses Vortrages von praktischer Art ist (nämlich die implementierte HTML-Anwendung) werden im folgenden lediglich Screenshots der konkret erstellten Seiten ("allgemein.html", "themen.html", "referenten.html" und "index.html") aufgeführt. Dabei besitzt nur die "index.html" keinen direkten Inhalt, sondern ist, wie bereits erwähnt, für die Bildschirmaufteilung

zuständig. Wer sich das ganze Ergebnis dennoch nicht entgehen lassen will, sollte die Internetseiten des Lehrstuhls 1 der Universität Dortmund besuchen ("www.uni-dortmund.de").

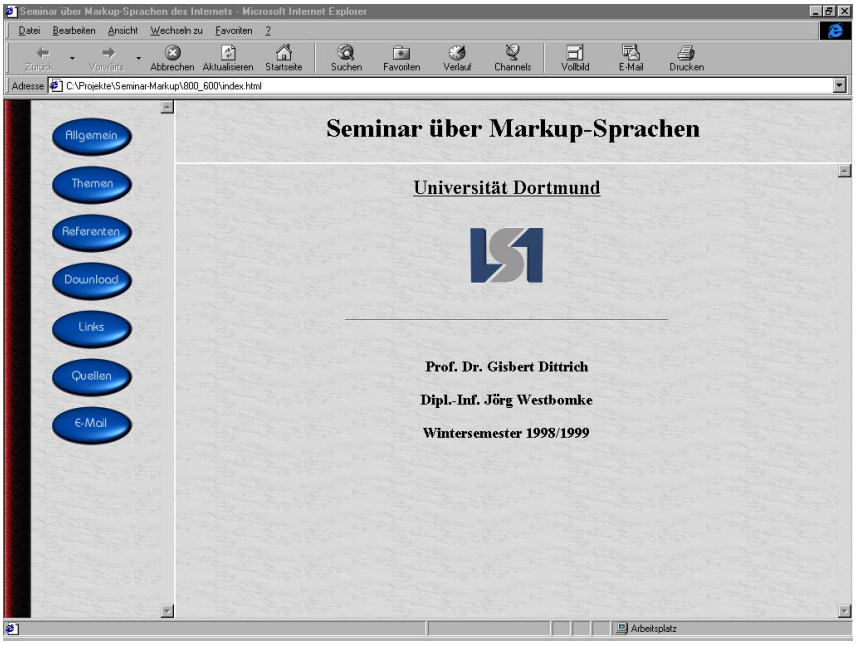


Abbildung 3-1: Die Seite "allgemein.html"

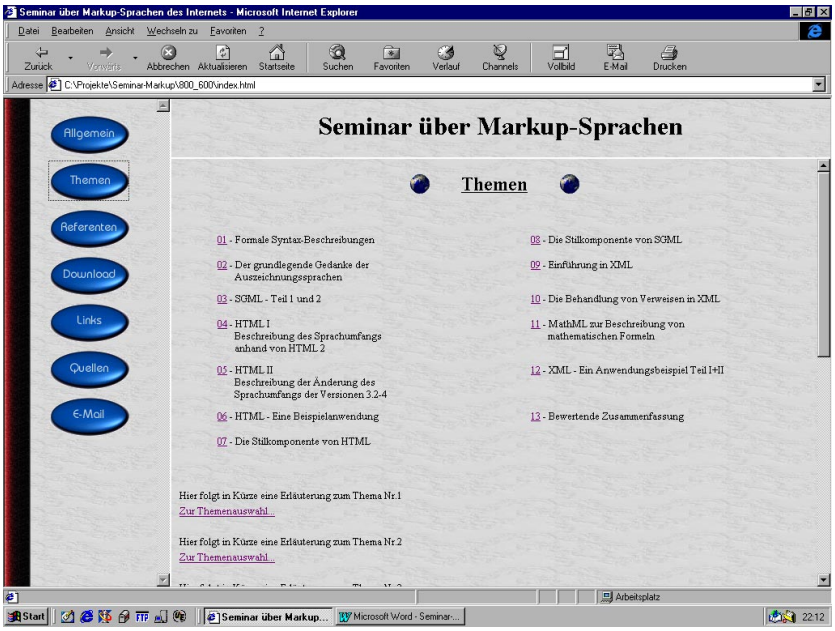


Abbildung 3-2: Die Seite "themen.html"

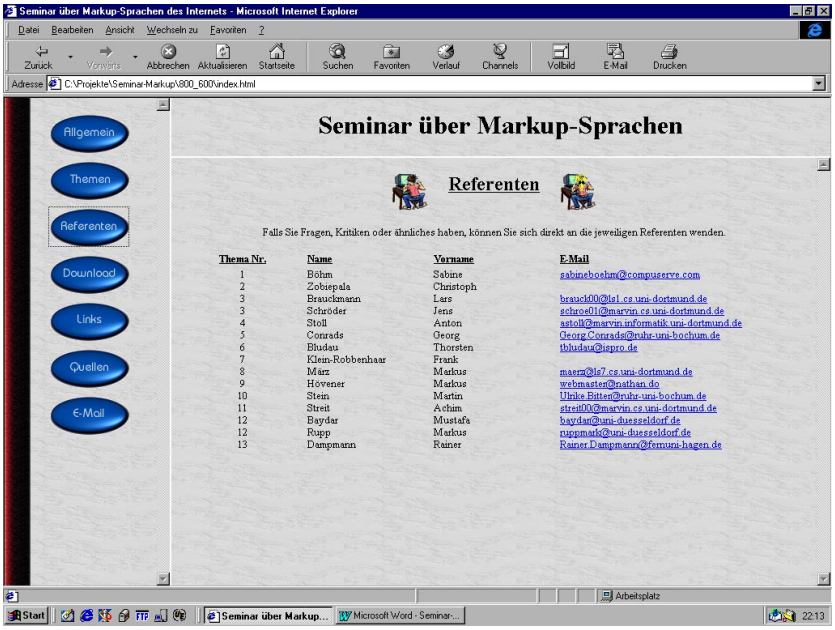


Abbildung 3-3: Die Seite "referenten.html"

4 Verwendete Tools

4.1 Allgemeines zu Editoren

Mittlerweile gibt es eine Unmenge von Text- und WYSIWYG-Editoren, so daß es an dieser Stelle nicht möglich ist zu sagen: "Der Editor XYZ ist der Beste !". Grundlegend kann man sich jedoch folgende "Regel" merken: Handelt es sich bei dem zu erstellenden Projekt z.B. um ein kleine Homepage für zu Hause, so ist es sicherlich dem Entwickler selbst überlassen, welchen Editor er verwendet. Werden die Projekte jedoch größer, so empfiehlt es sich im Allgemeinen, einen Text-Editor zu verwenden. Auch wenn es vielleicht etwas länger dauert die Seiten zu erstellen, so weiß man doch ganz genau, was wo warum an welcher Stelle steht. Werden jedoch die vom WYSIWYG-Editor angezeigten Seiten in HTML-Code umgesetzt, so kann es durchaus vorkommen, daß z.B. bei einer späteren Änderung der erzeugte Code nicht sofort durchschaubar ist oder daß sich die vom Editor präsentierte Ansicht von der im Browser dargestellten Ansicht unterscheidet.

4.2 Der verwendete Editor

Zum Erstellen dieses HTML-Beispiels wurde das Programm "Ultra Edit v5.2" verwendet. Es handelt sich hierbei um einen reinen Text-Editor, der es unter anderem ermöglicht, die verschiedenen Elemente eines HTML-Scripts in verschiedenen (vom Entwickler wählbaren) Farben zu kennzeichnen. So ist es zum Beispiel möglich, die Tags in rot und die dazugehörigen Attribute in blau darzustellen. Diese Unterscheidung ermöglicht schon während des Schreibens eine frühzeitige Fehlererkennung. Wird zum Beispiel das Attribut "border" in einer Tabelle verwendet und der Entwickler gibt versehentlich "broder" ein, so wird dieses Attribut im weiteren Verlauf nicht mit der entsprechenden Farbe gekennzeichnet.

4.3 Grafikprogramme

Um den HTML-Seiten einen gewissen optischen Eindruck zu verleihen, kommt man früher oder später nicht an einem oder mehreren Grafikprogrammen vorbei. Auch hier gibt es von kleinen Tools aus dem Free- und Sharewarebereich bis hin zu großen, komplexen Programmen eine große Auswahl an Möglichkeiten. Wichtig dabei ist, daß sowohl das Grafikformat GIF, als auch JPEG unterstützt werden müssen, da dies die beiden Standardformate des Internets sind. Wer an einem Tool zum Erstellen von animierten Bildern, wie sie oft zu finden sind, interessiert ist, kommt um das GIF-Format GIF89a und einen sogenannten GIF-Animator nicht herum. Dieser ermöglicht es, eine Folge von GIF-Bildern als einzelnes Bild zu speichern, so daß dieses Bildes später im Browser wie eine Art Daumenkino dargestellt wird. Desweiteren bieten GIF-Bilder den Vorteil. daß eine Farbe des Bildes als "transparent" deklariert werden kann., so daß an der Stelle im Bild, an der diese Farbe erscheinen würde, der hinter der Grafik liegende Hintergrund angezeigt wird.

Cascading Style Sheets

Frank Klein-Robbenhaar

FB Informatik, Uni Dortmund
su000201@access.uni-dortmund.de

Zusammenfassung

Die Trennung von Strukur- und Stilinformationen ist die grundlegende Idee oder Voraussetzung. Die strukturellen Informationen werden bei der Erzeugung von HTML-Dokumenten in Form von Auszeichnungen (HTML-Tags) in das Dokument eingebracht. Dagegen beschreibt das Konzept der Cascading Style Sheets (CSS) die Stilinformationen. Diese Ausarbeitung führt die benutzte Syntax, sowie die Funktionalität von CSS an.

1 Einleitung

Nehmen wir einmal an, daß man eine größere Anzahl von HTML-Dokumenten warten muß, und daß man die Aufgabe erhält alle Überschriften vom HTML-Typ H3, die dritte Alternative der Markup-Sprache HTML, rot zu färben. Dann wäre es also notwendig in allen HTML-Dokumenten jedes H3-Überschrifts-Tag zu ändern, in dem man das HTML-Attribut *color benützt.*

Hier setzt nun die Idee der Cascading Style Sheets an, die die Trennung von Struktur- und Stilinformationen aufgreift. Dabei beschreibt HTML nur die Struktur der Web-Seite, also die verschiedenen Teile des Textes innerhalb des Dokuments, die eine Rolle spielen. Ein Teil des Textes kann zum Beispiel eine Überschrift sein oder ein Absatz. HTML beschäftigt sich somit kaum mit dem Aussehen des Dokuments und kann es nur in begrenzten Maße beeinflussen. Im Gegensatz dazu beschreibt eine Cascading-Style-Sheet-Formatvorlage, eine Sammlung stilistischer Anweisungen, die zugehörigen Stilinformationen und wird mit dem HTML-Dokument verknüpft. Dadurch wird dem Autor einer Web-Seite ermöglicht, das Aussehen seines Dokuments zu steuern, in dem er die CCS-Formatvorlage in seinem Sinne verändert.

Mit Cascading Style Sheets kann man beispielsweise Stilelemente wie Farbe und Hintergrundfarbe, aber auch Leerräume um die HTML-Elemente oder das Positionieren von Bildern und Text bestimmen. Mit CCS ist also eine leistungsfähige „Sprache“ für Formatvorlagen geschaffen, die von dem herstellerneutralem World Wide Web Konsortiums, oder kurz W3C, im Jahre 1996 entwickelt wurde.

Im Laufe der Zeit entwickelten sie drei Empfehlungen oder auch zukünftige Standards; im August 1996 fing mit CSS alles an, dann folgte am 17.12.96 die CSS1-Spezifikation und mit CCS2 am 12.5.1998 setzt W3C noch nach, die damit noch größere Möglichkeiten schaffte. So unterscheiden sich CSS1 und CSS2 dadurch, daß CSS2 um eine Fülle von neuen Eigenschaften, wie beispielsweise das Positionieren von HTML-Elementen und bessere Benutzung von Webfonts, ergänzt wurde.

Der erste CSS-unterstützende Browser, der vor dem Erscheinen von CSS1 auf dem Markt kam , war der Microsoft Internet Explorer 3.0 (MSIE3). Rasch folgten weitere Browser wie beispielsweise der Netscape Navigator 4.0 und Arena.

Eine wichtige Fähigkeit von CSS ist die Kaskadierung von Formatvorlagen. Das bedeutet, daß mehrere Formatvorlagen an ein Dokument geknüpft werden können und alle das Aussehen des Dokuments

beeinflussen. Dadurch kann ein Autor eine Formatvorlage entwickeln, die das Aussehen der Seite festlegt, während der Leser seine eigene persönliche Formatvorlage bevorzugt, die das Aussehen der Seite nach seinen Wünschen regelt, wie zum Beispiel bei schlechten Augen eine größere Schrift adaptiert. Zusätzlich besitzt jeder Browser eine Standardformatvorlage, die für jedes Element eine Standard-Stilinformationen zur Verfügung stellt.

2 Tor zu Cascading Style Sheets

CSS basiert auf Anweisungen und Formatvorlagen. Eine Anweisung ist eine Aussage über einen stilistischen Aspekt eines oder mehrere HTML-Elemente. Eine Formatvorlage besteht aus einer oder mehrere Anweisungen, die sich auf ein HTML-Dokument beziehen. Betrachten wir den Aufbau einer Anweisung genauer.

2.1 Der Aufbau von Anweisungen

Der Aufbau einer einfachen CSS-Anweisungen besteht aus zwei Teilen, einmal aus dem Selektor, eventuell auch mehrere durch Kommata getrennte Selektoren-Gruppe, und einer Deklaration, die innerhalb von geschweiften Klammern liegt. Der Selektor ist das Verbindungsstück zwischen dem HTML-Dokument und den Style Sheets, somit sind alle HTML-Elemente mögliche Selektoren, die wir auch HTML-Element-Selektoren nennen. Da jeder Browser über eine Standard-CSS-Formatvorlage verfügt, ist eine vollständige vom Autor oder Benutzer erbrachte Formatvorlage nicht notwendig. Eine allgemeine Anweisung ist folgendermaßen aufgebaut:

- Selektor { Deklaration }

Außerdem lassen sich Selektoren mit gleicher Deklaration gruppieren, dadurch werden die Formatvorlagen kürzer und man kann sie schneller laden. Zur Notation wird hier der Kleensche Abschluß „(...)*“ verwendet.

- Selektor (‚Selektor)* { Deklaration }

Hierzu betrachte man zum Beispiel diese drei Anweisungen

- H1 { font-weight : bold }
- H2 { font-weight : bold }
- H3 { font-weight : bold }

Alle drei Anweisung haben dieselbe Deklaration, deshalb können sie zu einer Anweisung gruppiert werden und die Deklaration braucht nur einmal hingeschrieben werden.

- H1, H2, H3 { font-weight : bold }

Gehen wir auf das Einleitungsbeispiel zurück, so läßt sich die Überschrift H3 durch folgender Angabe von

- H3 { color: red;
  font-family: roman }

rot einfärben und mit Schriftart „Roman“ darstellen. Dabei werden die Eigenschaften (wie color, font-family, font-size, etc.) der Deklaration mit Semikola getrennt,

2.2 Der Aufbau einer Deklaration

Eine Deklaration besteht aus zwei durch einen Doppelpunkt getrennten Teilen und wird von geschweiften Klammern umfaßt. Der Teil vor dem Doppelpunkt ist die CCS-Eigenschaft (z.B. „color“) und der Teil nach dem Doppelpunkt bezeichnet man als Wert, der eine präzise, eigenschaftabhängige Spezifikation ist. Die Eigenschaft „color“ ist eine von den 50 Eigenschaften in CSS1, die das Aussehen eines HTML-Dokuments bestimmen können. CSS2 besitzt mehr als 100 Eigenschaften. Die geschweiften Klammern, und der Doppelpunkt ermöglichen es dem Browser, zwischen dem Selektor, der Eigenschaft und dem Wert zu unterscheiden. Eine allgemeine Anweisung schaut folgendermaßen aus:

- Selektor { Eigenschaft : Wert}

Auch können mehrere Deklarationen, die sich auf einen Selektor beziehen, in einer Liste zusammengeführt werden. Dabei trennen sich die einzelnen Deklaration in der Liste durch ein Semikolon, wobei am Ende der letzten Deklaration nicht unbedingt ein Semikolon stehen muß. Zusätzliche Leerzeichen oder Zeilenumbrüche sind in der Deklaration erlaubt und wünschenswert.

- Selektor { Eigenschaft1 : Wert1;
  Eigenschaft2 : Wert2 }

Es gehört also zu jeder Anweisung ein Selektor und eine Deklaration, die eventuell gleich mehrere Eigenschaften beschreibt, so daß der Browser diese neuen Eigenschaft dem Selektor oder der Gruppe von Selektoren zuordnen kann.
Die Entwickler von Cascading Style Sheet teilen die Eigenschaften in mehrere Kategorien ein:

- Schrift
- Grundlegende Strukturen
  - Listen
- Leerraum
  - Rand
  - Füllung
  - Rahmen
  - Textformationen
- Farbe und Hintergrund

3 Die Beschreibung der Selektoren

Der Selektor bildet das Verbindungsglied zwischen HTML-Dokument und der Formatvorlage. Weiter legt er fest, welche Elemente von der Deklaration betroffen sind. CSS bietet hierfür eine Vielzahl von Selekoren an, die es erlauben, die Teile in einem Dokument, die durch Formatierungsvorgaben beeinflußt werden, sorgfältig zu wählen. Weiter läßt sich CSS1 in vier Selektorenschemata teilen, dabei basiert jedes auf einem Merkmal eines Elements:

- dem Elementtyp
- einem Elementattribut
- dem Kontext, in dem das Element benutzt wird
- externen Informationen über das Element

Außerdem bietet CSS1 die Möglichkeit, Formatierungsanweisungen ohne den Gebrauch des Selektorenmechanismus; diese Methode benutzt das Attribut STYLE und widerspricht die Idee der Trennung von Strutur- und Stilinformationen.

3.1 HTML-Element Selektor

Jedem HTML-Element ist eine Selektor - den man HTML-Element- oder Typselektor nennt - gleichen Namens zugeordnet, dabei unterscheidet man nicht zwischen Groß- und Kleinschreibung der Typselektoren. Eine allgemeine Formatierungsanweisung sieht folgendermaßen aus :

- HTML-Element { Deklaration }

Um Platz von Style Sheet zu sparen, kann man mehrere HTML-Element Selektoren mit gleicher Deklaration:

- H1 { color : red }
- H2 { color : red }
- H3 { color : red }

Einfach in einer mit Kommata getrennten Liste gruppiert werden:

- H1,H2, H3 { color : red }

3.2 CSS2 : Universal-Selektor

Der Universal-Selektor, der mit dem Sternchen-Zeichen „*“ gekennzeichnet ist, entspricht genau einen beliebigen HTML-Element. Er steht für jedes Element in einem HTML-Dokument.

Beispiel:

- * { color : red }

Jedes HTML-Element in dem Dokument wird mit dieser Anweisung rötlich erscheinen.

3.3 Klassen-Selektor

Um die Granulation der Elemente zu erhöhen, bietet HTML das Attribut „CLASS“ an, so daß alle Elemente innerhalb des BODY-Elements klassifiziert werden können. Die Klassifizierung wird in den Style Sheets vorgenommen, mit folgender Syntax :

- Selektor.Klassennamen { Deklaration }

Nur ein einziger Selektor kann klassifiziert werden, so daß zum Beispiel die Anweisung „P.EM.Klassenname { Deklaration } nicht gültig ist. Der Klassennamen, der aus Buchstaben, Zahlen und dem Strichzeichen „-„ bestehen kann, darf keine Leerzeichen beinhalten. Auch wird ein Unterschied zwischen Groß- und Kleinschreibung gemacht.

Durch die normale Vererbung kann man mit einer allgemeineren Klassifizierung, die sich durch Wegfall des Selektors in CSS implementieren läßt, alle Elemente auf einem Mal klassifizieren :

- .Klassennamen { Deklaration }

CSS gibt mit dem Attribut CLASS eine enorme Flexibilität, so daß Elemente andere Elemente vom Aussehen her emulieren können. Dies liegt aber nicht im Sinne von CSS. Ein Struktur, basierend auf Klassifizierungen, ist nur dann sinnvoll, wenn sie eine Erweiterung darstellt, sogenannte Mutanten zu den Elementen.

In der Praxis sieht die Klassifizierung folgendermaßen aus. Zunächst seien neben den nicht klassifizierten Absatz einen als Zitat klassifiziert.

```
<P>Ein normaler Absatz <\P>
<P CLASS=zitat> eine klass. Anwendung beim Absatz <\P>
```

Nun kann der Autor Zitate von anderen Absätzen abheben, etwa durch die folgenden Zeilen:

- P.zitat { margin-left: 20%;
  margin-right: 20% }

Damit stellt der Web-Browser Zitat mit Rändern dar, die jeweils um 20% links sowie rechts vom Blattrand entfernt sind.

3.4 ID-Selektor

Wer es noch genauer mag, kann das Attribut ID verwenden, welches für die eindeutige Kennzeichnung eines Elements über das gesamte Dokument zuständig ist. Das heißt, daß der Parser eine Fehlermeldung erzeugt, wenn in einem Dokument zwei Elemente mit denselben ID-Attributen ausgestattet sind. Das ID-Attribut wird wie alle anderen Attribute in HTML verwendet und wird hauptsächlich für spezielle und wichtige Elemente, um Formatierungseigenschaften festzulegen, eingesetzt. Die CSS-Schreibweise lehnt sich an der Klassifizierungsnotation etwas an:

- Element#ID-Wert { Deklaration }

oder nur

- #ID-Wert { Deklaration }

Die Anweisung beginnt mit dem Element-Selektor, gefolgt von dem Zeichen „#“, welches die ID-Selektoren charakterisiert, und anschließender Kennzeichnung, dem ID-Wert. Die Anweisung ohne ein voranstehendes Element erlaubt das einmalige Anwenden des ID-Attibuts für ein beliebiges Element.

Beide Anweisungen enthalten immer das Raute-Zeichen „#“, das dem Browser zeigt, daß als nächstes ein ID-Wert kommt. Der gewählte ID-Wert ist unabhängig von Groß- und Kleinschreibung, und kann vom Autor frei gewählt werden.

Beispiel:

- #z98y { letter-spacing : 0.3em }
- H1#z98y { letter-spacing : 0.5em }

In HTML:

```
<P ID=z98y> großer Text </P>
```

Da der Wert des Attributs ID eindeutig sein muß, kann die Überschrift H1 nicht mit diesem Attribut noch einmal verwendet werden. Man muß verschiedene ID-Werte innerhalb eines Dokumentes benutzen.

3.5 Kontextsensitive Selektor

Ein kontextsensitiver oder auch kontextuellen Selektor beschreibt einen Selektor, der nur, wenn er in einem bestimmten Kontext steht, eine extra Formatierung einleitet. Dabei versteht man unter einem Kontext eine festgelegte Verschachtelung von einfachen Selektoren in einem HTML-Dokument. Alle Selektoren, die wir bisher kennengelernt haben, sind einfach: Jeder Typ-, Klassen- oder ID-Selektor ist ein einfacher Selektor. Auch die Kombination von Typ- und Klassenselektoren werden als einfach angesehen.

Kontextsensitive Selektoren bestehen aus zwei oder mehreren einfachen Selektoren, die durch ein Leerzeichen getrennt werden, und die den Kontext, d.h. die Verschachtelung, wiedergeben. Anschließend folgt wie bei allen Selektoren eine Deklaration, die wie schon gesagt, nur im kontextsensitiven Fall verwendet wird.

Zur Verdeutlichung betrachten wir folgendes Beispiel:

- H1 { color : red }
- EM { color : red }

Diese Anweisungen werden gut funktionieren. H1-Überschriften werden rot gefärbt, genauso wie EM-Elemente und eine Hervorhebung wird erreicht.

```
<H1>Diese Überschrift ist <EM>sehr</EM> wichtig </H1>
```

Hier aber fällt die Hervorhebung von dem Wort „sehr“ weg, weil sowohl EM als auch H1 auf Rot gesetzt sind. Ziel ist es, daß H1 und EM das ganze Dokument hindurch rot bleiben, aber für EM-Elemente innerhalb von H1-Überschriften anders betont werden. Dafür greift der kontextsensitive Selektor :

- H1 EM { color : blue }

Diese Anweisung bedeutet, daß EM innerhalb eines H1-Tags blau gefärbt wird. Die Farbe von EM ist kontextabhängig, EM erscheint also nur im Kontext von H1 blau, in allen anderen aber rot, wie zuvor.

Kontextsensitive Selektoren mit gleichen Deklarationen kann man mit Hilfe von Kommata gruppieren, um in den Style Sheets Platz zu sparen :

- H1 B, H2 B, H1 EM, H2 EM { color : red }

3.6 CCS2 : Kind-Selektor

Der Kind-Selektor stellt eine Erweiterung des kontextsensitiven Selektors dar. Wir kennen die Verschachtelung von Elementen in einem Dokument, die auch mit einer Baumstruktur dargestellt werden kann. Dadurch definieren sich die Begriffe wie Vater, Großvater oder Kind des Elements von selbst. Der Kind-Selektor ist eine „Abstammungs"-Relation, die zwei einfache Selektoren in eine Relation stellt, und zwar stellt sich dort die Frage, ob der erste Selektor ein Vater, Großvater, Urgroßvater usw. von dem zweiten Selektor ist.

Die Syntax des Kind-Selektors schreibt das Trennungszeichen „>" zwischen den ersten einfachen Selektor und den zweiten Selektor vor und die anschließende Deklaration gilt natürlich für das „Kind", der durch den zweiten einfachen Selektor gegeben ist.

Beispiel:

- BODY > P { line-height : 1.3 }

In diesem Beispiel gilt der Stil für alle Elemente P, die Abkömmlinge von BODY sind.

Das nächste Beispiel kombiniert ein Kind-Selektor mit einem kontextsensitiven Selektor.

- DIV OL>LI P { line-height : 1.3 }

Der Stil für P wird aktiv, falls P im Kontext mit LI steht, der wiederum von dem Element OL stammt, der im Kontext DIV steht. Eine Besonderheit ist das unbedingte Fehlen der Leerzeichen um das „Größer"-Zeichen „>", die in diesem Fall weggelassen werden.

3.7 CSS2 : Nachbarn-Selektor

Der Nachbarn-Selektor beschreibt zwei Elemente, die in dem Dokument direkt nebeneinander stehen, daher der Name, und die in dem Strukturbaum des Dokumentes denselben Vater besitzen. Verknüpft werden die beiden Elemente bei dem Nachbarn-Selektor mit dem Pluszeichen „+", die anschließende Deklaration bezieht sich auf das zweite Element, welches dem Pluszeichen folgt.

Beispiel:

- MATH + P { text-indent : 0 }

Diese Stilanweisung für das Element P rückt den Text von P genau dann nicht ein, wenn P dem Element MATH in dem Dokument folgt.

Statt den Elementen mit den zugehörigen HTML-Selektoren kann man auch andere einfache Selektoren benutzen, wie zum Beispiel die Klassifizierung eines Selektors.

Beispiel:

- H1.opener + H1 { margin-top : -5mm}

3.8 CSS2 : Attribut-Selektor

Der Attribut-Selektor, der ein Attribut in eckigen Klammern darstellt, adressiert alle Elemente, die das Attribut, welches in den eckigen Klammern steht, benutzen.

Die Syntax des Attribut-Selektors lautet:

- [att]

Dabei steht das Wort „att“ für ein HTML-Attribut wie z.B. href, usw..

Beispiel:

- [href] { color : red }

Diese Attribut-Selektor-Anweisung färbt alle Elemente rot, die das Attribut „href“ benutzen.

Eine Spezialisierung des Attribut-Selektors erreicht man, indem man dem Attribut-Selektor noch einen Attributwert mit auf den Weg gibt, der dann alle Elemente mit diesem Attribut und mit diesem Attributwert adressiert. Die Syntax dieses Attribut-Selektors sieht folgendermaßen aus :

- [att=value]

Manchmal beschreibt eine Liste von Attributwerten viele alternative Möglichkeiten für ein Attributwert. Um Element mit dieser Attributierung zu adressieren, nimmt man genau einen Wert aus der Liste und verwendet folgende spezielle Syntax dieses Attribut-Selektors:

- [att~=val]

Hier steht das Wort „val“ für einen Wert aus der alternativen Liste der Attributzuweisung zum Attibut „att“. Alle Elemente mit diesem Attributwert, der direkt oder durch eine Liste indirekt angeben ist, werden so addressiert.

Man kann die Attribut-Selektoren noch mit anderen Selektoren wie zum Beispiel HTML-Selektoren kombinieren, um den Adressierungsraum auf spezielle Elemente einzuschränken.

Beispiele:

- H1[title] { color : blue }

Dieses Beispiel gibt allen H1-Elementen, welche das Attribut „title“ nutzen, die Farbe blau.

- A[href=“http://www.w3.org/“] { color : yellow}

Weist die Farbe eines bestimmten Hyperlink auf gelb.

3.9 Pseudoklassen-Selektor

Wir kennen schon Klassenselektoren, die es ermöglichen als CLASS-Wertattribut von HTML-Elementen eine höhere Variation von Elementen zu erreichen. Im Gegensatz dazu, realisieren Pseudoklassen erstens keine

zusätzlichen CLASS-Attribute, d.h., daß Pseudoklassen nicht mit Hilfe des Attribut CLASS in HTML einfließen können. Und zweitens werden sie eigentlich nur in CSS1 (und nicht in HTML) deklariert, um Einflüsse auf Formatierungsprozesse zu bekommen, die in HTML nicht zugänglich sind. So besitzt beispielsweise das Element A, welches mit dem Attribut HREF ein Hyperlink darstellt, drei Hyperlink-Zustände, die jetzt nur mit Hilfe von Pseudoklassen adressiert werden können. Diese drei Pseudoklassen für die entsprechenden Hyperlink-Zustände lauten:

- link (für noch nicht besuchte Hyperlinks)
- visited (für alle Links, die schon einmal besucht wurden
- active (für Links, die in diesem Moment besucht werden, beispielsweise in einem anderen Frame)

Dadurch wird es möglich, diese Hyperlink-Zustände gewisse Formatierungsanweisungen zu zuweisen, die ohne eine Reformatierung des Textes dargestellt werden können. Das bedeutet, daß zum Beispiel eine Formatierung mit unterschiedlichen Schriftgrößen für die Hyperlink-Zustände keinen Effekt zeigt, weil dies eine Reformatierung des Textes impliziert, die nicht eingeleitet wird. So beschränken sich die Formatierungen für Hyperlink-Zustände bekanntermaßen auf die Farbe, Text-Dekoration u.v.a..

Zur Behandlung der Hyperlinks in den HTML-Dokumenten können also sogenannte Pseudoklassen-Selektoren in CSS verwendet werden. Dabei besteht ein Pseudoklassen-Selektor durch einen einfachen Selektor, gefolgt von

einem Doppelpunkt, der die Pseudoklasse kennzeichnet, und einem anschließenden Namen der Pseudoklasse.

Beispiel:

- A:link { text-decoration : underline }
- A:visited { color : red }
- A:active { color : blue }

Zusätzlich können Pseudoklassen mit Klassifizierungen von Selektoren verknüpft werden:

- A.Klassenname:link { Deklaration }

<A CLASS=“Klassenname“ HREF=“Referenzierung“> Beschreibung <\A>

Pseudoklassen können auch mit kontextsensitiven Selektoren verwendet werden und sind unabhängig von der Groß- und Kleinbuchstaben.

3.9.1 CSS2 : Pseudoklasse :fist-child

Diese Pseudoklasse adressiert den „ersten“ Sohn von einem beliebigen Element in einer Verschachtelung von Elementen. Somit adressiert zum Beispiel der Selektor „A:first-Child“ jedes Element A, das der erste Sohn eines anderen Elementes ist und damit in der Verschachtelung an erster Stelle steht. Das bedeutet, daß das Element vor dem Doppelpunkt eine Spezialisierung ist, denn genau dieses Element soll den ersten Sohn bilden.

Beispiele:

- DIV > P:first-child { text-ident : 0 }

Diese Anweisung würde das Element P, welches das erste Element ist, innerhalb von DIV im Stil beeinflussen:

```
<P> Das letzte P vor DIV
<DIV>
  <P> Das erste P innerhalb von DIV, dieser wird adressiert
  <P> Dieses P bleibt unbeeinflußt
</DIV>
```

Die Pseudoklasse „:first-child“ erlaubt auch eine Kombination mit kontextsensitiven oder einfachen Selektoren.

3.9.2 CSS2 : Die dynamischen Pseudoklassen

Wenn wir auf die Interaktionen des Benutzers eingehen wollen, dann benötigen wir zusätzliche allgemeine Pseudoklassen, die auf Benutzeraktionen reagieren. So bewegt der Benutzer zum Beispiel den Mauszeiger auf ein bestimmtes Element, dann erscheint dieses Element als Antwort auf die Benutzeraktion in einem anderen Stil. CSS2 unterstützt drei verschieden Pseudoklassen, wobei wir die Pseudoklasse „active“ aus CSS1 schon kennen:

- Die Pseudoklasse „:hover“ realisiert gerade, daß in dem Beispiel angesprochene, Wechselspiel zwischen dem interaktiven Dienst (Web-Site) und der Benutzeraktion auf dieser Web-Site. Dabei beachte man, daß der Benutzer noch nichts aktiviert hat, und nur lediglich auf die Position oder die bis dato gemachte Benutzeraktion geantwortet wird.
- Die Pseudoklasse „:active“ adressiert Elemente, die gerade von dem Benutzer aktiviert wurde.
- Die Pseudoklasse „:focus“ wird auf Elemente angewandt, die auf ein Keyboard-Ereignis oder sonstige textuelle Eingaben warten.

Diese drei Pseudoklassen schließen sich n i c h t gegenseitig aus, dadurch kann ein Element gleich durch mehrere dieser Pseudoklassen im Stil beeinflußt werden

Zusätzlich beachte man, daß diese Pseudoklassen keine Reformatierung des Textes implizieren, d.h. daß die Style-Sheet-Anweisung wie „font-size“ ignoriert werden. Zum Beispiel bedeutet dies, daß die Eigenschaft „font-size“ mit unterschiedlichen Werten für die Pseudoklasse „:active“ und „:link“ auch unterschiedliche Formatdarstellungen zur Folge hat, die dann ignoriert werden.

Beispiele:

- A:link { color : red }
- A:visited { color : blue }
- A:hover { color : yellow }
- A:active { color : lime }
- A:focus:hover { background : white }

3.9.3 CSS2 : Die Pseudoklasse für Sprachen

Wenn ein Dokument mehrere Elemente enthält, die in einer andere Sprache, z.b. französisch, realisiert oder unterstützt werden, dann ist es möglich Selektoren in CSS anzugeben, die diese Elemente adressieren. Dies geschieht mit der Pseudoklasse „:lang“, die es auch in HTML gibt.
Auch im weiteren Sinne kann man zum Beispiel mit der Pseudoklasse „:lang(C)“ Elemente adressieren, die mit der Programmiersprache C in Verbindung stehen.

3.10 Der Pseudoelement-Selektor

Pseudoelemente erlauben es, einen untergeordneten Teil des Inhaltes eines Elementes zu formatieren. Genauso wie Pseudoklassen existieren auch Pseudoelemente im HTML-Quelltext nicht. Durch Pseudoelemente werden Möglichkeiten im Design gegeben, die ohne sie nicht möglich wären.

CSS1 hat zwei Pseudoelemente: first-letter und first-line. Beide ermöglichen unabhängig von der Formatierung den ersten Buchstaben oder die erste Zeile eines Absatzes speziell zu formatieren.

Die Pseudoelement-Selektoren werden mit einem Doppelpunkt vor dem Pseudoelement gebildet und wird beinahe immer in Kombination mit anderen Selektoren gebraucht. Das liegt daran, daß man selten alle Pseudoelemente in einem Dokument formatieren möchte.

Beispiele:

- :first-letter { text-transform : uppercase }
- P:fist-line { text-transform : lowercase }
- P.Klassenname:first-letter { color : red }

Die beiden Pseudoelemente werden in HTML nur für Blockelemente eingesetzt und sind unabhängig von der Groß- und Kleinschreibung.

3.10.1 CSS2 : Die Pseudoelemente „before“ und „after“

Die Pseudoelemente „:before“ und „:after“ können benutzt werden, um einen schon in CSS festgelegten Teststring entweder an den Anfang oder an das Ende eines Elementinhalts anzuhängen.

- H1:before {content: counter(chapno, upper-roman) ". "}

Diese Anweisung fügt vor jeder Überschrift vom Typ H1 automatisch eine römische Numerierung ein.

- P.special:before { content : „Special!“ }
- P.special:first-letter { color : gold

4 Kommentar

Zur jeder Programmiersprache, wie auch in CSS, gibt es die Möglichkeit Kommentare zu setzen.

Beispiel:

- EM { color : red } /* Die Farbe von EM ist rot */

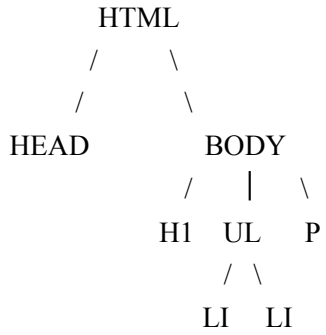
In CSS werden die Kommentare genau wie bei der Programmiersprache C gesetzt.

5 Die Vererbung von Eigenschaften

Kein HTML-Dokument kommt ohne Verschachtelung aus, so umschließt das Element HTML alle anderen Elemente wie HEAD oder BODY, wobei letzteres alle anderen Elemente wie H1 und UL umschließt. Und innerhalb einiger dieser Elemente sind wiederum andere enthalten, beispielsweise enthält eine ungeordnete Liste verschachtelt mehrere Listenelemente, einige Listenelemente enthalten verschachtelt fettgedruckte Wörter usw..

Diese Verschachtelung läßt sich in eine Baumstruktur übertragen. In der Baumstruktur eines HTML-Dokuments ist das Element HTML der älteste Vorfahr oder die Wurzel und steht ganz oben.

Alle anderen sind Söhne und Söhnessöhne, zum Beispiel sind TITLE und BODY Söhne von HTML, weiter hat BODY beispielsweise zwei Söhne H1 und UL. UL wiederum hat mehrere Li´s als Söhne.



Genau wie Söhne von ihrem Vater erben, erben die Html-Elemente die CSS-Formateigenschaften, wenn diese vererbbar sind. Das bedeutet, daß durch Vererbung CSS-Eigenschaften von einem Element, für das sie gesetzt wurden, den Baum hinunter an alle Nachkommen weiter gegeben werden, sofern diese nicht von speziellen Stilanweisungen für einige Nachkommen überschrieben werden.

```
<HTML>
  <HEAD>
     <TITLE> Titel </TITLE>
     <STYLE>
        H1 { color : red }
       EM { color : blue }
     </STYLE>
  </HEAD>
  <BODY>
     <H1> Die Überschrift <EM> ist </EM> wichtig </H1>
  </BODY>
</HTML>
```

Wenn keine stilistische Anweisung in der Formatvorlage wäre, dann würde durch Vererbung das Element EM rot dargestellt werden. Aber da wir in der Formatvorlage eine speziellere Anweisung über EM finden, werden die speziellen Eigenschaften genommen und der EM-TEIL der Struktur erscheint in blau.

Vorsicht, es gibt einige CSS-Eigenschaften die nichts vom Vater an den Sohn vererben. So zum Beispiel ist die *background*-Eigenschaft eine Eigenschaft, die sich nicht vererben läßt.

6 Maßeinheiten der Werte

Viele Eigenschaften akzeptieren Werte, die Zahlen oder eine bestimmte Anzahl von Einheiten eines bestimmten Maßes sein. Eine Zahl, die ein negativer oder positiver Wert ist, kann eine ganze Zahl (0, 1 , -1, 2,...) oder ein Bruch (0.5, -0.5, ...) sein. Wenn es sich um eine Länge, auch „length“, handelt, wird eine Maßeinheit direkt hinter der Zahl ohne Leerzeichen zwischen Zahl und Maßeinheit angegeben. Alle Einheiten werden mit zwei Buchstaben abgekürzt, ohne Punkt dahinter. Es gibt folgende Einheiten:

- Millimeter: mm
- Zentimeter: cm (1 cm = 10 mm)
- Zoll: in (1 Zoll = 25,4 mm)
- Punkt: pt (72 pt = 1 Zoll)
- Pica: pc (1 pc = 12 pt)
- em: em (1 em = Punktgröße der jeweiligen Schrift)
- Ex: ex (die Buchstaben-x-Höhe)
- Pixel px

Es gibt drei Sorten von Einheiten :

1. absolute - mm, cm, in, pt, pc
2. relative - em, ex
3. geräteabhängige - px

Eine absolute Einheit ist eine Einheit, die einen festen Wert definiert. Dagegen sind relative Maßeinheiten Einheiten, die in Abhängigkeit von der Schriftgröße des Elements. Zusätzlich genießen relative Werte gegenüber absoluten den Vorteil, daß sie automatisch skaliert werden. Das heißt beispielsweise, daß wenn man eine andere Schriftart wählt, dann brauchen die Eigenschaftswerte in ex und em nicht geändert werden. Eine geräteabhängige Einheit ist eine Maßeinheit, die vom benutzten Gerät abhängt. Aufpassen muß man bei Ausgabegeräten, wo sich die Pixeldichte unterscheidet (wie bei Druckern , Monitor), welches dann eine Anpassung der Pixelwerte nach sich zieht.

Obwohl Prozentwerte und Schlüsselwörter keine Maßeinheiten sind, sind sie weitere relative Formen der Angabe von Werten, d.h., sie werden automatisch skaliert. Normalerweise bezieht sich auch ein Prozentwert immer auf eine Längeneinheit des Vaters. Doch bildet die Eigenschaft line-height eine Ausnahme.

7 Die Eigenschaften von CSS1

7.1 Kategorie Schrift

Die Festlegung der Schrifteigenschaften gehört zu den häufigsten Anwendungsbereichen von Formatvorlagen. Leider beruht das Erscheinungsbild der Webseite, speziell auch der Schrift, auf festgesetzte Browser-Formatierungen, die den Einfluß auf das Aussehen schmälern. Dennoch wird der Browser sein Bestes tun, um alle CSS-Schriftvorgaben umzusetzen, und versuchen, im Einklang mit seinen Ressourcen die beste Entsprechung zu finden.

Unter CCS kann man folgende Schrifteigenschaften angeben:

- font-family
- font-style
- font-variant
- font-weight
- font-size
- font
- text-decoration
- text-transform

7.1.1 Die Eigenschaft font-family

Mit der Eigenschaft font-family kann man den Schriftartsatz, wie zum Beispiel Times, Garamond usw., festlegen, der dann dem Dokument zugrunde liegt. Dabei akzeptiert die Eigenschaft zwei Arten von Werten, einmal den Wert „family-name“, welcher den Name der Schriftart trägt. Weitere Schriftfamilien als Alternativen, falls die gewünschte nicht vorhanden ist, gibt man mit einer Liste an, in der alle Schriftfamilien mit einem Komma getrennt werden. Die Priorität fällt von links nach rechts.

- Body { font-family : Garamond, Times, „meine Schrift“}

Die Anführungszeichen bei dem letzten Schriftnamen ist erforderlich, weil der Name ein Leerzeichen enthält.

Die zweite Wertangabe ist ein Gattungsname, engl. „generic-family“ genannt, und kann anstelle oder zusätzlich zum Namen der Schriftfamilie gesetzt werden, welches dann wieder mit Hilfe eines Kommas getrennt wird. Es gibt folgende Gattungsnamen : serif, sans-serif, monospace, cursive, fantasy

Mit dem Gattungsname wird das Problem nicht verfügbarer Schriften, die eine Undefiniertheit implizieren, dadurch abgedeckt, daß der Browser aus seinem Repertoire schöpfen darf mit Blick auf die Gattung, d.h. daß zum Beispiel beim Gattungsname „serif“ der Browser eine Serifenschrift auswählt.

- Body { font-family : Garamond, Times, serif

7.1.2 Die Eigenschaft font-style

Mit der Eigenschaft font-style kann man zwischen normalen, kursiven oder schrägen Schriftschnitt innerhalb der Schriftfamilie festlegen. Die Schlüsselwörter der zulässigen Schriftschnitte in gleicher Reihenfolge sind normal, italic und oblique.

7.1.3 Die Eigenschaft font-variant

Die Eigenschaft font-variant legt innerhalb der aktuellen Schriftfamilie einen normalen oder einen verkleinerten Schriftformat fest. Dabei wird das verkleinerte Schriftformat mit Schlüsselwort „small-caps“ nicht nur verkleinert, sondern umproportioniert und Großbuchstaben ähnelnd geformt. Diese Schrift nennt man Kapitälchenschrift und wenn keine Kapitälchenschrift vorhanden ist, kann der Browser versuchen, eine aus einer vorhandenen Normalversion zu erstellen, indem er einige Großbuchstaben verkleinert und dann streckt.

7.1.4 Die Eigenschaft font-weight

Innerhalb der aktuellen Schriftfamilie wird mit der Eigenschaft font-weight die Strichstärke angegeben, die mit den Schüsselwörtern „lighter“, „normal“, „bold“ und „bolder“ beeinflußt werden können. Hierbei sind „normal“ und „bold“ absolute Strichstärken.

Auch kann man mit den neun Hundertern („100“, „200“, ... „„900“) alternativ absolute Schriftstärke definieren, wobei beispielsweise „700“ dasselbe ist wie „bold“ oder „400“ der normalen Schriftstärke entspricht. Die Schlüsselwörter „lighter“ und „bolder“ stehen in Relation mit dem Vaterelement, d.h. daß von der Strichstärke des Vaterelements ausgegangen wird und diese dann entweder kleiner oder größer übernommen wird.

