DTDs

Autor: Daniel Vogtland

e-mail: kontakt@daniel-vogtland.de

Begutachterin: Prof. Dr. Morik

Abgabe: 18.8.2003

1 Auszeichnungssprachen

Ein Dokument kann grundsätzlich beliebige Daten enthalten, doch im Zusammenhang mit Auszeichnungssprachen ist es sinnvoll, sich auf Textdokumente beschränken. Das schließt das Beinhalten anderer Daten (Bilder, akustische Signale, Videos etc.) keinesfalls aus, jedoch ist der vorherrschende Datentyp (Menschen-lesbarer) Text. (MINTERT, 1998)
Jedes Dokument besteht nach Mintert aus drei Komponenten: *Inhalt*, *Informationen zur (visuellen) Darstellung*, und die *(logische) Struktur*, die in „Computer-verständlicher" Form Teil des Dokuments sein müssen (MINTERT, 1998).
Die **Auszeichnungssprachen** (Markup Languages) nehmen eine explizite Trennung dieser Bereiche vor. Von einer anderen Perspektive aus betrachtet kann man auch sagen: Auszeichnungssprachen trennen die Verarbeitung, Darstellung (Präsentation) und Beschreibung von Information. Eine weitere wichtige Eigenschaft ist die *Plattformunabhängigkeit*.
Die populärste Auszeichnungssprache ist XML (Extensible Markup Language). Aber auch die viel ältere SGML (Standard Generalized Markup Language) ist heute noch gebräuchlich. (MINTERT, 1998) Bei der WML (Wireless Markup Language) als Bestandteil des WAP Protokollstacks handelt es sich ebenfalls um eine Auszeichnungssprache (WAP FORUM, 2000).

2 Werkzeuge für Auszeichnungssprachen

Die Werkzeuge der Verarbeitung stellen neben den Endgeräten (Bildschirm, Drucker, usw.) vor allem Browser und Editoren dar. Neben gewöhnlichen Texteditoren (emacs, winedit, …) existieren auch spezielle Editoren wie beispielsweise XMetal oder XMLSpy, die den Benutzer bei der Erstellung und Bearbeitung unter Berücksichtigung der getrennten Komponenten eines Dokuments (s.o.) unterstützen. Natürlich können auch beliebige Applikationen (das JDK bietet beispielsweise eine ansatzweise XML-Unterstützung) die Verarbeitung von Auszeichnungssprachen durchführen. Bei einem XML-Prozessor[1] handelt es sich ebenfalls um eine Applikation.
Um beim Umgang mit Auszeichnungssprachen eine Menge von Grundfunktionen (API) zur Verfügung zu stellen, wurde die Entwicklung an einem generellen Modell aufgenommen. Hierbei handelt es sich das *DOM* (Document Object Model), welches Objektunterstützung bietet und derzeit für XML und HTML (in Verbindung mit JavaScript) realisiert ist (MINTERT, 1998). Seit dem 13. November 2000 ist die Version 2.0 des DOM eine offizielle W3C[2]-Empfehlung (MÜNZ, 2001).

3 Präsentation

Die Separierung der Darstellungsinformation ermöglicht einerseits unterschiedliche „Sichten" desselben Dokuments für verschiedene Medien (Bildschirm, Drucker, Sprachausgabe, …). Auf der anderen Seite ist so auch eine Standardisierung möglich, so dass gleiche Informationstypen aus unterschiedlichen Dokumenten auch gleich dargestellt werden.
Für die Präsentation existieren diverse Standards. Der einfachste besteht in der Verwendung von *Stylesheets* wie man sie schon aus dem klassischen HTML basiertem Webdesign kennt. Mehr Möglichkeiten bietet XSL (Extensible Stylesheet Language), welche aus den Teilsprachen XSLT (XSL-Transform),selbst wieder eine XML-Auszeichnungssprache, und XPath, die ein Navigieren in der logischen Struktur des Dokuments über Baumstrukturen ermöglicht, besteht. Spezifikationen sind beim W3C zu finden. XSL hat oft die Transformation eines Typs von Dokument in einen anderen Dokumententyp als Ziel (SELIGMANN, 2001). Als Beispiel soll das *Hidden Web* dienen: Ein Internetbenutzer mit einem alten Browser (der kein XML unterstützt) möchte die Informationen einer bestimmten Seite abrufen. Diese ist auf dem Server in XML gespeichert. Nun wird die Anfrage über ein beliebiges Script (beispielsweise CGI) an den XSL-Prozessor geleitet. Die XSL wandelt die XML Daten dann in gewöhnliche HTML Daten um, welche an den Benutzer gesendet werden. Nun kann dieser die gewünschten Informationen in seinem Browser betrachten.
Die komplexesten Präsentationsmöglichkeiten sind nach Mintert mit der DSSL (Document Style Semantics and Specification Language) gegeben. Hierbei handelt es sich um eine Sprache mit Elementen der Programmiersprache Lisp (List Processing), die auch Rekursionen zulässt. Sie gilt als Grundlage von XSL. (MINTERT, 1998)

1 XML-Prozessor: Ein Parser für XML, also ein Programm dessen Aufgabe im Einlesen und Auswerten (z.B. Validierung) von XML-Dokumenten besteht.
2 W3C (World Wide Web Consortium): Ende 1994 gegründete Organisation, die es sich zur Aufgabe gemacht hat, die Standards für das Web zu entwickeln. Zu den Mitgliedern zählen hauptsächlich Firmen, die selbst an der Entwicklung beteiligt sind. (MINTERT, 1998)

4 DTD

Im Gegensatz zum klassischen WYSIWYG („what you see is what you get“) -Konzept, wie es von Mintert beschrieben wird, werden die verschiedenen Präsentationselemente nicht an eindeutigen visuellen Merkmalen wie Fettschrift oder Punktgröße festgemacht, was der Trennung von Struktur und Darstellung widersprechen würde, sondern an logischen Elementen bzw. der logischen Struktur eines Dokuments (MINTERT, 1998).
Gerade diese, man kann sie auch als *Grammatik* bezeichnen (HEFLIN, 2000) (MINTERT, 1998), wird bei den Auszeichnungssprachen durch die **DTD** (Document Type Definition) definiert. Genauer ausgedrückt legt eine DTD die logischen Elemente und ihre Beziehungen untereinander für einen Dokumenttyp, also eine **Klasse von Dokumenten**, fest (MINTERT, 1998). Dabei stellt jedes Dokument eine Instanz dieser Klasse dar.
Diese logischen Strukturinformationen werden durch Metadaten in einem Dokument beschrieben. Als Beispiel kann man sich einen Buchtext vorstellen: In einem klassischem visuell orientierten Programm würde der Titel groß und fett gedruckt, die Kapitel nur fett mit vorangestellter Nummerierung und der Absatz in der gewählten Standardschrift. Mit zunehmender Komplexität ist es für den Computer fast unmöglich, anhand bestimmter Schriftstile festzustellen, um welches logische Element es sich handelt (MINTERT, 1998). Bei einem Auszeichnungssprachenelement dagegen werden die reinen Inhaltsinformationen in umgebende *Auszeichnungen* (die aus XML und SGML die aus HTML bekannten „tags“) gekapselt. Diese Auszeichnungen wären beispielsweise „Titel“, „Kapitel“ oder „Absatz“. Ihre Darstellung wird durch die unabhängigen Präsentationsinformationen festgelegt und kann je nach Wunsch variieren, den Inhalt sogar transformieren (automatische Kapitelnummerierung, automatische Erstellung eines Inhaltsverzeichnisses).
Da die DTD die logische Struktur fest definiert ist eine einfache Validierung von Dokumenten möglich. So kann das Zusammenspiel mit der Darstellungskomponente realisiert werden, die auf alle logischen Elemente und deren Schachtelungen eingehen kann.
Neben der Strukturnutzung für die Präsentation ist auch Benutzer-abhängige Navigation möglich. Diese wird in XML durch *XPointer* innerhalb einer Instanz und Instanz-übergreifend durch *XLink* realisiert. (MINTERT, 1998)
Auch die Übersetzung von der Instanz eines Dokumenttyps in eine Instanz eines anderen ist durch Kenntnis der beiden DTDs automatisierbar. (z.B. mit XSL)