- Body { fontweight : bold }
- H1 { fontweight : bolder }

Wenn diese Formatvorlage als Grundlage für ein Dokument dient, dann werden dort alle HTML-Elemente außer H1 mit der Strichstärke „bold“ dargestellt, da sie die Strichstärke von dem Vaterelement BODY erben. Dagegen wird die Strichstärke des HTML-Elements H1 etwas fetter als alle anderen erscheinen.

7.1.5 Die Eigenschaft font-size

Mit der Eigenschaft *font-size* wird die Schriftgöße angegeben. Möglich sind wieder absolute oder relative Angaben. Für die absolute Angabe stehen einmal die Schlüsselwörter „xx-small“, „x-small“, „*small*“, „*medium*“, „*large*“, „*x-large*“ und „*xx-large*“, die selbst erklärend einige Größen darstellen. Auch kann man wieder direkt mit den Einheiten wie Punkt, Millimeter, Zentimeter, Zoll o.a. die Größe angeben, wie beispielsweise der Wert „14 pt“ eine 14 Punkt große Schrift hervorzaubert.

Die relativen Angaben beziehen sich auf der Größe des Vaterelements, die mit den Schlüsselwörtern „*smaller*“, „*medium*“ oder „*larger*“ verändert übernommen werden. Die andere relative Angabe sind die Prozentwerte, die sich auch die Schriftgröße des Vaters beziehen. Daher ergibt der Wert „120%“ eine um 20% größere Schrift als beim Vaterelement, „80%“ eine um 20% kleinere.

7.1.6 Die Eigenschaft font

Die Eigenschaft *font* erlaubt es, in einer einzigen Aktion alle anderen Schrifteigenschaften mit nur einem Leerzeichen voneinander getrennt und in folgender genau einzuhaltenden Reihenfolge *font-style, font-variant, font-weight, font-size*, line-height und font-family festzulegen. Dabei dürfen die Eigenschaften *font-size* und font-family nicht fehlen, die anderen werden im fehlenden Fall auf den normalen Wert gesetzt.

Zusätzlich gilt noch die Vereinbarung der traditionellen Typographie, die die beiden Eigenschaften font-size und line-height, sofern beide angegeben sind, statt mit einem Leerzeichen durch den schrägen Strich „/“ trennt.

Hierzu noch einige Beispiele:

- P { font : italic 12pt/14pt bodoni, bembo, serif}
- P { font : normal small-caps 120%/120% fantasy}
- P { font : x-large/100% „new century schoolbook, serif}

7.1.7 Die Eigenschaft text-decoration

Diese Eigenschaft verwendet man, um Unter-, Über- und Durchstreichungen oder blinkenden Text einzufügen. Die zugehörigen Schlüsselwörter heißen „underline“, „overline“, „line-through“ und „blink“, hierzu gehört auch der Anfangswert „none“, der keine Auszeichnung repräsentiert.

In vielen Browsern wird die Unterstreichung des Elements A verwendet, um den Status eines Hyperlinks zu markieren. Die Standardformatvorlage für diese Browser enthält eine Anweisung dieser Art:

- A:link, A:visited, A:active { text-decoration : underline }

Die Besonderheit ist, daß die Eigenschaft text-decoration nicht vererbt wird. Die Auszeichnung eines Vaterelements wird jedoch in Sohnelementen fortgesetzt. Die Wirkung dieser Fortsetzung entspricht aber nicht der Vererbung. Der Grund dafür, daß text-decoration nicht vererbt wird, hat mit möglichen späteren Erweiterungen dieser Eigenschaft zu tun. Gibt es in der Zukunft beispielsweise ein Schlüsselwort für die Umkreisung eines HTML-Elements, dann hätte eine Verschachtelung von Elementen mit dieser geerbten Eigenschaft eine oder mehrere Umkreisungen in einer Umkreisung zur Folge. Dabei ist doch nur eine äußere Umkreisung des ersten Elements, der die Eigenschaft der Umkreisung besitzt, wünschenswert, die zusätzlich in den verschachtelten Elementen intelligent fortgesetzt wird.

7.1.8 Die Eigenschaft text-transform

Der Eigenschaft *text-transform* stehen folgende Schlüsselwörter als Werte zur Verfügung : „*capitalize*“, „*uppercase*“, „*lowercase*“ und „*none*“.

„Capitalize“ bewirkt, daß der Erste Buchstabe Jedes Wortes Groß Geschrieben Wird, Damit Eine Wirkung Wie Diese Ensteht. „Uppercase“ wandelt alles in GROSSBUCHSTABEN WIE DIESE um; „lowercase macht das Gegenteil. „None“ neutralisiert einen vererbten Wert der Eigenschaft text-transform. Vor allem wird „lowercase“ zur Darstellung von Akronyme verwendet.

7.2 Kategorie : Grundlegende Strukturen

Wir wissen, daß HTML generell die Elemente in drei Gruppen einteilt : Blockelement, Inline-Element und Unsichtbares-Element. Auch wissen wir, daß ein HTML-Dokument aus Elementen innerhalb von anderen Elementen verschachtelt besteht. Ein Element EM kann sich z.B. innerhalb von anderen Elementen befinden, dieses innerhalb eines BODY und dieses innerhalb von HTML. Diese Anordnung könnte man sich auch als Kästchenmodell vorstellen, in dem kleinere Kästchen in immer größere Kästchen passen, wobei jedes Element ein Kästchen zugeordnet werden kann.

Ein Blockelement wie DIV und P wird normalerweise als eigenes Kästchen angezeigt. Ein Inline-Element wie EM und SPAN kann in mehrere kleinere Kästchen aufgeteilt werden, wenn es auf mehrere Zeilen verteilt ist

7.2.1 Die Eigenschaft display

Die Eigenschaft *display* bestimmt, ob ein Element als Block-, Inline- oder Listenelement angezeigt wird. Ein Element mit dem Wert „*block*“ beginnt immer auf einer neuen Zeile. Ein Element mit dem Wert „*inline*“, das Gegenstück zu „*block“*, beginnt und endet nicht auf einer neuen Zeile. Die Elemente einer Auflistung, die zwischen den Elementen UL oder OL sich befinden können, tragen den Eigenschaftswert „list-item“. Dabei bildet eine Folge von list-item’s entweder eine sortierte oder eine unsortierte Liste, die entweder eine Beschriftung hat oder nicht. Ob sie eine Beschriftung unterliegt, wird mit der Eigenschaft *list-style* festgelegt, die auch die Eigenschaften der Beschriftung festlegt.

Um ein Element vollständig unsichtbar zu machen, benützt man den Wert „none“. Das Element wird dann überhaupt nicht angezeigt.

Alle Elemente haben in der Standardformatvorlage des Browsers einen Wert für die Eigenschaft display, deshalb muß man die Eigenschaft *display* nicht so oft benutzen.

7.2.2 Die Eigenschaft list-style-type

Listenelemente können eine Beschriftung haben oder nicht, dabei unterscheidet man bei einer Beschriftung zwischen Spiegelstrich und Beschriftung mit einer Zahl. Die üblichere Beschriftung, der Spiegelstrich, ist ein vordefiniertes Symbol, der mit den Schlüsselwörtern „*disc*“, „*circle*“ und „*square*“ selbsterklärend gewählt werden kann. Eine Folge von Zahlen als Beschriftung, wie beispielsweise I,II,III oder 1,2,3 , werden mit den Schlüsselwörtern „*decimal“*, „*lower-roman“, „upper-roman“, „lower-alpha“* und *„upper-alpha“* gegeben. Das neunte Schlüsselwort „*none*“ unterdrückt die Beschriftung.

Dennoch unterdrückt „*none*“ nicht die Zählung in einer Numerierung, wie zum Beispiel in einer Liste mit drei Elementen, in der das zweite Element auf „none“ gesetzt ist und die restlichen mit einer Zahl beschriftet sind. Hier wird das erste Element mit „1“ gezählt und das letzte mit „3“, obwohl keine sichtbare „2“ neben dem zweiten Element steht.

Es folgen einige Beispielsanweisungen, die Beschriftungen festlegen:

- OL { list-style-type : lower-alpha }
- UL UL { list-style-type : square }
- LI.nolab { list-style-type : none }

7.2.3 Die Eigenschaft list-style-image

Anstelle einer Zahl oder eines vordefinierten Symbols kann auch ein kleines Bild als Aufzählungszeichen benutzt werden. Dafür gibt man einfach die URL-Adresse als Wert in Verbindung mit der Eigen-

schaft *list-style-image* an. Wenn *list-style-image* nicht „*none*“ ist, wird das angegebene Bild anstelle von *list-style-type* als Beschriftung verwendet. Wenn der Browser jedoch aus irgend einem Grund nicht in der Lage ist, das Bild herunter zu laden oder anzuzeigen, benutzt er wiederum *list-style-type.*

Beispiel:

- UL { list-style-image : url(http://png.com/ellipse.png)}

7.2.4 Die Eigenschaft list-style-position

Die Eigenschaft *list-style-position* steuert mit den Werten „*inline*“ und „*outside*“, ob bei einem Zeilenumbruch der Text am Anfang der Zeile oder erst ausgerichtet unter der Beschriftung anfängt.

Beispiel :

Elemente mit dem Wert Outside beginnen bei einem Zeilenumbruch erst nach der Beschrif tung, ist also ausgerichtet an der ersten Textzeile

Elemente mit dem Wert Inside beginnen bei einem Zeilenbruch mit der Beschriftung. Dieser Wert erzeugt eine kompaktere Liste.

7.2.5 Die Eigenschaft list-style

Die Eigenschaft list-style ist eine Kurzform, mit der sich die Beschriftung und deren Position gleichzeitig festlegen lassen. Ihre Werte sind die zulässigen Werte der Eigenschaften list-style-type, list-style-image und list-style-position, die nur mit einem Leerzeichen voneinander getrennt werden.

Beispiele:

- UL { list-style : disc inside }
- OL OL { list-style : circle outside }

7.3 Kategorie Leerraum

Um die Präsentation der HTML-Dokumente zu verbessern, verändert man die Schrift oder die Farbe der Elemente und zusätzlich den Leerraum. Vor CCS gab es in HTML drei Methoden, um den Leerraum zu steuern, nämlich mit Elementen, Bildern und Tabellen. Zum Beispiel verwendete man das Element PRE oder leere P-Element, um senkrechte Abstände zu vergrößern; durch Einfügen von transparenten Bildern, um kleine Änderungen an der Ausrichtung zu gestalten, oder mit Hilfe von Tabellen Leerraum zwischen Texten zu schaffen. Durch die Verwendung von CSS wird die Kontrolle über den Leerraum in Elementen und um diesen herum existierenden Leerraum stark erweitert.

In Übereinstimmung mit dem Kästchenmodell wird ein Blockelement, z. B. ein Absatz oder eine Überschrift, in ein imaginäres rechteckiges Kästchen gezeichnet, das den Text dicht umschließt. Außerhalb des umschließenden Kästchens liegen drei „Gürtel“, die in einer Formatvorlage bearbeitet werden können: margin(Rand), padding(Füllung) und border(Rahmen) (siehe Abbildung 7.3a).

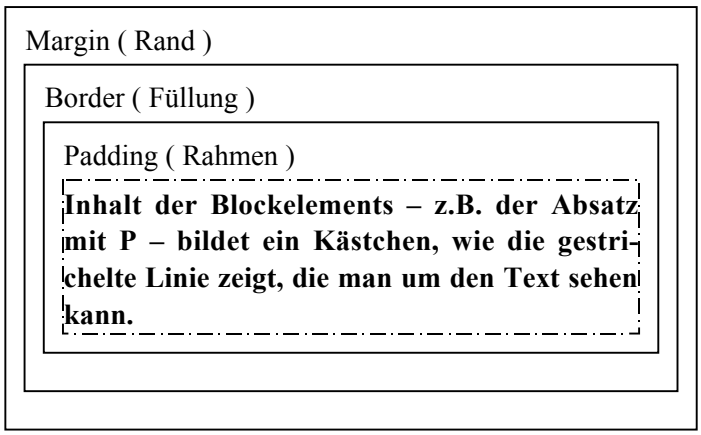


Abbildung 7.3a : Kästchenmodell für jedes Element

7.3.1 Randabstände und die Eigenschaften für den Randabstand

Der Rand ist der Raum zwischen dem umschließenden Kästchen des Elements und dem umschließenden Kästchen jedes angrenzenden Elements. Mit Hilfe der Eigenschaften „*margin-left*“, „*margin-right*“, „*margin-top*“ und „*margin-bottom*“ wird ein Zugriff auf den linken, rechten, oberen und unteren Rand ermöglicht. Den Eigenschaften können wieder Werte zugewiesen werden, so ist ein Längenwert - z.B. 10pt oder 2em - möglich, oder ein Prozentwert, der auf das Vaterelement Bezug nimmt.

Die fünfte Eigenschaft „*margin*“ erlaubt eine Angabe des Randes in Kurzform. Dies funktioniert folgendermaßen : Wenn bei dieser Eigenschaft nur ein Wert gesetzt wird, gilt dieser für alle vier Seiten. Wenn zwei oder drei Werte gesetzt werden, werden die fehlenden Seiten der gegenüberliegenden Seite übernommen. Angenommen, der obere Rand wird auf 3em und der rechte auf 2em gesetzt und für den unteren und linken Rand werden keine Werte zugewiesen. Der untere Rand liegt dem oberen gegenüber, deshalb wird er Wert für den unteren Rand übernommen: 3em. Der linke Rand liegt dem rechten Rand gegenüber, impliziert 2em für den rechten Rand. Wenn die Eigenschaft vier Werte mit Leerzeichen getrennt zugewiesen bekommt, gelten sie in der Reihenfolge oben/rechts/unten/links.

Beispiele:

- Body { margin : 2em } /* Alle Ränder betragen 2em */
- Body { margin : 1em 2em 3em } /* Der obere Rand berägt 1em,
  der linke und der rechte je 2em
  und der untere Rand 3em */

7.3.2 Die Eigenschaft der Füllung

Die Fülleigenschaften beschreiben, wieviel Raum zwischen einem Element und seinem Rand einzufügen ist, oder zwischen einem Element und seinem Rahmen, wenn ein Rahmen vorhanden ist. Die Breite der Füllung wird wieder durch vier verschiedene Eigenschaften gesteuert, die einzeln die Größe der Füllung festlegen : *padding-left, padding-right, padding-top* und *padding-bottom*.

Die Eigenschaften können wieder zwei verschiedene Werte annehmen, einmal in Form von einer Länge - z.B. 10pt - und die andere ist der Prozentwert, der zur Relation des Vaterelements steht.

Dann existiert noch die Eigenschaft *padding*, mit der alle vier Füllwerte auf einmal gesetzt werden kann. Das funktioniert folgendermaßen: Wenn bei dieser Eigenschaft nur ein Wert gesetzt wird, gilt dieser für alle vier Seiten. Wenn zwei oder drei Werte gesetzt werden, werden die fehlenden Werte von der gegenüberliegenden Seite übernommen.

Wenn bei dieser Eigenschaft vier Werte gesetzt werden, gilt die Reihenfolge oben/rechts/unten/links.

Beispiele:

/* Obere und untere Füllung betragen 1em, linke und rechte 2em */

- Body { padding : 1em 2em }

/* Alle Füllungen werden gesetzt, die Werte gelten in der Reihenfolge oben/rechts/unten/links */

- Body { padding : 1em 3em 5em 7em }

7.3.3 Die Rahmeneigenschaften

Der Rahmen bietet die Möglichkeit ein Element hervorzuheben. Dafür existieren zwölf Rahmeneigenschaften, mit denen man Breite, Farbe und Stil des Rahmens in verschiedenen Kombinationen festlegen kann. Mit den üblichen fünf Eigenschaften „*border-left*“, „*border-right*“, „*border-top*“, „*border-bottom*“ und „*border*“ wird die Breite des Rahmens bestimmt. Die Vorgehensweise ist dieselbe wie die Festlegung der Breite der Füllung.

Zusätzlich kommen die Eigenschaften „*border-color*“, „*border-style*“, „*border-left-width*“, „*border-right-width*“, „*border-top-width*“, „*border-bottom-width*“ und „*border-width*“ hinzu, die nur für ein Merkmal eingesetzt werden dürfen, d.h. nur die Breite, nur die Farbe oder bloß den Stil. Ein Rahmen gilt für jedes beliebige Element. Wenn der Rahmen sich auf ein Inline-Element bezieht, der sich über mehr als eine Zeile erstreckt, kann der Browser für jede Zeile einen Rahmen generieren und möglicherweise die Kanten weglassen.

Die Eigenschaft border-color

Die Eigenschaft *border-color* legt die Farbe des Rahmens fest. Die Farbe kann entweder als eine von 16 vordefinierten und benannten Farben oder als RGB-Farbnummer angegeben werden. Genaueres findet man unter der Eigenschaft *color*.

Weiter ist es möglich, jede Kante des Rahmens eine Farbe zu zuordnen, indem man mehrere Farbwerte, mit einem Leerzeichen getrennt, aufführt. Wenn ein Wert gesetzt ist, dann gilt dieser für alle Seitenkanten. Sind zwei Werte angegeben, dann werden oberer und unterer Rahmen auf den ersten, rechter und linker Rahmen auf den zweiten Wert gesetzt. Werden drei Werte gesetzt : Der obere Rahmen wird auf den ersten, der rechte und der linke Rahmen werden auf den zweiten, die untere Rahmenlinie wird auf den dritten Wert gesetzt. Bei vier Werten gilt die alt bekannte Reihenfolge oben/rechts/unten/links.

Die Eigenschaft border-style

Mit der Eigenschaft *border-style* wird ein Stil bestimmt, dabei legen folgende Schlüsselwörter das Erscheinungsbild des Rahmens fest:

- *none* - kein Rahmen, ohne Rücksicht auf eine evtl. gesetzte Rahmenbreite. Das ist die Standardeinstellung
- *dotted* - eine gepunktete Linie
- *dashed* - eine gestrichelte Linie
- *solid* - eine durchgezogene Linie
- *double* - eine doppelte Linie. Die Summe aus den beiden Linien und dem Zwischenraum entspricht dem Wert *border-width.*
- *groove* - eine 3D-Hohlkehle. Der Schatteneffekt entsteht durch die Verwendung von Farben, die ein wenig dunkler bzw. Heller sind als die in *bolder-color* oder *color* gesetzten.
- *ridge* - ein 3D-Wulst
- *inset* - eine 3D-Vertiefung
- *outset* - ein 3D-Erhebung

Jede Kante kann durch einer dieser Schlüsselwörter beschrieben werden, welche Kante nun welchen Stil unterliegt, wird wieder durch die Reihenfolge der Schlüsselwörter festgelegt:

- Ein Wert gesetzt: Dieser Wert gilt für alle vier Seiten
- Zwei Werte gesetzt: Oberer und unterer Rahmen werden auf den ersten, rechter und linker Rahmen auf den zweiten Wert gesetzt.
- Drei Werte gesetzt: Der obere Rahmen wird auf den ersten, der rechte und der linke Rahmen auf den zweiten, die untere Rahmenlinie auf den dritten Wert gesetzt.
- Vier Werte gesetzt: Die Werte gelten in der Reihenfolge oben/rechts/unten/links.

Die Eigenschaft border-width

Die *border-width*-Eigenschaften legen die Rahmenbreite der vier Seiten einzelnd oder in beliebiger Kombination fest. Es gibt für jede einzelne Rahmenseite eine Eigenschaft und zwar*: border-left-width, border-right-width, border-top-width* und *border-bottom-width*. Die fünfte, *border-width*, ist eine Möglichkeit, die erlaubt alle vier Rahmenbreiten in Kurzform auf einmal zu setzen. Es stehen sechs Wertetypen zur Verfügung :

- *thin* – dünn
- *medium* – Standardeinstellung
- *thick* – dick
- einen Längenwert - z. B. 2em
- einen Prozentwert
- *none*

Bei Verwendung der Schlüsselwörter „*thin*“, „*medium*“ oder „*thick*“ ist die tatsächliche Rahmenbreite vom Browser abhängig. Die Eigenschaft border-width, die als einzige unter diesen mehrere Werte haben kann, dient wieder als Kurzform, mit der alle Breitenangaben direkt angegeben werden können.

Die Festlegung der Rahmenlinien mit *border-width* folgt wieder ein Muster :

- Ein Wert gesetzt: Dieser Wert gilt für alle vier Seiten

- Zwei Werte gesetzt: Oberer und unterer Rahmen werden auf den ersten, rechter und linker Rahmen auf den zweiten Wert gesetzt.
- Drei Werte gesetzt: Der obere Rahmen wird auf den ersten, der rechte und der linke Rahmen auf den zweiten, die untere Rahmenlinie auf den dritten Wert gesetzt.
- Vier Werte gesetzt: Die Werte gelten in der Reihenfolge oben/rechts/unten/links.

Beispiel zur Eigenschaft border-width:

/* Alle Rahmen werden gleich gesetzt : 2em }

- BODY { border-width: 2em }

/* Oberer und unterer Rahmen sind 1em, linker und rechter Rahmen sind 2em */

- BODY { border-width: 1em 2em }

Die Eigenschaft border

Bisher ist es möglich, die Rahmeneigenschaften wie Rahmenbreite, -farbe und -stil zusammen für einen oder mehreren Rahmen festzulegen. Mit der Eigenschaft *border* lassen sich all diese Rahmeneigenschaften gleichzeitig mit einer Deklaration definieren. Der Wertetyp setzt sich zusammen aus dem Wertetyp der Eigenschaft *border-width*, gefolgt von dem Wertetyp der Eigenschaft *border-style* und der Eigenschaft *border-color*, die alle samt durch ein Leerzeichen getrennt sind. Die *border*-Eigenschaft akzeptiert also alle zulässigen Werte von *border-width, border-style* und *border-color*. Bei Fehlen eines Wertes der Eigenschaften *border-width, -style* und *-color* wird für diesen automatisch dessen Anfangswert angenommen und beeinflußt die Deklaration von *border* nicht. Hierzu schaue man sich dies Beispiel an :

- P { border : solid red }

Das bedeutet, daß alle Rahmen von dem Element P durchgezogen und in rot dargestellt werden. Da border-width nicht angegeben ist, wird dafür der Standartwert „medium“ angenommen. Auch die Reihenfolge, in der die drei Werte aufgelistet sind, spielt keine Rolle. Jede der folgenden Versionen liefert dasselbe Ergebnis :

- P { border: thin solid red }
- P { border: red thin solid }
- P { border: solid red thin }

Der Unterschied zu den Eigenschaften *margin* und *padding*, die beide in Kurzform Rand und Füllung deklarieren, kann die Eigenschaft *border* keine unterschiedlichen Werte für die vier Seiten festlegen. Dies resultiert aus der größeren Anzahl der Eigenschaften für den Rahmen. Deshalb kann man mit *border* nur denselben Stil, dieselbe Farbe und dieselbe Breite für alle vier Seiten angeben. Um unterschiedliche Werte vorzugeben, kommt man um die Eigenschaften wie z.B. boder-top-width, border-style nicht herum.

Eine weitere Besonderheit ist das Zusammenfallen der Ränder. Hierzu werden die Ränder ober- und unterhalb von verschiedenen Elementen nicht einfach addiert, um zu einem „Gesamtleerraum“ zwischen den beiden Elementen zu gelangen. Stattdessen übergeht der Browser den kleineren Abstand und verwendet den größeren, um die beiden Elemente voneinander abzusetzen. Dieser Vorgang wird als „Zusammenfallen der Ränder“ bezeichnet. Betroffen sind davon nur der obere und untere Rand zweier Elemente, die untereinander stehen.

7.3.4 Die Eigenschaft width

Mit der Eigenschaft *width* wird die Breite des Elements im Dokument festgelegt. Hierbei ist zu beachten, daß *width* bei Blockelementen selten verwendet wird, da sie einige Komplikationen mit sich bringen. Für diese Eigenschaft gelten wieder die drei Wertetypen : Länge; Prozentwert, der sich auf das Vaterelement bezieht; „*auto*“ - das ist die Standardeinstellung.

7.3.5 Die Eigenschaft height

Die Eigenschaft *height* legt die Höhe des Elements fest. Wie die Eigenschaft *width* wird auch die Eigenschaft *height* bei Blockelementen selten verwendet, und die Verwendung bei solchen Elementen bringt einige Komplikationen mit sich.

Doch wird sie bei Bildern häufig nützlich eingesetzt. Die Eigenschaft *height* akzeptiert zwei Wertetypen, erstens die klassische Längeneingabe und zweitens die Standardeinstellung „*auto*“. Die explizite Angabe der Höhe ist noch seltener als die Angabe der Breite. Falls sie benutzt wird, treten zwei Fälle auf: Wenn erstens der Text mehr Platz erfordert, als die Eigenschaft *height* erlaubt, kann der Browser einen Rollbalken oder ein ähnliches Hilfsmittel in das Element einfügen, so daß der Anwender an den Text gelangen kann, der nicht sichtbar ist. Wenn zweitens die Höhe größer ist, als es der Text des Elementes erfordert, wird der zusätzliche Platz als Füllung behandelt.

7.3.6 Die Eigenschaft float

In vielen Dokumenten gibt es vor allem Inline-Elemente, wie z. B. das Element IMG, die innerhalb eines Vaterelementes an der linken oder rechten Kante plaziert sind. Dies wird mit der Eigenschaft *float* und dessen zulässigen Werte „*left*“, „*right*“ und „*none*“ erreicht. Der Wert „*left*“ führt dazu, daß das Element soweit in Richtung der linken Kante seines Vaterelements verschoben wird, bis es an einen Rand, eine Füllung oder einen Rahmen eines anderen Blockelements trifft. „*Right*“ löst dieselbe Aktion auf der gegenüberliegenden Seite aus. Mit „*none*“ wird das Element an der Stelle angezeigt, an der es im Text erscheint. Wenn das zu-gleitende-Element an der angegebenen Kante nicht genug Platz zum Gleiten hat, bewegt es sich abwärts bis zur nächstliegenden geeigneten Stelle.

Wenn es einen negativen Rand hat, gleitet dieses Element im Vaterelement bis zum Rand und dann noch um den negativen Wert darüber hinaus. Beispielsweise schauen wir uns folgende Anweisungen an :

- IMG.icon { float : left;
  margin-left : 0 }
- IMG.icon { float : right;
  margin-right : -2em }

7.3.7 Die Eigenschaft clear

Die Eigenschaft *clear* wirkt mit der Eigenschaft *float* zusammen. Sie legt fest, ob ein Element gleitende Elemente an seiner Seite erlaubt, genauer gesagt listet sie auf, an welchen Seiten gleitende Elemente nicht akzeptiert werden.

Die Eigenschaft clear besitzt vier Werte :

- *none* - Das ist die Standardeinstellung

- *left*
- *right*
- *both*

Der Schlüsselwert „none“ erlaubt beidseitiges Einfügen von gleitenden Elementen. „*left*“ und „*right*“ bedeuten, daß das Element gleitende Elemente auf der linken bzw. rechten Seite nicht erlaubt. „*both*“ heißt, daß das Element auf beiden Seiten gleitende Element verbietet.

Zusätzlich kann die Eigenschaft *clear* auch für gleitende Elemente benutzt werden.

```
IMG {
        float : right;
        clear : right;
    }
```

Die Formatvorlage stellt zum Beispiel sicher, daß das eingefügte Bild an die rechte Kante des Vaterelements gleitet, und daß es nicht neben einem anderen gleitenden Element plaziert wird, das sich möglicherweise schon an der rechten Kante befindet. In diesem Fall verschiebt es sich so weit nach unten, bis es einen freien Platz findet des es auffüllen kann.

7.3.8 Die Eigenschaft text-align

Mit der Eigenschaft *text-align* legt man fest, ob der Text linksbündig, rechtsbündig, zentriert oder in einem Blocksatz ausgerichtet werden soll. Die vier zugehörigen Schlüsselwörter lauten : „*left*“, „*right*“, „*center*“ und „*justified*“.

Die Eigenschaft *text-align* wird vererbt, daher kann man die Ausrichtung des gesamten Dokumentes mit Hilfe des Elements BODY setzen, wie im folgenden Beispiel :

```
BODY { text-align : justify }
```

Noch zu beachten ist, daß die Textausrichtung sich nach der Breite des Elements richtet und nicht nach der Zeichenfläche. So wird der Text eines Elements, welches beispielsweise mit *text-align* auf „*center*“ gesetzt wurde, zwischen den Rändern des Elements zentriert, unabhängig davon, wo das Element auf der Zeichenfläche angeordnet wird. Daher mag der Text auf der Zeichenfläche vielleicht nicht zentriert erscheinen.

7.3.9 Die Eigenschaft text-indent

In einer von links nach rechts geschriebenen Sprache wie Deutsch oder Französisch rückt man am rechten Rand ein, d.h. alle Zeilen stehen schön linksbündig untereinander. Die Eigenschaft *text-indent* gibt den Einzug der ersten Zeile eines Absatzes an. Dabei kann die erste Zeile vor dem besagten Rand anfangen, oder erst später, alles je nachdem, ob der Wert ein negatives Vorzeichen oder ein positives besitzt. Der Wertetyp ist also eine Länge oder ein Prozentsatz der Breite des Absatzes; 10% heißt z. B., daß die erste Zeile des Absatzes um 10 Prozent der Breite des Absatzes eingerückt werden soll.

Da *text-indent* eine vererbte Eigenschaft ist, die aber nur berechnet übergeben wird, wird die Breite des Einzugs einmal für das Vaterelement berechnet, und an alle Sohnelemente vererbt. Der Wert wird für ein Sohnelement nicht neu berechnet.

7.3.10 Die Eigenschaft line-height

Die Eigenschaft *line-height* verwendet man, um die Entfernung der Zeilen eines Absatzes zu bestimmen, d.h. diese Eigenschaft gibt den minimalen Abstand zwischen den Grundlinien der benachbarten Zeilen an.

Weiterhin besitzt diese Eigenschaft drei mögliche Werte: eine Zahl oder eine Länge oder einen Prozentwert.

Die drei Werte sind absolut, relativ oder geräteabhängig, je nach dem welche Maßeinheit genommen wurde. Der einzige und sehr wichtige Unterschied, ist die Behandlung der Vererbung. Eine als Zahl angegebener Zeilenabstand wird für das Vaterelement und auch für jedes Tochterelement berechnet. Der Zeilenabstand, der dem Prozentwert entspricht, wird einmal berechnet, und alle Tochterelemente erben das Ergebnis. (Das gilt für alle Eigenschaften, für die ein Prozentwert oder eine em-Einheit als Wert zulässig sind.)

7.3.11 Die Eigenschaft word-spacing

Wenn man nun die Abstände zwischen den Wörtern einstellen möchte, dann verwendet man die Eigenschaft *word-spacing*. Jede Schrift besitzt einen normalen Wortabstand. Um bestimmte Effekte im Text zu erreichen, gibt es zudem das Schlüsselwort „*normal*“, welches eine Länge von 0 entspricht, die Möglichkeit, einen Längenwert direkt anzugeben, der dann zum normalen Wortabstand hinzuaddiert wird. *Word-spacing* ist eine vererbte Eigenschaft, aber nur der berechnete Wert vom Vater wird übergeben und weiter vererbt, ohne einer neuen Berechnung. Dadurch verursachen verschieden große Schriften der Tochterelemente eines Vaters, der den einmal berechneten Wortabstand vererbt, ein rücksichtsloses Aussehen.

7.3.12 Die Eigenschaft letter-spacing

Mit der Eigenschaft *letter-spacing* kann man den Abstand zwischen den Buchstaben einstellen. Wie beim Wortabstand besitzt jede Schrift auch einen normalen Zeichenabstand. Auch sind die beiden Wertetypen dieselben: „normal“, d.h. der Buchstabenabstand bleibt dem Browser überlassen. Dies ist die Standardeinstellung; eine Länge, die zu dem normalen Zeichenabstand hinzuaddiert wird. Für die Vererbung gilt, daß das Vaterelement einmal den Buchstabenabstand berechnet und diesen dann an alle Erben weitergibt, ohne daß eine neue Berechnung stattfindet.

Beispiel:

- Blockqoute { letter-spacing : 0.04in }
- P { letter-spacing : 0.1em }

7.3.13 Die Eigenschaft vertical-align

Mit der Eigenschaft *vertical-align* plaziert man die Buchstaben und Bilder oberhalb oder unterhalb der Grundlinie des Textes und ist ein weiteres Mittel zur Hervorhebung des Textes. Die unsichtbare Grundlinie ist die normale Zeilenhöhe, an der der Text normalerweise ausgerichtet wird. So sind für *vertical-align*, welches nur für inline-Elemente gilt, unterschiedliche Schlüsselwörter definiert. Sechs der acht verfügbaren Schlüsselwörter verhalten sich relativ zum Vaterelement, die anderen Schlüssel-

wörter, „*top*“ und „*bottom*“ dulden neben sich noch einen zusätzlichen Prozentwert als Wert, der besagt, um wieviel das Element höher oder tiefer angeordnet werden soll. Die sechs Schlüsselwörter, die relativ zum Vaterelement stehen :

- *baseline* - richtet die Grundlinie des Tochterelements an der Grundlinie des Vaterelements aus. Das ist die Standardeinstellung. Ein Inline-Element ohne Grundlinie, z.B. ein Bild oder Objekt, wird mit der Unterkante an der Grundlinie des Vaterelements ausgerichtet.
- *sub* - stellt das Element tief, d.h. richtet die Grundlinie des Elements an der bevorzugten Position des Vaterelements für Tiefstellungen aus . Diese hängt normalerweise von der Schrift des Vaterelements ab. Wenn die Schrift diese Position nicht explizit definiert, wählt der Browser eine „sinnvolle“ Position.
- *super* - stellt das Element hoch, d.h. richtet die Grundlinie des Elements an der bevorzugten Position des Vaterelements für Hochstellungen aus. Diese hängt normalerweise von der Schrift des Vaterelements ab. Wenn die Schrift diese Positionen nicht explizit definiert, wählt der Browser eine sinnvolle Position.
- *text-top* - richtet die Oberkante des Elements an der Oberkante des höchsten Buchstaben des Vaterelements aus. Einige Designer bevorzugen diese Möglichkeit der Ausrichtung, zum Beispiel bei N° anstelle von „super“.
- *middle* - richtet den vertikalen Mittelpunkt des Elements an der Grundlinie plus der halben x-Höhe des Vaterelements aus, d.h. an der Mitte der Kleinbuchstaben des Vaterelements. Genauer gesagt, wird das Element auf einer Linie zentriert, die 0,5ex über der Grundlinie liegt.
- *text-bottom* - richtet die Unterkante des Elements an der Unterkante der Schrift des Vaterelements aus.

Die Schlüsselwörter „*top*“ und „*bottom*“ haben Definitionen, die nicht schwieriger aussehen als die der ersten sechs gerade beschriebenen Schlüsselwörter. Das Element, für das *vertical-align* auf „*top*“ gesetzt ist, wird mit seiner Oberkante an der Oberkante des höchsten Objekts in der Zeile ausgerichtet. Der Wert „*bottom*“ richtet die Unterkante des Elements an der Unterkante des tiefsten Objekts in der Zeile aus.

7.4 Kategorie Farben und Background

Die Grundlage zur Darstellung von Farben ist das RGB-Farbmodell, dabei wird jede der drei Farben Rot, Grün und Blau einen Prozentwert von 0 bis 100% annehmen. Die Werte werden als Dreiergruppe angegeben, in der die erste Zahl für Rot, die zweite für Grün und die letzte für Blau steht. Schwarz wird durch (0,0,0) - keine Anteile der drei Farben - und Weiß durch (100%,100%,100%) definiert. Als Alternative zu Prozentwerten stehen noch Bytewerte, eine Zahl zwischen 0 und 255, zur Verfügung, die mit einem Tupel die Farbe bestimmen. An dieser Stelle sei noch gesagt, daß wenn man die Bytewerte in Hexadezimalzahlen umwandeln kann, diese dann ohne Leerzeichen hintereinander packt und eine Raute voranstellt, noch eine weitere Alternative gegeben ist.

Beispiel : Die Farbe rgb (255,91,19) in Byte-Darstellung entspricht in Hexadezimal-Darstellung dem Wert #FF5B1A.

Schließlich gibt es noch eine Variante, die nicht auf das RGB-Modell basiert. Diese definiert Farbnamen vor. Es werden in CSS1 genau 16 Farbnamen vorgegeben: aqua (ein helles Grünblau, auch Cyan), black, blue, fuchsia (helles Purpur/Pink), gray, green, lime (Hellgrün), maroon (Dunkelblau), olive, purple, red, silver, teal (Blaugrün), yellow, white.

7.4.1 Die Eigenschaft color

Mit der Eigenschaft *color* wird die Textfarbe eines Elements festgelegt. Dabei handelt es sich um die Vordergrundfarbe, dessen Wert - eine Farbe - mit Hilfe der oben anstehenden Theorie beschrieben werden muß.

Zusätzlich stehen die Schlüsselwörter der Farben wie „black“, „silver“, „red“ u.v.a. zur Verfügung.

Einige Beispiele:

- EM { color : red }
- P { color : rgb(255, 0, 0) /* Bytewert für rot */
- H1 { color : #f00 /* auch rot */
- H1 { color : #ff0000 /* rot */

Eine Farbe wird von Tochterelementen geerbt.

7.4.2 Die Eigenschaft background-color

Die Eigenschaft *background-color* legt die Hintergrundfarbe eines Elements fest. Wenn ein Wert für „Farbe“, wie oben angegeben, eingegeben wird, wird die betreffende Farbe hinter dem Text des Elements sichtbar. Inwieweit die Oberfläche diese Farbe annimmt, hängt von der Art des Elements und der Anzahl der Füllzeichen ab.

Der zweite Wert, der alternativ zur Farbe angegeben werden kann, lautet „transparent“ und läßt den Hintergrund transparent werden, so daß die Hintergrundfarbe oder Hintergrundbild, welches mit der Eigenschaft *background-image* eingefügt werden kann, des letzten nicht transparenten Vorfahren durchscheint.

7.4.3 Die Eigenschaft background-image

Ein Bild als Hintergrund wird mit der Eigenschaft *background-image* eingebunden, die als Wert eine URL-Adresse erhält. Wenn man ein Bild als Hintergrund wählt, sollte man auch eine Hintergrundfarbe mit der Eigenschaft *background-color* definieren. Man tut dies aus verschiedenen Gründen :

- Das Bild überlagert die Farbe an den nicht transparenten Stellen.
- Die Farbe kann man verwenden, um transparente Bereiche des Bildes auszufüllen; andernfalls bleiben diese Bereiche transparent.
- Die Farbe kann zum Ausfüllen des Bildschirms während des Ladevorgangs verwendet werden, zum Beispiel wenn dieser zu lange dauert.
- Die Farbe kann an die Stelle des Bildes treten, wenn das Bild nicht geladen werden kann, zum Beispiel wenn der Browser es nicht findet.

Wie immer gibt es für die Eigenschaft background-image einen Standardwert, dieser wird mit dem Schlüsselwort „none“ beschrieben.

Beispiel:

```
• P   { background-image : url(dot.gif);
        background-color : #FFAA00
      }
```

7.4.4 Die Eigenschaft background-repeat

Die Eigenschaft *background-repeat* bestimmt, ob und auf welche Weise ein Bild des Elements wiederholt wird, welches standardmäßig in der oberen linken Ecke des Elements beginnt. Diese Eigenschaft hat vier mögliche Werte, die unterschiedliche Wiederholungsarten erlauben. Wenn das Bild sowohl horizontal als auch vertikal so oft wie möglich wiederholt werden soll, um das gesamte Element auszufüllen, dann verwendet man die Standardeinstellung „*repeat*“. Diesen Vorgang nennt man auch kacheln.

Wenn das Bild nur horizontal (auf der x-Achse) in einer einzigen Zeile innerhalb des Elements wiederholt werden soll, und zwar sowohl links als auch rechts von der Ausgangsposition, dann nimmt man das Schlüsselwort „*repeat-x*“. Natürlich gibt es auch ein Schlüsselwort, welches die gleiche Vorgehensweise für eine Spalte realisiert, namens „*repeat-y*“. Wenn das Bild nicht wiederholt werden soll, sondern nur einmal in der oberen linken Ecke des Elements oder an der mit der Eigenschaft *background-position* festgelegten Stelle erscheinen soll, dann wird dies mit „no-repeat“ festgelegt.

Bei wiederholten Bildern handelt es sich oft um wiederholte Muster, wie zum Beispiel Punkt- und Wellenmuster.

7.4.5 Die Eigenschaft background-attachment

Wenn das Element mit einer Bildlaufleiste, die nach oben und unten verschoben werden kann, versehen ist, was passiert dann mit dem Hintergrundbild in diesem Element ? Die Eigenschaft *background-attachment* legt fest, ob das Bild auf dem Canvas, ein Element mit Bildlaufleiste, fest oder verschiebbar sein soll. Dazu stehen zwei Schlüsselwörter als Werte zur Verfügung. Einmal bestimmt das Schlüsselwort „scroll“, daß das Bild mit dem Inhalt mitlaufen soll, d.h. das Bild bleibt bei seinem Element, so daß das Bild dorthin mitläuft, wohin das entsprechende Element wandert. Dies ist auch die Standardeinstellung. Das andere Schlüsselwort, wo das Bild fest auf der Zeichenfläche bleiben soll, heißt „fixed“, welches auch bedeutet, daß der Hintergrund nicht mit dem Element verbunden ist.

Wenn der Benutzer einen Bildlauf durchführt, verschiebt sich zwar das Element, nicht aber der Hintergrund.

7.4.6 Die Eigenschaft background-position

Wir wissen, daß die Standardposition von Bildern in Elementen sich oben rechts befindet, die aber mit der Eigenschaft background-position geändert werden kann. Die Eigenschaft hebt als die Standardposition auf und zwar sowohl für ein einzelnes Bild als auch für ein wiederholendes Bild. Das neue Plazieren eines Hintergrundbildes kann man mit drei Methoden angehen. Die erste Methode ist das Plazieren von Bildern mit Hilfe von Prozentwerten, die im Verhältnis zur Größe des Elements sich befinden. Dabei werden zwei Prozentwerte mit einem Leerzeichen getrennt angegeben, so daß der erste die horizontale Position in Prozenten, die sich auf die horizontale Breite des Elements beziehen, beschreibt und der zweite Prozentwert die vertikale Position im Verhältnis zur Höhe des Elements angibt. Wenn nur ein Prozentwert angegeben wird, dies ist eine alternative Möglichkeit, dann gilt dieser Wert für die horizontale und vertikale Position. Um zum Beispiel ein Bild in einem Element zu zentrieren, würde man einfach 50% eingeben, und um ein Bild an der rechten Kante des Elements zu plazieren, bräuchte man nur 100% anzugeben.

Die zweite Methode ist das Plazieren von Bildern mit Hilfe absoluter Positionsangaben. Dabei werden die zwei Längenwerte anstatt zweier Prozentwerte durch Maßeinheiten, wie zum Beispiel 5mm, 1cm usw., eingegeben. Wie bei den Prozentwerten erfolgt auch hier bei Eingabe eines einzigen Wertes eine vertikale Zentrierung des Bildes.

Die letzte Methode ist das Plazieren mit Hilfe von Schlüsselwörtern. Man verwendet dabei eine beliebige Kombination zweier Schlüsselwörter, wobei eines aus den drei Schlüsselwörter - „*top*“, „*center*“, „*bottom*“ -, die das horizontale Maß darstellen, ist und das andere aus den Schlüsselwörten -„*left*“, “*center*“ und „*right*“ - das vertikale Maß bestimmt. Die Reihenfolge, in der die Schlüsselwörter angegeben sind, spielt keine Rolle. Wenn man nur eine Angabe macht, dann wird diese immer mit dem Schlüsselwort „*center*“ ergänzt. Daher führt die Anweisung

```
• Body {
        background-image : url(banner.jpeg);
        background-position :  top
  }
```

zum selben Ergebnis wie

```
• Body {
        background-image : url(banner.jpeg);
        background-position : top center
  }
```

Die Schlüsselwörter kann man nicht mit Prozent- oder -Absolutwerten kombinieren.

7.4.7 Die Eigenschaft background

Die Eigenschaft *background* ist eine abgekürzte Form zu gleichzeitigen Einstellung aller fünf vorgenannten Eigenschaften. Ihre Werte entsprechen den für diese fünf Eigenschaften möglichen Werten, die mit einem Leerzeichen voneinander getrennt werden. Ferner werden immer fünf Werte für die fünf Eigenschaften eingestellt, auch wenn man nicht für alle fünf ausdrücklich einen Wert angegeben hat. Damit werden unschöne Zufallsprodukte verhindert.

8 Kurzreferenz auf die neuen CSS2-Funktionalitäten:

Die einzelnen unterschiedliche Empfehlung wie CSS, CSS1 und CCS2 unterscheiden sich nur in ihrer Vielzahl an Eigenschaften und die dadurch ergebenen Möglichkeiten. Die nachfolgende Kurzreferenz von CSS2 ist eine unvollständige, kurze Aufzählung der Funktionen, die neu hinzugefügt worden sind.

8.1 Einführung von Media-Types

Die Medien wie Bildschirm oder Drucker können mit Media-Types in ihren Aussehen beeinflußt werden. So beschreibt die Anweisung

- @media print { Body : font-size : 10pt }

das Aussehen des Bildschirms. Dabei stehen alle Formatierungsanweisung innerhalb der von dem Ausdruck „@media print“ folgenden geschweiften Klammern.

8.2 Einführung der Eigenschaft „inherit“

In Cascading Style Sheets 1 existieren Eigenschaften, die nicht vererbt werden wie zum Beispiel die *background-color*, alle *margin-Eigenschaften* u.v.a. . Diese Besonderheit wird in CSS2 dadurch aufgehoben, daß Eigenschaften die geerbt werden sollen einfach den Wert „inherit“ zugewiesen bekommen.