Eine wichtige Funktion der DTD ist auch die Möglichkeit der Standardisierung. Alle Instanzen einer Dokumentenklasse befolgen die gleichen logischen Regeln. Indem die DTD zentral definiert wird und jede Instanz darauf zugreift wird nicht nur Speicherplatz gespart, sondern es ist auch gewährleitstet, dass alle Instanzen wirklich exakt derselben Klasse angehören und deren Eigenschaften zentral geändert werden können (eine sanfte Form der maximalen Änderungslokalität). Ein weiterer Aspekt einer solchen Zentralisierung ist die Bereitstellung globaler Informationen, die bei Bedarf durch eine lokale Änderung aktualisiert werden können. (MINTERT, 1998) Als Beispiel mag ein Webprojekt dienen, in dem mehrere Seiten auf gemeinsame andere Seiten verweisen. Bei einer veränderten url[1] muss nicht jede Seite einzeln bearbeitet werden, sondern es genügt eine Aktualisierung der zentralen linklist[2], welche in einer DTD festgehalten wird.
XML mit der DTD als Regelbeschreibung für spezifische Dokumenttypen im Plattform-unabhängigen Textformat findet heute häufige Verwendung als Datenaustauschformat im eCommerce (ABERER, 2001). Verteilte Anwendungen nutzen DTDs zur Kommunikation innerhalb von Client/Server-Architekturen und das GRID-computing[3] nutzt XML-RPC als Kommunikationsprotokoll (FOSTER, 2001).
Standardisierung führte zur Entwicklung *fester* DTDs. Diese Dokumenttyp-Definitionen sind meist öffentlich verfügbar und werden von einer Vielzahl Entwickler ohne Transformation genutzt. DUBLIN CORE ist eine SGML DTD, die zum Katalogisieren der Medien in Bibliotheken benutzt wird (JEFFERY, 1998). SMIL dient der Einbindung von Multimedia-Objekten (Pixelgrafiken, Videos, Audiodateien, …) in XML und deren Zusammenführung in Form von Animationen. SVG (Scalable Vector Graphics) realisiert skalierbare Vektorgrafiken mit der Möglichkeit des Einbindens von Pixelgrafiken in XML. In Nachrichtenagenturen werden mit der SGML-NITF und der XML-NewsML ebenfalls feste DTDs genutzt. Und auch beim eLearning existieren feste DTDs zum Zusammenführen von unterschiedlichen Lerninhalten bzw. deren Dokumenten. Eine Umsetzung von standardisierten Dokumentationsaufbauten durch DTDs war bereits 1998 für SGML durch DocBook und LinuxDoc

1 url (Uniform Resource Locator): Eindeutige Adresse im Internet, die einen Verweis auf ein bestimmtes Objekt (HTML-Datei, JPG-Bild, …) liefert.
2 linklist: Umgangssprachliche Bezeichnung für eine Liste von urls, die meist auf andere Websites verweisen.
3 GRID-computing: Mehrere Rechner teilen sich zumindest teilweise ihre Ressourcen (Speicher, CPU, …) um gemeinsam eine übergeordnete Aufgabe zu bearbeiten.

gegeben. (MINTERT, 1998)
Der Protokollstack des WAP nutzt ebenfalls feste DTDs (z.B. WML).

Die DTD legt *syntaktischen* Regeln fest, definiert jedoch *keine semantischen* Bedingungen. Da Auszeichnungssprachen auf „Menschen-lesbarer" Textcodierung arbeiten, könnte man schon allein durch die Namensgebung der logischen Elemente eine implizite semantische Bedeutung zumindest für den menschlichen Bearbeiter annehmen. Doch nicht einmal dies ist der Fall: Der Entwickler einer DTD wird seine persönliche semantische Interpretation bei der Benennung der Elemente einfließen lassen. Doch diese Interpretation ist abhängig von seinen individuellen Umweltbedingungen und seiner Erfahrung, also der aktuellen „Perspektive auf die Realität" oder auch „Sicht der Welt". Der Computer mag nun als Übermittler einer Instanz basierend auf dieser DTD zu einer anderen Person dienen, ohne die Möglichkeit eine ähnliche Interpretation nachzuvollziehen (Computer sind im Gegensatz zum Menschen rein Logik-orientiert). Die andere Person empfängt das Dokument und wertet nun die DTD im Hinblick auf mögliche semantische Bedingungen aus. Dazu muss sie jedoch ihre eigene „Perspektive der Realität" benutzen. In der Regel unterscheiden sich die Perspektiven verschiedener Menschen,- die Ideen des DTD Entwicklers sind für den Empfänger nicht implizit erschließbar.
Es muss also eine allgemeingültige Beschreibung der Semantik gelten und sie sollte auch Computer-verständlich sein. Um diese Funktionalität zu ermöglichen wird auf die DTD eine **Ontologie**, formuliert in einer KR (Knowledge Representation) Sprache wie beispielsweise SHOE (Simple HTML Ontology Extensions), aufgesetzt (HEFLIN, 2000). Ontologien arbeiten gewöhnlich mit Prädikatenlogik und so können semantische Bedingungen in eindeutiger Form definiert werden.
Ein praktischer Nutzen von Einbeziehung semantischer Bedingungen in Dokumente wäre beispielsweise eine (wirkliche) Strukturierung des Internets nach Inhalten, also das so genannte *Semantic Web*. Zur Realisierung würden sich unter anderem DAML und OIL anbieten (FENSEL, 2001). Durch konsequentes Einsetzen von DTDs und Ontologien würden Suchmaschinen wahrscheinlich erwünschte Informationen gezielter auffinden können. Leider unterstützen DTDs keine *Namensräume*, doch mit Ablösung der DTD durch seinen Nachfolger, dem XML *Schema* wird dieses Manko behoben werden.
Die Semantik, die durch das Zusammenwirken von Ontologien und DTDs bereitgestellt wird, bietet auch Unterstützung zur Realisierung von elektronischen Agenten (meist kleine Computerprogramme, die im Fall von Multi-Agenten-Systemen auch interagieren) die den Benutzer beim Auffinden und Zusammenfassen von Informationen unterstützen. So benutzt FIPA[1] in seiner ACL (Agent Communication Language) auf Modallogik basierende Ontologien (LABROU, 1999). Mögliche Anwendungsbereiche von Agentensystemen sind das Information Retrievel, Datenbank Management und Knowledge Sharing.
Im Zusammenhang von DTDs und Wissensmanagement ist vor allem die XTM (XML Topic Map) interessant. In der Realisierung ist ein Navigieren des Benutzers auf Informationsdaten in Form von stark verästelten Kreis-beinhaltenden Graphen möglich. Egal an welcher Stelle sich der Benutzer befindet, seine Perspektive stellt immer den Mittelpunkt des Informationsgraphen dar, von dem aus er in beliebige Richtung navigieren kann. Als ein Beispiel kann man das 1984-1992 von Klix[2] erweiterte semantische Netzmodell[3] (ein Modell für die Repräsentation von Begriffen in der Allgemeinen Psychologie) betrachten, das auch auf diesem Ansatz basiert.

5 DTD Deklaration

Der grundsätzliche Aufbau einer DTD folgt unabhängig von der gewählten Sprachvariante dem gleichen Prinzip (MINTERT, 1998):
Die DTD definiert folgende Eigenschaften der Dokumentenklasse:
- Welche **Elementtypen** gibt es und welchen Inhalt dürfen sie annehmen?
- Welche **Attribute** gibt es und welche Werte dürfen sie annehmen?