8.3 Formatvorlagen für gesprochenes HTML

Die Einführung der Aural-Style-Sheets ermöglichen ein gesprochenes HTML. Hierzu sind wieder viele Eigenschaften definiert, wie zum Beispiel *voice-family, volume*, um einige zu nennen.

8.4 Erweiterte Listen-Numerierungstypen

Zu den römischen, alphabetischen Numerierung gibt es in CSS2 auch Exoten wie hebräisch oder „decimal-leading-zero“ Numerierung, um nur zwei Exoten einmal zu nennen.

8.5 Erweiterte Font-Eigenschaften

Mit dem Ausdruck „@font-face“ und anschließenden geschweifenten Klammern kann innerhalb dieser eine ganz neue Schrift erstellt oder modifiziert werden. So stellt CSS2 die Eigenschaft *font-stetch, scr* u.v.m. zur Verfügung, die das Aussehen steuern können.

8.6 Beeinflussung der HTML-Desktop-Aufteilung

Die Teile wie HEAD, MAIN oder FOOTER können einzelnd refomartiert werden. Die Syntax lautet im Beispiel MAIN :

- #main { top : 5pt /* oberer Abstand = 5pt */ }

8.7 Eigenschaften für Tabellen

Das Handling mit Tabellen wird durch CSS2 einfacher gestaltet.

8.8 Erweiterte Selektoren

Zusätzliche Selektoren wie Child-Selektor oder Nachbar-Selektor wurden erschaffen, weiter wurden die Pseudoklassen und Pseudoelemente in CSS2 ergänzt.

8.9 Erzeugenden Text nun möglich

Mit den neuen Pseudoelementen „:before“ und „:after“, und der Eigenschaft „content“ ist es möglich erzeugende Text, d.h. einen String, der schon in CSS definiert ist, vor oder nach einem Text in HTML anzufügen.

9 Kaskadierung

Die Cascading Style Sheets stellen eine Formatvorlagensprache dar und wir wissen, daß diese Formatvorlage vom Browser oder vom Designer stammen, aber auch vom Benutzer kommen können. So kann zum Beispiel der Designer eine Überschrift mit einer serifenlosen Schrift setzen, der Benutzer würde aber dort eine Serifenschrift bevorzugen. Es entsteht also bei der Darstellung eines Dokumentes, welches sich auf überlappende Formatvorlagen unterschiedlicher Quellen bezieht, ein Konflikt, der mit Hilfe des Mechanismus der Kaskadierung zu lösen gilt.

Dabei wählt der gegenwärtige Mechanismus bei überlappenden Anweisungen immer nur eine einzige Anweisung aus, die dann den fraglichen Wert steuert.

Die Herausforderung besteht darin, die richtige Anweisung auszuwählen. Die Idee ist ein Sortieren aller Anweisungen, die für ein Element gelten, in der Reihenfolge, daß Anweisungen aus der Standardformatvorlage des Browsers die geringste Priorität haben, gefolgt von den Formatvorlagen des Benutzers und den Vorlagen des Designers mit höchster Priorität.

Manche betrachten dies als unfair, so daß CSS zwei Möglichkeiten anbietet, so daß die Benutzeranweisungen die Oberhand gewinnen können. Erste Möglichkeit ist das Ausschalten von Formatvorlagen mit Hilfe des Browsers. Die zweite Möglichkeit ist mit dem Schlüsselwort „!important“ jede Deklarationszeile zu vermerken, beispielsweise mit der Formatvorlage des Benutzers:

- Body {color : black !important;}

Zusätzlich kann aber der Designer, der die höchste normale Priorität besitzt, auch das Schlüsselwort „!important“ benutzen und setzt sich damit vor den Anweisungen des Benutzers, die eventuell auch mit „!important“ höher bewertet wurden, durch. In **CSS2** wird dies korrigiert, hier ist ein Benutzer mit der „!important“-Gewichtung vor dem Designer, der auch eine „!important“-Gewichtung ansetzt, zu finden. CSS2 behandelt die „!important“-Gewichtung“ also genau umgekehrt als CSS1.

Auch Vererbung kann zu einer überlappenden Anweisung beitragen, so vererbt ein Body mit serifenloser Schrift, alle Söhne, Enkel, Urenkel usw. diese Eigenschaft, daß eine andere Formatvorlage mit Serifenschrift für Elemente, welche in dem Stammbaum des Body´s zu finden sind, zu einer Kaskadierung führt. Dabei setzt sich der Kaskadierungsmechanismus gegenüber der Vererbung durch.

Die Kaskadierung läuft nach einem immer gleichenden Mechanismus ab, um die Anweisungen zu finden, die die höchste Priorität besitzen. Dabei baut man auf folgende Voraussetzung, daß alle Formatvorlagen, die mit dem Ausdruck „@import“ eingebunden werden, am Anfang der Style Sheets zu finden sind. Dies macht es einfach zu erkennen, welche Anweisungen sich durch den Import von anderen Formatvorlagen überlappen.

Nun folgt der Algorithmus, der zu jeder Eigenschaft und Element den sich durchsetzenden Wert bestimmt. Die Eingabe des Algorithmus ist also eine Eigenschaft und das damit verwickelte Element.

1. Suche alle anwendbaren Anweisungen, die die Eigenschaft im Deklarationsteil verwendet, zusammen. Wenn man keine Anweisungen zu dieser Eigenschaft findet, nimmt man den vererbten Wert, der mit Hilfe der Baumstruktur festgelegt wird. Wenn kein Wert für diese Eigenschaft vererbt wird, nimmt man den Standardwert, der vom Browser gegeben ist.
2. Sortieren der Anweisungen nach expliziter Gewichtung. Dabei genießen Deklarationen mit der Markierung „!important“ eine besondere Bedeutung als normale Deklarationen.
3. Sortieren nach Ursprung: Den Formatvorlagen des Designers wird ein größeres Gewicht beigemessen als den Formatvorlagen der Benutzer, die wiederum die Formatvorlagen des Browser überschreiben.
4. Sortieren nach spezieller Bedeutung: das Prinzip lautet, daß eine sehr spezielle Anweisung den Vorrang vor einer allgemeineren Anweisung hat. Um spezielle Anweisungen zu finden, vergibt

man eine Wertung, dabei zählt man die Anzahl der ID-Attribute des Selektors (a), die Anzahl der CLASS-Attribute des Selektors (b), und die Anzahl der Verschachtelungstiefe des Selektors (c) . Danach fügt man die Anzahl zu einer gesamten Zahl zusammen, die Anzahl von (a) mal Hundert, addiert zu der Anzahl von (b) mal Zehn und die Anzahl von (c) einfach noch hinzu addiert. Um so höher die Zahl, um so spezieller die Anweisung und damit um so wichtiger.

Beispiele:

- LI {...} /* a=0 b=0 c=1 -> Priorität = 1 */
- UL LI {...} /* a=0 b=0 c=2 -> Priorität = 2 */
- UL OL LI {...} /* a=0 b=0 c=3 -> Priorität = 3 */
- LI.red {...} /* a=0 b=1 c=1 -> Priorität = 11 */
- UL OL LI.red {...} /* a=0 b=1 c=3 -> Priorität = 13 */
- #x34y {...} /* a=1 b=0 c=0 -> Priorität = 100 */

5. Sortieren nach vorgegebener Reihenfolge: je später eine Anweisung in den Style Sheets angegeben wurde, desto größer ist ihre Bedeutung.

Die Suche nach den Wert der Eigenschaft kann beendet werden, wenn eine Anweisung gefunden ist, die höher ist als alle anderen.

10 Verknüpfung mit HTML-Dokument

CSS-Formatvorlage müssen grundsätzlich mit dem HTML-Dokument verbunden sein, damit die gewünschte Darstellungsänderung auch erzielt wird. Dafür existieren vier Arten für eine Anbindung.

1. Das Anbinden mit Hilfe des HTML-Elements STYLE innerhalb eines HTML-Dokument ist die erste Möglichkeit. Dafür stellt HTML extra STYLE-HTML-Tags zur Verfügung , in denen man in einer Formatvorlagensprache Anweisungen erstellen kann. Da es vielleicht in der Zukunft mehr als eine Formatvorlagensprache geben könnte, muß man dem Element STYLE mit dem Attribut TYPE die zugehörige Art der Formatvorlagensprache mitteilen. Der Attributwert von CSS, die bis jetzt einzige Formatvorlagensprache, lautet „text/css“. Zu finden ist dieses STYLE-Element, die Formatvorlage also, immer am Anfang des HTML-Dokuments vor dem BODY-Element. Ein Beispiel zur Anbindung mit Hilfe des Elements STYLE:

```
<HTML>
 <HEAD>
  <TITLE>Titel</TITLE>
  <STYLE TYPE="text"/css">
       H1      { color : red }
  </STYLE>
 </HEAD>
 <BODY>
       <H1>Grundlegendste Anbindungsmöglichkeit an ein Dokument</H1>
 </BODY>
</HTML>
```

2. Zusätzlich stellt HTML ein Attribut namens STYLE zur Verfügung, mit dessen man für jeden Teil der Struktur auch gleich sein Aussehen mitbestimmen kann. Dabei gebraucht man das Attribut

STYLE ausschließlich innerhalb des Body-Elementes und ist auch wie ein Attribut in HTML zu behandeln. Das bedeutet, daß in einem Anfangs-HTML-Tag, welches den Teil eine Struktur einleitet, das Attribut STYLE deklariert wird, indem man nach dem Gleichheitszeichen in Anführungsstrichen die CSS-Deklarationen zuweist. Ein Beispiel zur Anbindung mit dem Attribut STYLE:

```
<HTML>
 <TITLE>H1 in rot durch CSS in einzelnen Elementen</TITLE>
 <BODY>
      <H1 STYLE="color : red"> Überschrift ist rot </H1>
 </BODY>
</HTML>
```

3. Die dritte Möglichkeit ist mit dem HTML-Element LINK externe Formatvorlagen an das Dokument zu binden. Diese benutzt man meistens, um dem Browser eine alternative Formatvorlage vorzugeben, die er laden kann, wenn die eigentlich vorgesehene nicht eingesetzt werden kann. Die Verbindung des Dokuments mit einer Formatvorlage wird mit dem Attribut REL="stylesheet" eingeleitet, und danach wird mit dem anschließenden Attribut HREF verbindlich die URL angegeben, unter die die Formatvorlage zu finden ist. Ein Beispiel zur LINK-Anbindung:

```
<LINK REL="stylesheet" HREF="http://place.com/sty/rf.css">
```

4. Auch bietet die CSS-Notation eine Möglichkeit, um innerhalb der Formatvorlagensprache eine externe Formatvorlage zu importieren und automatisch mit der aktuellen zu verbinden. Dafür verwendet man die CSS-Notation @import „http://....URL-adress..".

Abschließend betrachten wir die CSS-Datei, welche mit einer Textdatei der Erweiterung „.css" repräsentiert wird, die die Formatvorlage und die Anweisungen nach der CSS-Grammatik speichert. Eine spezielle Kopf-Deklaration wie es bei HTML-Dokumenten der Fall ist, wird für die Style Sheets nicht benötigt.

11 Unterstützung durch Web-Browser

Die erste Browser, der die Cascading Style Sheets und CSS1 unterstützte, war der Microsoft Internet Explorer 3.0 und rasch folgte auch der Netscape Navigator 4.0. Für Cascading Style Sheets 2.0 existieren noch keine Browser, die diese Version vollständig unterstützen. Zwar setzte Microsoft im Herbst 1998 mit dem Internet Explorer 5.0 den ersten Schritt, in dem teilweise die neuen Funktionen und Eigenschaften von Cascading Style Sheets 2.0 unterstützt werden. Welche Funktionen aber genau unterstützt werden, war mir, dem Autor dieser Ausarbeitung, bis dato nicht zu gängig.

12 Literatur

[LieH 97] Lie, Hakon, Wium, Bert Bos: Cascading Style Sheets, Boon, Addison-Wesley, 1997

[Ragg 97] Dave Ragget et. al. : HTML 4, Addison-Wesley, 1997

[Zack 98] Craig Zacker: 10 minute guide to HTML Style Sheets

[Jone 97] Joseph R.Jones: Cascading Style Sheets

[www3 98] http://www.w3.org/Style/css: letzter Besuch 22.12.98

[awl 97] http://awl.com/css/: letzter Besuch 22.12.98

[dell 98] http://dell.com/outlet : letzter Besuch 30.11.98

[mmte 98] http://www.mmtert.com/ix : letzter Besuch 30.11.98

Einführung in XML

Markus Hövener

FB Informatik, Uni Dortmund
Markus.Hoevener@evision.de

Zusammenfassung

XML ist eine Teilmenge von SGML, die speziell für die Verwendung im Internet konzipiert wurde. XML-Dokumente sind nicht format- sondern strukturorientiert, zeichnen also innerhalb des Dokumentes logische Strukturen aus.

1 Einleitung

XML (Extensible Markup Language) ist eine vom W3C herausgegebene Empfehlung. Aktuell ist die am 10. Februar 1998 veröffentlichte Version 1.0. Bei XML handelt es sich um eine Teilmenge von SGML. Mit XML ist es möglich, strukturorientierte Sprachen zu definieren, in denen logische Elemente ausgezeichnet werden.

Im Gegensatz zu HTML, das eine feste, nicht erweiterbare Menge von Tags definiert , benötigt ein XML-Dokument i.d.R. eine Dokumenttyp-Definition (DTD), in der festgelegt wird, welche Elemente und Attribute im Dokument vorkommen dürfen.

Das folgende XML-Dokument bezieht sich daher in der zweiten Zeile auf die Datei buchladen.dtd, in der diese Strukturbeschreibung zu finden ist:

```
<?XML Version="1.0"?>
<!DOCTYPE buchladen SYSTEM "buchladen.dtd">
<buchladen spezialitaet="Computerbuecher">
    <buch schlagwort="XML" isbn="3-8273-1330-9">
        <titel>XML in der Praxis</titel>
        <autor>Henning Behme</autor>
        <autor>Stefan Mintert</autor>
        <preis waehrung="DM">69,90</preis>
        <verfuegbarkeit lieferbar="y" dauer="3 Tage"/>
    </buch>
    <buch schlagwort="XML" isbn="3-8273-1331-0">
        <titel>XML - Das &amp;quot;Kochbuch&amp;quot;</titel>
        <autor>Stefan Mintert</autor>
        <verfuegbarkeit lieferbar="n"/>
    </buch>
</buchladen>
```

Beispiel 1: XML-Dokument

Während die ersten beiden Zeilen den sogenannten Prolog bilden (bestehend aus XML-Deklaration und Dokumenttyp-Deklaration), stellt der Rest des Dokumentes die Instanz der DTD dar.

In XML ist es zwingend, zu jedem Start-Tag (z.B. <buchladen>) auch ein End-Tag (z.B. </buchladen) anzugeben; hier zeigt sich die erste gravierende Einschränkung gegenüber SGML. Einzige Ausnahme bilden die leeren Elemente, die lediglich über ihre Attribute definiert werden (im Beispiel: verfuegbarkeit).

Das äußere Element, das alle weiteren Elemente beinhaltet, heißt Wurzelelement (im Beispiel: buchladen). Der Name des gewünschten Wurzelelementes wird in der zweiten Zeile, der Dokumenttyp-Deklaration, definiert.

Die DTD zu diesem Dokument muß verschiedene Aspekte modellieren, z.B.

- Das Element buchladen kann beliebig viele Elemente vom Typ buch beinhalten.
- Das Element buch enthält genau einen titel, mindestens einen autor, evtl. einen preis und genau eine Angabe zur Verfügbarkeit.
- Das Element verfuegbarkeit beinhaltet keinen Text.
- Das Element buchladen hat ein optionales Attribut spezialitaet.
- Das Attribut waehrung kann nur die Werte DM oder Euro annehmen.

Die genaue Beschreibung der Elemente einer DTD liefert Kapitel 4. Eine mögliche DTD des Beispieldokumentes lautet wie folgt:

```
<!ELEMENT buchladen      (buch*)>
<!ELEMENT buch           (titel,autor+,preis?,verfuegbarkeit)>
<!ELEMENT titel          (#PCDATA)>
<!ELEMENT autor          (#PCDATA)>
<!ELEMENT preis          (#PCDATA)>
<!ELEMENT verfuegbarkeit  EMPTY>

<!ATTLIST buchladen
        spezialitaet CDATA     #IMPLIED>
<!ATTLIST buch
        schlagwort   CDATA     #IMPLIED
        isbn         CDATA     #REQUIRED>
<!ATTLIST preis
        waehrung     (DM|Euro) #REQUIRED>
<!ATTLIST verfuegbarkeit
        lieferbar    (y|n)     #REQUIRED
        dauer        CDATA     #IMPLIED>
```

Beispiel 2: XML-DTD (buchladen.dtd)

2 Die Grammatik

Der XML-Standard Version 1.0 verwendet für die Definition seiner kontextfreie Grammatik insgesamt 89 Produktion, die in EBNF (Erweiterte Backus-Naur-Form) vorliegen. Da in den folgenden Kapiteln viele dieser Produktionen angegeben werden, wird hier zuerst die Notation in EBNF erläutert.

2.1 EBNF

Alle Produktionen haben die Form:

```
Symbol ::= Ausdruck
```

Folgende atomare Ausdrücke sind möglich:

```
"abc"           String
'abc'       String
#xN             Einzelnes Zeichen, N (hexadezimal) bezeichnet
            seine Position im Zeichensatz
[012]           Menge von Zeichen (analog: '0' | '1' | '2')
[^012]          Passt auf alle Zeichen außer den angegebenen
```

Falls A und B Ausdrücke sind, dann auch:

```
A - B           A aber nicht B
A | B           A XOR B
A B             Konkatenation von A und B
```

Folgende Abschlüsse sind definiert:

```
A*              _0 Wiederholungen von A
A+              >0 Wiederholungen von A
A?              A ist optional
```

2.2 Grundlegende Produktionen

Alle Namen von Elementen und Attributen, die in der Grammatik immer mit dem Symbol Name bezeichnet werden, werden nach Regel [5] abgeleitet:

```
[5] Name     ::= (Letter | '_' | ':') (NameChar)*
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' |
                 CombiningChar | Extender
```

Bei einem Verweis auf eine externe Ressource (z.B. um die URL der DTD festzulegen), erscheint das Schlüsselwort SYSTEM gefolgt von der URL, die in Hochkommata eingeschlossen ist. Alternativ kann auch das Schlüsselwort PUBLIC erscheinen, dem eine textuelle Beschreibung (PubidLiteral) und dann die URL (SystemLiteral) folgen.

```
[75] ExternalID ::= 'SYSTEM' S SystemLiteral
                  | 'PUBLIC' S PubidLiteral S SystemLiteral
```

3 Dokument

Nachdem in der Einleitung bereits ein konkretes Beispiel für ein XML-Dokument gegeben wurde, geht es nun darum, den Dokument-Begriff zu formalisieren. Die XML-Dokumentation liefert hierzu Regel [1]:

```
[1] document ::= prolog element Misc*
```

Grundsätzlich besteht ein Dokument also aus den zwei Teilen prolog und element, wie auch schon aus dem Beispiel ersichtlich wird..

Der **Prolog** besteht aus zwei optionalen Teilen: der XML-Deklaration (Festlegung der verwendeten XML-Version, des verwendeten Zeichensatzes und der Standalone-Fähigkeit) und der Dokumenttyp-Deklaration (Festlegung der DTD). Das **Element** beinhaltet die konkrete Instanz der DTD (insofern vorhanden).

Bevor weiter auf die Regeln für den Prolog und das Element eingegangen wird, bleibt jedoch noch zu klären, was das Symbol **Misc** bedeutet, da es in vielen anderen Produktionen vorkommt.

3.1 Misc

Gemäß Regel [27] kann das Symbol Misc entweder zu einem Kommentar, einer Processing Instruction oder White Spaces erweitert werden.

```
[27] Misc ::= Comment | PI | S
```

Ein **Kommentar** wird eingerahmt von den Zeichenketten <!-- und -->. Der Kommentar selbst darf also die Zeichenkette --> nicht beinhalten.

Processing Instructions haben die Form <?Target Anweisung?>. Dieses sind Anweisungen, die der XML-Parser nicht verarbeiten kann, sondern an die mit Target adressierte Anwendung weiterreichen muß. Für das Target ist dabei lediglich zu beachten, daß der Bezeichner nicht XML lauten darf.

Das Symbol **S** steht für sogenannte White Spaces: eine Folge von Leerzeichen, Tabulatoren, LineFeeds und CarriageReturns).

3.2 Der Prolog

Der Prolog eines XML-Dokumentes enthält (abgesehen von Kommentaren, Processing Instructions und White Spaces) die beiden optionalen Bestandteile XML-Deklaration und Dokumenttyp-Deklaration:

```
[22] prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

3.2.1 Die XML-Deklaration

Die **XML-Deklaration** muß - insofern sie in einem XML-Dokument aufgeführt wird - in jedem Fall die verwendete XML-Versionsnummer beinhalten. Optional sind die Angabe des verwendeten Zeichensatzes und der Standalone-Fähigkeit:

```
[23] XMLDecl ::= '<?xml' VersionInfo
                 EncodingDecl? SDDecl? S? '?>'
```

Die **Versionsnummer** (VersionInfo) wird in der Form version="Versionsnummer" notiert (zur Zeit version="1.0">.

Den verwendeten **Zeichensatz** (EncodingDecl) kann man über das Attribut encoding="Zeichensatz" angeben. Möglich sind hier z.B. UTF-8 (Unicode) oder ISO-8859-1 (Latin1).

Die Deklaration der **Standalone-Fähigkeit** (SDDecl) erlaubt zwei mögliche Schreibweisen: standalone="yes" und standalone="no". Mit der Wahl "yes" wird dem XML-Parser bekanntgegeben, daß es keine externe DTD-Untermenge gibt.

3.2.2 Die Dokumenttyp-Deklaration

Über die XML-Deklaration wird dem XML-Parser der Name des Wurzelelements und die verwendete DTD übergeben:

```
[28] doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)? S?
                     ('[' (markupdecl|PEReference|S)* ']' S?)?
                     '>'
```

Eingerahmt durch die Zeichenketten <!DOCTYPE und > enthält dieses Tag den Namen des Wurzelelements, optional die URL einer externen DTD-Untermenge (Symbol ExternalID) und ebenfalls optional eine interne DTD-Untermenge (zweite Zeile der Regel [28]). Die Bedeutung der Symbole markupdecl und PEReference werden erst in Kapitel 4 erläutert.

Es gibt drei Möglichkeiten, dieses Tag zu benutzen:

- Die DTD ist extern:
  <!DOCTYPE WurzelElementName SYSTEM "URL">
- Die DTD ist intern:
  <!DOCTYPE WurzelElementName [DTD]>
- Es gibt sowohl eine interne als auch eine externe DTD-Untermenge:
  <!DOCTYPE WurzelElementName SYSTEM "URL" [DTD]>

Sollte sowohl eine interne als auch eine externe DTD-Untermenge Verwendung finden, überdecken die Deklarationen der internen die der externen DTD-Untermenge.

3.3 Dokument-Inkarnation

Nach der Erklärung des Symbols prolog folgt nun das zweite zentrale Symbol der Produktion [1], das element:

```
[39] element ::= EmptyElemTag | STag content ETag
```

Im einleitenden Beispiel sind die beiden unterschiedlichen Element-Arten bereits zu erkennen. **Leere Elemente** (im Beispiel: verfuegbarkeit) bestehen lediglich aus einem einzelnen Tag, während **Elemente mit Inhalt** (im Beispiel: buchladen) aus einem Start- und End-Tag bestehen, zwischen denen sich der Inhalt des Elementes befindet.

Die Produktionen für diese beiden Möglichkeiten lauten:

```
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>
```

```
[40] STag ::= '<' Name (S Attribute)* S? '>
[42] ETag ::= '</' Name S? '>'
```

Leere Elemente werden eingerahmt durch die Zeichenketten < und /> und beinhalten den Element-Namen und eine Folge von Attributnamen und den zugeordneten Werten. Die Benutzung von Elementen mit Inhalt erfolgt analog zu HTML: eingeschlossen in ein Start- und End-Tag ist der dem Element zugeordnete Inhalt.

Die Belegung von Attributen erfolgt gemäß Regel [41]. Auf den Attributnamen folgen (umschlossen von beliebig vielen White Spaces) ein = sowie der in einfache oder doppelte Hochkommata eingeschlossene Wert des Attributes:

```
[41] Attribute ::= Name Eq AttValue
[25] Eq        ::= S? '=' S?
[10] AttValue  ::= '"' ([^<&"] | Reference)* '"'
                |  "'" ([^<&'] | Reference)* "'"
```

3.3.1 Element-Inhalt

Was noch zu klären bleibt, ist die Frage, wie der Inhalt (Symbol content) aussieht, der zwischen dem Start- und End-Tag eingeschlossen ist. Die Antwort liefert Regel [43]:

```
[43] content ::= (element | CharData | Reference | CDSect
                 | PI | Comment)*
```

Neben weiteren Elementen, Kommentaren und Processing Instructions kann der Inhalt eines Elementes also auch Zeichendaten (CharData), Referenzen auf Entities und einzelne Zeichen (Reference) und geschützte Sektionen (CDSect) beinhalten.

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

Das Symbol CharData umfaßt nahezu beliebigen Text als Inhalt eines Elementes (im Beispiel: der Text zwischen <autor> und </autor>). Regel [14] definiert lediglich zwei Einschränkungen:

- Der Text darf weder ein < noch ein & enthalten. Man muß sich hier mit den Entities < und & behelfen.

- Ebenfalls verboten ist die Teilzeichenkette]]>. Stattdessen greift man auch hier auf Entities zurück:]]>

```
[67] Reference ::= EntityRef | CharRef
[68] EntityRef ::= '&' Name ';'
[66] CharRef   ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'
```

Referenzen auf Entities bestehen aus dem Namen der Entity, der von den Zeichen & und ; umschlossen wird. Im Beispiel-Dokument (Kapitel 2.1) verwenden wir z.B. die Entity quot. Sobald der XML-Parser auf die Referenz " stößt, wird er diese durch doppelte Hochkommata ersetzen.

Auch **Referenzen auf Zeichen** sind möglich. Dazu muß man die Position dieses Zeichens im zugrundegelegten Zeichensatz (siehe XML-Deklaration im Prolog) kennen. Durch Voranstellung von &# (dezimale Darstellung) oder &#x (hexadezimale Darstellung) kann man nun einzelne Zeichen anhand ihrer Position im Zeichensatz angeben (z.B.   oder   für ein Space).

```
[18] CDSect  ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20] CData   ::= (Char* - (Char* ']]>' Char*))
[21] CDEnd   ::= ']]>'
```

Geschützte Sektionen beinhalten Text, der i.d.R. als Markup erkannt würde. Umschlossen werden solche Sektionen von den Zeichenketten <![CDATA[(Terminal CDStart) und]]> (Terminal CDEnd).

Im folgenden Beispiel wird eine solche Sektion benutzt, um das IMG-Tag und die Entity-Referenz davor zu schützen, vom XML-Parser als Markup erkannt zu werden:

```
<text>
Das IMG-Tag benutzen Sie bitte wie folgt:<br/>
<![CDATA[ <IMG SRC="..."> &amp; ]]>
</text>
```

Beispiel 3: Benutzung einer geschützten Sektion

4 DTD

Die DTD ist eine formale Strukturbeschreibung von XML-Dokumenten. Sie enthält u.a eine Auflistung aller möglichen Elemente und Attribute sowie deren Inhaltsmodelle. Die DTD zu unserem Beispieldokument muß z.B. folgende Regeln enthalten:

- Das Element buchladen darf beliebig viele Elemente vom Typ buch enthalten.
- Das Element verfuegbarkeit ist ein leeres Element.
- Das Attribut isbn des Elementes buch muß angegeben werden.

Eine DTD besteht aus einer Folge von Markup-Deklaration, definiert in Regel [29]:

```
[29] markupdecl ::= elementdecl | AttlistDecl | EntityDecl |
                    NotationDecl | PI | Comment
```

Neben Processing Instructions und Kommentaren enthält eine DTD (unabhängig davon, ob sie sich extern in einer separaten Datei oder intern im Prolog des XML-Dokumentes befindet) also Deklarationen von Elementen, Attributen, Entities oder Notationen.

4.1 Deklaration von Notationen

```
[82] NotationDecl ::= '<!NOTATION' S Name S
                      (ExternalID | PublicID) S? '>'
```

Notationen dienen dazu, einen Bezeichner (Symbol Name) mit einer Applikation oder einer Beschreibung zu versehen. Diese Bezeichner können dann bei der Deklaration von Attributen und Entities Verwendung finden.

Die folgenden drei Beispiele verknüpfen den Bezeichner GIF mit einer Applikation und der Kurzbeschreibung (es ist jeweils nur eine der drei Varianten innerhalb einer DTD erlaubt):

```
<!NOTATION GIF SYSTEM "/usr/local/bin/gimp">

<!NOTATION GIF PUBLIC "Graphics Interchange Format">

<!NOTATION GIF PUBLIC "Graphics Interchange Format"
                      "/usr/local/bin/gimp">
```

Beispiel 4: Deklaration von Notationen

4.2 Deklaration von Elementen

Die Deklaration eines Elementes definiert neben dem Elementennamen (Symbol Name) das Inhaltsmodell (Symbol contentspec). Es muß also festgelegt werden, welcher Inhalt sich zwischen dem Start- und End-Tag befinden darf (insofern das Element nicht als leeres Element definiert wird). Die Produktionen [45] und [46] geben Aufschluß über die Deklaration eines Attributes:

```
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
```

Es existieren vier verschiedene Inhaltsmodelle:

- EMPTY: Das Element ist ein leeres Element.
- ANY: Das Element kann beliebigen Inhalt haben.
- Mixed: Neben Text enthält das Element evtl. auch noch andere Elemente.
- children: Das Element enthält keinen Text, sondern nur weitere Elemente.

Die ersten beiden Inhaltsmodelle werden durch die Schlüsselworte EMPTY bzw. ANY deklariert, während Mixed und children noch weiterer Erklärung bedürfen.

4.2.1 Elemente mit gemischtem Inhalt (Mixed)

Regel [51] gibt Aufschluß darüber, wie das Inhaltsmodell für Elemente mit gemischtem Inhalt deklariert werden muß:

```
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
               | '(' S? '#PCDATA' S? ')'
```

Abhängig davon, ob das Element neben Text auch noch weitere Elemente beinhalten soll, greift entweder die erste oder die zweite Zeile der Regel:

- Das Element soll auch noch weitere Elemente beinhalten (erste Zeile).
  Neben Text sind die Elemente $E_1..E_n$ erlaubt:
  (#PCDATA|E1|...|En)*
- Das Element soll lediglich Text beinhalten (zweite Zeile):
  (#PCDATA)

4.2.2. Elemente mit weiteren Elementen (children)

Das Inhaltsmodell children führt zwei Symbole ein: choice und seq. Eine Auswahl (choice) definiert eine Menge von Möglichkeiten, eine Sequenz eine strikte Abfolge:

```
[47] children ::= (choice | seq) ('?' | '*' | '+')?
[48] cp       ::= (Name | choice | seq) ('?' | '*' | '+')?
[49] choice   ::= '(' S? cp ( S? '|' S? cp )* S? ')'
[50] seq      ::= '(' S? cp ( S? ',' S? cp )* S? ')'
```

Diese etwas unübersichtliche Regelmenge läßt sich wie folgt informal erfassen:

- Eine Auswahl (choice) schreibt sich als:
  $(A_1|...|A_n)$
- Eine Sequenz (seq):
  $(A_1,...,A_n)$

wobei jedes A_i entweder ein Elementname, eine Auswahl oder eine Sequenz ist. Jedes A_i kann auch mit einem der Abschlüsse ? (optional), * (_ 0 mal) und + (_ 1 mal) versehen werden.

4.3 Deklaration von Attributen

Auch bei der Deklaration von Attributen wird ein Inhaltsmodell festgelegt (Symbol AttType). Attributdeklarationen beginnen mit der Angabe des Elementnamens, zu dem Attribute definiert werden sollen. Diesem folgen beliebig viele Tripel (Symbol AttDef), die den Attribut-Namen, den Attribut-Typ und die Default-Deklaration beinhalten:

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
[53] AttDef      ::= S Name S AttType S DefaultDecl
```

4.3.1 Attribut-Typ

Welche Werte ein Attribut annehmen kann, wird über die Angabe des Attribut-Typen festgelegt:

```
[54] AttType    ::= StringType | TokenizedType | EnumeratedType
[55] StringType ::= 'CDATA'
```

Ein Attribut, das den Typ StringType (gekennzeichnet durch das Schlüsselwort CDATA) besitzt, kann beliebigen Text als Inhalt annehmen. Die beiden Symbole TokenizedType und EnumeratedType werden im folgenden näher erklärt.

4.3.1.1 Attribut-Typ: TokenizedType

Das Symbol TokenizedType faßt sieben verschiedene Attribut-Typen zusammen:

```
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS'
                     | 'ENTITY' | 'ENTITIES'
                     | 'NMTOKEN' | 'NMTOKENS'
```

Ist ein Attribut vom Typ **ID**, muß der Autor des XML-Dokumentes sicherstellen, daß es keine zwei Attribute vom Typ ID gibt, die den gleichen Wert annehmen. Es muß also zu jedem ID-Wert, der in einem Dokument benutzt wird, genau ein Element geben, das diese ID als Attribut-Wert hat.

Über Attribute vom Typ **IDREF**/**IDREFS** kann ein Link-Mechanismus realisiert werden. Für Attribute von diesem Typ gilt: ihr Wertebereich ist die Menge der benutzten ID-Werte eines XML-Dokumentes. Während IDREF nur eine einzige ID bezeichnet, können Attribute vom Typ IDREFS mehrere durch Leerzeichen getrennte IDs aufnehmen.

```
<!ELEMENT artikel        (#PCDATA)>
<!ELEMENT AngebotDesTages EMPTY>

<!ATTLIST artikel
        artikelNr ID #REQUIRED>
<!ATTLIST AngebotDesTages
        referenz IDREF #REQUIRED>

<artikel artikelNr="1192">Bananen</artikel>
<AngebotDesTages referenz="1192"/>
```

Beispiel 5: Benutzung von ID/IDREF

Attribute vom Typ **ENTITY**/**ENTITIES** können nur Werte annehmen, die mit dem Namen einer externen binären Entity übereinstimmen. ENTITIES steht auch hier für die Mehrzahl (Werte durch Leerzeichen getrennt). Ein Beispiel zu deren Verwendung findet sich im Kapitel 4.4.4 (nicht-analysierte Entities).

Während Attribute vom Typ CDATA praktisch keinen Einschränkungen unterliegen, gelten für Attribute vom Typ **NMTOKEN**/**NMTOKENS** die gleichen Einschränkungen, die auch für alle Bezeichner (Symbol Name) gelten:

```
[7] Nmtoken  ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
[4] NameChar ::=  Letter | Digit | '.' | '-' | '_' | ':' |
                  CombiningChar | Extender
```

4.3.1.2 Attribut-Typ: EnumeratedType

Wie der Symbol-Name EnumaretedTyp bereits suggeriert, handelt es sich bei diesem Attribut-Typ um Aufzählungen. Dieses können entweder Aufzählungen von Bezeichnern oder von vereinbarten Notationen sein:

```
[57] EnumeratedType ::= NotationType | Enumeration
[58] NotationType   ::= 'NOTATION' S
                        '(' S? Name (S? '|' S? Name)* S? ')'
[59] Enumeration    ::= '(' S? Nmtoken (S? '|' S? Nmtoken)*
                        S? ')'
```

Eingeschlossen in Klammern und getrennt durch | werden die Bezeichner aufgelistet, die das Attribut annehmen darf. Im folgenden Beispiel werden beide Möglichkeiten aufgezeigt.

```
<!ATTLIST daten
        sort(aufsteigend|absteigend|unsortiert) #REQUIRED>

<!NOTATION GIF  SYSTEM "/usr/local/bin/gimp">
<!NOTATION JPEG SYSTEM "/usr/local/bin/jpeg_view">
<!NOTATION TGA  SYSTEM "/usr/local/bin/tga_view">
<!ATTLIST img type NOTATION (GIF|JPEG|TGA) #IMPLIED>
```

Beispiel 6: Verwendung von Aufzählungen

4.3.2 Default-Deklaration

Zu jedem Attribut kann über die Default-Deklaration festgelegt werden, ob dieses Attribut gesetzt werden muß, darf oder kann:

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED'
                   | (('#FIXED' S)? AttValue)
```

Es gibt insgesamt vier verschiedene Möglichkeiten einer Default-Deklaration

- #REQUIRED
  Das Attribut muß gesetzt werden.
- #IMPLIED
  Das Attribut darf gesetzt werden
- FIXED "value"
  Das Attribut kann nicht verändert werden und hat stets den Wert value.
- "value"
  Das Attribut kann verändert werden, und hat den Anfangswert value.

4.3.3 Sprachidentifikation

Das Attribut xml:lang ist dafür reserviert worden, um Element-Inhalten eine Sprache zuzuordnen. Es ist allerdings nötig, dieses Attribut explizit innerhalb der DTD zu deklarieren. Der Wert eines solchen Attributes muß der Produktion [33] genügen:

```
[33] LanguageID ::= Langcode ('-' Subcode)*
```

```
[34] Langcode   ::= ISO639Code |  IanaCode |  UserCode
[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36] IanaCode   ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37] UserCode   ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38] Subcode    ::= ([a-z] | [A-Z])+
```

4.4 Entities

Entities sind Speicherungseinheiten, auf die über einen Bezeichner zugegriffen werden kann. Einige davon sind vordefiniert und auch schon aus HTML bekannt:

- < <
- > >
- " "
- & &
- ' '

Sobald der XML-Parser auf eine Entity-Referenz stößt, wird diese Referenz (z.B. ") durch den definierten Ersetzungstext ersetzt.

Entities werden in verschiedene Kategorien eingeteilt:

- **analysiert**/**nicht-analysiert** (parsed/unparsed)
  Bei analysierten Entities besteht der Ersetzungstext aus Text, in dem der XML-Parser nach dem Ersetzen nach Markup sucht. Falls der Ersetzungstext jedoch z.B. binäre Daten enthält, mit denen der Parser sich nicht beschäftigen soll, müssen nicht-analysierte Entities benutzt werden.
- **Allgemeine Entities**/**Parameter-Entities** (general/parameter entities)
  Referenzen auf Allgemeine Entities werden in der Dokument-Instanz erkannt und aufgelöst, während Parameter-Entities innerhalb der DTD Verwendung finden. Um diese beiden Entity-Arten zu unterscheiden, gibt es unterschiedliche Deklarationen und Referenzierungsnotationen (&Entityname; bei allgemeinen Entities, %Entityname; bei Parameter-Entities).
- **intern**/**extern**
  Interne Entities erhalten bei der Deklaration direkt ihren Ersetzungstext zugewiesen. Der Ersetzungstext externer Entities wird indirekt über Angabe einer URL festgelegt: Der Inhalt der über diese URL bezeichneten Datei ist der Ersetzungstext einer externen Datei.

4.4.1 Deklaration von Entities

Wie Regel [70] zeigt, unterliegt die Deklaration von Allgemeinen Entities (Symbol GEDecl) einer anderen Notation als die von Parameter-Entities (Symbol PEDecl):

```
[70] EntityDecl ::= GEDecl | PEDecl
[71] GEDecl     ::= '<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl     ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef  ::= EntityValue | (ExternalID NDataDecl?)
[74] PEDef      ::= EntityValue | ExternalID
[76] NDataDecl  ::= S 'NDATA' S Name
```

Die Deklaration von Allgemeinen Entities und Parameter-Entities unterscheidet vor allem durch das Prozent-Zeichen bei Parameter-Entities (Regel [72]). Es gibt jedoch noch eine weitere Einschränkung:

Durch Angabe des Schlüsselwortes NDATA und dem nachgestellten Namen einer Notation (Regel [76]) kann eine nicht-analysierte definiert werden. Dieses ist jedoch nur bei Allgemeinen Entities möglich.

Die Angabe des Ersetzungstextes kann entweder direkt erfolgen (eingeschlossen in Hochkommata (Symbol EntityValue)) oder indirekt (Symbol ExternalID).

In der Praxis kommen vier verschiedene Entity-Arten vor, die im folgenden anhand einiger Beispiele erläutert werden.

4.4.2 Interne, allgemeine Entities

Interne, allgemeine Entities werden sowohl im Element-Inhalt (Symbol content) als auch im Attribut-Wert (Symbol AttValue) erkannt und durch ihren Ersetzungstext ersetzt.

```
<!ENTITY JavaVersion "1.1">

<text>
<java version="&JavaVersion;">
Sie benutzen Java in der Version &JavaVersion;
</text>
```

Beispiel 7: Interne, allgemeine Entity

4.4.3 Externe, analysierte, allgemeine Entities

Das folgende Beispiel zeigt die Verwendung einer externen, analysierte, allgemeinen Entity. Entities dieser Art werden lediglich im Dokument-Inhalt (Symbol content) erkannt.

```
<!ENTITY Verfasser SYSTEM "verfasser.xml">

<text>
<java version="&JavaVersion;">
&Verfasser;
</text>
```

Beispiel 8: Externe, analysierte, allgemeine Entity

Sobald der Parser auf die Referenz &Verfasser; stößt, wird diese Referenz durch den Inhalt der Datei verfasser.xml ersetzt.

4.4.4 Nicht-analysierte Entities

Wie bereits im Kapitel 4.4.1 beschrieben, müssen nicht-analysierte Entities in ihrer Deklaration das Schlüsselwort NDATA beinhalten. Daran schließt sich der Bezeichner einer Notation an, damit der XML-Parser diese Daten an eine Anwendung weiterreichen kann. Erlaubt sind solche Entities nur als Werte eines Attributes vom Typ ENTITY/ENTITIES.

```
<!ENTITY Bild  SYSTEM "bild.gif" NDATA GIF>
<!NOTATION GIF SYSTEM "/usr/local/bin/gimp">

<!ELEMENT img EMPTY>
```

```
<!ATTLIST img src ENTITY>

<text><img src="Bild"></text>
```

Beispiel 9: Nicht-analysierte Entity

4.4.5 Parameter-Entities

Parameter-Entities werden lediglich innerhalb der DTD und als Bestandteil einer Entity-Deklaration dereferenziert:

```
<!ENTITY % headings "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL|OL">

<!ELEMENT BODY (%headings;|P|%list;)>

<!ENTITY Headings "%headings">
```

Beispiel 10: Parameter-Entity

4.5 Bedingte Abschnitte

Es existieren zwei Arten von bedingten Abschnitten: inkludierte und ignorierte Abschnitte. Innerhalb der DTD können beliebige Anweisungen in die Zeichenketten <![INCLUDE [und]]> (inkludiert) bzw. <![IGNORE [und]]> (ignoriert) eingeschlossen werden. Wie die Namen bereits andeuten, werden die eingeschlossenen Anweisungen entweder beachtet oder ignoriert.

```
[61] conditionalSect ::= includeSect | ignoreSect
[62] includeSect   ::= '<![' S? 'INCLUDE' S?
                       '[' extSubsetDecl ']]>'
[63] ignoreSect    ::= '<![' S? 'IGNORE' S?
                       '[' ignoreSectContents* ']]>'
[31] extSubsetDecl ::= (markupdecl | conditionalSect |
                        PEReference | S)*
```

Das folgende Beispiel zeigt, daß über Parameter-Entities verschiedene Varianten einer DTD gesteuert werden können:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[ <!ELEMENT buch (kommentare*, titel, text)> ]]>
<![%final;[ <!ELEMENT buch (titel, text)> ]]>
```

Beispiel 11: Bedingte Abschnitte

5 Wohlgeformtheit & Gültigkeit

Die Regeln der Grammatik (insgesamt 89) reichen nicht aus, um XML-Dokumente korrekt und umfassend zu beschreiben. Das folgende Beispiel zeigt ein Dokument, das sich an alle Regeln der Grammatik hält, aber dennoch nicht XML-konform ist.

```
<?XML version="1.0"?>
<!DOCTYPE buch SYSTEM "buch.dtd">
<titel>
      <vorname>Richard W.</vorname>
</nachname>
```

Beispiel 12: Fehlerhaftes Dokument

Es stimmen weder das in der Dokumenttyp-Deklaration definierte Wurzelelement mit dem tatsächlichen Wurzelelement überein (buch bzw. titel), noch tragen Start- und End-Tag des Wurzelelements den gleichen Bezeichner (titel bzw. nachname).

Als Konsequenz wurden zusätzlich sprachlich formulierte Beschränkungen (constraints) eingeführt, die zum einen Wohlgeformtheit und zum anderen Gültigkeit sichern sollen.

5.1 Wohlgeformtheit

Ein wohlgeformtes XML-Dokument muß drei Ansprüchen genügen:

- Es muß zu Produktion [1] passen.
- Es muß alle Wohlgeformtheitsbeschränkungen einhalten.
- Jede analysierte Entity muß wohlgeformt sein.

Im folgenden werden lediglich drei die drei wichtigsten Wohlgeformtheitsbeschränkungen aufgeführt und erläutert:

- **Element-Übereinstimmung**
  Der Bezeichner im Start-Tag muß mit dem Bezeichner im End-Tag übereinstimmen.
- **Eindeutige Attribut-Spezifikation**
  Jeder Attributname darf in einem Element höchstens einmal erscheinen.
- **Keine Rekursion**
  Analysierte Entities dürfen weder direkt noch indirekt eine rekursive Referenz auf sich selbst enthalten.

5.2 Gültigkeit

Um ein gültiges Dokument zu erhalten, muß die Dokument-Instanz mit der formalen Strukturbeschreibung, der DTD, abgeglichen werden. Gültige Dokumente müssen also diese zwei Anforderungen erfüllen:

- Eine Dokumenttyp-Definition muß vorhanden sein.
- Alle Gültigkeitsbeschränkungen müssen eingehalten werden.

Von diesen Gültigkeitsbeschränkungen werden auch hier lediglich drei exemplarisch aufgeführt:

- **Wurzel-Elementtyp**
  Der Name des Wurzelelementes aus der Dokumenttyp-Deklaration muß mit dem Elementtyp des Wurzelelementes übereinstimmen.
- **Gültiges Element**
  Das deklarierte Inhaltsmodell (EMPTY, ANY, children, Mixed) stimmt mit dem tatsächlichen Inhalt überein.
- **Typ des Attributwertes**
  Der Attributwert muß von dem Typ sein, der dafür deklariert wurde.

6 Ausblick

Der Ausblick soll unter anderem Technologien aufführen, die nicht Bestandteil von XML selbst sind, damit aber in direktem Zusammenhang stehen.

Ein XML-Dokument definiert zwar logische Strukturen, mit diesen kann ein Browser aber nicht umgehen, da nicht bekannt ist, wie diese Strukturen zu formatieren sind. Zu diesem Zweck gibt es drei verschiedene Stilkomponenten: **CSS** (Cascading Style Sheets), **DSSSL** (Document Style Semantics und Specification Language) und das vom W3C bevorzugte **XSL** (Extensible Style Language).

Darüber hinaus existieren mit den **XLink** und **XPointer** noch nicht standardisierte Ansätze, Links in XML-Dokumente einzubinden. Diese können sich sowohl auf Teilbereiche von XML-Dokumente als auch auf externe Ressourcen beziehen. Im Gegensatz zu HTML sind aber nicht nur unidirektionale Links möglich.

Auf die XSL-Pattern-Syntax stützt sich **XQL** (Extensible Query Language). Mit Hilfe dieser Erweiterung soll es möglich sein, Elemente innerhalb eines Dokumentes auszuwählen.

Abschließend geht es noch um die zentrale Frage, ob XML HTML ersetzen wird. Diese Frage ist nicht eindeutig zu beantworten, da sie von der Anwendung abhängt. Kleinere Projekte, die hauptsächlich grafisch orientiert sind, werden auch zukünftig in HTML leichter und schneller zu realisieren sein. Der Erfolg von XML zur Realisierung von großen Projekten wird vor allem davon abhängen, ob es hinreichende Unterstützung durch Werkzeuge gibt. Insbesondere in Kombination mit XSL und XQL werden dem Autor umfangreiche Kenntnis relativ komplexer Sprachen abverlangt.

7 Literaturangaben

[Bray 98] Extensible Markup Language (XML) 1.0
http://www.w3.org/TR/1998/REC-xml-19980210.html

[Robi 98] XML Query Language (XQL)
http://www.w3.org/Style/XSL/Group/1998/09/XQL-proposal.html

Die Behandlung von Verweisen in XML

Martin Stein

FB Informatik, Uni Dortmund
Ulrike.Bitter@ruhr-uni-bochum.de

Zusammenfassung

In der Spezifikation 1.0 von XML sind Verweise nicht enthalten. Aus diesem Grund wird die XML Pointer Language (XPointer) und die XML Linking Language (XLink) entwickelt. Während XLink beschreibt, wie Verweise in einem XML-Dokument eingesetzt werden, behandelt XPointer wie man interne Strukturen innerhalb eines Dokumentes anspricht. Mittels Namensräumen ist es möglich Elemente aus verschiedenen DTDs in einem Dokument nutzen zu können. Dies hat nicht nur den Vorteil der Wiederverwendbarkeit, sondern bietet sich z.B. auch an, wenn die Benutzung von bestimmten Elementen vorgeschrieben ist. Auch für den Austausch von Daten liegt es nahe eine gemeinsame DTD als Basis zu verwenden. Alle drei behandelten Bereiche befinden sich noch im "Working-Draft Status", d.h. ihre Entwicklung ist noch nicht abgeschlossen (Stand Dezember 1998).

1 Einleitung

1.1 Terminologie

Im folgenden werden einige, in Verbindung mit XLink und XPointer, verwendete Begriffe erläutert:

Elementen-Baum: Eine Repräsentation der Struktur, der Tags und Attribute, in einem XML-Dokument.

Link: Eine Beziehung zwischen zwei oder mehreren Datenobjekten, oder Teilen von Objekten.

Linkelement: Ein Element, das die Existenz eines Links feststellt und ihn beschreibt.

multidirektionaler Link: Ein Link dessen Verwendung durch mehr als eine beteiligte Ressource ausgelöst werden kann.

Inline-Link: Ein Link, der sich selbst als Ressource dient. D.h. der Inhalt des Link-Elementes dient als beteiligte Ressource.

Out-of-line-Link: Ein Link, dessen Inhalt nicht als beteiligte Ressource dient. Solche Links setzen eine Notation, wie erweiterte Link-Gruppen, voraus, die einer Anwendung anzeigen, wo nach dem Link zu suchen ist. Out-of-line-Links werden generell für multidirektionale Links benötigt.

lokale Ressource: Der Inhalt eines Inline-Linkelementes.

entfernte Ressource: Irgendeine, an einem Link, beteiligte Ressource, auf die mit einem Locator gezeigt wird.

Locator: Teil eines Links, der die referenzierte Ressource bezeichnet.

Ressource: Adressierbare Einheit (Dateien, Grafiken,…).

1.2 Verwandtschaft zu bestehenden Standards

Drei Standards dienen XLink und XPointer als Grundlage:

- *HTML*: Definiert verschiedene SGML Elemente die Links repräsentieren.
- *HyTime*: Definiert Inline und Out-of-line Linkstrukturen, sowie einige semantische Eigenschaften.
- *Text Encoding Initiative Guidelines (TEI P3)*: Bietet Strukturen zur Linkerstellung und Ansammlung von Links.

1.3 Notation

Die in diesem Text verwendeten Definitionen benutzen die Erweiterte Backus-Naur Form (EBNF), wie sie in der XML-Spezifikation beschrieben ist.

2 XML Linking Language (XLink)

2.1 Locator Syntax

Ein Locator enthält normalerweise einen Uniform Resource Identifier, kurz URI, wie er in den RFCs 1738 [IETF 91] und 1808 [IETF 95] definiert ist. Das heißt unter anderem, daß die aus HTML bekannten URLs auch unter XML verwendet werden können. Dem URI kann ein „#“ und ein *Fragment Identifier* folgen, dessen Interpretation von dem verwendeten Datentyp der aufgezeigten Ressource abhängt. XPointer werden in Verbindung mit URI-Strukturen als Fragment Identifier benutzt, um präzisere Sub-Ressourcen anzusprechen. Des weiteren ist es, per Definition des URI, möglich Anfragen an das Dokument zu stellen. Um XML-Dokumente, oder Teile davon, zu adressieren, enthält ein Locator einen URI, oder einen Fragment Identifier, oder beide zusammen. Jeder Fragment Identifier muß ein XPointer sein.

Locator:

[1] Locator::= URI
| Connector (XPointer | Name)
| URI Connector (XPointer | Name)

[2] Connector::= ´#´ | ´|´

[3] URI ::= URIchar*

Im Vergleich mit HTML-Links fallen drei Unterschiede auf:

- Auf den Connector kann ein XPointer folgen.
- Das Fragezeichen als Einleitung einer Anfrage wird nur durch einen URI ermöglicht.
- Neben dem Connector ´#´existiert auch eine Pipe (´|´).

Durch die Möglichkeit eines Hash-Zeichen im URI können zwei Hash-Zeichen auftreten, da der Connector eines definiert. Auf dieses Problem geht die XLink-Spezifikation nicht näher ein.

Falls nach dem ´#´ in der Adreßangabe kein expliziter XPointer folgt, sondern ein *Name*, wird davon ausgegangen, daß es sich um einen XPointer handelt und *Name* wird durch *id(Name)* ersetzt.

Tritt das Hash-Zeichen ´#´ auf wird das gesamte Dokument „geholt“. Tritt als Connector das Pipe-Zeichen auf, signalisiert dies, daß nicht feststeht, wie die Ressource weiter behandelt werden soll.

Beinhaltet der URI eine Anfrage, die vom Server ausgewertet werden soll, so ist diese folgendermaßen zu stellen:

Anfrage:

[4] Query::= ´XML-XPTR=´(XPointer | Name)

2.2 Link Erkennung

Linkelemente müssen durch die Anwendung zuverlässig erkannt werden, um das gewünschte Verhalten und die gewünschte Darstellung zu gewährleisten. XML-Linkelemente werden durch das Attribut *xml:link* erkannt, es hat die möglichen Werte *simple*, *extended*, *locator*, *group*, und *document.* Ein Element in dessen Start-Tag solch ein Attribut auftritt wird als ein Element des aufgezeigten XLink-Typs behandelt wie es die Spezifikation vorschreibt. Hier ein Beispiel:

```
<A xml:link="simple" href="Fehler! Hyperlink-Referenz ungültig."> The W3C</A>
```

XML kann ohne Angabe des Attributwertes nicht erkennen um welche Art von Link es sich handelt. Daher muß entweder der XML-Autor das Attribut angeben, oder der Wert des Attributes wird in der DTD festgelegt.

Soll das Attribut einen festen Wert erhalten, muß die Definition folgendermaßen aussehen:

```
<!ATTLIST A   xml:link        CDATA        #FIXED        "simple" >
```

Soll der XML-Autor angeben wie der Wert des Attributes xml:link im Dokument sein soll ist in der DTD die Attributbeschreibung so aussehen:

```
<!ATTLIST A   xml:link        CDATA        #REQUIRED >
```

Beide Methoden haben Vor- und Nachteile. Wird der Attributwert in der DTD festgelegt, weil nur einfache Links verwendet werden, können nur validierende XML-Prozessoren (die auch die DTD lesen) einen Link erkennen. Wird der Attributwert jedesmal im Start-Tag angegeben, erhöht dies nicht nur die Fehleranfälligkeit, sondern es wird auch die Netzwerk-Bandbreite durch eine große Anzahl von Links belastet. Hier bietet sich als Ausweg die Möglichkeit von XML default Wert für Attribute festzulegen.

2.3 Link Elemente

XLink definiert zwei Arten von Linkelementen:

- Einfache Links, die normalerweise inline und immer uni-direktional sind.
- Erweiterte Links, die entweder inline oder out-of-line sind und für multidirektionale Links, Links die von read-only-Ressourcen stammen, usw. benutzt werden müssen.

Beide Arten von Links können (oder müssen) mit verschiedenen Informationen verbunden sein. In den folgenden Abschnitten werden diese Informationen durch die Prozessor Instruktion ENTIY zu Gruppen zusammengefaßt:

2.3.1 Locator

Ein Locator-String adressiert die zugehörige Ressource. Für jede entfernte Ressource muß ein Link einen Locator bereitstellen. Hierzu wird das Attribut *href* verwendet.

Beispiel Deklaration:

```
<! ENTITY % Locator.att
 ″href    CDATA          #REQUIRED″
>
```

2.3.2 Bedeutung der Attribute inline und role

Jeder Link ist entweder inline oder out-of-line. Dies wird durch das Attribut *inline,* mit den möglichen Werten *true* oder *false,* angezeigt. Ist der Wert *true* so ist der betreffende Link vom Typ inline, ansonsten out-of-line. Handelt es sich um einen inline Link, zählt sein Inhalt als lokale Ressource.

Das Attribut *role* informiert die Anwendung über die Bedeutung des Links. Jeder am Link beteiligten Ressource kann ein eigenes role-Attribut zugewiesen werden.

Beispiel Deklaration:

```
<!ENTITY % link-semantics.att
 ″inline          (true | false)      ´true´
  role            CDATA          #IMPLIED″
>
```

Da einfache Links bereits ein Attribut *role* mit einer unterschiedlichen Bedeutung haben, muß für diese eine andere Deklaration gewählt werden und das role entfallen:

Beispiel Deklaration:

```
<!ENTITY % simple-link-semantics.att
 ″inline          (true | false)      ´true´″
>
```

2.3.3 Entferne Ressourcen

Folgende Informationen können entfernten Ressourcen zugefügt werden:

- Die Rolle der Ressource, um der Anwendung mitzuteilen welche Bedeutung sie für den Link hat. (Dem Link als ganzem kann ein eigenes *role* Attribut, wie in 2.3.2 beschrieben, gegeben werden). Der Autor kann hierfür das optionale Attribut *role* verwenden.
- Ein Titel für die Ressource, der zur Anzeige dient, um dem Benutzer mitzuteilen welche Rolle die Ressource bei dem Link spielt. Der Autor kann hierfür das optionale Attribut *title* verwenden.
- Verhaltensweise bei der Benutzung des Links. Hierzu dienen dem Autor die optionalen Attribute *actuate* und *show*. Das *show* Attribut kann die Werte *new, replace* oder *embed* annehmen. Der Wert entscheidet darüber, ob die Ressource an dieser Stelle eingeblendet wird (*embed*), den gegenwärtigen Inhalt ersetzt (*replace*), oder in einem neuen Fenster erscheint (*new*). Das *ac-*

tuate Attribut hat entweder den Wert *auto* oder *user*. Es legt fest, wann zu einer Ressource verzweigt wird. *behavior* ist inhaltlich nicht festgelegt und kann Hinweise auf die Art der Verzweigung beinhalten.

Beispiel Deklaration:

```
<!ENTITY % remote-resource-semantics.att
″role        CDATA                    #IMPLIED
title        CADTA                    #IMPLIED
show         (embed | replace | new)  #IMPLIED
actuate      (auto | user)            #IMPLIED
behavior     CDATA                    #IMPLIED″
>
```

2.3.4 lokale Ressourcen

Die folgenden Informationen können einer lokalen Ressource mitgegeben werden, wenn es sich um einen inline Link handelt:

- Die Rolle der Ressource, um der Anwendung mitzuteilen welche Bedeutung sie für den Link hat. (Dem Link als ganzem kann ein eigenes *role* Attribut, wie in 2.3.2 beschrieben, gegeben werden). Der Autor kann hierfür das optionale Attribut *content-role* verwenden.
- Ein Titel für die Ressource, der zur Anzeige dient, um dem Benutzer mitzuteilen welche Rolle die Ressource bei dem Link spielt. Der Autor kann hierfür das optionale Attribut *content-title* verwenden.

Beispiel Deklaration:

```
<!ENTITY % local-resource-semantics.att
″content-role    CDATA    #IMPLIED
 content-title   CDATA    #IMPLIED″
>
```

2.3.5 Einfache Links

Um lediglich die Eigenschaften des von HTML bekannten Linkens zu nutzen, dienen die einfachen Links. Sie haben jedoch einen etwas größeren Funktionsumfang. Einfache Links besitzen nur einen Locator und seine Funktionen können mit dem Locator in einem einzigen Element zusammengefaßt werden.

Es folgt eine Beispieldeklaration mit den aus 2.3, Link Elemente, bekannten Entities. Es beinhaltet alle möglichen Attribute. Das Attribut *xml:link* muß vom Typ *simple* sein:

```
<!ELEMENT simple ANY>
<!ATTLIST simple
        xml:link        CDATA               #FIXED ″simple″
        %locator.att;
        %remote-resource-semantics.att;
        %local-resource-semantics.att;
        %simple-link-semantics.att;
>
```

Hier nun ein konkretes Beispiel, wie es in einem echten Dokument auftreten könnte, gefolgt von einer Beispiel-DTD:

```
<mylink xml:link=″simple″ title=″Citation″
        href=″http://www.w3c.org/xml/xlink.xml″ show=″new″
        content-role=″Reference″>as discussed in Smith(1997) </mylink>

<!ELEMENT mylink (#PCDATA)>
<!ATTLIST mylink
        href            CDATA       #REQUIRED
        content-role    CDATA       #IMPLIED
```

2.3.6 Erweiterte Links

Im Gegensatz zu den einfachen Links, erlauben es die erweiterten Links mehr als eine Ressource zu enthalten. Außerdem ist es möglich Verweise von Read-only-Medien oder Dateien zu anderen Stellen einzurichten, oder Links von und zu Dateien einzurichten, die überhaupt kein Linking unterstützen.

Ein Linkelement für erweiterte Links beinhaltet eine Liste von Kindelementen, die als Locator dienen, von denen jeder seine eigenen Attribute hat.

Auch hier eine Beispieldeklaration:

```
<!ELEMENT extended ANY>
>ATTLIST extended
        xml:link        CDATA       #FIXED      ″extended″
        %link-semantics.att;
        %local-resource-semantics.att;
>
<!ELEMENT locator ANY>
<!ATTLIST locator
```

```
        xml:link        CDATA       #FIXED      ″locator″
        %locator.att;
        %remote-resource-semantics.att;
>
```

Beispiel im XML-Dokument:

```
<commentary xml:link=″extended″      inline=″false″>
        <locator        href=″smith.1″  role=″Essay″/>
        <locator        href=″jones″    role=″Rebuttal″/>
        <locator        href=″robin″    role=″Comparison″7>
</commentary>
```

3 XML Pointer Language (XPointer)

XPointer arbeiten mit dem Baum, der durch die Elemente und andere Markup-Konstrukte eines XML-Dokumentes aufgespannt wird.

Ein XPointer kann aus verschiedenen Verweisausdrücken bestehen, wie es die Definition zeigt:

XPointer:

```
[1] XPointer::= AbsTerm ´.´OtherTerms
                | AbsTerm
                | OtherTerms
[2] OtherTerms::= OtherTerm
                | OtherTerm ´.´ OtherTerm
[3] OtherTerm::= RelTerm
                | SpanTerm
                | AttrTerm
                | StringTerm
```

3.1 Absolute Verweisausdrücke

Absolute Verweisausdrücke:

```
[4] AbsTerm::= ´root()´ | ´Origin()´ |  IdLoc | HTMLAddr
[5] IdLoc::=    ´id(´ Name ´)´
[6] HTMLAddr::=´html(´ SkipLit ´)´
```

Mit den vier Schlüsselwörtern *root*, *origin*, *id* und *html* wird die absolute Adreßangabe bestimmt.

Enthält ein Verweisausdruck keines der Schlüsselwörter, gehen die Anwendungen davon aus, daß sie mir *root* anfangen. Beginnt ein solcher Ausdruck mit *root*, startet die Suche beim Wurzel-Element des Dokumentes, das vor dem Fragmentbezeichner genannt wird. Das Schlüsselwort *origin* bezieht sich auf die Teilressource, von der aus die Verzweigung begonnen hat. Es ist ein Fehler, *origin* in einem Locator zu benutzen der eine andere Ressource bezeichnet, als die von der die Verzweigung gestartet wurde. Wird *id(Name*) in einem XPointer verwendet, so ist die Verweisquelle das Element mit dem Attribut *id* und dem Wert *Name*. Schließlich bedeutet *html(…)* die in HTML übliche Adreßangabe mit Hilfe von #. *html(AttrWert)* sucht nach dem ersten Element A, das den Ausdruck *name=″AttrWert″* enthält.

3.2 Relative Verweisausdrücke

Auch die hier behandelten Verweisausdrücke benötigen eine Quelle. Fehlt sie, wird das Wurzel-Element der referenzierten Ressource angenommen. Relative Verweisausdrücke dienen zum Navigieren im Elementenbaum.

Relative Verweisausdrücke:

[7] RelTerm::= Keyword? Arguments

[8] Keyword::= ´child´	/* Kindelement */
| ´descendant´	/* Nachfahre innerhalb der Verweisquelle */
| ´ancestor´	/* Vorfahre, der die Verweisquelle enthält */
| ´preceding´	/* Elemente, vor der Verweisquelle */
| ´following´	/* Elemente, die der Verweisquelle folgen */
| ´psibling´	/* Ältere Geschwister */
| ´fsibling´	/* Jüngere Geschwister */

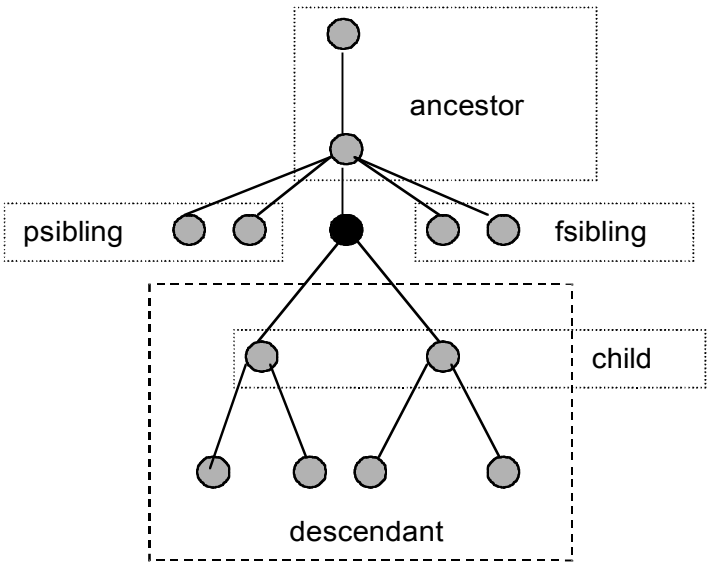


Abbildung 3-1: jedes Schlüsselwort deckt jeweils einen anderen Bereich des Dokumentes ab.

Werden mehrere gleiche Schlüsselwörter hintereinander benutzt, so muß es nur am Anfang des Verweisausdruckes stehen, die folgenden können weggelassen werden. Die beiden unten stehenden XPointer sind also gleich:

```
child(2,SECTION).child(1,SUBSECTION)
child((2,SECTION).(1,SUBSECTION)
```

Alle relativen Verweisausdrücke arbeiten mit der selben Menge an Argumenten:

Argumente:

```
[9] Arguments::= ´(´ InstanceOrAll
        (´,´ NodeType (´,´ Attr ´,´ Val)*)? ´)´
[10] InstaneOrAll::= ´all´| Instance
[11] Instance::= (´+´ |´-´)? [1-9] Digit*
```

Ist die Instanznummer *n* positiv, wird das *n-te* Element ausgewählt, ist sie negativ, wird vom Schluß gezählt. Ist der Wert *all*, werden alle möglichen Elemente ausgewählt.

Ebenso wie mit Zahlen kann man sub-Ressourcen auch mit Knotentypen ansprechen. NodeType kann dabei folgende Werte annehmen:

Knotentypen:

```
[12] NodeType::= Name
                | ´#element´
                | ´#pi´
                | ´#comment´
                | ´#text´
                | ´#cdata´
                | ´#all´
```

Name wählt einen Elementtyp aus, *#element* steht für ein beliebiges Element. Fehlt der Knotentyp, wird *#element* verwendet. *#pi* und *#comment* sprechen Processing Instructions bzw. Kommentare an. *#text* wählt Zeichendaten aus und *#cdata* Zeichendaten in cdata-Regionen. Der Knotentyp *#all* bestimmt alle Knotentypen zusammen.

Wie die Definition der relativen Verweisausdrücke zeigt, können Elemente auch anhand ihres Argumentes oder ihres Wertes ausgewählt werden.

Attributverweisausdrücke:

```
[13] Attr::= ´*´            /* irgend ein Attributname */
           | Name
[14] Val::= ´#IMPLIED´  | ´*´ | Name | SkipLit
```

3.3 Verweisausdrücke für Bereiche

Das Schlüsselwort *span* benötigt zwei XPointer als Parameter. Es verweist nicht auf ein Element, sondern auf einen Bereich, der mehrere Elemente umfassen kann.

Verweisausdruck für Bereiche:

[15] SpanTerm::= ´span(´ XPointer ´,´ XPointer ´)´

4 Namespaces

Die Einbeziehung von Namespaces in XML-Dokumente bedeutet im wesentlichen, daß Elemente aus DTDs verwendet werden können, die aus einer externen Quelle stammen. Ziel ist es häufig verwendete Komponenten nicht immer neu definieren zu müssen, sondern standardisierte Elemente verwenden zu können. Hier ist nicht nur die Wiederverwendbarkeit von Vorteil, es ist auch möglich, daß der Gebrauch von manchen Elementen zwingend vorgeschrieben ist. Außerdem könnten Suchmaschinen das WWW effektiver durchsuchen, wenn in Dokumenten dieselben Elemente die selbe Bedeutung hätten.

4.1 Deklaration von Namespaces

Namespaces bestehen aus einem qualifizierten Namen, der ein Doppelpunkt enthält, der als Trennzeichen dient, und einem Namespaces Prefix und einem Lokalem Teil.

Namespace Deklaration:

[1] NSAttName::=	PrefixedAttName
	| DefaultAttName
[2] PrefixedAttName::=	´xmlns:´ NCName
[3] DefaultAttName::=	´xmlns´
[4] NCName ::=	(Letter |´_´) (NCNameChar)*
[5] NCNameChar::=	Letter | Digit | ´.´ |´-´ | ´_´
	| CombiningChar | Extender

Der Wert eines Attributes ist eine URI-Referenz, der Namespace Name.

Hier ein Beispiel, daß den Namespaceprefix edi mit dem Namespace Namen **Fehler! Hyperlink-Referenz ungültig.**a verbindet.

```
<?xml version="1.0"?>
<x xmlns:edi=´Fehler! Hyperlink-Referenz ungültig.a´>
</x>
```

4.2 Qualifizierte Namen

In einem XML-Dokument können qualifizierte Namen auftreten, die wie folgt definiert sind:

Qualifizierte Namen:

[6] QName::=(Prefix ´:´)? LocalPart

[7] Prefix::= NCName

[8] LocalPart::=NCName

Das Prefix dient lediglich als Platzhalter für einen Namespaces Namen. Anwendungen sollten den Namespaces Namen und nicht das Prefix verwenden.

4.3 Die Verwendung von qualifizierten Namen

In Dokumenten, die zu der Spezifikation konform sind, werden Elementtypen als qualifizierte Namen verwendet:

[9] STag::= ´<´ QName (S Attribute)* S? ´>´

[10] ETag::= ´</´ QName S? ´>´

[11] EmptyElemTag::= ´<´ (S Attribute)* S? ´/>´

[12] Attribute::= NSAttName Eq AttValue
| QName Eq AttValue

Abschließend ein Beispiel in dem zwei Prefixe als Attribute eines Einzelnen Elementes deklariert werden:

```
<?xml version="1.0" ?>
<bk:book xmlns:bk=´urn:loc.gov:books´
                xmlns:isbn=´urn:ISBN:0-395-36341-6´>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

5 Literatur

[BHLa 98] Bray, Tim – Hollander, Dave – Layman, Andrew: Namespaces in XML
http://www.w3c.org/TR/1998/PR-xml-names-19981117 , 1998

[EMSD 98a] Maler, Eve – DeRose, Steve : XML Linking Language (XLink)
http://www.w3c.org/TR/1998/WD-xptr-19980303 , 1998

[EMSD 98b] Maler, Eve – DeRose, Steve : XML Pointer Language (XPointer)
http://www.w3c.org/TR/1998/WD-xptr-19980303 , 1998

[IETF 91] IETF RFC 1738: Uniform Resource Locators , 1991

[IETF 95] IETF RFC 1808: Relative Uniform Resource Locators , 1995

[XMLC 98] unbekannt : Extensible Marup Language (XML) Part 2 Linking
http://xml.com/xml/pub/w3j/s2.linkspec.html , 1998

MathML zur Beschreibung von mathematischen Formeln

Achim Streit

FB Informatik, Uni Dortmund
streit00@marvin.informatik.uni-dortmund.de

Zusammenfassung

Grundsätzlich sind bei der Darstellung mathematischer Ausdrücke im Web zwei voneinander unabhängige Probleme vorhanden. Zum einen muß der mathematische Ausdruck dargestellt werde, und zwar so, daß eine möglichst hohe Darstellungsqualität gegeben ist, daß er sich optimal in seine Umgebung einpaßt, und daß er durchsuch- und indizierbar ist. Bei der Darstellung sind nicht nur einfache Ausgabegeräte, wie Drucker und Monitore, zu beachten, sondern auch komplexere, wie Sprachsynthetisierer oder Geräte zur Erzeugung von Brailleschrift.
Zum anderen muß auch der Inhalt eines mathematischen Ausdrucks beschreibbar sein, um diesen in Computer Algebra Systemen weiterzuverwenden. Diese Inhaltsbeschreibung wird in diesem Text jedoch in groben Zügen vorgestellt.
Nachdem zu Beginn kurz auf die Geschichte und die Entwicklungsziele von MathML eingegangen wird, folgen in Kapitel 3 einige Grundlagen zu MathML und einige Festlegungen zur Notation der Syntax zum besseren Verständnis dieses Textes.
In Kapitel 4 werden die Elemente zur Darstellung der Notation eines mathematischen Ausdrucks aufgeführt. Zu den einzelnen Elementen wird eine Kurzbeschreibung und eine Übersicht der verwendbaren Attribute geliefert. Teilweise werden noch einige Verwendungsbeispiele dargestellt. Unterteilt werden die Darstellungselemente nach Zeichen- und Layoutelementen, nach Elementen zur Erzeugung von Indizes und Akzenten und nach Elementen mit denen sich Tabellen und Matrizen erstellen lassen.

1 Einführung

Zu Beginn wird eine kurze Einführung zur Sensibilisierung des Themas gegeben. Nach einer Problembeschreibung folgt ein Rückblick in die Vergangenheit, um die ersten Ansätze von MathML zu erläutern. Alsdann werden die Entwicklungsziele der Sprache MathML aufgeführt. Abschließend wird darauf eingegangen, welche Rolle die Sprache MathML im Web heutzutage spielt.

1.1 Die Mathematik und ihre Notation

Unabhängig vom eigentlichen mathematischen Problem existiert für den Autor mathematischer Ausdrücke häufig auch das Problem der Notation. So ist es zur Zeit üblich, daß Variablen und Buchstaben, die Zahlen repräsentieren (z.B. die eulersche Zahl), in Italic-Fonts dargestellt werden. Andere Festlegungen definieren den Leerraum um Symbole für mathematische Operationen, wie +, -, * und /, wel-

cher etwas abweicht vom Leerraum eines Leerzeichens zwischen zwei Wörtern in einem Fließtext. Obwohl wir beim ständigen Betrachten mathematischer Ausdrücke in Büchern und anderen gedruckten Dokumenten eine Vielzahl an Festlegungen als selbstverständlich hinnehmen, hat die genaue Definition solcher Notationsfestlegungen gerade für elektronische Dokumente eine ungeheuer große Bedeutung. Es ist dabei nicht nur die einfache Darstellung mathematischer Ausdrücke von Bedeutung, sondern es müssen auch die zusätzlichen Gegebenheiten elektronischer Dokumente beachtet werden. So muß ein mathematischer Ausdruck durchsuchbar und indizierbar dargestellt werden.

1.2 Herkunft und Ziele

1.2.1 Die Geschichte von `MathML`

Das eigentliche Problem der Darstellung mathematischer Ausdrücke in computerlesbarer Form ist schon viel älter als das Web selbst. So waren zahlreiche Darstellungssprachen, wie etwa `TeX` bereits im Jahre 1992 schon in Benutzung, als sich das Web gerade erst durchsetzte. Das Web erwies sich schnell als eine effektive Methode Informationen einer breiten Schicht von Personen zugänglich zu machen. Obwohl es ursprünglich entwickelt wurde, eine Kommunikationsmöglichkeit von Wissenschaftlern für Wissenschaftler zu schaffen, wurde die Fähigkeit zur Beschreibung mathematischer Ausdrücke in HTML nur sehr stiefmütterlich behandelt. So mußte man sich damit behelfen, wissenschaftliche Ausdrücke als Grafiken darzustellen. Das World Wide Web Consortium (W3C) hatte dieses Problem schon längerfristig beobachtet und begann 1994 mit der Ankündigung von HTML-Math, welches bei der darauffolgenden WWW IV Conference eingehend diskutiert wurde. In den darauffolgenden zwei Jahren wurde die Gruppe, die sich mit diesem Thema beschäftigte, ständig erweitert, bis sie sich schließlich als *W3C HTML-Math working group* neu konstituierte. `MathML` repräsentiert die ganz besonderen Interessen einer Teilgruppe dieser W3C HTML-Math working group, obwohl sich deutlich der Einfluß anderer Teilgruppen erkennen läßt. Insbesondere wurde `MathML` durch das OpenMath project (ISO 12083) und die Arbeiten von Stilo Technologies bezüglich eines semantischen Math-DTD Fragmentes stark beeinflußt.

1.2.2 Voraussetzungen mathematischer Beschreibungssprachen

Die grafikbasierte mathematische Beschreibung sollte aufgrund zahlreicher Probleme, wie etwa unterschiedliche Auflösungen zwischen Grafik und umgebenden Text, die sich besonders beim Drucken bemerkbar machen, nicht länger verfolgt werden. Ebenso sind solche grafikbasierten Methoden nicht durchsuchbar und es gibt sehr große Probleme bei der Darstellung der Grafiken im Fließtext.

Während des Entwicklungsprozesses einer Sprache müssen die Wünsche der potentiellen Benutzer sehr sorgsam betrachtet werden. Das Spektrum reicht dabei bei einer Sprache zur Beschreibung mathematischer Ausdrücke von der Ausbildung bis zur Wissenschaft. Gerade im Bildungsbereich ist es wichtig, daß sich mathematische Ausdrücke ohne viel zeitlichen und materiellen Aufwand implementieren lassen, obwohl auch sehr komplexe mathematische Ausdrücke darstellbar sein müssen. Desweiteren ist es von großer Bedeutung, daß es möglich ist, den mathematischen Inhalt schnell und einfach zu implementieren, die Benutzung intuitiv und einfach zu erlernen ist und daß preisgünstige Werkzeuge verfügbar sind.

Auf der anderen Seite des Spektrums steht die akademische Forschung, welche ganz andere Anforderungen an eine Sprache zur Beschreibung mathematischer Ausrücke stellt. So ist es üblich, daß wissenschaftliches Material in großen Datenbanken gespeichert wird, um einen besseren Zugriff zu gewährleisten. Daraus ergeben sich spezielle Eigenschaften, wie etwa die Unterstützung großer Dokumente mit automatisierter Suche und Indizierung. Da bereits sehr viele Dokumente in anderen Beschreibungssprachen existieren, vor allem in `Tex`, ist es ebenfalls sehr wichtig, daß entsprechende

Konvertierungsmöglichkeiten existieren. Aus dem Bereich der industriellen Forschung stammt die Forderung, daß es möglich sein muß, Daten aus Experimenten und Simulationen zu dokumentieren.

Eine weitere Forderung besteht darin, daß eine Ausgabe der mathematischen Ausdrücke in andere Medien, wie etwa gesprochenes Wort oder auch in Brailleschrift, möglich sein muß. Verlage hingegen verlangen, daß es möglich sein muß, direkt einen hochqualitativen Druck zu ermöglichen und daß eine Kompatibilität zu ihren bereits vorhandenen Systemen, hauptsächlich SGML, existiert.

1.2.3 Entwicklungsziele von `MathML`

Um die genannten Bedürfnisse unterschiedlicher Gruppen befriedigen zu können, wurde `MathML` mit den folgenden Zielen entwickelt:

- Darstellung mathematischer Ausdrücke benutzbar für alle Bereiche des Lernens und der Wissenschaft
- Darstellung der mathematischen Notation und auch der mathematischen Bedeutung
- Konvertierung in und von anderen Formaten unter Beibehaltung der Darstellung und der Bedeutung. Ausgabeformate sollten folgendes beinhalten:
  - Grafische Anzeige
  - Sprachsynthetisierer
  - Eingabe für Computer Algebra Systeme
  - Andere mathematische Beschreibungssprachen, wie etwa `TeX`
  - plain-text Anzeigegeräte, wie etwa VT100
  - Druckmedien, einschließlich Braille
- Übergabe spezieller Informationen für Darstellungsprogramme
- Unterstützung des effizienten Durchsuchen langer Ausdrücke
- Erweiterbarkeit
- vom Menschen einfach zu benutzen, ebenso einfache Verarbeitung durch Software

Zusätzliche Forderungen der W3C HTML-Math working group:

- zufriedenstellende Darstellung von `MathML`-Ausdrücken in HTML Webseiten unter Berücksichtigung der Darstellungswünsche des Lesers und des Autors und in der besten Qualität, die das System ermöglicht
- HTML-Dokumente mit `MathML`-Ausdrücken sollten ein gutes Druckbild ergeben und bei möglichst hohen Druckerauflösungen druckbar sein
- `MathML`-Ausdrücke in Webseiten sollten auf Mausaktivitäten reagieren können
- Editierwerkzeuge sollten rasch verfügbar sein, um die Generierung von HTML-Dokumenten mit `MathML`-Ausdrücken zu vereinfachen

1.3 Die Rolle von `MathML` im Web

Da `MathML`-Ausdrücken in Webseiten integriert werden, müssen folgende drei Eigenschaften erfüllt sein.

i.) Es muß möglich sein, bereits existierende mathematische Beschreibungssprachen in `MathML` zu konvertieren und bereits vorhandene Autorenwerkzeuge zu erweitern, so daß sie `MathML` erzeugen können.

ii.) Zusätzlich muß gewährleistet sein, das sich MathML-Konstrukte ohne große Probleme in HTML-Dokumente einfügen lassen, und zwar in der Form, daß zukünftige Browser, Suchmaschinen, und andere Web-Applikationen keine Probleme damit bekommen.

iii.) Schließlich muß es ermöglicht werden, daß MathML-Ausdrücke in aktuellen Browsern sichtbar sind, auch wenn diese Darstellung noch nicht ideal ist.

Die große Akzeptanz von HTML hat dazu geführt, daß immer neue Anforderungen an die Sprache HTML und die Browser-Hersteller gestellt worden sind. Deswegen mußte ein universeller Erweiterungsmechanismus geschaffen werden. Man beschäftigte sich zunächst mit einfachen plain-text Erweiterungen, die sich aber schnell als ungeeignet herausstellten. XML stellte sich schnell als die beste Methode für einen universellen Erweiterungsmechanismus heraus. XML steht dabei für *Extensible Markup Language* und stellt eine vereinfachte Version von SGML dar, die zur Definition der Grammatik und Syntax von HTML benutzt wird. Wie der Name schon erklärt, ist die Erweiterbarkeit eine Schlüsselfunktion der Sprache. Autoren ist es somit ermöglicht worden, Erweiterungen in einfacher Form hinzuzufügen, während XML dafür sorgt, daß weiterhin eine automatische Generierung und Wartbarkeit möglich ist.

2 MathML Grundlagen

Nachdem im vorherigen Kapitel eine Motivation zur Entwicklung der Sprache MathML gegeben wurde, werden im folgenden Kapitel die Grundlagen von MathML vorgestellt. Nachdem Ausdrucksbäume definiert worden sind, erfolgt abschließend eine Unterscheidung in Presentation Markup und Content Markup.

Es werden dabei die englischen Begriffe benutzt und die entsprechende deutsche Übersetzung wird nur kurz bei der Definition des Begriffes nachgestellt. Um die Fachtermini auch in anderen Texten schneller wiederzuerkennen, wird den englischen Begriffen der Vorzug gegeben.

2.1 MathML Überblick

Grundsätzlich ist zwischen zwei Arten der mathematischen Beschreibung zu unterscheiden. Zum einen gibt es die Beschreibung der mathematischen Notation und zum anderen die Beschreibung der mathematischen Bedeutung eines Ausdrucks. Dabei ist für den Betrachter die Notation eines Ausdrucks viel wichtiger als die Bedeutung einer Formel. So kann eine klare Strukturierung in der Notation auf die logische Struktur hinweisen. Dagegen ist die mathematische Bedeutung für Computer Algebra Systeme von Bedeutung, die den beschriebenen Ausdruck für Berechnungen verwenden, wobei dann die Notation wiederum von geringerer Bedeutung ist. MathML erlaubt die Gegenwart beider Beschreibungsalternativen und auch ihre Mischung.

2.1.1 Einordnung von MathML-Elementen

MathML Elemente lassen sich grob in drei Kategorien einteilen. Es gibt presentation elements (Darstellungelemente) , content elements (Aussageelemente) und interface elements (Schnittstellenelemente). Presentation elements beschreiben die Struktur der Darstellung eines mathematischen Ausdrucks. Zwei wichtige Beispiele sind <mrow>, welches zur Darstellung einer horizontalen Hintereinanderreihung von Zeichen benutzt wird, und <msup>, welches bei der Darstellung einer Basis und eines hochgestellten Ausdrucks zum Einsatz kommt. Grundsätzlich gehört jedes presentation element zu einem speziellen Bereich der Notation des mathematischen Ausdrucks, wie etwa einer Zeile (row), eines hoch- oder tiefgestellten Ausdrucks (superscript oder underscript). Jedes presentation element besitzt eine große Auswahl an Attributen, um die verschiedenen auftretenden Varianten zu verwirkli-

chen. Beispielsweise gibt das Attribut *superscriptshift* an, um wieviel der hochgestellte Ausdruck minimal nach oben verschoben wird.

Content elements beschäftigen sich direkt mit der mathematischen Bedeutung des Ausdrucks. Typische Elemente sind etwas <plus/>, für die normale Addition, oder <vector>, für Vektoren in der linearen Algebra. Jedes content element ist dabei streng nach einem mathematischen Konzept definiert.

Alle MathML Elemente sind entweder presentation elements oder content elements, bis auf eines, das <math> Element. Dieses interface element ist auf der obersten Ebene angesiedelt und stellt die Kommunikation und Parameterübergabe (Stileigenschaften) an MathML-Programme sicher. Eine zweite Funktion des interface element beschäftigt sich mit der Kommunikation mit dem Web-Browser. Hierbei werden Informationen übermittelt, die klarstellen, wie ein MathML Ausdruck dargestellt wird und wie er in der ihn umgebenden HTML-Seite eingebettet wird. Bei der zukünftig zu erwartenden Unterstützung von XML in Web-Browsern werden eventuell noch einige Parameter hinzukommen.

2.1.2 Ausdrucksbäume und Zeichenelemente

Häufig zerfallen mathematische Ausdrücke in Teile oder Unterausdrücke. So zerfällt $(a+b)^2$ in die Basis und den Exponenten. Unterausdrücke können dabei wiederum in weitere Unterausdrücke zerteilt werden. Diese häufig streng hierarchische Struktur mathematischer Ausdrücke spiegelt sich auch in der MathML-Beschreibung wieder. So gibt es parent schema (Eltern-Schemata), die child schema (Kinder-Schemata) enthalten können. MathML-Ausdrücke lassen sich deshalb einfacher in Bäumen ausdrücken, wobei jeder Knoten ein MathML-Element repräsentiert und die Blätter des Baumes Zahlen oder Zeichen darstellen. Die meisten Blätter in MathML-Ausdrucksbäumen sind entweder leere Elemente oder Zeichenelemente. Leere Elemente repräsentieren direkt ein Aussageelement, wie beispielsweise <plus/>. Zeichenelemente sind dagegen die einzigen Elemente , die direkt Zeichen, also Buchstaben bzw. Zahlen enthalten können. Diese Zeichen können einerseits normale ASCII-Zeichen sein, aber andererseits auch MathML-Einheiten mit denen sich bestimmte mathematisch notwendige Zeichen darstellen lassen, wie etwa das Summenzeichen oder einen Funktionspfeil. Ein weiteres mögliches Element, das in einem Blatt vorkommen kann, ist das Anmerkungselement zur Darstellung von Daten, die nicht in einem MathML-Format vorliegen.

Die wichtigsten Darstellungselemente sind <mi>, <mn> und <mo>, die jeweils Variablen, Zahlen und Operatoren kennzeichnen. Typischerweise wird das Programm, welches für die grafische Darstellung des MathML-Ausdrucks zuständig ist, für jedes dieser drei Elemente unterschiedliche Schriftarten, Schriftgrößen und Schriftattribute verwenden.

Die wichtigsten Aussageelemente sind <ci> und <cn> für Variablen und Zahlen. Spezielle Elemente für Funktionen und Operatoren werden zur Verfügung gestellt, wobei das <fn> Element dem Benutzter erlaubt, eigene Erweiterungen hinzuzufügen.

Die meisten MathML-Elemente besitzen ein Start- oder Endezeichen(start tag, end tag), welches den entsprechenden Inhalt umgibt (z.B. <mrow> und </mrow>). Eine andere Variante sind kanonisch leere Elemente, die keinen Inhalt haben und durch ein einfaches Zeichen dargestellt werden (z.B. <plus/>).

Das Beispiel $(a+b)^2$ läßt sich wie folgt darstellen:

Darstellungsbeschreibung	Aussagebeschreibung
<msup> <mfenced>	<apply> <power/>

<mrow> <mi> a</mi> <mo>+</mo> <mi>b</mi> </mrow> </mfenced> <mn>2</mn> </msup>	<apply> <plus/> <ci>a</ci> <ci>b</ci> </apply> <cn>2</cn> </apply>

Tabelle 2-1: Beispiel $(a+b)^2$

2.1.3 Presentation Markup

Es existieren insgesamt 28 verschiedene Elemente mit jeweils rund 50 verschiedenen Attributen, wobei die meisten 2-dimensionale Elemente sind, mit denen sich beispielsweise Basis und Exponent, Brüche, oder Tabellen beschreiben lassen. Die Elemente lassen sich in Gruppen unterteilen. Die eine Gruppe beschäftigt sich mit der horizontalen Ausrichtung der verschiedenen Zeichen. Hier gibt es <msub>, <munder>, <mmultiscript>. Eine andere Gruppe beschäftigt sich mit dem generellen Layout, wofür dann <mrow>, <mstyle> und <mfrac> benutzt werden können. Eine dritte Gruppe schließlich beschäftigt sich mit Tabellen.

Wichtig ist dabei die Reihenfolge der Argumente. So ist bei <mfrac> das erste Argument der Zähler und das zweite der Nenner.

2.1.4 Content Markup

Hier gibt es etwa 75 Elemente mit jeweils einigen Attributen. Die meisten Elemente sind leere Elemente, die für eine Vielzahl an Operatoren, Beziehungen und Funktionen gedacht sind. Beispiele hierzu sind <partialdiff>, <leq>, <tan>. Andere Elemente werden dazu benutzt, um verschieden mathematische Datentypen zu umschreiben, wie etwa <martix> und <set>.

Das <apply>-Element ist allerdings das wichtigste. Mit ihm lassen sich neue mathematische Objekte aus anderen zusammensetzen. Es ist wiederum die Reihenfolge der Argumente von Bedeutung. Ebenso kann dieses Element in prefix-Schreibweise benutzt werden, ähnlich wie schon <plus/>.

Weiterhin ist es möglich mit dem <declare>-Element einer bestimmten Variablen einen bestimmten Datentyp zuzuweisen. Dies wird insbesondere dann zur Anwendung kommen, wenn später die Verwendung eine Computer Algebra Systems in Erwägung gezogen wird.

2.1.5 Mischung beider Beschreibungen

Manchmal ist es notwendig, eine Mischung aus Darstellungs- und Aussagebeschreibung zu benutzen. Dies kann bei einigen Anwendungsprogrammen oder auch bei pädagogisch-orientierten Autorenwerkzeugen der Fall sein.

3 MathML-Syntax und Grammatik

Aufgrund der Tatsache, daß MathML ein Anwendung von XML ist, ist seine Syntax durch die von XML bestimmt. Die Grammatik von MathML ist in einer DTD (Document Type Definition) spezifiziert. Die Verwendung von tags, Attributen, usw. sind in der XML Sprachbeschreibung festgelegt, und die Besonderheiten von MathML-Elementen und Attributen sind in der MathML DTD definiert.

Es werden aber von MathML noch zusätzliche Syntax- und Grammatikregeln festgelegt, die nicht von XML übernommen worden sind. Eine wichtige neue Regel ist, daß es in MathML möglich ist, zusätz-

liche Kriterien an einen Wert eines Attribut zu stellen. So kann man beispielsweise festlegen, daß der Wert eines Attributes eine positive Ganzzahl sein muß. Eine zweite wichtige Regel legt genauere Beschränkungen für die Argumente fest. So läßt sich in XML nicht spezifizieren, daß das erste Argument auf die eine Weise interpretiert werden soll, und das zweite auf eine andere Weise.

3.1 XML Syntax: Kurzform

XML Daten werden aus Unicode Zeichen aufgebaut, die die normalen ASCII Zeichen beinhalten und aus 'entity references' (kurz entities), die besondere erweiterte Zeichen darstellen, wie etwa einen Zeilenvorschub, und aus 'elements' bestehen. Solche 'elements' sind beispielsweise `<mi>x</mi>`, die meistens aus einem Start- und Endezeichen aufgebaut sind und einen Inhalt besitzen, auf den sie sich beziehen. Es gibt aber auch empty elements, die keinen Inhalt besitzen und die deshalb auch kein Endezeichen haben (z.B. `<plus/>`). Die Startzeichen können zudem noch weitere Parameter beinhalten, die als Attribute bezeichnet werden (`<mi fontstyle="normal">`).

Weiterhin sei noch zu bemerken, daß in XML ein Unterschied zwischen Groß- und Kleinschreibung von Zeichen gemacht wird. Allerdings wird in `MathML` nahezu alles klein geschrieben.

3.2 `MathML` Attributwerte

Attribute müssen entweder der Form `attribute-name = "value"` oder der Form `attribute-name = 'value'` gehorchen, wobei die Leerzeichen vor und nach dem Gleichheitszeichen optional sind. Die Möglichkeit Attributwerte entweder in einfachen oder doppelten Anführungsstrichen zu schreiben, macht es möglich, daß das entsprechend andere Zeichen in dem Attributwert selber vorkommen kann. Weiterhin können die Zeichen `"`, `'` und `&` auch durch `&qout;`, `&apos;` und `&amp;` ersetzt werden.

3.2.1 Notationsformen

Die folgenden Konventionen werden bei der Beschreibung von Attributen verwendet:

number	Ganzzahl oder reelle Zahl mit einem Dezimalpunkt, optional startend mit einem negativen Vorzeichen
unsigned-number	Ganzzahl oder reelle Zahl, ohne Vorzeichen
integer	Ganzzahl, optional mit negativen Vorzeichen
unsigned-integer	Ganzzahl ohne Vorzeichen, nicht die Zahl Null
string	beliebige Zeichenkette
character	einzelnes Zeichen, nicht Leerzeichen
h-unit	Einheit für horizontale Längen
v-unit	Einheit für vertikale Längen
form+	eine oder mehrere Instanzen von from
form*	keine oder mehrere Instanzen von form
f1 f2 ... fn	nacheinander eine Instanz von jedem, eventuell getrennt durch ein Leerzeichen
f1 | f2 | ... | fn	eine beliebige Instanz der angegebenen
[form]	optionale Instanz von form
(form)	das gleiche wie form

3.2.2 Attribute mit Einheiten

Einige Attribute erlauben es, daß Einheiten zusätzlich zu einem Wert angegeben werden. In obiger Notation entspräche es `number [h-unit]`. Es sind folgend einige mögliche Einheiten aufgelistet:

em	ems (zeichensatzrelative Größe für horizontale Längen)
ex	exs (zeichensatzrelative Größe für vertikale Längen)
px	Pixel oder Pixelgröße einer Anzeige
in	Inch (1 inch = 2,54 centimeter)
cm	Centimeter
mm	Millimeter
pt	Punkte (1 point = 1/72 inch)
pc	picas (1 pica = 12 points)
%	Prozentsatz eines Standardwertes

3.3 Leerzeichen

Generell brauchen keine Leerzeichen für die Eingabe verwendet werden. Sie dienen hauptsächlich der besseren Lesbarkeit. Ebenfalls in diese Kategorie fallen Tabulatoren, neue Zeilen und Wagenrücklauf. Beispielsweise sind `<mo> ( </mo>` und `<mo>(</mo>` genau das gleiche. Ähnlich verhält es sich mit:

```
<mtext>
  Theorem
  1:
</mtext>
```

und `<mtext>Theorem 1:</mtext>`.

4 Presentation Markup

Mit den Elementen des presentation markups wird die Visualisierung eines mathematischen Ausdrucks definiert. Unterschiedliche Varianten der Visualisierung läßt sich über verschiedene Attribute festelegen. Die Elemente des presentation markups lassen sich unterteilen in Zeichenelemente, die zur Darstellung von Zahlen, Buchstaben und mathematischen Symbolen verwendet werden, und in Layoutelemente, die sich mit der Anordnung der Zeichen befassen.

4.1 Wozu presentation elements?

Die Darstellungsbeschreibung besteht aus einzelnen Elementen, den presentation elements (Darstellungselemente). Diese drücken die Struktur eines mathematischen Ausdrucks aus, und zwar ähnlich wie in Büchern der Titel, die Kapitel und Abschnitte. Aufgrund dessen wird ein einzelner Ausdruck, wie "x + a/b" nicht nur durch ein einziges `<mrow>`-Element dargestellt, sondern durch mehrere geschachtelte `<mrow>`-Elemente, um die Struktur des mathematischen Ausdrucks sichtbar zu machen. Man wird den Ausdruck in `MathML` wie folgt beschreiben:

```
<mrow>
  <mi> x </mi>
  <mo> + </mo>
  <mrow>
    <mi> a </mi>
    <mo> / </mo>
    <mi> b </mi>
  </mrow>
</mrow>
```

Ähnlich verhält es sich mit Exponenten. Hier wird man, etwa beim Ausdruck 465^2, nicht nur der letzten Zahl 5 den Exponenten zuordnen, sondern dem kompletten Ausdruck, der die Basis darstellt. Dies kann zu einer besseren Qualität bei der Darstellung des mathematischen Ausdrucks führen, da z.B. zwischen die einzelnen Zahlen keine Abstände gesetzt werden.

Desweiteren existieren spezielle erweiterte Zeichen, die in normaler Schreibweise durch spezielle Symbole dargestellt werden. Hierunter fallen etwa das Differentiationssymbol, die eulersche Zahl e und auch die imaginäre Zahl i bzw. j. Diese Zeichen sollten zwecks einer besseren Erkennbarkeit nicht in der gleichen Schriftart und –größe und mit den gleichen Schriftattributen dargestellt werden, wie etwa die Zeichen, die folgen oder vorhergegangen sind. In MathML werden diese Symbole daher durch "ⅆ", "ⅇ" und "ⅈ" ausgedrückt.

In MathML existieren spezielle Zeichen, um auch nichtgeschriebene Ausdrücke darstellen zu können. Etwa wird "4a" mit einem "⁢" versehen, oder die Funktion "f(x)" wird mit dem "⁡" Zeichen beschrieben. Für komplizierte Indizeschreibweisen, wie etwa "x_{31}" wird ein Zeichen "⁣" verwendet. Diese speziellen Zeichen erlauben es dem Programm, welches diese mathematischen Ausdrücke später darstellen soll, entsprechende Abstände zu setzen, damit der Leser erkennt, welche Bedeutung der Ausdrucks besitzt. Falls als Ausgabe nicht eine optisches Gerät (z.B. Monitor) benutzt wird, sondern ein Gerät, welches den Text vorliest, so besteht durch diese speziellen Zeichen die Möglichkeit, den eigentlich nur nacheinander gesprochenen Zeichen einen Sinn zu geben. "4a" kann dann so gesprochen werden, wie es jeder tut, nämlich '4 mal a'.

4.2 Benötigte Argumente

Viele der in MathML vorhandenen Elemente verlangen die Angabe einer bestimmten Anzahl an Argumenten (in der Regel eins, zwei oder drei). Es wird über die Numerierung der Argumente festgelegt, welche Position das child element in dem zughörigen parent element einnimmt.

4.2.1 Implizit vorhandene <mrow>-Elemente

Solche implizit vorhandenen <mrow>-Elemente (inferred <mrow>‘s) dienen zur Erleichterung des Erstellens von MathML-Ausdrücken. So können an einigen Stellen (z.B. beim Wurzelzeichen) die <mrow> Elemente weggelassen werden. In MathML wird der Ausdruck

```
<msqrt>
  <mo> - </mo>
  <mi> 1 </mi>
</msqrt>
```

so behandelt, als wäre innerhalb der Wurzel ein <mrow>-Element vorhanden.

```
<msqrt>
```

```
    <mrow>
      <mo> - </mo>
      <mi> 1 </mi>
    </mrow>
  </msqrt>
```

4.2.2 Auflistung notwendiger Elemente

In der folgenden Tabelle sind einige beispielhafte Elemente mit der Anzahl der benötigten Argumente und der Bedeutung der Argumente angegeben. Später werden diese näher erläutert.

Element	Anzahl der Argumente	Bedeutung der Argumente
<mrow>	null oder mehr als eins	-
<mfrac>	2	Zähler und Nenner
<msqrt>	1	-
<mroot>	2	Basis und Index
<mfenced>	null oder mehr als eins	-
<msub>	2	normal- und tiefgestelltes
<msup>	2	normal- und hochgestelltes
<msubsup>	3	normal-, tief- und hochgestelltes
<munder>	2	normal- und daruntergestelltes
<mover>	2	normal- und darübergestelltes
<munderover>	3	normal-, darunter- und darübergestelltes
<mmultiscript>	eins oder mehr	(normalgestelltes (tiefgestelltes hochgestelltes)* [<mprescripts/> (voran- & tiefgestellt voran- & hochgestellt)*])
<mtable>	null oder mehrere Zeilen	<mtr>s, inferred, falls notwendig
<mtr>	null oder mehrere Tabellenelemente	<mtd>s, inferred, falls notwendig
<mtd>	1	-

Tabelle 4-1: wichtige MathML-Elemente

4.3 Zeichenelemente

Zeichenelemente bestehen aus null oder mehreren Zeichen und können mit Attributen ausgestattet werden. Grundsätzlich haben alle Zeichenelemente einige gemeinsame Attribute, die mit der Textformatierung bei der Anzeige verbunden sind. Zusätzlich gibt es noch weitere Attribute, die aber nur bei einem bestimmten Zeichenelement vorhanden sind.

Name	Wert	Standardwert
fontsize	number v-unit	geerbt
fontweight	normal | bold	geerbt
fontstyle	normal | italic	normal
fontfamily	string | css-fontfamily	geerbt
color	#rgb | #rrggbb | html-color-name	geerbt

Tabelle 4-2: gemeinsame Attirbute aller Zeichenelemente

Geerbte Standardwerte bedeutet, daß diese aus der Umgebung des Darstellungsprogrammes gesetzt werden. Damit kann der Anwender des Darstellungsprogrammes die Visualisierung des mathematischen Ausdrucks noch in einigen Bereichen variieren.

4.3.1 <mi>

Ein <mi>-Element repräsentiert einen symbolischen Namen oder einen willkürlichen Text. Dieser kann aus Variablen, Funktionsnamen und symbolischen Konstanten bestehen. <mi>-Elemente akzeptieren die Attribute aus Kapitel 4.3, allerdings hat das Attribut fontstyle einen anderen Standardwert, der wiederum vom Inhalt abhängig ist. Besteht der Inhalt aus einem einzigen Zeichen, so hat das Attribut fontstyle den Standardwert italic, andernfalls normal.

Beispiele:

```
<mrow>
  <mi> sin </mi>
  <mo> &ApplyFunction; </mo>
  <mi> x </mi>
</mrow>

<mrow>
  <mn> 1 </mn>
  <mo> + </mo>
  <mi> ... </mi>
  <mo> + </mo>
  <mi> n </mi>
</mrow>
```

Die Verwendung von Konstanten geschieht ebenfalls über <mi> Elemente.

```
<mi> &pi; </mi>

<mi> &ImaginaryI; </mi>
```

4.3.2 <mn>

Ein <mn>-Element repräsentiert ein numerisches Literal oder andere Daten. Anders ausgedrückt ist ein numerisches Literal als Folge von Zahlen, die eventuell Dezimaltrennzeichen (Dezimalpunkt) beinhalten, zu betrachten, daß entweder eine vorzeichenlose Ganzzahl oder reelle Zahl darstellt. Dabei akzeptiert das <mn>-Element alle Attribute aus Kapitel 4.3.

Beispiele:

```
<mn> 2 </mn>
<mn> 0.123 </mn>
<mn> 1,000,000 </mn>
<mn> 2.1e10 </mn>
<mn> 0xFFEF </mn>
<mn> MCMLXIX </mn>
<mn> thirty six </mn>
```

Allerdings sollten nicht alle mathematischen Zahlen als <mn>-Element dargestellt werden. So sind zur Darstellung negativer Zahlen, komplexer Zahlen und Zahlenverhältnisse andere Elemente zu verwenden.

Beispiele:

```
<mrow> <mo> - </mo> <mn> 1 </mn> </mrow>

<mrow>
  <mn> 2 </mn>
  <mo> + </mo>
  <mrow>
    <mn> 3 </mn>
    <mo> &InvisibleTimes; </mo>
    <mi> &ImaginaryI; </mi>
  </mrow>
</mrow>

<mfrac> <mn> 1 </mn> <mn> 2 </mn> </mfrac>
```

4.3.3 <mo>

Ein <mo>-Element repräsentiert Operatoren oder alles, das wie eine Operator angezeigt werden soll. Dabei sind als Operatoren nicht nur die gewöhnlichen mathematischen Operatoren anzusehen, sondern auch Abgrenzungen, wie Klammerungen und Absolutbalken, und Trennzeichen, die aus Komma und Semikolon bestehen können. Desweiteren können auch Akzente, wie beispielsweise Tilden, Striche, Dächer oder Punkte über Symbolen, mit <mo> dargestellt werden.

Als Attribute werden neben den Standardattributen auch zusätzlich noch folgende akzeptiert:

Name	**Wert**	**Standardwert**
form	prefix | infix | postfix	abhängig von der Position des Operanden in der <mrow> Umgebung
fence	true | false	false
separator	true | false	false
lspace	number h-unit	0.27777em
rspace	number h-unit	0.27777em
maxsize	number [v-unit | h-unit] | infinity	infinity
minsize	number [v-unit | h-unit]	1
largeop	true | false	false
moveablelimits	true | false	false
accent	true | false	false

Tabelle 4-3: zusätzliche Attribute von <mo>

Unterschied in der Darstellung können sich aufgrund differierender Zeichensätze ergeben. So wird beispielsweise <mo> ≤ </mo> auf einem Textterminal als <= dargestellt. Generell gilt der Grundsatz, daß der Inhalt eines <mo> Elementes so genau wie möglich darzustellen ist. Das führt da-

zu, daß <mo> ≤ </mo> und <mo> <= </mo> unterschiedlich dargestellt werden, nämlich einmal ≤ und beim zweiten Mal <=.

Es folgen nun einige Beispiele zur Darstellung von Klammerungen mittels <mo>-Elemente:

(a+b)

```
<mrow>
  <mo> ( </mo>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mrow>
  <mo> ) </mo>
</mrow>
```

f(x,y)

```
<mrow>
  <mi> f </mi>
  <mo> &ApplyFunction </mo>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mi> x </mi>
      <mo> , </mo>
      <mi> y </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
</mrow>
```

<mo> Elemente können auch für unsichtbare Operatoren verwendet werden, wie etwa für &InvisibleTime;. Dabei sind folgende Abkürzungen erlaubt:

voller Name	Abkürzung	Verwendung
&InvisibleTime	⁢	xy
&ApplyFunction	⁡	f(x) oder sin x
&InvisibleComma	⁣	m_{12}

Tabelle 4-4: Abkürzungen für unsichtbare Operatoren

4.3.4 <mtext>

<mtext>-Elemente werden zur Repräsentation beliebigen Textes verwendet, der auch als solcher dargestellt werden soll. Hauptsächlich wird das <mtext>-Element zur Darstellung von Kommentaren verwendet, der eine untergeordnete Rolle bei der Bedeutung des mathematischen Ausdrucks spielt. Das <mtext>-Element verwendet dabei die Standardattribute aus Kapitel 4.3.

Es existieren auch einige Spezialzeichen, die zwar als Leerraum dargestellt werden, allerdings auf die Darstellung doch einen Einfluß haben.

Name	**Verwendung**
NewLine	neue Zeile, nicht eingerückt
IndentingNewLine	neue Zeile, eingerückt
NoBreak	Zeilenumbruch an dieser Stelle nicht erlaubt
GoodBreak	gute Position für einen Zeilenumbruch, falls dieser gebraucht wird
BadBreak	schlechte Position für einen Zeilenumbruch, falls dieser gebraucht wird

Tabelle 4-5: Spezialzeichen zur Darstellung von Leerraum

4.3.5 <mspace/>

Durch ein `<mspace/>`-Element wird ein Leerraum jeder beliebigen Größe dargestellt, der allerdings durch die Attribute des Elementes angegeben ist. Es existieren dazu drei Attribute:

Name	**Wert**	**Standardwert**
width	number h-unit	0em
height	number v-unit	0ex
depth	number v-unit	0ex

Tabelle 4-6: `<mspace>`-Attribute

4.3.6 <ms>

Das `<ms>`-Element wird zur Repräsentation von Zeichenketten in mathematischen Ausdrücken benutzt, die durch ein Computer Algebra System auswertbar sind. Standardmäßig werden solche Zeichenketten innerhalb von doppelten Anführungsstrichen dargestellt. Es werden neben den Attributen aus Kapitel 4.3 auch noch zwei weitere Attribute unterstützt:

Name	**Wert**	**Standardwert**
lquote	string	&qout;
rquote	string	&qout;

Tabelle 4-7: `<ms>`-Attribute

Mit diesen beiden Attributen lassen sich die rechten und linken Zeichen, die eine Zeichenkette begrenzen, verändern.

4.4 Layoutelemente

Nachdem nun einige Zeichenelemente behandelt wurden, werden jetzt einige Elemente zur Darstellung von Brüchen, Wurzeln vorgestellt und die sich ganz allgemein mit dem Aussehen des mathematischen Ausdrucks befassen.

4.4.1 <mrow>

`<mrow>`-Elemente werde benutzt, um einen beliebig lange, horizontal angeordnete Teilausdrücke zu gruppieren. Diese Gruppe besteht gewöhnlich aus einem oder mehreren `<mo>`-Elementen, die sich

wie Operatoren verhalten, und einer Reihe von anderen Unterausdrücken, die die Rolle der dazugehörigen Operanden übernehmen. Einige Zeichenelemente behandeln dabei ihre Argumente so, als würden diese in einer <mrow>-Umgebung stehen, obwohl dies nicht der Fall ist.

Falls das <mrow>-Element nur ein Argument besitzt, so wird das Argument so dargestellt, als wäre das <mrow>-Element nicht vorhanden, es sei denn, das <mrow> Element enthält noch zusätzliche Attribute. Die beim <mrow>-Element verwendbaren Attribute sind die Kapitel 4.3 aufgeführten.

Eine Gruppierung von Teilausdrücken durch <mrow>−Element sollte durch den Autor so vorgenommen werden, daß diese mit der mathematischen Bedeutung des Ausdrucks und mit dem zugrundeliegendem Syntaxbaum übereinstimmt. Insbesondere sollten sich Operatoren in einer extra <mrow>-Umgebung befinden. Eine sinnvolle Gruppierung hat mehrere Hintergründe. So werden hierdurch die Abstände bei der Darstellung kontrollierbarer und der Zeilenumbruch langer Ausdrücke wird erleichtert. Außerdem lassen sich semantische Interpretationsmöglichkeiten für Computer Algebra System und Sprachsynthetisierer darstellen.

4.4.2 <mfrac>

Um Brüche darzustellen, werden <mfrac>-Elemente benutzt. Ebenfalls lassen sich mit diesem Element Ausdrücke erzeugen, die eine ähnliches Aussehen haben, wie Brüche, beispielsweise Binomialkoeffizienten.

Das <mfrac>-Element hat die Syntax <mfrac> *zähler nenner* </mfrac>. Es existiert nur ein Attribut, welches die Dicke des horizontalen Balkens angibt.

Name	**Wert**	**Standardwert**
linethickness	number [v-unit] | thin | medium | thick	1

Tabelle 4-8: <mfrac>-Attribut

Dabei wird bei einer Linienstärke von 0 kein Bruchstrich erzeugt, und größere Werte als 1 lassen sich bei der Darstellung geschachtelter Brüche verwenden. Neben der Angabe eines Zahlenwerts mit Einheit, ist auch die Angabe von Schlüsselwörtern möglich, die dann wie ein relativer Multiplikationsfaktor zum aktuellen Wert benutzt werden.

4.4.3 <msqrt> und <mroot>

Diese beiden Elemente erzeugen Wurzelausdrücke. Das <msqrt>-Element wird zur Erzeugung des Spezialfalles Quadratwurzel verwendet.

Syntax:

```
<msqrt> basis </msqrt>
<mroot> basis index </mroot>
```

Das <mroot>-Element verlangt dabei exakt zwei Argumente. Das <msqrt>-Element kommt auch mit mehr als einen Argument zurecht. Es erzeugt dann eine <mrow>-Umgebung.

4.4.4 <mstyle>

Zur Veränderung von stylistischen Merkmalen der Darstellung wird das <mstyle>-Element verwendet. Es können alle Attribute verwendet werden, die von jedem beliebigen anderen MathML Element akzeptiert werden. Einfacher ausgedrückt kann man mit dem <mstyle>-Element den Standardwert eines Attributes für das Element ändern, das in der <mstyle>-Umgebung enthalten ist.

Die eigentliche Änderung kann auf zwei verschiedene Arten durchgeführt werden.

- Einige Attribute werden aus der Umgebung geerbt, falls diese nicht explizit gesetzt sind. Mit `<mstyle>` kann der geerbte Wert der child elements geändert werden. Überschreibt ein child element wiederum diesen Wert, so wird dieser neue Wert an die nächste Generation weitergegeben.
- Andere Attribute, wie `linethickness` bei `<mfrac>`, besitzen Standardwerte, die normalerweise nicht vererbt werden. Wird etwa bei einem übergeordneten `<mfrac>` dieses Attribute geändert, so erhalten die untergeordneten `<mfrac>`-Elemente in seiner Umgebung nicht den neuen Wert. Wird dagegen mit `<mstyle>` der Standardwert diese Attributes geändert, so erhalten alle `<mfrac>`-Elemente in der `<mstyle>`-Umgebung den neuen Standardwert. Natürlich kann individuell der Attributwert bei einzelnen `<mfrac>`-Elementen überschrieben werden.

Folgende Attribute werden von `<mstyle>` noch zusätzlich akzeptiert:

Name	**Wert**	**Standardwert**
scriptlevel	['+', '-'] vorzeichenlose Ganzzahl	geerbt
displaystyle	true | false	geerbt
scriptsizemultiplier	number	0.71
scriptminsize	number v-unit	8pt
color	#rgb | #rrggbb | html-color-name	geerbt
background	#rgb | #rrggbb | transparent | html-color-name	transparent

Tabelle 4-9: `<mstyle>`-Attribute

Die beiden Attribute *scriptlevel* und *displaystyle* sind für die orthogonale Ausrichtung der Ausdrücke zuständig. Das Attribut *displaystyle* bestimmt, ob spezielle Attribute (z.B. *largeop* und *moveablescripts* des `<mo>`-Elementes) anderer Elemente, die sich in der `<mstyle>`-Umgebung befinden, eine Wirkung besitzen, oder nicht. Das Attribut *scriptlevel* regelt hauptsächlich die Schriftgröße, Typischerweise wird die Schriftgröße bei steigendem *scriptlevel* kleiner.

Immer wenn der *scriptlevel* geändert wird, so wird die aktuelle Schriftgröße mit dem Wert des Attributes *scriptsizemultiplier* entweder multipliziert oder durch ihn dividiert. Dabei gibt *scriptminsize* eine minimale Untergrenze an.

4.4.5 <merror>

Das `<merror>`-Element stellt seinen Inhalt als Fehlermeldung dar, wobei diese aus anderen `MathML` Elementen bestehen kann. Durch dieses Element ist es möglich, einen standardisierten Weg zur Beschreibung von Fehler für Programme zur Verfügung zu stellen, die ihre Eingabe aus anderen Quellen beziehen und `MathML` erzeugen.

Dabei wird der fehlerbehaftete Teil in eine `<merror>`-Umgebung eingebettet und der restliche Teil wird so gut wie möglich verarbeitet. Dies erleichtert die anschließende Fehlersuche für den Autor erheblich.

Beispiel:

```
<mfraction>
  <mrow>
    <mn> 1 </mn>
    <mo> + </mo>
    <msqrt> <mn> 5 </mn> </msqrt>
  </mrow>
```

```
    <mn> 2 </mn>
  </mfraction>
```

In diesem Beispiel wird statt <mfrac> das nicht-MathML-Element <mfraction> benutzt. Folgende Fehlermeldung wird erzeugt:

```
  <merror>
    <mtext>  Unrecognized element: &lt;mfraction&gt;;
             arguments were: 
    </mtext>
    <mrow>
      <mn> 1 </mn>
      <mo> + </mo>
      <msqrt> <mn> 5 </mn> </msqrt>
    </mrow>
    <mtext>  and  </mtext>
    <mn> 2 </mn>
  </merror>
```

Interessant ist, das die Eingabe nicht MathML konform ist, allerdings wird MathML konformer Code erzeugt.

4.4.6 <mpadded>

Manachmal ist es notwendig, daß der Abstand um einige Elemente verändert wird. Dies kann das <mpadded>-Element bewerkstelligen. Es wird dabei nicht der Inhalt selbst vergrößert oder geschrumpft, sondern die Größe und Position des den Inhalt umgebenen Rechtecks wird geändert. Bei Sprachsynthetisieren wird geeigneterweise eine Zeitverzögerung verwendet.

Zulässige Attribute sind:

Name	Wert	Standardwert
width	[+ | -] vorzeichenlose Zahl (% [pseudo-unit] | pseudo-unit | h-unit)	genauso wie Inhalt
lspace	[+ | -] vorzeichenlose Zahl (% [pseudo-unit] | pseudo-unit | h-unit)	0
height	[+ | -] vorzeichenlose Zahl (% [pseudo-unit] | pseudo-unit | v-unit)	genauso wie Inhalt
depth	[+ | -] vorzeichenlose Zahl (% [pseudo-unit] | pseudo-unit | v-unit)	genauso wie Inhalt

Tabelle 4-10: <mpadded> Attribute

Falls der Wert ein Vorzeichen enthält, wird der alte Wert um diesen Betrag vergrößert bzw. verkleinert. Ist kein Vorzeichen vorhanden, so wird der Attributwert direkt auf den angegebenen Wert gesetzt. Pseudo-Unit bedeutet, daß eine der 4 möglichen Attribute benutzt wird. Beispiele dafür sind depth="100% height" oder depth="1.0 height". Eine Verdoppelung kann ausgedrückt werden durch depth="+100%" oder depth="200%".

Das Attribut *width* beschreibt die horizontale Weite des den Inhalt umgebenen Rechtecks. Allerdings wird eine Vergrößerung dieses Attributes nur ein Einfügen zusätzlichen Raumes am rechten Rand dem Umgebungsrechteckes zur Folge haben und nicht symmetrisch links und rechts. Das *lspace* Attribut gibt den Abstand zwischen linkem Rand des Umgebungsrechteckes und dem Beginn des Inhaltes an. Auf diese Weise läßt sich am linken Rand zusätzlicher Freiraum einfügen.

Die Attribute *heigth* und *depth* geben den Abstand zwischen der Grundlinie, auf der die meisten Buchstaben "stehen", und der oberen bzw. unteren Grenze des umgebenen Rechtecks an.

Das `<mpadded>` Element sollte allerdings nicht dazu benutzt werden, um negative Abstände zu generieren oder neue Zeichen bzw. Ausdrücke zu schaffen. Gründe dafür sind, daß es zu abweichenden Ergebnissen bei der Verwendung unterschiedlicher Darstellungswerkzeuge kommen kann. Bei anderen Programmen, die `MathML` als Eingabe benutzen, kann es auf diese Weise zu groben Fehlern kommen, etwa bei Sprachsynthetisierer oder Computer Algebra Systemen.

4.4.7 <mphantom>

Das `<mphantom>` Element dient zur unsichtbaren Darstellung seines Inhaltes. Die Position und Größe des Inhaltes wird beibehalten, es ist eben nur nichts sichtbar. Dies kann sehr vorteilhaft sein, um etwa eine gewisse Ordnung in mathematische Ausdrücke zu bringen. Beispielsweise bei der Aufstellung von linearen Gleichungsystemen würde diese Element Anwendung finden, oder auch bei Brüchen:

MathML	**Darstellung**

`<mfrac>`
`  <mrow>`
`  <mi> x </mi>`
`  <mo> + </mo>`
`  <mi> y </mi>`
`  <mo> + </mo>`
`  <mi> z </mi>`
`  </mrow>`
`  <mrow>`
`  <mi> x </mi>`
`  <mphantom>`
`  <mo> + </mo>`
`  </mphantom>`
`  <mphantom>`
`  <mi> y </mi>`
`  </mphantom>`
`  <mo> + </mo>`
`  <mi> z </mi>`
`  </mrow>`
`</mfrac>`
	x + y + z
	--------- 	
	x + z

Tabelle 4-11: Beispiel zu `<mphantom>`

4.4.8 <mfenced>

Mittels dieses Elementes wird dem Autor ein einfaches Hilfsmittel an die Hand gelegt, um geklammerte Ausdrücke, eventuell auch mit Trennzeichen, darzustellen. So erzeugt `<mfenced>`

<mi>x</mi> </mfenced> die gleiche Darstellung wie <mrow> <mo> (</mo> <mi>x</mi> <mo>) </mo> </mrow>.

Grundsätzlich kann der Inhalt einer <mfenced>-Umgebung null oder mehrere Argumente beinhalten, die zwischen geklammerten Ausdrücken dargestellt werden. Als Zeichen zur Klammerung muß nicht notwendigerweise eine normale runde Klammer zum Einsatz kommen. Vorstellbar sind auch ekkige Klammern, Mengenklammern, oder gar ganz andere Zeichen, wie Sternchen oder Doppelkreuz. Sind mehrere Argumente vorhanden, so werden diese durch ein Trennzeichen unterteilt. Dieses Trennzeichen ist standardmäßig ein Komma. Es kann aber auch verändert werden, und so durch jede andere Zeichenkette ausgedrückt werden. Die Attribute sind folglich:

Name	Wert	Standardwert
open	Zeichenkette	(
close	Zeichenkette	)
separators	Zeichen*	,

Tabelle 4-12: <mfenced>-Attribute

Wird dem Attribut eine Sequenz von Zeichen zugeordnet, so werden bei mehreren Argumente diese durch das entsprechende Trennzeichen getrennt. Sind mehr Argumente als Trennzeichen vorhanden, so wird das letzte Trennzeichen mehrfach verwendet. Im umgekehrten Fall werden die überflüssigen Trennzeichen einfach ignoriert.

Beispiel:

```
<mfenced open="opening-fence"
          close="closing-fence"
          separators="sep#1 sep#2 ... sep#(n-1)" >
  arg#1
  ...
  arg#n
</mfenced>
```

4.5 Erstellung von Indizes und Akzenten

Mit diesen Elementen ist es möglich, eine der beliebtesten Darstellungsformen in den Naturwissenschaften zu beschreiben, die Indizierung. So können einem Basisausdruck vier mögliche Indexpositionen (davor unten und oben, dahinter unten und oben) bereitgestellt werden, und noch zusätzlich zwei Positionen für Akzente (darunter und darüber). Es lassen sich aber nicht nur Indizes erstellen, sondern mit diesem Hilfsmittel kann man auch bequem Exponenten darstellen.

Natürlich sollte, um die Struktur eines Gesamtausdrucks beizubehalten, nicht nur das am weitesten rechts stehende Element eines Basisausdrucks einen Index bzw. Exponenten erhalten, sondern der gesamte Basisausdruck.

4.5.1 <msub>

Das <msub>-Element fügt zu einer Basis einen nach- und tiefergestellten Ausdruck hinzu. Die Syntax ist: <msub> *base subscript* </msub>.

Es gibt nur ein Attribut, welches den minimalen Abstand angibt, um das die Grundlinie des Tiefergestellten nach unten wandert, im Vergleich zur Grundlinie des Basisausdrucks.

Name	Wert	Standardwert
subscriptshift	number v-unit	automatic (typische Einheit ist ex)

Tabelle 4-13: <msub>-Attribut

4.5.2 <msup>

Im Gegensatz zum <msub>-Element wird beim <msup>-Element einem Basisausdruck ein nach- und höhergestellter Ausdruck hinzugefügt. Die Syntax ist ähnlich:

<msup> *base superscript* </msup>.

Es existiert ebenfalls nur ein Attribut, welches den minimalen Abstand angibt, um das die Grundlinie des Höhergestellten im Vergleich zur Grundlinie des Basisausdrucks nach oben wandert.

Name	Wert	Standardwert
Superscriptshift	number v-unit	automatic (typische Einheit ist ex)

Tabelle 4-14: <msup>-Attribute

4.5.3 <msubsup>

Mit diesem Element ist es möglich gleichzeitig etwas Tiefer- und Höhergestelltes einem Basisausdruck nachzustellen. Dabei werden die beiden tiefer- und höhergestellten Ausdrücke so nahe wie möglich an den Basisausdruck herangestellt, so daß diese vertikal die gleiche Position einnehmen. Daraus ergibt sich folgende Syntax:

<msubsup> *base subscript superscript* </msubsup>.

Folglich sind zwei Attribute zulässig, die die entsprechende Bedeutung besitzen, wie schon bei <msub> und <msup>:

Name	Wert	Standardwert
subscriptshift	number v-unit	automatic (typische Einheit ist ex)
superscriptshift	number v-unit	automatic (typische Einheit ist ex)

Tabelle 4-15: <msubsup>-Attribute

So läßt sich ein Integral darstellen $\int_0^1 e^x \partial x$ als:

```
<mrow>
  <msubsup>
    <mo> &int; </mo>
    <mn> 0 </mn>
    <mn> 1 </mn>
  </msubsup>
  <mrow>
    <msup>
      <mi> &ExponentialE; </mi>
      <mi> x </mi>
    </msup>
    <mo> &InvisibleTimes; </mo>
```

```
      <mrow>
        <mo> &DifferentialD; </mo>
        <mi> x </mi>
      </mrow>
    </mrow>
  </mrow>
```

4.5.4 <munder>

Mit diesem MathML Element läßt sich ein Akzent unter einen Basisausdruck hinzufügen. Entsprechend ist die Syntax:
<munder> *base* *underscript* </munder>

Es existiert ein Attribut, welches angibt, ob das Argument *underscript* als Akzent geschrieben wird, d.h. in normaler Größe, oder verkleinert wird.

Name	Wert	Standardwert
accentunder	true | false	automatic

Tabelle 4-16: <munder>-Attribut

Der Standardwert des Attributes ist false. Ausnahmen gibt es, wenn als *underscript* Argument beispielsweise ein <mo>-Element benutzt wird.

Beispiel:

MathML	Darstellung
<mrow> <munder accentunder="true"> <mrow> <mi> x </mi> <mo> + </mo> <mi> y </mi> <mo> + </mo> <mi> z </mi> </mrow> <mo> ⏟ </mo> </munder> <mtext> vs </mtext> <munder accentunder="false"> <mrow> <mi> x </mi> <mo> + </mo> <mi> y </mi> <mo> + </mo> <mi> z </mi> </mrow> <mo> ⏟ </mo>	$\underbrace{x+y+z}$ vs $\underbrace{x+y+z}$

</munder> </mrow>	

Tabelle 4-17: <munder>-Beispiel

4.5.5 <mover>

Ähnlich zum <munder>-Element lassen sich mit dem <mover>-Element Akzente über einen Basisausdruck schreiben. Die Syntax ist entsprechend, wie bei <mover>:

<mover> *base overscript* </mover>

Das einzige Attribut gibt ebenfalls an, ob das Argument verkleinert dargestellt werden soll, oder als normalgroßer Akzent.

Name	Wert	Standardwert
accent	true | false	automatic

Tabelle 4-18: <mover>-Attribut

Ein einfacheres Beispiel mit einem Dach ist:

MathML	Darstellung
<mrow> <mover accent="true"> <mi> x </mi> <mo> &hat; </mo> </mover> <mtext> vs </mtext> <mover accent="false"> <mi> x </mi> <mo> &hat; </mo> </mover> </mrow>	$\hat{x}$ vs $\hat{x}$

Tabelle 4-19: <mover>-Beispiel

4.5.6 <munderover>

Das <munderover>-Element vereint die beiden letztgenannten Element in der Form, das das *underscript* und das *overscript* in vertikaler Richtung die gleiche Position einnehmen. Die Syntax ist also:

<munderover> *base underscript overscript* </munderover>

Entsprechend existieren auch zwei Attribute, die die gleiche Bedeutung besitzen, wie bie <munder> und <mover>.

Name	Wert	Standardwert
accent	true | false	automatic
accentunder	true | false	automatic

Tabelle 4-20: <munderover>-Attribute

Der Unterschied zwischen <munderover> und der Hintereinanderschaltung von <munder> und <mover> ist auf Bildschirmen nicht auszumachen, allerdings auf hochauflösenden Drucker lassen sich die Unterschiede erkennen.

MathML	**Darstellung**
<mrow> <mover> <munder> <mo> ∫ </mo> <mn> 0 </mn> </munder> <mi> &infinity; </mi> </mover> <mtext> vs </mtext> <munderover> <mo> ∫ </mo> <mn> 0 </mn> <mi> &infinity; </mi> </munderover> </mrow>	$\int_0^\infty$ vs $\int_0^\infty$

Tabelle 4-21: <munderover>-Beispiel

4.5.7 <mmultiscripts>

Die Syntax diese Elementes lautet:

```
<mmultiscripts>
  base
  ( subscript superscript ) *
  [ <mprescripts/> ( presubscript presuperscript ) * ]
</mmultiscripts>
```

Hiermit ist es möglich alle vier Positionen um einen Basisausdruck anzusprechen, die zudem noch vertikal ausgerichtet sind. Falls eine Position nicht benutzt wird, so ist an entsprechender Stelle in MathML ein leeres Element <none/> zu setzen. Die Benutzung der beiden vorangestellten Positionen ist optional, da diese Indizierung recht selten benutzt wird.

Anzumerken sei noch, daß die Argumente von <mmultiscripts> alle auf der gleichen Ebene in einem Ausdrucksbaum sind. Dies gilt auch für das leere Element <mprescripts/>. Die verwendbaren Attribute sind exakt die gleichen, wie bei dem <msubsup>-Element aus 4.5.3.

Ein kurzes Beispiel dient zur Erläuterung:

MathML	**Darstellung**
<mmultiscripts> <mi> R </mi> <mi> k </mi> <mi> j </mi> <mn> 5 </mn> <none/> <mprescripts/> <mi> i </mi> <mn> 1 </mn> </mmultiscripts>	${}_{i}^{1}R_{k5}^{j}$

Tabelle 4-22: <mmultiscripts>-Beispiel

4.6 Tabellen und Matrizen

Um Matrizen, Tabellen und andere tabellenähnliche Ausdrücke darzustellen, werden die MathML Elemente <mtable>, <mtr> und <mtd> benutzt. Diese Elemente ähneln sehr denen in HTML, allerdings um die Attribute erweitert, die eine bessere Kontrolle des Layout ermöglichen, um etwa Matrizen zu erzeugen.

4.6.1 <mtable>

Grundsätzlich wird eine Tabelle oder Matrix mit dem <mtable>-Element erzeugt. Für die genauere Struktur werden innerhalb des <mtable>-Elements die Zeilen mit <mtr> und die einzelnen Zellen mit <mtd> erzeugt. Falls das <mtr>-Element nicht vorhanden ist, wird eine einspaltige Ausführung erzeugt, und eine <mtd>-Umgebung impliziert.

Um eine Matrix zu erzeugen, müssen auch die Klammern hinzugefügt werden. Dies geschieht entweder mit <mrow> und <mo>-Elementen oder komfortabler mit dem <mfenced>-Element.

Es gibt eine Vielzahl von Attributen, von denen die wichtigsten aufgelistet sind:

Name	**Wert**	**Standardwert**
align	(top | bottom | center | baseline | axis) [rownumber]	axis
rowalign	(top | bottom | center | baseline | axis)+	baseline
columnalign	(left | center | right)+	center
rowspacing	(number v-unit)+	1.0ex
columnspacing	(number h-unit)+	0.8em
framespacing	number h-unit number v-unit	0.4em 0.5ex
rowlines	(none | solid | dashed)+	none
columnlines	(none | solid | dashed)+	none
frame	none | solid | dashed	none
equalrows	true | false	true
equalcolumns	true | false	true

Tabelle 4-23: <mtable>-Attribute

Das Attribut *align* beschreibt, in welcher Form die Tabelle in ihre Umgebung eingebettet wird. Als Richtwert bei der Ausrichtung wird immer die Grundlinie der Umgebung benutzt. Wird noch optional eine Zeilennummer angegeben, so wird die angegebene Zeile der Tabelle entsprechend ausgerichtet.

Die beiden Attribute *rowalign* und *columnalign* beschreiben die Ausrichtung der Zelleninhalte innerhalb einer Zeile bzw. Spalte. Dabei sind die Attribute in gleicher Weise zu besetzen, wie es aus TeX bekannt ist. So bedeutet etwa "left center right left left" für das Attribut *columnalign*, daß der Inhalt der ersten Spalte linksbündig, der zweiten Spalte zentriert, der dritten Spalte rechtsbündig, usw. ausgerichtet werden.

Die beiden Attribute *rowspacing* und *columnspacing* bestimmen, wieviel Freiraum zwischen jeder Zeile und Spalte eingefügt werden soll. Dabei gibt das Attribut *framespacing* den Spezialfall der Randzellen an.

Damit die Art der Linien zwischen Zeilen und Spalten angeben werden kann, sind die beiden Attribute *rowlines* und *columnlines* vorhanden. Für die Gestaltung des Rahmens um eine gesamte Tabelle gibt es ein entsprechendes Attribut *frame*.

Um eine gleiche Zeilenhöhe bzw. eine gleiche Spaltenbreite zu erhalten, wurden die beiden Attribute *equalrows* und *equalcolumn* eingeführt.

Abschließend sei noch ein Beispiel angegeben, wie sich eine Matrix implementieren lassen könnte:

MathML	**Darstellung**

`<mfenced>`
`  <mtable>`
`  <mtr> <mn>1</mn> <mn>0</mn> <mn>0</mn> </mtr>`
`  <mtr> <mn>0</mn> <mn>1</mn> <mn>0</mn> </mtr>`
`  <mtr> <mn>0</mn> <mn>0</mn> <mn>1</mn> </mtr>`
`  </mtable>`
`</mfenced>`
	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Tabelle 4-24: `<mtable>`-Beispiel

4.6.2 <mtr>

Das `<mtr>`-Element repräsentiert eine komplette Zeile einer Tabelle und ist nur als direkter Unterausdruck von `<mtable>` erlaubt. Jedes Argument von `<mtr>` wird in einer anderen Spalte positioniert, wobei mit der weitesten links befindlichen Spalte begonnen wird.

Alle Zeilen, die weniger Spalten besitzen als andere Zeilen werden mit leeren `<mtd>`-Elementen entsprechend ausgefüllt. Die Anzahl der benötigten `<mtd>` ist abhängig von den Attributen rowspan und columnspan des `<mtd>`-Elementes.

Folgende Attribute sind zulässig:

Name	**Wert**	**Standardwert**
rowalign	top | bottom | center | baseline | axis	geerbt
columnalign	(left | center | right)+	geerbt

Tabelle 4-25: `<mtr>`-Attribute

Mit den beiden Attributen *rowalign* und *columnalign* wird es einer Zeile bzw. Spalte ermöglicht die entsprechend in `<mtable>` gesetzten Werte zu überschreiben. Wie schon bei `<mtable>` beschrieben, werden überflüssige Einträge ignoriert, und bei nicht ausreichender Anzahl an Einträgen der letzte Eintrag dupliziert.

4.6.3 <mtd>

Dieses Element repräsentiert die kleinste Einheit in einer Tabelle, nämlich eine Zelle. Das <mtd>-Element ist nur als direkter Unterausdruck des <mtr>-Elementes erlaubt. Es werden beliebig viele Argumente akzeptiert, wobei bei mehr als einem Argument eine <mrow>-Umgebung automatisch hinzugefügt wird.
Folgende Attribute sind für das <mtd>-Element erlaubt:

Name	Wert	Standardwert
rowspan	number	1
columnspan	number	1
rowalign	top | bottom | center | baseline | axis	geerbt
columnalign	left | center | right	geerbt

Tabelle 4-26: <mtd>-Attribute

Dabei können durch die Attribute *rowspan* und *columnspan* einzelne Zellen den Platz von mehreren Zeilen bzw. Spalten benutzen.
Durch die beiden Attribute *rowalign* und *columnalign* könne die geerbten Werte von <mtr> oder <mtable> speziell für diese eine Zelle verändert werden.

5 Document Type Definition

Aufgrund der Tatsache, daß MathML eine Anwendung von XML ist, wird die DTD auszugsweise vorgestellt, um den Zusammenhang zwischen MathML und XML darzustellen. Dabei wird anhand des <mrow>-Elementes der Aufbau erklärt. Zunächst aber das DTD-Fragment:

```
1     <!-- general attribute definitions for class & style & id & other >
2     <!-- : attributes shared by all mathml elements >
3     <!ENTITY % attglobalatts 'class CDATA #IMPLIED
4                               style CDATA #IMPLIED
5                               id    ID    #IMPLIED
6                               other CDATA #IMPLIED' >
7     ...
8     <!-- presentation token schemata with content>
9     <!ENTITY % ptoken "mi|mn|mo|mtext|ms" >
10    ...
11    <!-- empty presentation token schemata >
12    <!ENTITY % petoken  "mspace" >
13    ...
14    <!-- presentation layout schemata: script empty elements >
15    <!ENTITY % pscreschema "mprescripts|none" >
16    ...
17    <!-- presentation general layout schemata >
18    <!ENTITY % pgenschema "mrow|mfrac|msqrt|mroot|mstyle|merror|mpadded|
19                           mphantom|mfenced" >
20    <!ATTLIST mrow %attglobalatts; >
```

```
21    ...
22    <!-- presentation  layout schemata : scripts and limits >
23    <!ENTITY % pscrschema "msub|msup|msubsup|munder|mover|
24                           munderover|mmultiscripts" >
25    ...
26    <!-- presentation layout schemata: tables >
27    <!ENTITY % ptabschema "mtable|mtr|mtd" >
28    ...
29    <!ENTITY % plschema  "%pgenschema|%pscrschema|%ptabschema" >
30    ...
31    <!-- empty presentation layout schemata >
32    <!ENTITY % peschema "maligngroup|malignmark" >
33    ...
34    <!-- presentation action schemata >
35    <!ENTITY % pactions "maction" >
36    ...
37    <!-- Presentation entity all presentation constructs >
38    <!ENTITY % Presentation "%ptoken;|%petoken;|%pscreschema;|
39                             %plschema;|%peschema;|%pactions;">
40    <!ENTITY % ContInPres "ci|cn|apply|fn|lambda|reln|interval|list|
41                           matrix|matrixrow|set|vector" >
42    ...
43    <!-- recursive definition for content of expressions              >
44    <!-- include presentation tag constructs at lowest level         >
45    <!-- so presentation layout schemata hold presentation or Content >
46    <!ENTITY % PresExpression "(%Presentation;|%ContInPres;)* " >
47    ...
48    <!-- presentation layout schema contain tokens, >
49    <!-- layout and content schema                  >
50    <!ELEMENT mrow (%PresExpression;) >
```

Bei der folgenden Beschreibung wird vorausgesetzt, daß sich der Leser mit der XML-Notation auskennt.

In Zeile 50 erkennt man, daß das <mrow>-Element aus genau einer Entity %PresExpression; besteht. Dies wiederum kann aus einer der Entities, wie z.B. %Presentation; oder %ContInPres;, bestehen (siehe Zeile 46). Die Entity %Presentation; beinhaltet wiederum genau eine Entity, wie in Zeile 38 und 39 angegeben. Möglich sind beispielsweise &ptoken; oder %plschema;.

In der Entity &ptoken; sind elementare presentation elements, wie etwa <mi>, <mo> und <mn> enthalten (siehe Zeile 9). Die Entity %plschema; läßt sich noch weiter unterteilen in %ptabschema; zur Darstellung von Tabellen und Matrizen (siehe Zeile 27) oder in %pscrschema; zur Darstellung von Ausdrücken, die mit Indizes oder Akzenten behaftet sind (siehe Zeile 23 und 24). Als dritte Option ist die Entity %pgenschema; möglich, in der Elemente enthalten sind, die sich mit der generellen Darstellung von Elementen mit Inhalt (z.B. <mi>) beschäftigen (siehe Zeile 18 und 19). Hierunter fällt dann auch wieder <mrow>, so daß geschachtelte <mrow>-Umgebungen erzeugt wer-

den können. Es sind aber auch andere Elemente möglich, wie beispielsweise `<mfrac>`, `<mpadded>` oder `<mphantom>`.

In Zeile 20 sind dann noch die Attribute angegeben, die mit dem `<mrow>`-Element benutzt werden können. Hierbei ist die Entity `%attglobalatts;` angegeben, die wiederum vier mögliche Attribute enthält (siehe Zeile 3 bis 6). Hier werden jetzt die konkreten Attributnamen, die Attributtypen und die Standardwerte festgelegt.

6 Literatur

[PaMi 98] Patrick Ion & Robert Miner (Editoren), Stehpen Buswell, Stan Devitt, Angel Diaz, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt (Autoren): Mathematical Markup Language, W3C Proposed Recommendation, http://www.w3c.org/TR/PR-math, Version: 24.02.98

[Topp 98] Paul Topping: MathML Requirements, http://www.w3.org/Math/W3Cdocs/mathrequ.html, Version: 14.05.98

[BuSu 98] Stehpen Buswell, Robert S. Sutor (Editoren), Angel Diaz, Patrick Ion: Mathematical Markup Language (MathML) Frequently Asked Questions (FAQ), http://www.w3.org/Math/mathml-faq.html

[Math 97] The Math Forum: Math Typesetting for the Internet, http://forum.swarthmore.edu/typesetting/index.html, Version: 22.08.97

XML – Ein Anwendungsbeispiel

Mustafa Baydar & Markus Rupp

FB Informatik, Uni Dortmund
baydar00@marvin.cs.uni-dortmund.de & marupp00@marvin.cs.uni-dortmund.de

Zusammenfassung

In der vorliegenden Arbeit erläutern wir die Verwendung von der Extensible Markup Language (XML) am konkreten Beispiel unseres Seminars. Zunächst konzipieren wir eine Dokumenttyp-Definition (DTD) für die Klasse von Seminar-Dokumenten. Danach implementieren wir unser Seminar als konkretes Dokument konform zu der konzipierten DTD. Anschließend zeigen wir, daß die Extensible Stylesheet Language (XSL) die geeignetste Stilsprache für XML Dokumente ist. Es folgt eine kurze Einführung in XSL, da diese Sprache im übrigen Seminar nicht erklärt wurde. Abschließend geben wir noch verschiedene Sichten auf das implementierte XML-Dokument mit XSL an.

1 Entwicklung der Dokumenttyp-Definition

Die Dokumenttyp-Definition, im nachfolgenden kurz DTD genannt, ist das Herzstück eines jeden XML-Dokuments. In ihr wird die Grundstruktur für eine Klasse von Dokumenten festgelegt. Sie erfüllt zwei Grundlegende Aufgaben:

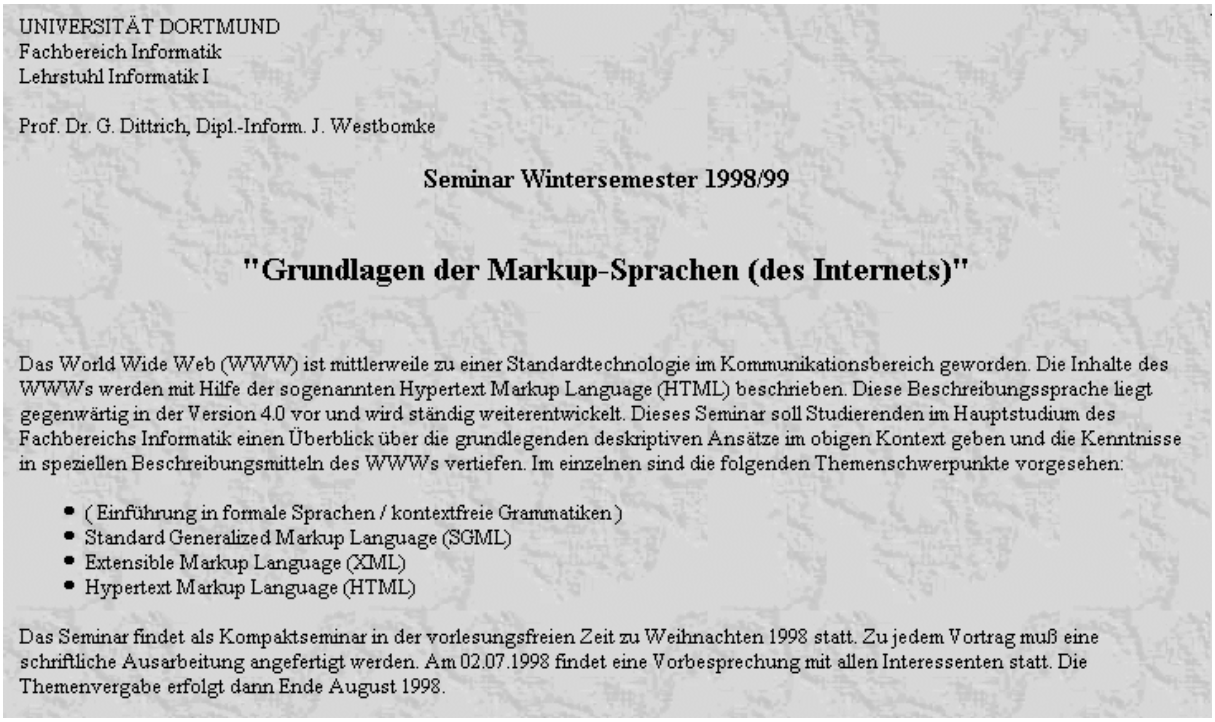
1. Die Verfasser von Dokumenten müssen ihre Dokumente konform zu der DTD erstellen.
2. Programmierer wissen welche Elemente (wie oft und in welcher Reihenfolge) ihre Anwendungen verarbeiten müssen.

Um eine konkrete DTD zu konzipieren ist es sehr wichtig sich an bereits „fertigen“ Dokumenten zu orientieren, da es später möglich sein soll, die fertigen Dokumente konform zur konzipierten DTD zu implementieren. Für unsere Beispiel DTD orientieren wir uns an der Ankündigung unseres Seminars, welche in folgenden Abbildung wiedergegeben ist.

Grundsätzlich gibt es für die Konzipierung einer DTD zwei Vorgehensweisen:

1. Top-Down: Ausgehend vom Wurzelelement (in unserem Fall Seminar) werden für jedes Element die Unterelemente identifiziert. Hierbei wird festgelegt, ob ein Element ein oder mehrmals vorkommen kann bzw. darf. Wird ein Element nicht mehr weiter in Unterelemente eingeteilt, so ist es atomar, und wir definieren den Elementtyp als (#PCDATA). Hierbei ist Suche nach dem „richtigen“ Abstraktionsgrad am schwierigsten, d.h. zu bestimmen, ob ein Element atomar ist oder aus weiteren Unterelementen besteht.
2. Bottom-Up: Zunächst werden alle atomaren Elemente identifiziert. Anschließend werden die Elemente immer weiter zu größeren Elementen zusammengefaßt, bis schließlich ein Element erhalten wird.

Wir werden mit Hilfe der ersten Methode die DTD für die Klasse von Seminaren konzipieren.



UNIVERSITÄT DORTMUND
Fachbereich Informatik
Lehrstuhl Informatik I

Prof. Dr. G. Dittrich, Dipl.-Inform. J. Westbomke

Seminar Wintersemester 1998/99

"Grundlagen der Markup-Sprachen (des Internets)"

Das World Wide Web (WWW) ist mittlerweile zu einer Standardtechnologie im Kommunikationsbereich geworden. Die Inhalte des WWWs werden mit Hilfe der sogenannten Hypertext Markup Language (HTML) beschrieben. Diese Beschreibungssprache liegt gegenwärtig in der Version 4.0 vor und wird ständig weiterentwickelt. Dieses Seminar soll Studierenden im Hauptstudium des Fachbereichs Informatik einen Überblick über die grundlegenden deskriptiven Ansätze im obigen Kontext geben und die Kenntnisse in speziellen Beschreibungsmitteln des WWWs vertiefen. Im einzelnen sind die folgenden Themenschwerpunkte vorgesehen:

- (Einführung in formale Sprachen / kontextfreie Grammatiken)
- Standard Generalized Markup Language (SGML)
- Extensible Markup Language (XML)
- Hypertext Markup Language (HTML)

Das Seminar findet als Kompaktseminar in der vorlesungsfreien Zeit zu Weihnachten 1998 statt. Zu jedem Vortrag muß eine schriftliche Ausarbeitung angefertigt werden. Am 02.07.1998 findet eine Vorbesprechung mit allen Interessenten statt. Die Themenvergabe erfolgt dann Ende August 1998.

Abbildung 1-1: Auszug aus der Seminarankündigung

1.1 Das Wurzelelement Seminar

Betrachten wir die Seminarankündigung, so lassen sich schon einige Elemente eines Seminars identifizieren. Bei diesen Elemente handelt es sich, von oben nach unten betrachtet, um einen Veranstalter, ein Semester, einen Titel und einer Kurzbeschreibung. Als weitere Elemente haben wir einen Zeitplan, und mindestens einen meistens mehrere Vorträge. Die Elemente Semester und Titel haben wir als atomar identifiziert. Hieraus ergibt sich die folgende Definition des Elements Seminar, wobei wir atomaren Unterelemente gleich mit definieren werden:

```
<!ELEMENT seminar (veranstalter, semester, titel, zeitplan, beschreibung,
                   vortrag+)>
<!ELEMENT semester (#PCDATA)>
<!ELEMENT titel (#PCDATA)>
```

Die Elemente Veranstalter, Zeitplan und Vortrag werden in den folgenden Abschnitten vollständig definiert.

1.2 Das Element Veranstalter

Der Veranstalter besteht, wie aus der Seminarankündigung ersichtlich ist, aus einer Universität, einem Fachbereich, einem Lehrstuhl und mindestens einem Leitenden. Bis auf das Element Leitender sind alle Elemente atomar, woraus sich folgende Definition des Elementtyps Veranstalter ergibt:

```
<!ELEMENT veranstalter        (uni, fachbereich, lehrstuhl, leitender+)>
<!ELEMENT uni                 (#PCDATA)>
<!ELEMENT fachbereich         (#PCDATA)>
<!ELEMENT lehrstuhl           (#PCDATA)>
```

1.3 Das Element Leitender

Ein Leitender besitzt, wie aus der Seminarankündigung ersichtlich ist, einen Namen. Dies erscheint uns jedoch ein wenig zu abstrakt. Wir haben uns daher entschlossen, den Namen in einen akademischen Titel und dem Namen zu unterteilen. Desweiteren ist es sicherlich wichtig die Telefonnummer oder Email-Adresse zur Kontaktaufnahme zu kennen. Die nun festgelegten Unterelemente akademischer Titel, Namen, Telefonnummer und Email des Elements Leitender sind atomar, woraus sich die folgende Elementtyp-Definition für das Element Leitender ergibt:

```
<!ELEMENT leitender          (akadtitel?, name, telefon*, email*)>
<!ELEMENT akadtitel          (#PCDATA)>
<!ELEMENT name               (#PCDATA)>
<!ELEMENT telefon            (#PCDATA)>
<!ELEMENT email              (#PCDATA)>
```

1.4 Das Element Zeitplan und das Element Slot

Zur Definition des Zeitplans haben wir folgendes Beispiel für einen Zeitplan konstruiert.

Montag (4.1.)	Dienstag (5.1.)	Mittwoch (6.1.)
8:30 Uhr – 8:45 Uhr Begrüßung / Orga	8:30 Uhr – 9:45 Uhr HTML II – Beschreibung der Änderung des Sprachumfangs der Versionen 3.2 – 4	8:30 Uhr – 9:45 Uhr Die Behandlung von Verweisen in XML
8:45 Uhr – 10:00 Uhr Formale Syntax-Beschreibungen	9:45 Uhr – 11:00 Uhr HTML – Eine Beispielanwendung	9:45 Uhr – 11:00 Uhr MathML zur Beschreibung von mathematischen Formeln
10:00 Uhr – 11:15 Uhr Der Grundgedanke der Auszeichnungssprachen	11:15 Uhr – 12:30 Uhr Die Stilkomponente von HTML	11:15 Uhr – 12:30 Uhr XML – Ein Anwendungsbeispiel Teil I
11:30 Uhr – 12:45 Uhr SGML – Teil I	13:30 Uhr – 14:45 Uhr Die Stilkomponente von SGML	13:30 Uhr – 14:45 Uhr XML – Ein Anwendungsbeispiel Teil II
13:45 Uhr – 15:00 Uhr SGML – Teil II	14:45 Uhr – 16:00 Uhr Einführung in XML	14:45 Uhr – 16:00 Uhr Bewertende Zusammenfassung
15:00 Uhr – 16:15 Uhr HTML I – Beschreibung des Sprachumfangs anhand von HTML 2		16:00 Uhr – 16:45 Uhr Abschlußdiskussion / Orga

Abbildung 1-2: Ein beispielhafter Zeitplan

Aus der Abbildung erkennen wir, daß ein Seminar mindestens einen Veranstaltungstag besitzt, an dem mindestens eine Veranstaltung (Slot) stattfindet. Jede Veranstaltung besitzt hierbei eine Anfangs- und Endzeit, sowie einen Titel. Aus diesen Vorüberlegungen ergibt sich dann unmittelbar die vollständige Definition des Elements Zeitplan:

```
<!ELEMENT zeitplan           (datum,slot+)+>
<!ELEMENT datum              (#PCDATA)>
<!ELEMENT slot               (uhrzeit,titel)>
<!ELEMENT uhrzeit            (#PCDATA)>
```

Das Elemente Titel wurde nicht mehr definiert, da es bereits schon im Abschnitt 1.1 definiert wurde.

1.5 Das Element Beschreibung und das Element Liste

Eine Kurzbeschreibung besteht, wie die Seminarankündigung zeigt, aus Textabschnitten und Listen. Diese Elemente können in beliebiger Anzahl und in verschiedenen Reihenfolgen auftreten. Eine Liste ihrerseits besteht wieder aus einzelnen Listenpunkten, die atomar sind. Ein Absatz ist sicherlich ebenfalls atomar. Hieraus ergibt sich sofort die vollständige Definition des Elements Beschreibung.

```
<!ELEMENT beschreibung          (absatz|liste)*>
<!ELEMENT absatz                (#PCDATA)>
<!ELEMENT liste                 (listenelement)+>
<!ELEMENT listenelement         (#PCDATA)>
```

Grundsätzlich kann das Element Beschreibung natürlich viel ausführlicher definiert werden. Sobald das W3C die Arbeit an der HTML-DTD in XML beendet hat, können wir dann beispielsweise den Elementtyp vom Element Beschreibung als HTML definieren.

1.6 Das Element Vortrag

Ein Vortrag hat grundsätzlich einen Titel, mindestens einen Referenten sowie eine Kurzbeschreibung. An dieser Stelle könnten wir auch noch sagen, daß jeder Vortrag Folien und eine Ausarbeitung besitzt, jedoch haben wir hiervon abgesehen, da die DTD durch den Elementtyp Folien zum einen zu komplex wird, und zum anderen die Ausarbeitung über einen Link-Mechanismus verankert werden müßte, das W3C jedoch die Arbeit an den Elementen XLink und XPointer noch nicht beendet hat, und somit im Augenblick kein Link-Mechanismus auf externe Dateien zur Verfügung steht. Insgesamt kommen wir zu folgender Definition für das Element Vortrag.

```
<!ELEMENT vortag                (titel,referent+,beschreibung)>
```

Der Elementtyp Titel wurde bereits in Abschnitt 1.1 und der Elementtyp Beschreibung im vorherigen Abschnitt definiert.

1.7 Das Element Referent

Das Element Referent besitzt ebenfalls, wie das Element Leitende einen Namen. Desweiteren besitzt er auch zur Kontaktaufnahme eine Telefonnummer oder Emailadresse. Da es aber auch vorkommen kann, daß ein Student eines Fachbereichs einen Titel in einem anderen Fachbereich besitzt, kann ein Referent ebenso wie ein Leitender einen akademischen Titel besitzen. Aus diesen Vorüberlegungen ergibt sich nun unmittelbar die Definition des Elementtyps Referent.

```
<!ELEMENT referent              (akadtitel?, name, telefon*, email*)>
```

Die Elemente akademischer Titel, Name, Telefon und Email wurden bereits im Abschnitt 1.3 definiert und müssen daher nicht nochmals definiert werden.

1.8 Das Parameter Entity Person

Was aufgefallen sein sollte, ist daß der Elementtyp Leitender und Referent die gleichen Unterelemente besitzen (vgl. Abschnitte 1.3 und 1.7). Damit wir nicht für jede neue Person diese Definition angeben müssen, definieren wir uns das Entity Person. Desweiteren bringt das Entity den Vorteil, daß falls wir einer Person beispielsweise noch eine Postadresse zuordnen wollen, wir dies nicht für jede einzelne Person tun müssen, sondern nur einmal in der Entity-Definition. Das Entity Person sowie die Elemente Leitender und Referent werden dann, wie folgt, definiert.

```
<!ENTITY  % person            "akadtitel?, name, telefon*, email">
<!ELEMENT leitender           (%person;)>
<!ELEMENT referent            (%person;)>
```

1.9 Einsatz der Attribute vom Typ ID und IDREF

Die meisten Veranstaltung des Zeitplans sind Vorträge. Da sich diese Veranstaltungen auf konkrete Vorträge beziehen, die im späteren XML-Dokument implementiert werden, werden wir einen Link-Mechanismus implementieren, um die Beziehung zwischen einer Veranstaltung aus dem Zeitplan mit einem konkreten zu ermöglichen. Da jeder Vortrag als Veranstaltung im Zeitplan vorkommen sollte, werden wir festlegen, daß jeder Vortrag auch ein eindeutiges Attribut vom Typ ID erhalten muß. Da andererseits nicht jede Veranstaltung ein Vortrag ist, werden wir festlegen, daß jede Veranstaltung eine Referenz durch ein Attribut vom Typ IDREF erhalten kann. Dies definieren wir in den Attributlisten.

```
<!ATTLIST slot
            vortrag IDREF #IMPLIED>
<!ATTLIST vortrag
            nummer ID #REQUIRED>
```

1.10 Der komplette Sourcecode der DTD (seminar.dtd)

```
<!ELEMENT seminar (veranstalter, semester, titel, zeitplan, beschreibung,
                   vortrag+)>
<!ELEMENT veranstalter        (uni, fachbereich, lehrstuhl, leitender+)>
<!ELEMENT uni                 (#PCDATA)>
<!ELEMENT fachbereich         (#PCDATA)>
<!ELEMENT lehrstuhl           (#PCDATA)>
<!ENTITY  % person "akadtitel?, name, telefon*, email*">
<!ELEMENT leitender           (%person;)>
<!ELEMENT akadtitel           (#PCDATA)>
<!ELEMENT name                (#PCDATA)>
<!ELEMENT telefon             (#PCDATA)>
<!ELEMENT email               (#PCDATA)>
<!ELEMENT semester            (#PCDATA)>
<!ELEMENT titel               (#PCDATA)>
<!ELEMENT zeitplan            (datum, slot+)+>
<!ELEMENT datum               (#PCDATA)>
<!ELEMENT slot                (uhrzeit, titel)>
<!ATTLIST slot
            vortrag           IDREF #IMPLIED>
<!ELEMENT uhrzeit             (#PCDATA)>
<!ELEMENT beschreibung        (absatz|liste)*>
```

```
<!ELEMENT absatz              (#PCDATA)>
<!ELEMENT liste               (listenelement+)>
<!ELEMENT listenelement       (#PCDATA)>
<!ELEMENT vortrag             (titel, referent+, beschreibung)>
<!ATTLIST vortrag
            nummer            ID #IMPLIED>
<!ELEMENT referent            (%person;)>
```

2 Das Seminar als XML-Dokument

Nachdem die DTD festgelegt wurde, kann nun das XML-Dokument implementiert werden. Hierbei binden wir die konzipierte DTD als externe DTD in unser Dokument ein.

2.1 Der komplette Sourcecode des Dokuments (seminar.xml)

```
<?xml version="1.0"?>
<!DOCTYPE seminar SYSTEM "seminar.dtd">

<seminar>

<veranstalter>
  <uni> Universitaet Dortmund </uni>
  <fachbereich> Fachbereich Informatik </fachbereich>
  <lehrstuhl> Lehrstuhl I </lehrstuhl>
  <leitender>
    <akadtitel> Prof. Dr. </akadtitel>
    <name> Gisbert Dittrich </name>
    <telefon> 0231 / 755 6444 </telefon>
    <email> dittrich@ls1.cs.uni-dortmund.de </email>
  </leitender>
  <leitender>
    <akadtitel> Dipl.-Inform. </akadtitel>
    <name> Joerg Westbomke </name>
    <telefon> 0231 / 755 6326 </telefon>
    <email> westbomke@ls1.cs.uni-dortmund.de </email>
  </leitender>
</veranstalter>

<semester> Wintersemester 1998/99 </semester>
<titel> Grundlagen der Markup-Sprachen (des Internets) </titel>

<zeitplan>
  <datum> Montag (4.1.) </datum>
  <slot>
    <uhrzeit> 8:30 Uhr </uhrzeit>
    <titel> Begruessung / Orga </titel>
  </slot>
  <slot vortrag="v01">
    <uhrzeit> 8:45 Uhr - 10:00 Uhr </uhrzeit>
    <titel> Formale Syntax-Beschreibungen </titel>
  </slot>
  <slot vortrag="v02">
    <uhrzeit> 10:00 Uhr - 11:15 Uhr </uhrzeit>
    <titel>
      Der grundlegende Gedanke der Auszeichungssprachen
    </titel>
  </slot>
  <slot vortrag="v03">
```

```
  <uhrzeit> 11:30 Uhr - 12:45 Uhr </uhrzeit>
  <titel> SGML - Teil I </titel>
</slot>
<slot vortrag="v03">
  <uhrzeit> 14:00 Uhr - 15:15 Uhr </uhrzeit>
  <titel> SGML - Teil II </titel>
</slot>
<slot vortrag="v04">
  <uhrzeit> 15:15 Uhr - 16:30 Uhr </uhrzeit>
  <titel>
    HTML I - Beschreibung des Sprachumfangs anhand von HTML 2
  </titel>
</slot>
<datum> Dienstag (5.1.) </datum>
<slot vortrag="v05">
  <uhrzeit> 8:30 Uhr - 9:45 Uhr </uhrzeit>
  <titel>
    HTML II - Beschreibung der Aenderung des Sprachumfangs der Versionen
    3.2 - 4
  </titel>
</slot>
<slot vortrag="v06">
  <uhrzeit> 9:45 Uhr - 11:00 Uhr </uhrzeit>
  <titel> HTML - Eine Beispielanwendung </titel>
</slot>
<slot vortrag="v07">
  <uhrzeit> 11:15 Uhr - 12:30 Uhr </uhrzeit>
  <titel> Die Stilkomponente von HTML </titel>
</slot>
<slot vortrag="v08">
  <uhrzeit> 13:30 Uhr - 14:45 Uhr </uhrzeit>
  <titel> Die Stilkomponente von SGML </titel>
</slot>
<slot vortrag="v09">
  <uhrzeit> 14:45 Uhr - 16:00 Uhr </uhrzeit>
  <titel> Einfuerung in XML </titel>
</slot>
<datum> Mittwoch (6.1.) </datum>
<slot vortrag="v10">
  <uhrzeit> 8:30 Uhr - 9:45 Uhr </uhrzeit>
  <titel> Die Behandlung von Verweisen in XML </titel>
</slot>
<slot vortrag="v11">
  <uhrzeit> 9:45 Uhr - 11:00 Uhr </uhrzeit>
  <titel>
    MathML zur Beschreibung von mathematischen Formeln
  </titel>
</slot>
<slot vortrag="v12">
  <uhrzeit> 11:15 Uhr - 12:30 Uhr </uhrzeit>
  <titel>
    XML - Ein Anwendungsbeispiel Teil I
  </titel>
</slot>
<slot vortrag="v12">
  <uhrzeit> 13:30 Uhr - 14:45 Uhr </uhrzeit>
  <titel>
    XML - Ein Anwendungsbeispiel Teil II
  </titel>
</slot>
```

```
  <slot vortrag="v13">
    <uhrzeit> 14:45 Uhr - 16:00 Uhr </uhrzeit>
    <titel> Bewertende Zusammenfassung </titel>
  </slot>
  <slot>
    <uhrzeit> 16:00 Uhr - 16:45 Uhr </uhrzeit>
    <titel> Abschlussdiskussion / Orga </titel>
  </slot>
</zeitplan>

<beschreibung>
<absatz>
Das World Wide Web (WWW) ist mittlerweile zu einer Standardtechnologie im
Kommunikationsbereich geworden. Die Inhalte des WWWs werden mit Hilfe der
sogenannten Hypertext Markup Language (HTML) beschrieben. Diese Beschrei-
bungssprache liegt gegenwaertig  in der Version 4.0 vor und wird staendig
weiterentwickelt. Dieses Seminar  soll Studierenden im Hauptstudium des
Fachbereichs Informatik einen Ueberblick ueber die grundlegenden deskripti-
ven Ansaetze im obigen Kontext geben  und die Kenntnisse in speziellen Be-
schreibungsmitteln des WWWs vertiefen. Im einzelnen sind die folgenden The-
menschwerpunkte vorgesehen:
</absatz>
<liste>
  <listenelement>
    ( Einfuehrung in formale Sprachen / kontextfreie Grammatiken )
  </listenelement>
  <listenelement>
    Standard Generalized Markup Language (SGML)
  </listenelement>
  <listenelement>
    Extensible Markup Language (XML)
  </listenelement>
  <listenelement>
    Hypertext Markup Language (HTML)
  </listenelement>
</liste>
</beschreibung>

<vortrag nummer="v01">
<titel> Formale Syntax-Beschreibungen </titel>
<referent>
  <name> Sabine Boehm </name>
  <telefon> 0234 / 286459 </telefon>
  <email> sabineboehm@compuserve.com </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v02">
<titel>
  Der grundlegende Gedanke der Auszeichungssprachen
</titel>
<referent>
  <name> Zobuepla </name>
  <telefon> 02305 / 23816 </telefon>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v03">
```

```
<titel> SGML </titel>
<referent>
  <name> Lars Brauckmann </name>
  <telefon> 02361 / 29315 </telefon>
  <email> brauck00@ls1.cs.uni-dortmund.de </email>
</referent>
<referent>
  <name> Jens Schroeder </name>
  <telefon> 02303 / 41712 </telefon>
  <email> schroe01@marvin.cs.uni-dortmund.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v04">
<titel>
  HTML I - Beschreibung des Sprachumfangs anhand von HTML 2
</titel>
<referent>
  <name> Anton Stoll </name>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v05">
<titel>
HTML II - Beschreibung der Aenderung des Sprachumfangs der Versionen
  3.2 - 4
</titel>
<referent>
  <name> Georg Conrads </name>
  <telefon> 0231 / 7275349 </telefon>
  <email> georg.conrads@ruhr-uni-bochum.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v06">
<titel> HTML - Eine Beispielanwendung </titel>
<referent>
  <name> Thorsten Bludau </name>
  <telefon> 02051 / 62847 </telefon>
  <email> tbludau@ispro.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v07">
<titel> Die Stilkomponente von HTML </titel>
<referent>
  <name> Frank Klein-Robbenhaar </name>
  <telefon> 0231 / 102672 </telefon>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v08">
<titel> Die Stilkomponente von SGML </titel>
<referent>
  <name> Markus Maerz </name>
```

```
  <telefon> 0201 / 753029 </telefon>
  <email> maerz@ls7.cs.uni-dortmund.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v09">
<titel> Einfuehrung in XML </titel>
<referent>
  <name> Markus Hoevener </name>
  <telefon> 02364 / 12836 </telefon>
  <email> webmaster@nathan.do </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v10">
<titel> Die Behandlung von Verweisen in XML </titel>
<referent>
  <name> Martin Stein </name>
  <telefon> 0231 / 7282492 </telefon>
  <email> ulrike.bitter@ruhr-uni-bochum.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v11">
<titel>
  MathML zur Beschreibung von mathematischen Formeln
</titel>
<referent>
  <name> Achim Streit </name>
  <telefon> 0231 / 756299 </telefon>
  <email> streit00@marvin.cs.uni-dortmund.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

<vortrag nummer="v12">
<titel> XML - Ein Anwendungsbeispiel </titel>
<referent>
  <name> Mustafa Baydar </name>
  <telefon> 0211 / 341570 </telefon>
  <email> baydar@uni-duesseldorf.de </email>
  <email> baydar00@marvin.cs.uni-dortmund.de </email>
</referent>
<referent>
  <name> Markus Rupp </name>
  <telefon> 0211 / 8114232 </telefon>
  <email> ruppmark@uni-duesseldorf.de </email>
  <email> marupp00@marvin.cs.uni-dortmund.de </email>
</referent>
<beschreibung> <absatz> Unser Seminar </absatz> </beschreibung>
</vortrag>

<vortrag nummer="v13">
<titel> Bewertende Zusammenfassung </titel>
<referent>
  <name> Rainer Dampmann </name>
  <telefon> 02304 / 68525 </telefon>
```

```
  <email> rainer.dampmann@fernuni-hagen.de </email>
</referent>
<beschreibung> <absatz></absatz> </beschreibung>
</vortrag>

</seminar>
```

3 Auswahl der Stilsprache

Da XML eine neue Sprache ist, stellt sich für den Benutzer die Frage, welche Stilsprache man benutzen soll, um XML-Dokumente auszugeben. Vor ca. einem halben Jahr vor Erstellung dieses Textes (August 1998, Erstellung des Textes Januar 1999) bestanden die Alternativen in CSS und DSSSL. Nun wurde im August 1998 dem W3C eine neue Sprache vorgeschlagen, die Extensible Stylesheet Language. Die Gründe, warum man es notwendig sah, eine neue Sprache zu entwickeln, sollen im Folgenden erklärt werden.

3.1 CSS als Stilsprache für XML

Hierzu sollte man sich zunächst klarmachen, wie mit CSS und XML Ausgaben generiert werden. Ein Browser geht sequentiell das XML-Dokument durch und gibt die Elemente entsprechend der CSS-Angaben aus. Aus der Tatsache, daß CSS für HTML entwickelt worden ist, einer (zukünftigen) DTD in XML, läßt sich schon erahnen, daß CSS in einigen Situationen ungeeignet für XML sein wird.

Die folgenden zwei Punkte sind Schwächen von CSS1, die in CSS2 nicht mehr existieren, aber sie werden erläutern, damit man sieht, welche Schwächen existiert haben:

- CSS1 kann kein Text erzeugen: In einigen Fällen kann es sein, daß man XML-Elemente durch erklärenden Text versehen möchte. Mit CSS1 ist dies nicht möglich.

  Beispiel:

  ```
  <person>
        <telnr>  01234/567890 </telnr>
        <faxnr> 01234/555555 </faxnr>
  </person>
  ```

  In diesem Beispiel würden nur die Rufnummer ausgegeben werden, ohne das dem Betrachter ersichtlich wäre, daß es sich um eine Telefonnummer und um eine Faxnummer handelt. Hier wäre es wünschenswert bei der Ausgabe die Strings ‘Tel-Nr.’ und ‘Fax-Nr.’ einzufügen.

  Natürlich kann man das Problem durch einfügen der Strings in das XML-Dokument lösen, aber bei längeren Dokumenten bedeutet dies, daß man sich ständig Gedanken über die Ausgabe machen müßte. Außerdem geht man dann davon aus, daß man das XML-Dokument ändern darf bzw. selbst erstellt hat. Wenn man nur die Aufgabe hat, ein Stylesheet anzugeben, steht man vor einem unlösbaren Problem.

- CSS1 kann keine Bilder erzeugen

Die folgenden zwei Punkte sind Schwächen von CSS2, die in zukünftigen Versionen ausgeräumt werden können:

- Teilunterstützung von Listen: In CSS2 ist es zwar möglich XML-Elemente als Listen zu kennzeichnen, aber es ist nicht möglich XML-Elemente als Listeneinträge zu kennzeichnen. Man muß zu diesem Zweck immer <li> benutzen.

  Beispiel:

  XML-Dokument:

  <Bücherliste>

  <Buch> XML in der Praxis </Buch>

  <Buch> Cascading Style Sheets </Buch>

  </Bücherliste>

  Es ist nicht möglich diese Liste auszugeben, da man <Buch> nicht als Listeneintrag definieren kann. Dagegen würde für

  <Bücherliste>

   XML in der Praxis

   Cascading Style Sheets

  </Bücherliste>

  folgende CSS-Definition genügen: Bücherliste { display: listitem }.

  Dies liegt daran, daß <LI> aus HTML+CSS dem Browser bekannt ist. Doch natürlich wäre die Benutzung von <LI> in XML-Dokumenten eine zu große Einschränkung.

- keine Unterstützung von Tabellen: Während Listen nur teilweise unterstützt werden, gibt es in CSS2 keine Elemente zu Generierung von Tabellen.

Der nachfolgende Punkt ist dagegen eine prinzipielle Schwäche von CSS, d.h. es wird wohl noch in zukünftigen Versionen existieren:

- Ein XML-Dokument muß mit CSS sequentiell abgearbeitet werden - ein Browser geht also das XML-Dokument sequentiell durch und beim Auftreffen auf ein Element wird dieser entsprechend der CSS-Angabe ausgegeben. Die Ausgabe erfolgt unmittelbar, es gibt keine Möglichkeit sich an nachfolgenden Elementen zu orientieren. Dies bedeutet, daß man ein XML-Dokument (und auch die DTD) im Hinblick auf die Ausgabe entwerfen müßte, da die Einträge im XML-Dokument in der Reihenfolge in der sie im Dokument stehen, ausgeben werden.

  Beispiel:

  <!ELEMENT personen (student | wimi | professor)*>

  <personen>

  <student> Anton Stoll </student>

  <wimi> Jörg Westbomke </wimi>

<student> Georg Conrads </student>

<student> Thorsten Bludau </student>

<professor> Gisbert Dittrich </professor>

<student> Sabine Böhm </student>

</personen>

In diesem Beispiel ist es mit CSS nicht möglich die Ausgabe zu sortieren, d.h. die Professoren vor den wissenschaftlichen Mitarbeitern und diese vor den Studenten auszugeben. Auch ist es nicht möglich die Personen alphabetisch zu sortieren.

Fazit: CSS ist für den Einsatz mit XML praktisch ungeeignet. Bei größeren DTDs wird es auch ziemlich schwierig sein, das XML-Dokument im Hinblick auf die Ausgabe zu entwerfen. Wenn die DTD simpel und kurz ist, ist CSS anwendbar. Allerdings ist bereits die DTD für unser Seminar (siehe Teil 1) für CSS zu komplex.

3.2 XSL als Stilsprache für XML

XSL wird vom W3C für XML entwickelt und gefördert. Es ist noch in der Entwicklung und die im folgenden beschriebenen Eigenschaften sowie der Sprachumfang können sich ändern.

Im Grunde besteht XSL aus zwei Teilen: Erstens enthält es Sprachelemente zur Konvertierung von XML-Dokumenten und zweitens Sprachelemente zur Formatierung von XML-Dokumenten. Zur Zeit ist XSL darauf ausgelegt, XML-Dokumente in HTML-Dokumente umzuformen, aber später soll es auch möglich sein, kompliziertere Dokumente (z.B.: Word) zu generieren.

Eine wichtige Eigenschaft von XSL ist die nicht sequentielle Abarbeitung. Im XSL-Dokument stehen die Angaben, welche Elemente im XML-Dokument in welcher Art und Weise ausgegeben werden sollen. Dies bedeutet auch, daß die Reihenfolge und die Häufigkeit der XML-Elemente bei der Ausgabe beeinflußt werden können. Das XSL-Dokument wird auch nicht sequentiell abgearbeitet. Es ist in Regeln (Templates) gegliedert und diese Regeln können sich gegenseitig aufrufen.

Weiterhin handelt es sich bei XSL-Dokumenten eigentlich um XML-Dokumente, d.h. es wird demnächst auch eine DTD für XSL existieren.

Außerdem soll XSL nicht-visuelle Ausgabemedien unterstützen und eine Programmiersprache (EcmaScript) enthalten.

3.3 DSSSL als Stilsprache für XML

Das W3C will DSSSL so normieren, daß es eine Obermenge zu XML wird. Zu Zeit gibt es nur eine große Schnittmenge zwischen den beiden Sprachen. Da DSSSL eine Obermenge von XML sein wird, heißt es, daß mit DSSSL alles möglich ist, was auch mit XML möglich ist. Theoretisch gibt es also keinerlei Einschränkungen im Einsatz von DSSSL mit XML. Praktisch gesehen, enthält der Sprachumfang von DSSSL unnötige Elemente, die im Gebrauch mit XML nicht notwendig sind.

Für den Benutzer bedeutet DSSSL+XML auch, daß er zwei verschiedene Syntaxregeln lernen muß.

4 Einführung in Extensible Stylesheet Language (XSL)

Eingeführt wird hier in die Extensible Stylesheet Language, die in dem Microsoft Internet Explorer5.0beta implementiert ist. Es sollte beachtet werden, daß diese XSL-Version sich von der, im W3C Working Draft (vom August 1998) beschriebenen XSL Version, in einigen Punkten unterscheidet. So heißt 'process-children' beim IE5beta 'apply-templates'.

Im folgenden werden wir uns hauptsächlich mit der Generierung nach HTML/CSS beschäftigen, da die Formatierungselemente von XSL unzureichend dokumentiert sind. Konvertierung bzw. Formatierung (d.h. es gibt keinen prinzipiellen Unterschied) von XML-Dokumenten mit XSL geschieht in zwei Schritten. Zunächst generiert der Browser einen Baum, der der Struktur des XML-Dokumentes entspricht. Mit Hilfe dieses Baumes kann dann der Benutzer das Zieldokument generieren bzw. das XML-Dokument formatieren.

Beispiel-Dokument:

```
<folien>
        <titel> 1. Aufbereitung eines XML-Dokumentes </titel>
        <auflistung   typ="unnummeriert">
                <eintrag> Wurzel: ... </eintrag>
                <eintrag> Knoten: ... </eintrag>
        </auflistung>
</folien>
```

Aus dem obigen Dokument wird der folgende Baum generiert:

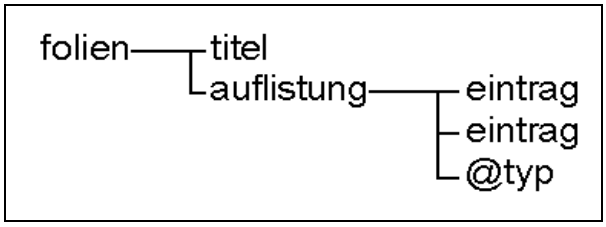


Abbildung 4-1

Das "@"-Symbol wird Attributen vorangestellt, um diese kenntlich zu machen. Es entspricht der MS-IExplorer 5.0beta-Syntax, aber es ist äußert zweifelhaft, daß "@" auch in der endgültigen Beschreibung von XSL verwendet wird.

4.1 Generierung des Zieldokumentes

Unter Generierung verstehen wir die Formatierung bzw. die Konvertierung eines XML-Dokumentes. Bei der Generierung eines Zieldokumentes wird immer ein Knoten des Baumes zum aktuellen Knoten

gewählt und von diesem Knoten aus wird dann weitergearbeitet. Dazu benötigt man Filteroperationen/Suchoperationen:

select="."	Wählt den aktuellen Knoten
select="/"	Wählt den Wurzelknoten
select="*name*"	Wählt einen Nachfolgerelement mit Namen *name*
select="@*name*"	Wählt einen Nachfolgerattribut mit Namen *name*
select="*"	Wählt einen Nachfolgerelement mit beliebigem Namen
select="@*"	Wählt einen Nachfolgerattribut mit beliebigem Namen
select="ancestor(x)"	Wählt den Vorgängerknoten des Knotens x
select="x//name"	Wählt den Knoten mit dem Namen *name,* der von dem Knoten x aus erreichbar ist (mit Tiefensuche). //name sucht dabei von der Wurzel aus .//name sucht vom aktuellen Knoten aus

Man kann Suchangaben mit Eigenschaften versehen, die in eckigen Klammern stehen. Generell: a[b] wählt den Knoten a, der die Eigenschaft b hat, d.h. Knoten a wird der aktuelle Knoten. b kann wieder aus Suchangaben bestehen.

Beispiele:

referent[email]	Referent mit Email-Knoten als Nachfolger
referent[email = "y@z.de"]	Referent mit der Emailadresse y@z.de
slot[@vortrag]	Slot mit Attribut-Knoten als Nachfolger
slot[@vortrag = "v01"]	Slot mit Attributwert v01
slot[0] slot[index() = 0]	Erstes Auftauchen von Slot
slot[end()] slot[index() = end()]	Letztes Auftauchen von Slot
slot[*x*] slot[index() = *x*]	*x*-tes Auftauchen von Slot

4.2 XSL-Elemente

Die folgenden Beispiele, an denen die XSL-Elemente erläutern werden, beziehen sich auf das XML-Seminar-Dokument (siehe ersten Teil).

- <xsl:value-of select="...">
  Liefert den Inhalt des angegebenen Knotens.

```
<BODY>
    <h1> <xsl:value-of select="//titel" /> </h1>
</BODY>
```

Dies generiert:

```
<BODY>
        <h1>  Grundlagen der Markup-Sprachen (des Internets)  </h1>
</BODY>
```

- <xsl:for-each select="..."> ... </xsl:for-each>

  Schleifenkonstrukt zum Anwenden von Operationen auf Knoten desselben Types.

```
<BODY>
        <xsl:for-each select="//titel">
                <p> <xsl:value-of select= "." />  </p>
        </xsl:for-each>
</BODY>
```

Dies generiert:

```
<BODY>
        <p> Grundlagen der Markup-Sprachen (des Internets)  </p>
        <p> Begruessung / Orga  </p>
        <p> Formale Syntax-Beschreibungen </p>
        <p> Der grundlegende Gedanke der Auszeichnungssprachen </p>
        <p> SGML - Teil I </p>

        ...  Beispiel verkürzt.

</BODY>
```

- <xsl:choose select="..."><xsl:when select="..."><xsl:otherwise>

  Bedingte Ausführung:

  'choose' wählt nacheinander Knoten des angegebenen Typs.

  Mit 'when' kann man den Typ des Knoten überprüfen, der mit 'choose' ausgewählt wurde. Es können mehrere 'when'-Abschnitte definiert werden. Der optionale 'otherwise'-Abschnitt wird nur dann ausgeführt, wenn keine der 'when'-Abschnitte ausgeführt wurde.

  Für das nächste Beispiel befinden wir uns im Zeitplan-Element im Seminar-Dokument:

```
<xsl:choose select="*">
        <xsl:when select="Datum">
```

```
                Datum:<xsl:value-of select= “.“ />
            </xsl:when>
            <xsl:when select="Slot[@vortrag]">
                Slot:<xsl:value-of select= “.“ />
            </xsl:when>
            <xsl:otherwise>
                Sonstiger Slot:<xsl:value-of select= “.“ />
            </xsl:otherwise>
        </xsl:choose>
```

Dies generiert:

Datum: Montag (4.1.)
Sonstiger Slot: 8:30 Uhr Begruessung / Orga
Slot: 8:45 Uhr - 10:00 Uhr Formale Syntax-Beschreibungen
Slot: 10:00 Uhr - 11:15 Uhr Der grundlegende Gedanke der Auszeichungssprachen
Slot: 11:30 Uhr - 12:45 Uhr SGML - Teil I
Slot: 14:00 Uhr - 15:15 Uhr SGML - Teil II
Slot: 15:15 Uhr - 16:30 Uhr HTML I - Beschreibung des Sprachumfangs anhand von HTML 2
Datum: Dienstag (5.1.)
Slot: 8:30 Uhr - 9:45 Uhr HTML II - Beschreibung der Aenderung des ...
Beispiel verkürzt

- <xsl:element name=“...“> <xsl:attribute name=“...“>

  Erzeugt Elemente/Attribute

```
<xsl:element name=“a“>
    <xsl:attribute name=“href“>
        seminar.html
    </xsl:attribute>
    Unser Seminar
</xsl:element>
```

Dies generiert: Unser Seminar

4.3 Aufbau eines XSL-Dokumentes

XSL ist eine DTD in XML. Ein XSL-Dokument besteht aus einer oder mehreren Regeln, die Templates genannt werden (zumindest in der MS-IExplorer5beta-Syntax).

Beispiel:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/"> <xsl:apply-templates select=“seminar“>  </xsl:template>
        <xsl:template match="seminar“> <xsl:apply-templates select=“veranstalter“> </xsl:template>
        <xsl:template match="veranstalter“> ... </xsl:template>
        ...
</xsl:stylesheet>
```

Es existiert eine Start-Regel, die mit ‘ match=“/“ ’ formuliert wird. Im obigen Beispiel wird innerhalb der Startregel das Element “seminar“ zum aktuellen Element erklärt. Daher wird die zweite Regel aktiv, die dann ihrerseits das Veranstalter-Element zum aktuellen Element erklärt.

Beispiel für Konvertierung:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/">
                <html>
                        <head>  <style> ... </style>  </head>
                        <body>
                                <xsl:apply-templates select="*"/>
                        </body>
                </html>
        </xsl:template>
        <xsl:template match="seminar">
                <p>  <xsl:value-of select="."/> </p>
        </xsl:template>
</xsl:stylesheet>
```

Ergebnis:

```
        <html>
                <head> <style> ... </style> </head>
                <body>
```

```
                <p> Alle Strings im Seminar-Dokument </p>
            </body>
        </html>
```

Kommt man nur mit einer Regel aus, kann man auch direkt das Format des Zieldokuments angeben:

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <head>
            <style> ... </style>
        </head>
        <body>
            <p> <xsl:value-of select="seminar" /> </p>
        </body>
</HTML>
```

Das Ergebnis ist wie oben.

Wenn man XSL-Stylesheets benutzen will, muß man diese in die Stellen einfügen, wo in den vorherigen Beispielen die HTML/CSS Angaben stehen. Also:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
            xmlns:fo="http://www.w3.ord/TR/WD-xsl/FO"
            result-ns="fo"
            indent-result="yes">

        <xsl:template match="/">
            <fo:page-sequence font-family="serif">
                <fo:simple-page-master name="scrolling"/>
                <fo:queue queue-name="body">
                    <xsl:apply-templates select="*"/>
                </fo:queue>
            </fo:page-sequence>
        </xsl:template>

        <xsl:template match="doc">
            <fo:block font-weight="bold">
                <xsl:apply-templates select="*"/>
```

```
            </fo:block>
        </xsl:template>
        ...
<xsl:stylesheet>
```

5 Stilkomponenten für die Seminar-DTD

Die kompletten Source-Texte der Stilkomponenten befinden sich im Anhang. Hier werden nur die schwer verständlichen Teile erläutert.

5.1 CSS

Wenn man mit CSS XML-Dokumente ausgeben will, dann muß man sich eigentlich nur überlegen, welche Elemente ausgegeben werden sollen und welche nicht. Die Elemente, die nicht ausgegeben werden, können einfach mit {display: none; } ausgeblendet werden.

5.1.1 Zeitplan in CSS

Wenn man sich den Zeitplan, mit der im Anhang angegebenen Stilkomponenten, ausgeben läßt, so wird man feststellen, daß die Ausgabe nicht korrekt ist. Das liegt daran, daß wir in der Definition der DTD reguläre Ausdrücke benutzt haben:

```
<!ELEMENT zeitplan          (datum, slot+)+>
```

Der Browser erkennt, daß das Element ’zeitplan’ ‘datum’ und ‘slot’ Elemente enthält. Da aber Tage nicht als solche gekennzeichnet sind, kann man dem Browser mit CSS nicht mitteilen, daß ein Datum mit mehreren Slots eine Logische Einheit bildet.

Um dieses Problem zu lösen, hätte man folgendes definieren müssen:

```
<!ELEMENT zeitplan          (tag)+>
<!ELEMENT tag               (datum, slot+)>
```

Dies bedeutet natürlich wieder, daß man sich Gedanken über die Ausgabe, während der Erstellung der DTD machen muß.

5.2 XSL

5.2.1 Einführungs-Ansicht in XSL

Hier sollen die Leitenden eines Seminars nacheinander, durch Kommata getrennt, ausgegeben werden. Dazu werden die ‘leitender’-Elemente zum aktuellen Element gewählt und immer überprüft, ob es nicht der letzte ‘leitender’-Element ist. Da der letzte ‘leitender’-Element vom aktuellen ‘leitender’-Element nicht erreichbar ist, muß man es mit “ancestor(.)/leitender[end()]“ filtern:

```
<xsl:for-each select="leitender[. != ancestor(.)/leitender[end()]]"><SPAN CLASS="leitender">
        <SPAN CLASS="akadtitel"><xsl:value-of select="akadtitel"/> </SPAN>
        <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
         ,
```

```
</SPAN></xsl:for-each>
```

Der letzte ‘leitender’-Element wird mit "leitender[end()]" gefiltert:

```
<xsl:for-each select="leitender[end()]"><SPAN CLASS="leitender">
        <SPAN CLASS="akadtitel"><xsl:value-of select="akadtitel"/> </SPAN>
        <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
</SPAN></xsl:for-each>
```

5.2.2 Zeitplan in XSL

Mit der aktuellen Version von XSL ist es auch nicht (direkt) möglich (datum, slot+) als eine Einheit zu identifizieren. Man müßte einen regulären Ausdruck als Suchangabe benutzen, wie etwa “select=’(datum,slot+)’ “. Damit würden aber auch mehrere Elemente als aktuelles Element gewählt werden - zur Zeit ist nur ein Element möglich.

Dieses Problem wird hier durch einen Trick gelöst: Beim Auftreffen auf ein ‘datum’-Element wird ein schließendes und öffnendes <table>-Tag eingefügt. Und damit die Klammerung stimmt, wird der Zeitplan mit <table> und </table> geklammert.

Beispiel:

```
                <table>
datum                   </table><table>
slot
slot
datum                   </table><table>
slot
slot
slot
datum                   </table><table>
slot
                </table>
```

Wie man hier sieht, enthält der Zeitplan drei Datumselemente. Es werden vier Tabellen erzeugt, wobei die erste leer ist.

6 Stilkomponenten zum Seminar-Dokument

6.1 CSS-Stilkomponenten

6.1.1 Einführungs-Ansicht

```
uni             {text-transform: uppercase; display: block; }
fachbereich     {display: block; }
lehrstuhl       {margin-bottom: 1em; display: block; }
leitender       {margin-bottom: 1em; display: block; }
telefon         {display: none; }
email           {display: none; }
semester        {margin-bottom: 1em; display: block; font-weight: bold; text-align: center; }
titel           {margin-bottom: 1em; font-size: large; font-weight: bold; display: block;
                text-align: center; }
zeitplan        {display: none; }
beschreibung    {margin-bottom: 1em; display: block;}
absatz          {text-indent: 2em; text-align: justify; display: block; }
liste           {display: block; }
listenelement   {margin-left: 1em; display: block; }
vortrag         {display: none; }
```

6.1.2 Inhaltansicht

```
uni             {text-transform: uppercase; display: block; }
fachbereich     {display: block; }
lehrstuhl       {margin-bottom: 1em; display: block; }
leitender       {display: none; }
semester        {margin-bottom: 1em; display: block; font-weight: bold; text-align: center; }
titel           {margin-bottom: 1em; font-size: large; font-weight: bold; display: block;
                text-align: center; }
zeitplan        {display: none; }
datum           {display: block; border: thin solid black;
                border-bottom-width: medium; width:33.32%; top center;}
slot            {height: 5em; display: block; border: thin solid black; width: 33.32%}
slot titel      {margin-bottom: 0em; font-size: medium; font-weight: normal;
```

```
                text-align: left; }
beschreibung    {margin-bottom: 1em; display: block;}
absatz          {padding: 1ex 1ex 1ex 1ex; text-indent: 2em; text-align: justify;
                display: block; }
liste           {display: block; }
listenelement   {margin-left: 1em; display: block; }
vortrag         {border: thin solid black; width: 99%}
vortrag titel   {margin-bottom: 1em; font-size: medium; font-weight: normal;
                display: block;  text-align: center; }
referent        {display: none; }
```

6.1.3 Personen-Ansicht

```
seminar         {background-color: rgb(255,255,180); display:block; }
veranstalter    {display: block; }
uni             {display: none; }
fachbereich     {display: none; }
lehrstuhl       {display: none; }
semester        {display: none; }
titel           {display: none; }
zeitplan        {display: none; }
beschreibung    {display: none; }
leitender       {display: block; margin-bottom: 1em;
                background-color: rgb(255,220,150); }
referent        {display: block; margin-bottom: 1em;
                background-color: rgb(255,220,150); }
telefon         {display: block; }
email           {display: block; }
vortrag         {display: block; }
```

6.1.4 Titel-Ansicht

```
seminar         {display: block; }
uni             {text-align: center; margin-bottom: 1em; text-transform: uppercase;
                display: block; }
fachbereich     {text-align: center; margin-bottom: 1em; display: block; }
```

```
lehrstuhl       {display: none; }
leitender       {display: none; }
semester        {margin-bottom: 1em; display: block; font-weight: bold; text-align: center; }
titel           {margin-bottom: 1em; display: block; font-size: large; font-weight: bold;
                text-align: center; }
zeitplan        {display: none; }
beschreibung    {display: none; }
vortrag         {display: none; }
```

6.1.5 Zeitplan-Ansicht

```
uni             {display: none; }
veranstalter    {display: none; }
semester        {display: none; }
titel           {display: none; }
zeitplan        {border: medium solid black; width:100%; }
datum           {display: block; border: thin solid black;
                border-bottom-width: medium; width:33.32%; }
slot            {height: 5em; display: block; border: thin solid black; width: 33.32%; }
slot titel      {display: block; }
beschreibung    {display: none; }
vortrag         {display: none; }
```

6.2 XSL-Stilkomponenten

6.2.1 Einführungs-Ansicht

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <style>
    .uni            {text-transform: uppercase; display: block; }
    .fachbereich    {display: block; }
    .lehrstuhl      {margin-bottom: 1em; display: block; }
    .semester       {margin-top: 1em; display: block; font-weight: bold;
                    text-align: center; }
```

```
      .titel                {margin-bottom: 1em; font-size: large; font-weight: bold;
                            display: block; text-align: center; }
      .beschreibung         {margin-bottom: 1em; display: block;}
      .absatz               {text-indent: 2em; text-align: justify; display: block; }
   </style>

<BODY>
<xsl:for-each select="seminar"><SPAN CLASS="seminar">

<xsl:for-each select="veranstalter"><SPAN CLASS="veranstalter">
        <SPAN CLASS="uni"> <xsl:value-of select="uni"/> </SPAN>
        <SPAN CLASS="fachbereich"> <xsl:value-of select="fachbereich"/></SPAN>
        <SPAN CLASS="lehrstuhl"> <xsl:value-of select="lehrstuhl"/></SPAN>
        <xsl:for-each select="leitender[. != ancestor(.)/leitender[end()]]">
           <SPAN CLASS="leitender">
                <SPAN CLASS="akadtitel"><xsl:value-of select="akadtitel"/> </SPAN>
                <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
                ,
            </SPAN>
        </xsl:for-each>
        <xsl:for-each select="leitender[end()]"><SPAN CLASS="leitender">
                <SPAN CLASS="akadtitel"><xsl:value-of select="akadtitel"/> </SPAN>
                <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
        </SPAN></xsl:for-each>
</SPAN></xsl:for-each>

<xsl:for-each select="semester"><SPAN CLASS="semester">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>
<xsl:for-each select="titel"><SPAN CLASS="titel">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>

<xsl:for-each select="beschreibung"><SPAN CLASS="beschreibung">
        <xsl:for-each select="absatz"><SPAN CLASS="absatz">
                <xsl:value-of select="."/>
        </SPAN></xsl:for-each>
```

```
        <xsl:for-each select="liste"> <ul>
                <xsl:for-each select="listenelement">
                        <li> <xsl:value-of select="."/> </li>
                </xsl:for-each>
        </ul></xsl:for-each>
</SPAN></xsl:for-each>

</SPAN></xsl:for-each>
</BODY>
</HTML>
```

6.2.2 Inhaltansicht

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <style>
      .uni              {text-transform: uppercase; display: block; }
      .fachbereich      {display: block; }
      .lehrstuhl        {margin-bottom: 1em; display: block; }
      .semester         {margin-bottom: 1em; display: block; font-weight: bold;
                        text-align: center; }
      .titel            {margin-top: 1em; margin-bottom: 1em; font-size: large;
                        font-weight: bold; display: block; text-align: center; }
      .beschreibung     {margin-bottom: 1em; display: block;}
      .absatz           {padding: 1ex 1ex 1ex 1ex; text-indent: 2em; text-align: justify;
                        display: block; }
      .vortrag .titel   {margin-top: 0em; margin-bottom: 1em; font-size: medium;
                        font-weight: normal; display: block; text-align: center; }
   </style>

<BODY>
<xsl:for-each select="seminar"><SPAN CLASS="seminar">

<xsl:for-each select="veranstalter"><SPAN CLASS="veranstalter">
        <SPAN CLASS="uni"> <xsl:value-of select="uni"/> </SPAN>
        <SPAN CLASS="fachbereich"> <xsl:value-of select="fachbereich"/></SPAN>
        <SPAN CLASS="lehrstuhl"> <xsl:value-of select="lehrstuhl"/></SPAN>
```

```
</SPAN></xsl:for-each>

<xsl:for-each select="semester"><SPAN CLASS="semester">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>
<xsl:for-each select="titel"><SPAN CLASS="titel">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>

<xsl:for-each select="vortrag">
        <a>
                <xsl:attribute name="href">
                        #<xsl:value-of select="@nummer"/>
                </xsl:attribute>
                <xsl:attribute name="target">
                        oben
                </xsl:attribute>
                <div style="font-weight: bold; margin-left: 1em; margin-bottom: 1ex; ">
                        <xsl:value-of select="titel"/>
                </div>
        </a>
</xsl:for-each>

<hr></hr>

<xsl:for-each select="titel"><SPAN CLASS="titel">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>

<xsl:for-each select="beschreibung"><SPAN CLASS="beschreibung">
        <xsl:for-each select="absatz"><SPAN CLASS="absatz">
                <xsl:value-of select="."/>
        </SPAN></xsl:for-each>
        <xsl:for-each select="liste"> <ul>
                <xsl:for-each select="listenelement">
                        <li>  <xsl:value-of select="."/>  </li>
                </xsl:for-each>
```

```
        </ul></xsl:for-each>
</SPAN></xsl:for-each>

<xsl:for-each select="vortrag">
        <hr style="width: 90%"></hr>
        <DIV class="vortrag">
                <a>
                        <xsl:attribute name="name">
                                <xsl:value-of select="@nummer"/>
                        </xsl:attribute>
                        <SPAN CLASS="titel"> <xsl:value-of select="titel"/> </SPAN>
                </a>
                <xsl:for-each select="beschreibung"><SPAN CLASS="beschreibung">
                        <xsl:for-each select="absatz"><SPAN CLASS="absatz">
                                <xsl:value-of select="."/>
                        </SPAN></xsl:for-each>
                        <xsl:for-each select="liste"> <ul>
                                <xsl:for-each select="listenelement">
                                        <li>  <xsl:value-of select="."/>   </li>
                                </xsl:for-each>
                        </ul></xsl:for-each>
                </SPAN></xsl:for-each>
        </DIV>
</xsl:for-each>

</SPAN></xsl:for-each>
</BODY>
</HTML>
```

6.2.3 Personen-Ansicht

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <style>
      .seminar          {background-color: rgb(255,255,180); display:block; }
      .leitender        {display: block; margin-bottom: 1em; background-color: rgb(255,220,150); }
      .referent         {display: block; margin-bottom: 1em; background-color: rgb(255,220,150); }
```

```
      .telefon          {display: block; }
      .email            {display: block; }
      .vortrag          {display: block; margin-bottom: 1em; }
   </style>

<BODY>
<xsl:for-each select="seminar"><SPAN CLASS="seminar">

<DIV style="text-align: center; font-size: large; margin-bottom: 1ex;">
   Personenverzeichnis
</DIV>

<xsl:for-each select="veranstalter"><SPAN CLASS="veranstalter">
        <xsl:for-each select="leitender"><SPAN CLASS="leitender">
                <SPAN CLASS="akadtitel"><xsl:value-of select="akadtitel"/> </SPAN>
                <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
                <xsl:for-each select="telefon"><SPAN CLASS="telefon">
                        Tel-Nr.: <xsl:value-of select="."/>
                </SPAN></xsl:for-each>
                <xsl:for-each select="email"><SPAN CLASS="email">
                        EMail:  <xsl:value-of select="."/>
                </SPAN></xsl:for-each>
        </SPAN></xsl:for-each>
</SPAN></xsl:for-each>

<xsl:for-each select="vortrag"> <SPAN CLASS="vortrag">
        <xsl:for-each select="referent"><DIV CLASS="referent">
                <SPAN CLASS="akadtitel"> <xsl:value-of select="akadtitel"/> </SPAN>
                <SPAN CLASS="name"> <xsl:value-of select="name"/> </SPAN>
                (Thema: <xsl:value-of select="ancestor(.)/titel"/>)
                <xsl:for-each select="telefon"><SPAN CLASS="telefon">
                        Tel-Nr.: <xsl:value-of select="."/>
                </SPAN></xsl:for-each>
                <xsl:for-each select="email"><SPAN CLASS="email">
                        EMail:  <xsl:value-of select="."/>
                </SPAN></xsl:for-each>
        </DIV></xsl:for-each>
```

```
</SPAN></xsl:for-each>

</SPAN></xsl:for-each>
</BODY>
</HTML>
```

6.2.4 Titel-Ansicht

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <style>
      .uni              {text-align: center; margin-bottom: 1em;
                         text-transform: uppercase; display: block; }
      .fachbereich      {text-align: center; margin-bottom: 1em; display: block; }
      .semester         {margin-bottom: 1em; display: block;
                         font-weight: bold; text-align: center; }
      .titel            {margin-bottom: 1em; display: block;
                         font-size: large; font-weight: bold; text-align: center; }
   </style>

<BODY>
<xsl:for-each select="seminar"><SPAN CLASS="seminar">

<xsl:for-each select="veranstalter"><SPAN CLASS="veranstalter">
        <SPAN CLASS="uni"> <xsl:value-of select="uni"/> </SPAN>
        <SPAN CLASS="fachbereich"> <xsl:value-of select="fachbereich"/></SPAN>
</SPAN></xsl:for-each>

<xsl:for-each select="semester"><SPAN CLASS="semester">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>
<xsl:for-each select="titel"><SPAN CLASS="titel">
        <xsl:value-of select="."/>
</SPAN></xsl:for-each>

</SPAN></xsl:for-each>
</BODY>
```

```
</HTML>
```

6.2.5 Zeitplan-Ansicht

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <style>
      .zeitplan          {border: medium solid black; width: 100%; }
      .slot .titel       {display: block; }
   </style>

<BODY>
<xsl:for-each select="seminar"><SPAN CLASS="seminar">

<DIV style="text-align: center; font-size: large; margin-bottom: 1ex;">
        Zeitplan fuers Seminar
</DIV>

<xsl:for-each select="zeitplan"><span class="zeitplan">
<table>
<xsl:choose select="*">
        <xsl:when select="datum">
                <xsl:element name="/table"/>
                <xsl:element name="table border='1' style='width: 33.32%; float: left '"/>
                <tr><th>
                        <xsl:value-of select="."/>
                </th></tr>
        </xsl:when>
        <xsl:when select="slot[@vortrag]"><span class="slot">
                <tr style="height: 6em; background-color: rgb(235,235,235)"><td>
                        <xsl:for-each select="uhrzeit"><span class="uhrzeit">
                                <xsl:value-of select="."/>
                        </span></xsl:for-each>
                        <xsl:for-each select="titel"><span class="titel">
                                <xsl:value-of select="."/>
                        </span></xsl:for-each>
                </td></tr>
```

```
        </span></xsl:when>
        <xsl:otherwise><span class="slot">
                <tr style="height: 6em"><td>
                        <xsl:for-each select="uhrzeit"><span class="uhrzeit">
                                <xsl:value-of select="."/>
                        </span></xsl:for-each>
                        <xsl:for-each select="titel"><span class="titel">
                                <xsl:value-of select="."/>
                        </span></xsl:for-each>
                </td></tr>
        </span></xsl:otherwise>
</xsl:choose>
</table>
</span></xsl:for-each>

</SPAN></xsl:for-each>
</BODY>
</HTML>
```

7 Literatur

[BeMi 98a] Behme, H. – Mintert, S.: XML in der Praxis, Addison-Wesley Verlag 1998

[BeMi 98b] Behme, H. – Mintert, S.: http://www.mintert.com/xml/trans/REC-xml-19980210-de.html (Extensible Markup Language (XML) 1.0) [Stand: 10.2.1998]

[ClDe 98] Clark, J. – Deach, S.: http://www.w3c.org/TR/1998/wd-xsl-19980818.html (Extensible Styleshet Language (XSL) Version 1.0) [Stand: 18.8.1998]

[Dünh 98] Dünhölter, K.: http://members.aol.com/xmldoku/index.html (Das Web automatisieren mit XML) [Stand: 1.9.1998]

[LieB 97] Lie, Kaon Wium - Bert Bos: Cascading Style Sheet, Boon,Addison-Wesley Verlag 1997

[Micr 98] Microsoft Corporation: http://www.microsoft.com/xml/xsl/downloads/msxsl.asp [Stand: 31.10.98]

[Münz 98] Münz, S.: http://www.netzwelt.com/selfhtml/ (SELFHTML) [Stand: 27.4.1999]

Bewertende Zusammenfassung von XML und SGML

Rainer Dampmann

FB Informatik, Uni Dortmund
Rainer.Dampmann@Fernuni-Hagen.de

Zusammenfassung

Der Beitrag beschäftigt sich zusammenfassend mit SGML und XML. Nach einem kurzen Einstieg in die bisherige Entwicklung der Sprachen geht es nun vornehmlich um einen Vergleich der beiden Sprachen im Bezug auf ihre Syntax und ihre Verwendbarkeit. Es wird sich herausstellen, daß XML wesentlich leichter für einen Parser zu verifizieren ist. Hier beginnt dann der nächste Abschnitt über die Werkzeuge zur Sprache, welche momentan von der Erzeugung bis zur Darstellung zur Verfügung stehen. Auch die geplante, zukünftige Unterstützung durch die Softwarehäuser wird ein Thema sein. Abschließend wird auf die Aussichten von XML als (Internet-)Sprache der Zukunft eingegangen.

1 Kurzer Einstieg

Der Ursprung der heutigen Markup-Sprachen liegt Ende der sechziger Jahre und entspringt aus einer Idee von William Tunnicliffe. 1967 stellte er das Konzept des „generic coding“ vor, welches die Trennung des Informationsgehalts eines Dokuments von seiner äußeren Form vorsah. Kurze Zeit später führte Stanley Rice die „editorial structure tags“ ein, woraus sich das heutige „generic markup“ entwickelte.

1.1 SGML

SGML, die Standard Generalized Markup Language, ist die wohl umfangreichste Markup-Sprache und wurde erstmalig 1986 in der ISO 8879 standardisiert. Dieser Standard wird allerdings in regelmäßigen Abständen den aktuellen Gegebenheiten angepaßt, so daß es mittlerweile mehrere Anhänge (Annex) gibt, die die Sprache an die heutige Informationstechnik anpassen. In diesem Zusammenhang seien hier nur zwei erwähnt, die im weiteren von Bedeutung sind. Der Annex J - Extended Naming Rules - führt beispielsweise die Case-Sensitivität, also den Unterschied zwischen Groß- und Kleinschreibung im Markup optional ein. Der Annex K - Web SGML Adaptations - hebt die Kapazitäts- und Quantitätsbeschränkungen auf, die bei Einführung des Standards aufgrund der noch nicht so leistungsfähigen Rechenmaschinen bestanden. Im wesentlichen sorgen diese beiden Anhänge dafür, daß XML eine Teilmenge von SGML ist, womit wir bei der zweiten hier relevanten Sprache sind.

1.2 XML

XML, die Extended Markup Language, entstammt einer Idee des W3 Consortiums. Dieses sah sich seit Mitte der neunziger Jahre aufgrund des sich immer mehr verbreiternden Internets veranlaßt, einen mächtigeren Nachfolger für HTML, die Hypertext Markup Language, zu schaffen. Weil SGML aber viel zu umfangreich und vor allem zu schwer zu lernen war, entschied man sich, SGML um die nicht relevanten Teile abzuspecken und das Markup im Dokument vollständig zu fordern. Die so entstandene Sprache XML wurde erstmals Anfang 1996 auf einer Konferenz besprochen. Bereits zwei Jahre später, am 10. Februar 1998, existiert dazu die erste Empfehlung (Recommendation) in der Version 1.0. Weitere Anhänge und Teilsprachen wie MathML, eine Markup-Sprache für den mathematischen Bereich sowie die zugehörigen Linksprachen XPointer und XLink sind auch schon zumindest als Arbeitspapiere (Working Drafts) verfügbar. Während die Entwicklung von SGML annähernd zwanzig Jahre gedauert hat und sich die Sprache bis heute noch nicht auf dem Markt durchgesetzt hat, hat sich XML binnen drei Jahren von einer Idee bis fast zu einem Standard entwickelt, den laut Presseberichten noch im Jahre 1999 die meisten Software-Produkte und -Häuser unterstützen wollen.

Doch zunächst zu den Unterschieden, die zwischen XML und SGML bestehen.

2 Unterschiede zwischen den beiden Markup-Sprachen

Die Unterschiede bzw. Einschränkungen, die XML von SGML abgrenzen, lassen sich in vier Bereiche einteilen. Auf die ersten beiden wird dabei detailliert eingegangen, während die zwei übrigen nur der Vollständigkeit halber kurz angerissen werden.

Die vier Aspekte sind:

- Generelle Einschränkungen / Unterschiede
- Unterschiede in der Syntax (Entity, Attribut, Element, ...)
- Unterschiede bei Verweisen (Linking)
- Unterschiede der Stilkomponenten (StyleSheets)

Zur Veranschaulichung werden ein paar der Unterschiede an einem ausgewählten Beispiel deutlich gemacht. Es handelt sich dabei um eine DTD und ein Dokument, die als eine Email angesehen werden können. Dabei sind allerdings nur die grundlegenden Eigenschaften einer Email implementiert, die was aber zur Verdeutlichung des Sachverhalts völlig ausreichen.

2.1 Generelle Einschränkungen / Unterschiede

Dieser Abschnitt beschäftigt sich mit all jenen Unterschieden, deren Auswirkungen sich in der gesamten Sprache XML bemerkbar machen.

2.1.1 Nicht verfügbare SGML-Optionen

Zunächst zu den Optionen, die in SGML möglich, aber in XML grundsätzlich verboten sind. Die wohl wichtigste Einschränkung ist das Verbot von OMITAG, welches in SGML die Möglichkeit eröffnet, Tags wegzulassen. Dies wird in SGML durch zusätzliche Informationen in den Elementen sichergestellt. Die fehlende Verwendungsmöglichkeit in XML erhöht nicht nur die Lesbarkeit, sondern erleichtert auch dem Parser des Dokuments die Arbeit, da zu jedem Start- auch ein End-Tag vorhanden ist. Die nicht sehr schwierige Überprüfung eines Dokumentes ist Ziel auch der nächsten Einschränkungen, nämlich dem Untersagen von DATATAGs. Sie stellen in SGML eine

Art Parsen eines durch Markup gekennzeichneten Bereiches dar, in welchem die Applikation selber die Tags setzen muß. Dazu gehört auch die Option USEMAP, die selbiges in SGML erst ermöglicht und in XML ebenso nicht verfügbar ist. Weiterhin ist es in XML nicht gestattet, zwei oder mehr unterschiedliche Sichten auf ein Dokument zu haben (CONCUR). Auch externe Daten mit zugehöriger DTD (Document Type Definition), die komplett eingebunden werden, sind wegen „SUBDOC NO“ nicht möglich. Außerdem ist das Einbinden mittels FORMAL von externen Entitäten, die von der ISO standardisiert wurden, nicht zulässig. Der letzte, größere Unterschied zwischen SGML und XML ist das Linkkonzept. Die in SGML vorhanden Links (IMPLICIT, EXPLICIT und SIMPLE) werden in XML nicht übernommen. Auf dieses Thema wird in Kapitel 2.3 näher eingegangen.

Damit die Auswirkungen der Einschränkungen deutlich werden, hier zur Veranschaulichung das bereits oben erwähnte Beispiel einer Email:

Quelltext der DTD in SGML:

```
<!ELEMENT EMAIL - - (HEADER, CONTENT)>
<!ELEMENT HEADER  o o (RECEIVER+, SENDER, SUBJECT?)>
<!ELEMENT RECEIVER - o (#PCDATA)>
<!ATTLIST RECEIVER SEX (Male | Female) "Male">
<!ELEMENT SENDER - o (#PCDATA)>
<!ELEMENT SUBJECT - o (#PCDATA)>
<!ELEMENT CONTENT - - (#PCDATA)>
<!ENTITY SIGNATURE “Rainer Dampmann - Student der Informatik“>
```

In SGML ist, wie oben bereits beschrieben das Weglassen von Markup-Information (OMITAG) möglich, was durch ein „o“ hinter dem Namen des Elements angegeben wird. Das erste „o“ steht dabei für den Start-, das zweite für den End-Tag. In XML fallen diese beiden Zeichen weg.

Dies führt in SGML dazu, das ein zugehöriges Dokument wie folgt aussehen könnte:

```
<EMAIL>
<RECEIVER SEX=“Male“>Jörg Westbomke
<SENDER>Rainer Dampmann
<SUBJECT>Eine Mail
<CONTENT>Dies ist der Inhalt</CONTENT>
</EMAIL>
```

Hier fehlt zum Beispiel der schließenden Tag von RECEIVER, und auch der Start- und End-Tag vom HEADER sind nicht vorhanden. Das entsprechende Dokument in XML hat die Option des OMITAG nicht und verfügt deshalb über das komplette Markup.

```
<EMAIL><HEADER>
<RECEIVER SEX=“Male“>Jörg Westbomke</RECEIVER>
<SENDER>Rainer Dampmann</SENDER>
<SUBJECT>Eine Mail</SUBJECT></HEADER>
<CONTENT>Dies ist der Inhalt</CONTENT>
</EMAIL>
```

Bei den Beispielen fehlt jeweils die Einbindung der entsprechenden DTD sowie die Angabe der verwendeten Version der Sprache. Zur Veranschaulichung der Unterschiede ist es hier und auch im weiteren Verlauf nicht nötig.

Die bisherigen Einschränkungen, die sich durch Setzen von SGML-Optionen auf NO ergaben, sind die wohl weitreichendsten. Daraus lassen sich noch ein paar weitere mit generellen Charakter ableiten.

2.1.2 Weitere, generelle Unterschiede

In XML gibt es zwei definierte Dokumentzustände, nämlich wohlgeformt und gültig. Wohlgeformtheit eines Dokuments ist vorhanden, wenn im Dokument das Markup korrekt verwendet wird, d.h. die Schachtelungen eindeutig sind und es ein umschließendes Element gibt. Ein Dokument ist gültig, wenn neben der Wohlgeformtheitsbedingung auch eine dazu passende DTD existiert, deren Anwendung in dem Dokument korrekt und vollständig erfolgt. Es gibt in XML keine Kapazitäts- und Quantitätsbeschränkungen wie in der ersten Standardisierung von SGML. 1986 war die Anzahl der Elemente, deren Schachtelung sowie Namenslänge aufgrund der beschränkten Rechenkapazitäten begrenzt worden. Durch den Annex K wurde dies mittlerweile aufgehoben. Weiterhin existieren in XML einige Namenskonventionen, die in SGML nicht gelten. So sind in XML die Namen grundsätzlich case-sensitiv und auch Sonderzeichen wie Unterstrich, Komma usw. sind in Namen erlaubt. Die Buchstabenkombination [Xx][Mm][Ll] ist zusätzlich noch reserviert und sollte deshalb nicht durch den Programmierer oder Anwender verwendet werden.

Ein weiterer Unterschied zu SGML besteht bei den markierten Abschnitten. Diese sind in XML ausschließlich vom Typ CDATA - Character Data - und werden nicht geparst, sondern ungeprüft als Text eingesetzt. In SGML gibt es zusätzlich noch RCDATA - Replaceble Character Data - , die vor dem Einfügen noch nach Entitäten durchsucht werden, welche dann ersetzt werden. Zum Themenbereich Entitäten ist hier noch ein weiterer Punkt anzusprechen. In XML existieren einige Standardentitäten, die nicht explizit durch den Programmierer oder Anwender definiert werden müssen. Dazu zählen unter anderem die Entitäten „&amp“ (ampersand) bzw. „&lt“ (less than) für „&“ und „<“. Werden die Zeichen „&“ und „<“ im Text benötigt, so müssen sie durch „&amp“ bzw. „&lt“ angegeben werden. Erforderlich wird dies durch die Verwendung der Zeichen. „<“ Steht in XML für den Beginn von Tags. Bei einem Vorkommen dieses Zeichens im Text könnte der Parser Markup vermuten und Fehler generieren. Das Zeichen „&“ steht vor jeder Entität und auch hier könnte es zu Verwechslungen kommen.

Als letzte Aspekte in diesem Abschnitt sind noch einige Einschränkungen zu nennen, die in den Bereich Vollständigkeit des Markup aus Kapitel 1.2 fallen. In XML existieren weder Nicht-geschlossene-Start-Tags noch Nicht-geschlossene-End-Tags. Gleiches gilt für leere Start-Tags und leere End-Tags. Auch diese sind in XML nicht verfügbar.

Das folgende SGML-Dokument zeigt, welche Auswirkungen sich daraus ergeben. Der nicht geschlossene Start-Tag von EMAIL ist ebenso in XML nicht erlaubt, wie der leere End-Tag von SENDER und der nicht geschlossene End-Tag von SUBJECT.

```
<EMAIL<HEADER>
<RECEIVER “Male“>Jörg Westbomke</RECEIVER>
<SENDER>Rainer Dampmann</>
<SUBJECT>Eine Mail</SUBJECT</HEADER>
<CONTENT>Dies ist der Inhalt</CONTENT>
</EMAIL>
```

Das entsprechende XML-Dokument unterscheidet sich nicht von dem in Kapitel 2.1.1 aufgeführten. Es wird aber deutlich, daß durch das Weglassen von Markup-Information oder auch nur Teilen davon die Parserfreundlichkeit und Lesbarkeit beeinträchtigt wird.

Damit ist der Abschnitt mit den wichtigsten Einschränkungen bzw. Unterschieden, die den gesamten Sprachbereich von XML betreffen, abgeschlossen. Doch auch in der Syntax von XML gibt es wesentliche Unterschiede zu SGML, die im folgenden besprochen werden.

2.2 Unterschiede in der Syntax

Der Bereich der Unterschiede in der Syntax beschäftigt sich grundlegender mit der eigentlichen Sprachbeschreibung von XML und den daraus resultierenden Unterschieden zu SGML. Dabei werden die wichtigen Typen der Sprache wie Entitäten, Attribute, Elemente und Verarbeitungsanweisung explizit besprochen. Es werden die Auswirkungen für den Programmierer oder Anwender ebenso deutlich gemacht wie die Folgen für die Parsbarkeit und Lesbarkeit der Dokumente.

2.2.1 Entitäten (Entity)

Entitäten sind Speicherungseinheiten, die über ihren Namen referenziert werden und auf den zugehörigen Inhalt verweisen. Hierbei treten zu SGML in zwei Bereichen Unterschiede auf. In SGML wie auch in XML müssen Referenzen mit einem Semikolon abgeschlossen werden. Außerdem muß selbstverständlich die referenzierte Entität verfügbar sein, da sonst Definitionslücken entstehen würden, die von der Applikation nicht sinnvoll geschlossen werden können. In XML gelten aber noch einige weitere Einschränkungen. So dürfen externe Entitäten, also solche mit System-Identifier, nicht als Attributwert verwendet werden. Damit wird verhindert, daß auch die Attribut-Werte neben der eigentlichen Überprüfung bei bestimmten Attributtypen (zum Beispiel: ID) eine weitere Kontrolle durchlaufen müssen. Weiterhin dürfen in XML Parameter-Entitäten nur in der DTD verwendet werden, was zu einer Erleichterung beim Parsen des Dokumentes führt. Denn die Einstellungen müssen somit fest in der DTD vorgenommen werden und nicht erst im Dokument. Gleiches gilt für Parameter-Entitäten in SGML.

Neben den Referenzen, die einige Unterschiede zu SGML aufweisen, existieren selbige auch bei der Deklaration. So muß jede verwendete Entität eine korrekte Ersetzung ermöglichen. Auch gibt es in XML weder externe noch interne CDATA-Entitäten. Hier steht wiederum die Parserfreundlichkeit und die korrekte Ersetzung im Vordergrund. Außerdem ist es in XML verboten, Verarbeitungsanweisungen oder gar Markup in Entitäten zu codieren. Dies würde dazu führen, daß das Markup im Dokument vollständig erscheint, da es nicht wegkodiert werden kann.

In SGML wäre es also vorstellbar, folgende Entität zu definieren:

```
<!ENTITY STARTINGTAGS “<EMAIL><HEADER><RECEIVER>“>
```

Die entsprechende SGML-Instanz sähe dann wie folgt aus:

```
&STARTINGTAGS;Jörg Westbomke
...
<CONTENT>Dies ist eine Inhalt.</CONTENT>
...
```

Ein Parser hätte hier zuerst die entsprechende Entität aufzulösen, bevor er die Korrektheit des Dokuments sicherstellen kann. Der Aufwand für den Parser steigt hier ebenso wie der des Anwenders, das Dokument zu lesen.

Im Unterschied zu SGML muß in XML jede externe Entität die geforderte System-Identifier

besitzen. Außerdem existiert im Dokument keine #DEFAULT Entität, welche in SGML immer dann zum tragen kommt, wenn die referenzierte Entität nicht definiert worden ist.

Abschließend zu diesem Bereich sei noch erwähnt, daß Entitäten in XML keine Attribute haben können. Dies bleibt ausschließlich den Elementen vorbehalten.

2.2.2 Attribute (attribute)

Im letzten Abschnitt wurde bereits erwähnt, daß Attribute für Entitäten nicht zur Verfügung stehen. Selbiges gilt auch für Notationen, welche durch ihren Namen das Format von nicht-analysierten Entitäten identifizieren. Attribute werden verwendet, um Elemente näher zu spezifizieren. Dabei ist in XML darauf zu achten, daß eine Attributdeklaration nur genau für ein Element vorgenommen werden kann. Namensgruppen sind nicht erlaubt. Weiterhin muß in XML der Standardwert definiert sein. Dies kann zum einen ein Wert aus der entsprechenden Aufzählung sein oder auch die vordefinierten Typen #REQUIRED, #IMPLIED oder #FIXED mit dem entsprechenden Wert. Attributwerte dürfen ferner nur mit „Oder“ (OR) verknüpft werden, so daß jedem Attribut genau ein Wert zugewiesen wird. Während in SGML ein Attribut auch die Möglichkeit hat, vom Typ NUTOKEN(S), NAME(S) oder NUMBER(S) zu sein, ist das in XML nicht zulässig. Diese drei, mit Mehrzahl sechs Typen stehen in XML von vornherein nicht zur Verfügung. Hingegen dürfen ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN und NMTOKENS verwendet werden. Auch das #CURRENT, welches in SGML zulässig ist, taucht in dieser Liste nicht auf. Diese Nichtberücksichtigung hat den Effekt, daß sich die Applikation nicht die aktuelle Einstellung für ein Attribut merken muß. Dies führt, wie auch das Verbot von Namensgruppen und die Einschränkung auf nur Oder-Verknüpfungen, zu einer Erleichterung für den Parser. Dieser hat zu jedem Zeitpunkt genau einen explizit angegebenen Attributwert und muß nicht erst Werte aus der Vergangenheit verwenden.

In SGML wäre der nachfolgende Teil einer DTD erlaubt:

```
<!ELEMENT RECEIVER - o (#PCDATA)>
<!ELEMENT SENDER - o (#PCDATA)>
<!ATTLIST (RECEIVER, SENDER) SEX (Male | Female) "Male">
```

Im Gegensatz dazu muß in XML, will man ähnliches wie in SGML erreichen, das Attribut einzeln für jedes Element definiert werden.

```
<!ELEMENT RECEIVER (#PCDATA)>
<!ELEMENT SENDER (#PCDATA)>
<!ATTLIST RECEIVER SEX (Male | Female) "Male">
<!ATTLIST SENDER SEX (Male | Female) "Male">
```

Der eindeutige Schlüssel für die Attribute ist der Attributname, über den die Referenzierung gegenüber den Elementen erfolgt.

2.2.3 Elemente (element)

Ein Element ist eine Struktureinheit des XML-Dokuments, deren Inhalt durch Attribute näher beschrieben werden kann. Allerdings muß hier im Gegensatz zu SGML darauf geachtet werden, daß der Attributname immer vorhanden ist. Ein Weglassen führt bei der Applikation sofort zu einem Fehler. In SGML ist das Auslassen des Attributnamens bei Attributen, die vorhanden sein müssen (#REPUIRED), erlaubt. Die Applikation in XML hat dadurch den Vorteil, daß nicht nach dem entsprechenden, passenden Attributnamen gesucht werden muß, sondern er immer explizit davor steht. Dies fördert ebenso die Parserfreundlichkeit wie auch den Umstand, daß im Inhaltsmodell eines Elements nur „Oder“ („|“) und der Aufzählungstyp („,“) verwendet werden dürfen. Ein

„AND“ kann in XML nicht auftreten. Das „AND“ ermöglicht in SGML eine beliebige Reihenfolge für die so verknüpften Elemente. Die strikte Reihenfolge, wie sie folglich in XML notwendig ist, erleichtert die Arbeit des Parsers um einiges.

Ähnlich wie bei Attributen ist auch bei Elementen eine Deklaration mittels Namensgruppen nicht gestattet. Des weiteren gibt es für einen gemischten Inhalt („mixed content“) ein strikte Ordnung. So befindet sich in XML am Anfang eines jeden „mixed content“ ein Bereich mit Zeichendaten (#PCDATA), gefolgt von einer beliebigen Anzahl von Elementen, die Kindelemente genannt werden. Hier steht, wie auch bei den anderen erwähnten Unterschieden die Vereinfachung im Vordergrund. Ein Parser hätte Schwierigkeiten, einen PCDATA-Abschnitt inmitten von Kindelementen zu trennen, weil dieser nicht durch Markup eingeführt wird.

Auch hier ein Beispiel, welches die Verwendung und Definition von Elementen verdeutlicht. Da in SGML Namensgruppen und das „AND“ erlaubt sind, ist der folgende Teil der DTD korrekt:

```
<!ELEMENT HEADER  o o (RECEIVER+ & SENDER & SUBJECT?)>
<!ELEMENT (RECEIVER,SENDER,SUBJECT) - o (#PCDATA)>
<!ATTLIST RECEIVER SEX (Male | Female) "Male">
```

In XML, in der beide SGML-Optionen verboten sind, müßte die DTD (siehe Kapitel 2.1.1) verwendet werden.

Wie schon angesprochen, kann ein Element vom Typ PCDATA sein. Dies ist in XML auch die einzige Möglichkeit, Elemente als Zeichenketten zu deklarieren. Der Inhalt von CDATA ist nicht erlaubt. Außerdem gibt es in XML keine Inklusionen und Exklusionen wie in SGML. Dort werden sie benutzt, um zusätzliche Daten für eine größere Anzahl von Elementen zur Verfügung zu stellen, beispielsweise für alle Abschnitte und Unterabschnitte eines Dokuments die Verwendung von Fußnoten. Es ist müßig zu erwähnen, daß auch hier wieder die Parserfreundlichkeit und die Vereinfachung für den Programmierer oder Anwender der Vater des Gedankens ist.

2.2.4 Verarbeitungsanweisungen (processing instructions)

Verarbeitungsanweisungen entsprechen eigentlich nicht direkt dem strikten Konzept der Markup-Sprachen. Sie dienen dazu, Anweisungen für die angrenzenden Applikationen oder die Umgebung zur Verfügung zu stellen, was der Systemunabhängigkeit widerspricht. Schon in SGML gab es Verarbeitungsanweisungen, die auf Grund der Programmierer eingeführt wurden. Anfangs haben diese die Anwendungen in Kommentaren codiert, was dazu führte, daß schließlich auch die Kommentare geparst wurden. Um dem entgegenzuwirken und die Kommentare wieder zu dem zu machen, wozu sie eigentlich gedacht waren, wurden kurzerhand die Verarbeitungsanweisungen eingeführt. Sie stellen vom Typ her nichts anderes dar als einen Kommentar, also eine nicht geparste Zeichenkette. Allerdings werden hierin natürlich systemspezifische Informationen codiert, während der Kommentar eine Information für den Programmierer oder Anwender ist. Während in SGML die Verarbeitungsanweisungen fast beliebig sind, ist in XML zugunsten der Parserfreundlichkeit eine vorgegebene Struktur eingeführt worden. So beginnt jede Verarbeitungsanweisung mit einem Namen („PITarget“) und endet mit „?>“. Wie schon in Kapitel 2.1.2 erwähnt, ist hier die Buchstabenkombination [Xx][Mm][Ll] reserviert für die eigentliche Sprache. Diese Kombination darf zusätzlich noch nur am Anfang eines Dokuments erscheinen. Sie steht zum Beispiel für den Typ der verwendeten XML-Version und dieser sollte im Dokument sicherlich nur einmal festgelegt werden.

Alles in allem sollte man Verarbeitungsanweisungen deshalb nur höchst selten verwenden, denn sie enthalten Informationen, die sich meist auf umgebende Applikationen beziehen. Damit sind sie vom System abhängig und könnten in einer anderen Umgebung zu völlig anderen Aktionen führen.

2.2.5 Weitere Unterschiede

Zwischen SGML und XML existieren noch eine ganze Reihe kleinerer Unterschiede, von denen an dieser Stelle die etwas wichtigeren der Vollständigkeit halber noch genannt werden. Für einen SGML Programmierer, der nun auf XML umsteigen soll, fallen sie aber nicht großartig ins Gewicht.

Die Verwendung von Kommentaren („comment“) beispielsweise ist in XML nicht mehr überall möglich. War es in SGML erlaubt, Kommentar auch im Markup anzugeben, ist dies in XML nicht mehr gestattet. Auch bei Zeichenreferenzen („character references“) haben sich einige Änderungen zu SGML ergeben. In XML müssen Zeichenreferenzen entweder über ihre dezimale oder ihre hexadezimale Darstellung angegeben werden. Ein Vergeben von Namen für bestimmte Zeichen ist nicht mehr vorgesehen. Die letzten Unterschiede betreffen die markierten Bereiche („marked section“). Sie sind in XML nur noch vom Typ CDATA zulässig. Weiterhin sind „INCLUDE“ bzw. „IGNORE“ ausschließlich der Verwendung in der DTD vorbehalten.

Damit sind die Unterschiede in der Syntax mehr oder weniger vollständig besprochen.

2.3 Unterschiede bei Verweisen (Linking)

Das Linking aus SGML ist in XML nicht übernommen worden. Das heißt, in XML werden bisher weder EXPLICID, IMPLICID noch SIMPLE Links unterstützt. Das W3-Consortium bemüht sich um eine völlig neue Linksprache für XML. Mittlerweile sind zwei unterschiedliche Sprachen auch schon als Working Draft verfügbar.

Zum einen ist da die Sprache XPointer. Sie soll später in XML auf Markup-Strukturen verweisen und somit direkt an die gewünschten Information verzweigen. Dieses Verweiskonzept findet sich in HTML überhaupt nicht.

Die Grundzüge des aus HTML bekannten Konzepts von Verweisen auf andere Dokumente sind auch in XML vorhanden. Zusätzlich umfaßt die Sprache XLink noch weitere Strukturen. So wird es möglich sein, über einen Verweis mehrere Dokumente gleichzeitig anzusprechen.

Obwohl aber zwei, speziell für XML entwickelte Sprachen das Linking übernehmen, bleibt selbstverständlich die Kompatibilität nicht auf der Strecke. So wird in XML auch das Linkformat von HTML unterstützt und auch an eine Kompatibilität zu SGML ist gedacht.

2.4 Unterschiede der Stilkomponenten (StyleSheets)

Die Stilkomponente („StyleSheet“) ermöglicht dem Programmierer, die äußere Form der Elemente festzulegen. Zwischen den Sprachen existieren jedoch die unterschiedlichsten Stilsprachen mit den unterschiedlichsten Konzepten.

CSS - die Cascading Style Sheets - ist die Stilsprache von HTML. Diese Sprache ermöglicht ausschließlich eine Formatierung der in HTML vorhandenen Elemente. Somit ist sowohl ihr Umfang als auch ihre Möglichkeiten begrenzt.

Die Stilsprache von SGML ist DSSSL. Die Document Style Semantics and Specification Language dient nicht nur der reinen Formatierung der Elemente, sondern enthält zusätzlich noch eine komplette Programmiersprache. Mit ihr kann man das Aussehen beispielsweise anhand von Werten, Information oder Systemen zur Laufzeit individuell gestalten.

Die Entwicklung der Stilsprache von XML ist noch nicht komplett abgeschlossen. Was sich aber abzeichnet ist, daß XSL (Extensible Style Language) eine Stellung zwischen CSS und DSSSL einnehmen wird. Sie wird, wie auch die beiden anderen Sprachen, die Formatierung der Elemente

übernehmen. Zusätzlich wird es eine stark eingeschränkte Programmiersprache mit einigen wenigen Befehlen geben, die nur einen Bruchteil der Variabilität von DSSSL hat.

2.5 Zusammenfassung der Unterschiede

Eine Zusammenfassung des Kapitels über die Unterschiede der beiden Markup-Sprachen SGML und XML ist trotz der doch recht langen Teilabschnitte einfach. Die Anzahl der Unterschiede ist zwar recht groß, aber ihre Auswirkung für SGML-Programmierer, die jetzt auf XML umsteigen, nicht von großer Bedeutung.

Alles in allem beziehen sich die Unterschiede zum Großteil auf Restriktionen in der Syntax. Während in SGML eine Menge Informationen (Markup) weggelassen werden können, ist dies in XML nicht mehr möglich. Hier wird strikte und vollständige Verwendung des Markup verlangt. Minimalisierung ist an fast keinem Punkt möglich. Einzige Ausnahme ist der Empty-Tag, der auch in XML verfügbar ist.

Zwar hat XML sowohl eine neue Linksprache als auch eine neue Stilkomponente, doch ist hier die Kompatibilität zu den bereits vorhandenen Sprachen existent. Die Linksprache wird sowohl die aus HTML als auch den Großteil der aus SGML bekannten Linkstrukturen unterstützen. Ähnliches gilt für XSL. Die Stilkomponente von XML ist zwar eine eigene Sprache, doch sie wird sich an CSS und DSSSL ausrichten, so daß ein Umstieg hier leichtfallen sollte.

Außerdem haben beide Sprachen zur Bearbeitung meist Unterstützung durch entsprechende Werkzeuge, die genaue Kenntnisse der Sprache nicht unbedingt erforderlich machen.

3 Werkzeuge

Die Entwicklung und Akzeptanz einer Sprache hängt vor allem von der Unterstützung durch Hard- und Software ab. Im Falle der Markup-Sprachen ist der Bereich von Software der wohl ausschlaggebendere. Sicherlich gibt es Beispiele, bei denen der Durchbruch auch ohne Unterstützung durch Werkzeuge gelungen ist, doch das ist im allgemeinen die Ausnahme. Doch selbst eine gute Unterstützung durch Werkzeuge garantiert den Durchbruch einer Sprache keinesfalls. Als bestes Beispiel dafür dürfte SGML stehen. Den Standard gibt es seid nunmehr zwölf Jahren und auch Werkzeuge sind seid geraumer Zeit verfügbar. Trotzdem hat sich SGML als Sprache nicht durchsetzten können. Heutzutage nutzen nur wenige Firmen den Vorteil der Markup-Sprache für ihre Informationssysteme.

XML scheint die gleichen Voraussetzungen zu haben, da die in Kapitel 2 besprochenen Unterschiede zu SGML von nur geringer Bedeutung sind. Trotzdem entwickelt sich XML anders, als es SGML bisher getan hat. Während es bei SGML relativ lang gedauert hat, bis Werkzeuge vorhanden waren, stehen für XML bereits Applikationen kurz vor der Fertigstellung oder wurden bereits auf den Markt gebracht. Dies ist um so erstaunlicher, als daß die Standardisierung der Sprache bis jetzt nicht einmal vollständig abgeschlossen ist. Weder ist XML bis heute zu einem ISO-Standard geworden, noch stehen die Linksprache oder die Stilkomponente in ihrer endgültigen Version zur Verfügung.

Obwohl noch viele Unwägbarkeiten vorhanden sind, gibt es wie erwähnt bereits die ersten Werkzeuge. Diese werden in den folgenden Abschnitten besprochen.

3.1 Editoren

Die erste große Gruppe von Werkzeugen sind die Editoren. Sie werden zur Erstellung sowohl von DTD wie auch vom Dokument benötigt und sollen dem Programmierer oder Anwender die Arbeit beim Erstellen erleichtern.

3.1.1 DTD-Editoren

Bei den DTD-Editoren läßt sich, wie in SGML, eine Unterteilung in zwei Bereich vornehmen.

Textuelle Editoren

Der Erste ist die, in allen anderen Sprachen wohlbekannte, textuelle Erstellung einer DTD. Dabei wird die DTD in einem Editor der Text manuell eingegeben. Diese verlangt vom Benutzer die Kenntnis der Regeln von XML und bietet keinerlei Unterstützung, um grundlegende Bestimmungen vorzugeben. Die textuelle Erstellung der DTD ist folglich nur etwas für versierte Programmierer, denen der Standard geläufig ist.

Graphische Editoren

Ein normaler Benutzer hätte sicherlich gern mehr Unterstützung durch die Applikation. Dies ist bei der graphischen Entwicklung der Fall. Hier wird über eine Oberfläche die DTD erzeugt, so daß der Benutzer die dahinterstehende Textversion gar nicht zu Gesicht bekommt. Da es zur Zeit noch kein Beispiel für XML gibt, hier der Near&Far-Editor von Microstar Software (Abbildung 3-1).

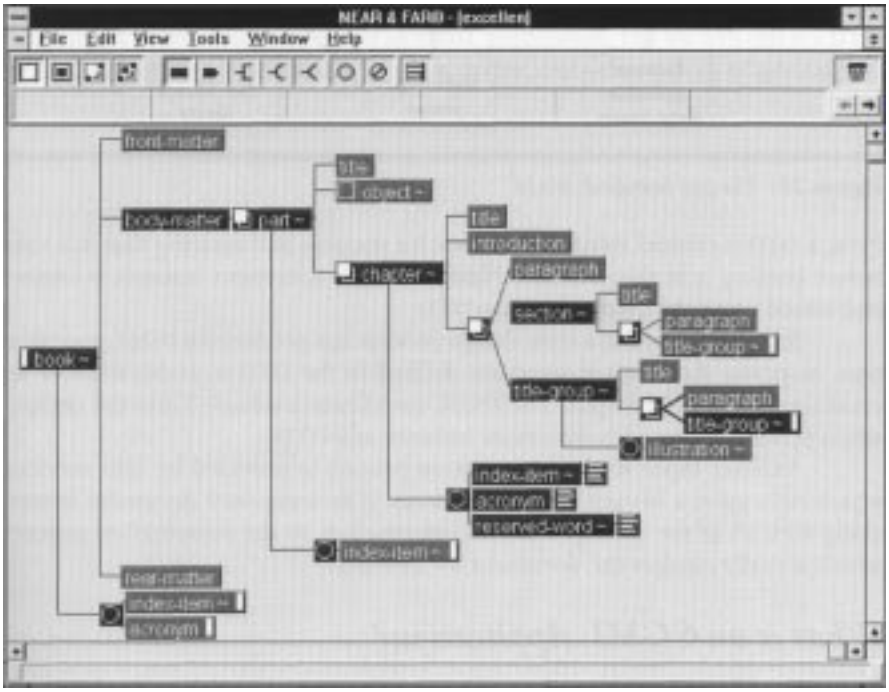


Abbildung 3-1: Near&Far von Microstar Software

Hierbei sieht man deutlich den baumartigen Aufbau einer DTD. Eine DTD besteht aus einem Element, welches alle anderen vorhandenen Elemente einschließt. Auch kann man durch die darüberliegenden Menüs eine Verwendung von nicht definierten Typen weitestgehend ausschließen. Somit wird eine Reihe von Fehlern, die bei der textuellen Erstellung auftreten können, durch die Applikation von vornherein ausgeschlossen. Es ist deshalb relativ einfach, eine korrekte DTD zu erzeugen, die dann später bei der Erstellung des Dokuments zum Einsatz kommen kann.

3.1.2 Dokument-Editoren

Die Dokument-Editoren unterstützen den Benutzer beim Erstellen des eigentlichen Dokuments. Auch hier gibt es wieder eine Reihe von Arten von Editoren, aus denen der Benutzer wählen kann. Dabei spielt die textuelle Entwicklung, wie sie auch bei den DTD-Editoren vorhanden war, sicherlich wieder nur für professionelle Anwender eine Rolle. Denn hier muß der Programmierer die verwendete DTD mit ihren Elementen, Attributen und Entitäten kennen, um ein korrektes Dokument zu erstellen.

Viel interessanter für die meisten Anwender dürften die Editoren sein, die die DTD zumindest kennen und ihre Elemente, Attribute und Entitäten dem Benutzer mehr oder weniger komfortabel zur Verfügung stellen.

„View Tags Mode"-Editoren

Der textuellen Erstellung am nächsten kommt dabei die Editoren im „View Tags Mode". Hier werden die Tags graphisch hervorgehoben, während gleichzeitig die Elemente noch als normaler Text zu sehen sind.

„In Context"-Editoren

In der nächsten Stufe verschwinden die Tags aus dem sichtbaren Teil des Dokuments. Diese „In Context"-Editoren haben, wie im Beispiel des XML-Notepads von Microsoft (Abbildung 3-2), eine zwei-geteilte Oberfläche. Während rechts das eigentliche Dokument in Textform ohne Formatierung zu sehen ist, findet man links die entsprechende Strukturinformation zum rechts stehenden Textabschnitt.

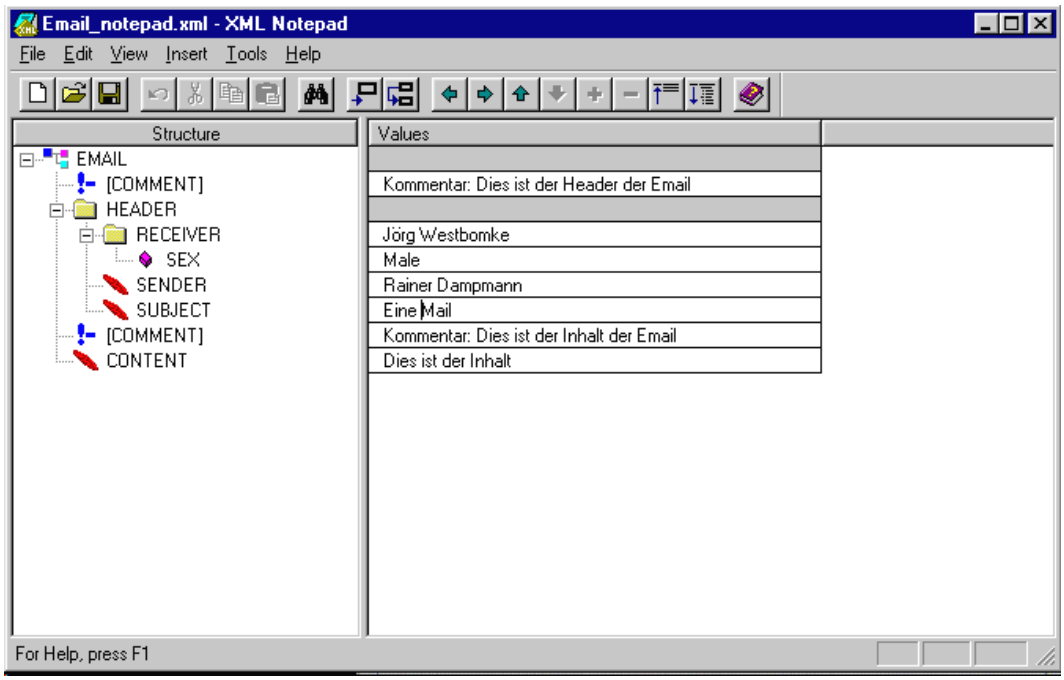


Abbildung 3-2: XML-Notepad von Microsoft

Die letzte Gruppe von Editoren unterscheidet sich grundsätzlich von den bisher genannten. Sah man bis jetzt immer nur die Struktur und den eigentlichen Text ohne seine Formatierung, so steht jetzt die Sicht auf das eigentliche Aussehen im Vordergrund.

„WYSIWYG“-Editoren

„What You See Is What You Get“-Editoren sind die für den Benutzer komfortabelsten Produkte. Hier wird die Struktur des Dokuments vollständig vor dem Anwender verborgen. Dieser bekommt das Dokument in seiner formatierten Art und Weise zu Gesicht und kann in diesem Modus auch Änderungen oder Erweiterungen vornehmen. Als Beispiel sei hier auf ein SGML-Produkt verwiesen, das auch die anderen vorher genannten Modi, unterstützt. Es handelt sich um Author/Editor von Softquad (Abbildung 3-3).

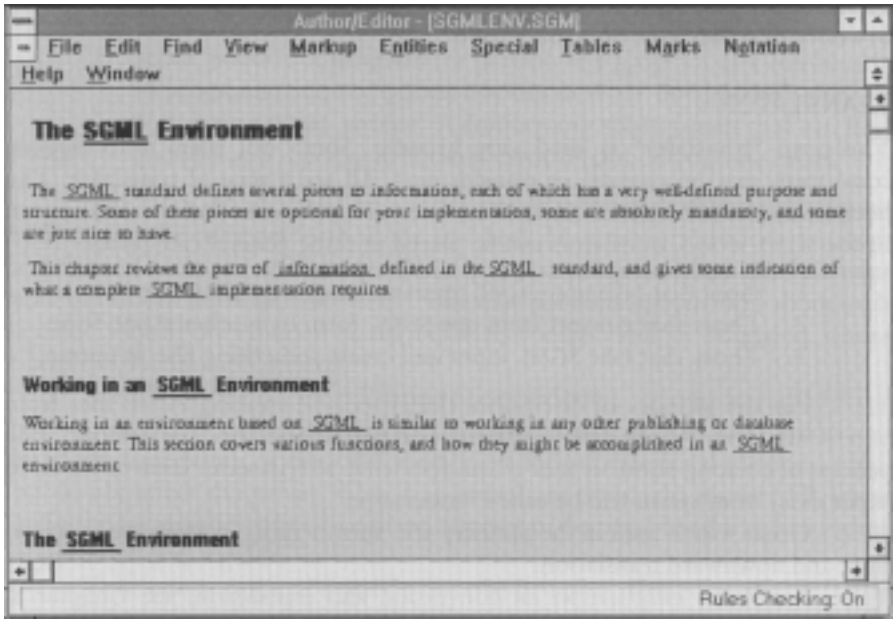


Abbildung 3-3: Author/Editor von SoftQuad

Neben den rein für SGML oder XML entwickelten Editoren gibt es auch noch den großen Bereich der Werkzeuge, die auf schon vorhandenen Programmen basieren.

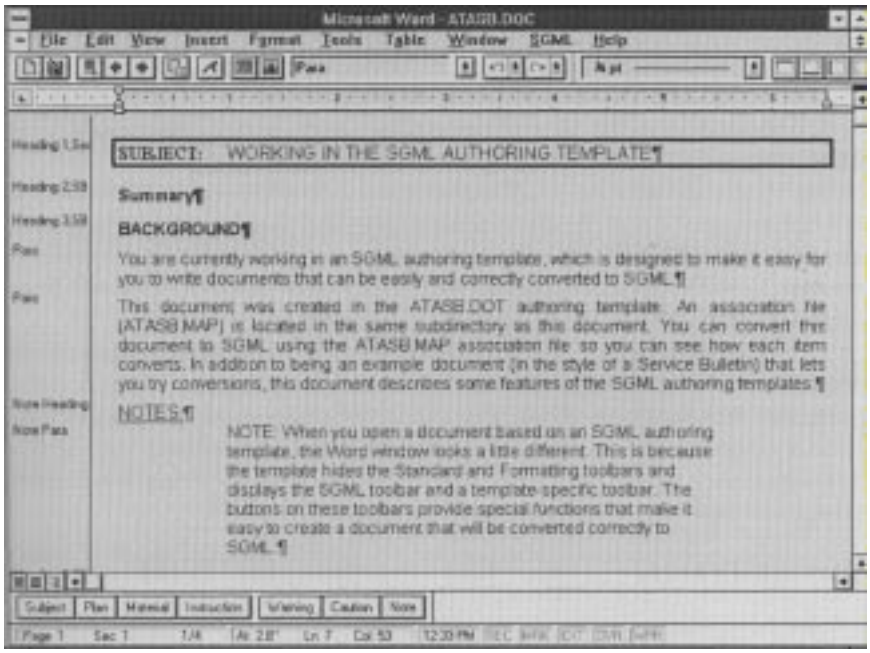


Abbildung 3-4: Add-In für Winword von Microsoft

Add-Ins

Add-Ins erweitern existente Applikationen um einen neuen Standard, eine neue Sprache. Ein gutes Beispiel, welches auch in Abbildung 3-4 erscheint, ist Microsoft Word. Dieses als Schreibprogramm für Textdokumente konzipierte Produkt kennt neben den ursprünglichen Formaten mittlerweile eine ganze Menge weitere, die mit dem anfänglichen nur wenig zu tun haben. So ist es möglich, in Word beispielsweise HTML-Dokumente zu erstellen.

Ein Problem stellt sich aber all diesen Dokument-Editoren. Soll ein Dokument mit einer vorhandenen DTD erstellt werden, muß die Applikation die DTD vorher einlesen und darf nur die dort verwendeten Elemente, Attribute und Entitäten zulassen. Folglich müßten die Menüs und verfügbaren Elemente dynamisch an die zugrundeliegende DTD angepaßt werden. Erforderlich ist aber auch, daß eine korrekte DTD eingelesen und gegebenenfalls überprüft werden kann, womit schon die nächste Gruppe von Werkzeugen angesprochen ist, die Parser.

3.2 Parser

Die Parser waren schon in Kapitel 2 ein zentrales Thema. Ging es dort darum, wie XML die Arbeit der Parser durch die strikte Strukturierung erleichtert, so geht es jetzt um die Funktionen eines Parsers.

Ein Parser hat allgemein die Aufgabe, die Korrektheit eines Dokuments zu prüfen. Ein Parser für die Markup-Sprache XML hat genau die gleiche Funktion, nur daß man hier noch zwei aufeinander aufbauende Parsertypen unterscheidet.

3.2.1 Normale Parser

Die hier angesprochene Gruppe der Parser überprüfen ein XML-Dokument auf dessen Korrektheit. Dabei wird nur kontrolliert, ob das Markup im Dokument korrekt gesetzt wurde, ohne auf eine eventuell zugrundeliegende DTD zurückzugreifen. Das bedeutet, diese Parser überprüfen nur die korrekte Verwendung der Regeln aus der XML-Spezifikation. Beispielsweise wird getestet, ob das gesamte Dokument von einem Element eingeschlossen wird. Auch kann das Dokument auf eventuell ungültige Verschachtelungen durchgegangen werden, so daß immer nur der zuletzt geöffnete Tag auch geschlossen werden kann.

3.2.2 Validierende Parser

Die validierenden Parser haben selbstverständlich auch die Funktionalität der normalen Parser, doch sie können weit mehr. Zur Kontrolle der Korrektheit beziehen sie auch die DTD mit ein und überprüfen auf deren Basis das zugrundeliegende Dokument. Hierbei muß die Struktur, die in der DTD angegeben ist, auch komplett und korrekt im Dokument verwendet worden sein.

Ein Programm von Microsoft, der XML-Parser (Abbildung 3-5), ist ein Beispiel für einen solchen validierenden Parser, der gleichzeitig auch als normaler Parser verwendet werden kann. Neben der reinen Überprüfung zeigt das Programm auch die Struktur durch Einrückung der entsprechenden Tags an. Außerdem werden das Markup, Kommentare und der eigentliche Inhalt in unterschiedlichen Farben dargestellt.

Ein weiterer Vorteil ist die Umgebung, in welcher der XML-Parser zum Einsatz kommt. Er funktioniert zwar ausschließlich im Internet Explorer (ab Version 4.0) von Microsoft, bietet dem Benutzer aber gerade deshalb eine intuitive Bedienung.

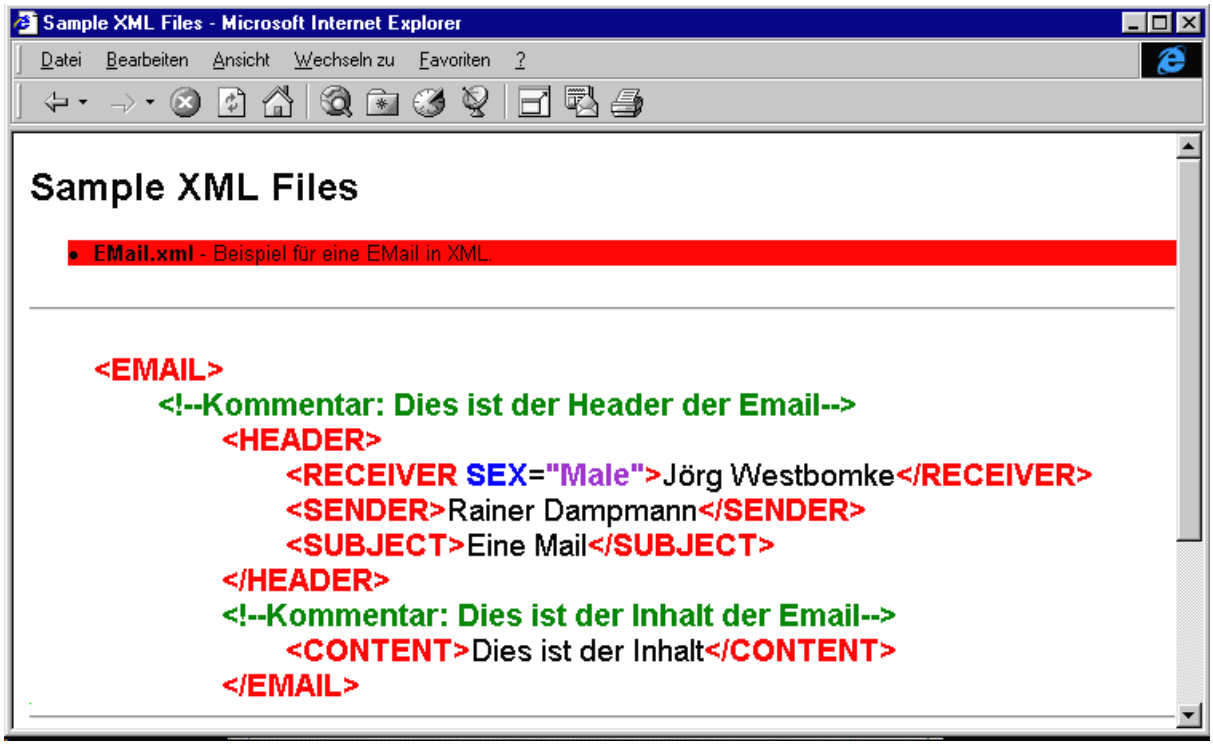


Abbildung 3-5: XML-Parser von Microsoft

Tritt ein Fehler im Dokument auf, so wird dieser mittels einer Fehlermeldung auf dem Bildschirm angezeigt. Allerdings stimmt der gemeldete Fehler, wie in Abbildung 3-6, nicht immer mit dem im Dokument vorhandenen Fehler überein. Insbesondere bei Elementen, die aus einer Elementengruppe bestehen, kommt es immer wieder zu ungenauen Fehlermeldungen.

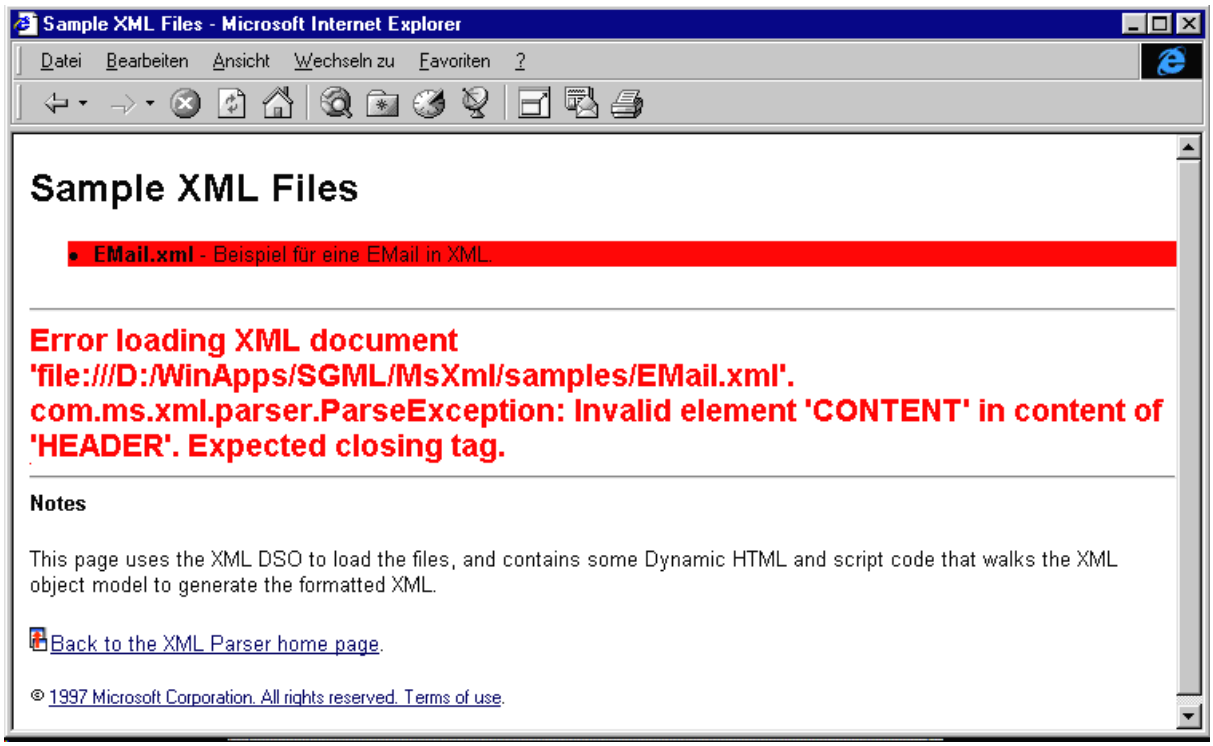


Abbildung 3-6: Fehlermeldung des XML-Parsers

Der erwähnte Vorteil der intuitiven Bedienung durch die wohlbekannte Oberfläche ist nur auf den

Internet Explorer bezogen. In anderen Browsern ist der XML-Parser nicht einzusetzen.

3.3 Browser

Browser dienen zum Anzeigen der Informationen eines Dokuments auf dem Bildschirm und ermöglichen eine Navigation über die erwähnten Links. Als Beispiel sei hier nur der Internet Explorer 5.0 von Microsoft (Abbildung 3-7) angeführt.

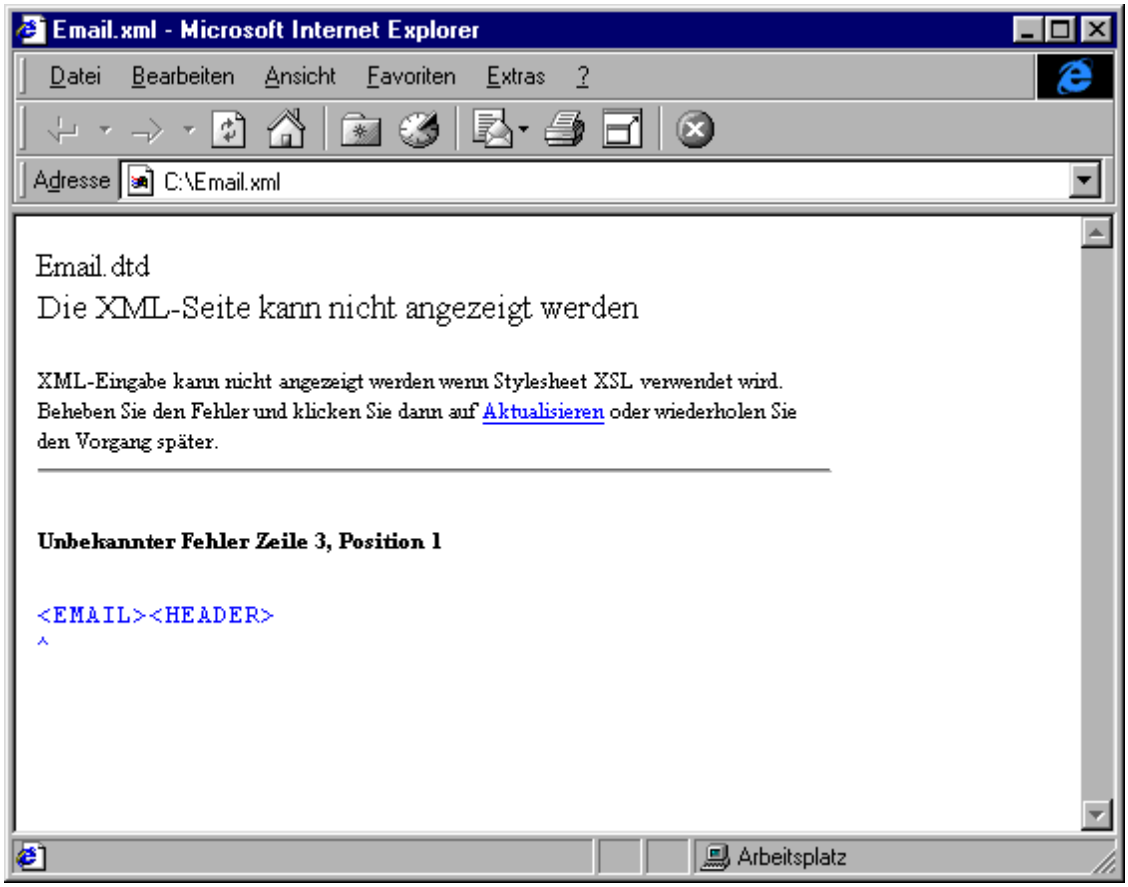


Abbildung 3-7: Internet Explorer 5.0 von Microsoft

Doch wie schon in der Abbildung 3-7 zu sehen, tauchen erste Probleme auf. Der Browser kann das Dokument nicht anzeigen, obwohl es zuvor vom Parser der gleichen Firma als korrekt überprüft worden ist. Dies kann zum einem am Stadium des Browser liegen, der zum Zeitpunkt des Seminars noch in der Beta-Version vorlag. Zum anderen könnte es bedeuten, daß der Standard, wie er vom W3-Consortium vorgeschlagen worden ist, schon jetzt wieder durch die verschiedenen Firmen unterschiedlich interpretiert und umgesetzt wird. Ein Beispiel aus jüngster Zeit ist HTML, welches in unterschiedlichen Browser auch unterschiedliche Funktionen zur Verfügung stellt. Netscape, Microsofts schärfster Rivale auf dem Browser-Markt, hat angekündigt, XML im Communicator 5.0 vollständig und standardgetreu zu unterstützen. So wird sich wohl erst in Zukunft zeigen, ob die Einheit der Sprache erhalten bleibt.

Eine weitere Frage, die nur am Rande mit der Sprache XML zusammenhängt, ist die Übertragung zu den Browsern. Bei lokalen Dateien ergeben sich keine Probleme. Ein Dokument in XML besteht, wenn der Standard erst verabschiedet ist, aus mindestens drei unterschiedlichen Dateien. Neben dem Dokument und der DTD kommt zusätzlich die Datei mit der Stilkomponente hinzu. Die dort und in der DTD beschriebenen Objekte müssen aber nicht alle im Dokument verwendet werden, so daß eventuell viel zuviel Information übertragen wird. Ein Ausweg wäre die

Konvertierung auf Serverseite in eine HTML-Datei. Dies würde zum einen die XML-Browser in Frage stellen, zum anderen müßte auf Serverseite eine Mehrfachhaltung gleicher Daten vermieden werden. An dieser Stelle besteht sicherlich noch Handlungsbedarf.

3.4 Weitere Werkzeuge

Neben den Editoren, Parsern und Browsern, die sicherlich auch für die Stilkomponente von XML zur Verfügung stehen sollten, gibt es noch eine Reihe weiterer Werkzeuge. So wären zum Beispiel auch Komplettpakete denkbar, ähnlich dem Netscape Communicator mit seinen zahlreichen Anwendungen. Dieser bietet für HTML Werkzeuge von der Erstellung bis zur Darstellung.

Doch XML ist nicht ausschließlich für das Internet entwickelt worden. Die Sprache gilt zwar als Nachfolger von HTML, bietet aber viel mehr Funktionalität und ist in anderen Bereichen ebenso einsetzbar. In naher Zukunft wird es wohl Konvertierungsprogramme geben, um beispielsweise die Dokument auf eine Ausgabe auf Drucker oder CD vorzubereiten. Auch Konvertierungen in andere Formate oder in relationale Datenbanken sind denkbar.

Damit ist aber schon die Brücke in die Zukunft von XML geschlagen.

4 Zukunftsaussichten von XML

Die Markup-Sprache XML soll einmal die Nachfolge von HTML als Internetsprache antreten. Dazu wäre es wichtig, die Vorteile von XML gegenüber HTML zu kennen.

HTML hat eine begrenzte Anzahl von Tags. Es können keine eigenen Tags definiert werden. Damit ist es in HTML nicht möglich, eigene Strukturen und damit Strukturinformationen in ein Dokument einzufügen.

SGML hingegen läßt frei definierbare Tags zu und somit auch eine Strukturierung der Daten. Allerdings ist SGML eine sehr umfangreiche und vor allem eine schwer zu lernende und lesende Sprache.

XML soll die Vorteile beider Sprachen vereinen. Sie bietet zum einen frei definierbare Tags, die eine Strukturierung der Daten ermöglichen. Zum anderen ist sie aber so einfach gehalten (im Vergleich zu SGML), daß ein potentieller Benutzer nicht zu viel Zeit benötigt, um sich in XML einzuarbeiten.

Damit stehen die Grundvoraussetzungen für eine positive Entwicklung der Sprache recht gut. Hinzu kommt noch die Unterstützung durch die Industrie. Die meisten großen Software-Firmen haben ihre Absicht erklärt, XML zu unterstützen. Neben den schon genannten Firmen Microsoft und Netscape bekunden dies so namhafte Firmen wie IBM, Sun, Adobe und Oracle.

Mit Oracle unterstützt dann auch eins der wohl bekanntesten Datenbankprogramme XML, was zu einem weiteren Schub führen dürfte. Durch die Datenbankanbindung wird der Bereich des Electronic Commerce weiter wachsen. Denn hinter all den elektronischen Geschäften steht meist eine große Liste mit Produkten und Preisen, aus denen der Kunde auswählen kann. Bis zum heutigen Tage wird unter HTML meist mittels Java oder Javascript auf die entsprechenden Daten zugegriffen. Dies könnte durch XML überflüssig werden.

Abhängig ist die Zukunft aber vor allem von der Einheit der Sprache. Kommt es zu verschiedenen Interpretationen des Standards oder der Erweiterungen, so wird auch XML wie HTML nicht mehr plattform-, applikations- und systemunabhängig sein.

Doch nicht nur die Software-Häuser arbeiten an der Umsetzung von XML. Auch das W3-Consortium treibt die Weiterentwicklung voran. Mittlerweile existieren dort sechs Gruppen, die sich um die Zukunft von XML bemühen.

Die XML Coordination Group übernimmt dabei, wie der Name schon vermuten läßt, die Organisation und Koordination der einzelnen Gruppen. Sie weist den Gruppen die Aufgaben zu, kümmert sich um den Arbeitsfluß und schlichtet in Fällen, die zwischen zwei Bereichen liegen.

Die XML Schema Working Group beschäftigt sich mit der automatischen Verarbeitung von Dokumenten. Dazu ist eine schärfere Reglementierung im Bereich von Elementen, Attributen usw. vonnöten. Deshalb arbeitet die Gruppe an Definitionen für die Struktur, den Inhalt und die Semantik.

Die XML Linking Working Group hat die Aufgabe, das Linkkonzept für XML zu entwickeln. Hier wird in Kürze der endgültige Entwurf sowohl für XPointer als auch für XLink vorgelegt werden.

Die XML Information Set Working Group setzt sich mit den Applikationen und deren Kompatibilität auseinander. So werden hier abstraktere Beschreibungen für die Sprache gesucht, die dann in den Applikationen umgesetzt werden können.

Die XML Fragment Working Group sucht nach einer Möglichkeit, Aktualisierungen möglichst effizient zu gestalten. So soll nicht bei jeder kleinen Änderung gleich das gesamte Dokument beispielsweise zum Kunden geschickt werden, sondern nur der geänderte Teil. Dieser wird dann automatisch an der richtigen Stelle eingefügt.

Die letzte der Gruppen ist die XML Syntax Working Group. Sie hält die Spezifikation auf dem neuesten Stand und paßt gegebenenfalls die Spezifikation an neue Anforderungen seitens der Anwender oder der Informationstechnik an. Zu ihrem Aufgabenbereich zählt auch die Entwicklung der Stilkomponente, wo ebenfalls Anfang 1999 als endgültige Version verfügbar sein soll.

Alles in allem beschäftigen sich eine große Anzahl von Personen mit der Weiterentwicklung von XML. Die Anstrengungen, die seit der ersten Empfehlung am 10. Februar 1998 unternommen wurden, deuten auf eine rasante Entwicklung von XML hin. Dies wird noch durch die vielen Forschungseinrichtungen und Universitäten vorangetrieben, die schon jetzt über eine Umstellung ihrer Online-Angebote in Lehre und Forschung nachdenken. Bleibt zu hoffen, daß XML nicht wie HTML nach kurzer Zeit durch firmenspezifische Interpretationen vom Standard zum Stückwerk wird.

5 Literatur

[BeMi 98] Behme, H. - Mintert, S. : XML in der Praxis, Addison-Wesley Longman Verlag 1998

[Bosa 97] Bosak, J. : XML, Java and the future of the Web, http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm [Stand: 10.12.1998]

[Brya 97] Bryan, M. : Web SGML and HTML 4.0 explained, http://www.personal.u-net.com/~sgml/book/home.htm [Stand: 10.12.1998]

[Clar 97] Clark, J. : Comparison of SGML and XML, http://www.w3.org/TR/NOTE-sgml-xml-971215 [Stand: 10.12.1998]

[ClDe 98] Clark, J. - Deach, S. : Extensible Stylesheet Language, http://www.w3.org/TR/WD-xsl [Stand: 10.12.1998]

[CoBr 98] Connolly, D. - Bray, T. : XML Activity, http://www.w3.org/XML/Activity.html [Stand: 10.12.1998]

[Gars 98] Garshol, L.M. : Free XML software, http://birk105.studby.uio.no/linker/XMLtools.html [Stand: 10.12.1998]

[MaRo 98] Maler, E. - DeRose, S. : XML Linking Language (XLink), http://www.w3.org/TR/1998/WD-xlink-19980303 [Stand: 10.12.1998]

[MaRo 98a] Maler, E. - DeRose, S. : XML Pointer Language (XPointer), http://www.w3.org/TR/1998/WD-xptr-19980303 [Stand: 10.12.1998]

[TrWa 95] Travis, B. - Waldt, D. : The SGML Implementation Guide, Springer Verlag 1995

[BrPS 98] Bray, T. - Paoli, J. - Sperberg-McQueen, C.M., Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/1998/REC-xml-1998-19980210 [Stand: 10.12.1998]