Elementtypen stellen die logischen Elemente der Dokumentenklasse zur Beschreibung von Teilen des eigentlichen Inhalts und Attribute wiederum eine Beschreibung der Elemttypen (also Meta-Meta-Daten) dar.
In diesem Text wird der Aufbau einer XML DTD erläutert, die nach Mintert den SGML Bestimmungen relativ ähnlich ist. Jedoch sind sie nicht kompatibel, da XML in gewisser Weise nur eine Teilmenge von SGML darstellt und nicht alle Funktionen unterstützt. (MINTERT, 1998) Detaillierte Informationen

1 FIPA (Foundation for Intelligent Physical Agents): Eine nicht kommerzielle Institution, die sich die Schaffung von Standards für Software-Agenten zur Aufgabe gemacht hat.
2 Klix, Friedhart: International anerkannter Psychologe, studierte auch Mathematik. 1927 in Sachsen geboren.
3 semantisches Netzmodell: Eine Beschreibung findet man beispielsweise in dem Vorlesungsscript „Allgemeine Psychologie 1", Kapitel „Der Ansatz von KLIX" von Prof. Dr. Metz-Göckel (METZ-GÖCKEL, 1997).

findet man in den entsprechenden Spezifikationen des W3C.
Die allgemeinen Syntaxregeln entsprechen weitgehend denen einer XML-Instanz (<!...> tags), Abweichungen sollten nach der XML Spezifikation 1.0 von einem Verarbeitungsprozess nicht akzeptiert werden. Im Gegensatz zu SGML wird bei XML zwischen Groß- und Kleinschreibung unterschieden. D.h. das Element „Titel“ ist von dem Element „titel“ verschieden. Die Namensgebung der Elemente muss eindeutig sein.
Die verwendete Kodierung wird von Mintert wie folgt angegeben:
Der Zeichensatz ist bestimmt durch die Spezifikation in ISO/IEC 10646. Gültige Zeichen sind somit Tab (Tabulator), Carriage Return (Wagenrücklauf), Line Feed (Zeilenvorschub) sowie die Grafikzeichen von Unicode und ISO/IEC 10646. Von der Verwendung von »Kompatibilitätszeichen« (compatibility characters), wie sie in Abschnitt 6.8 von [Unicode] definiert werden, wird abgeraten. (MINTERT, 1998)

Die verwendete DTD eines XML Dokuments werden in dessen **Dokumenttyp-Deklaration**, welche die Klasse eines Dokumentes bestimmt angegeben. Dabei ist es in der XML auch möglich, keine DTD anzugeben. Entspricht das Dokument der XML-Syntax gilt es als *wohlgeformt*. Wird eine DTD angeben und werden ihre Definitionen nicht verletzt, so gilt das Dokument zusätzlich als *gültig*. (MINTERT, 1998)
DTDs können intern in einem Dokument, also in einer Instanz, definiert sein. Zentralisierung wird alternativ über das Einbinden einer externen DTD realisiert. Bedingung für die Nutzung einer externen DTD ist die Unterstützung von *Processing Instructions*[1] durch die verarbeitende Software.

Die DTD beginnt mit der **Elementtyp-Deklaration**, der wichtigsten Komponente. (MINTERT, 1998) Sie listet die logischen Elemente einer Dokumentenklasse auf. Instanzen dieser Dokumentenklasse können keine Elemente enthalten, die nicht an dieser Stelle deklariert wurden.

Eine Deklaration hat die folgende Syntax:
<!ELEMENT *Name* (*Inhalt*)>

Name setzt die Bezeichnung für das Element, z.B. Kapitel oder Titel. Ein Name beginnt mit einem Buchstaben, einem Unterstrich oder einem Doppelpunkt. Als weitere Symbole sind dann auch Zahlen, der Bindestrich und der Punkt gestattet. (MINTERT, 1998)
Inhalt bestimmt den Inhalt des Elements. Es sind drei Basis-Inhaltstypen definiert:

1. **ANY**
   Dieser Inhaltstyp kann beliebige Elemente aufnehmen. Wichtig ist, dass dieser Inhalt nicht geparsed wird. Das bedeutet, es findet keine Fehlerüberprüfung statt. (MARUGG, 2002)
2. **EMPTY**
   Dieser Inhaltstyp ist ein leerer Inhalt. Keine Daten können eingetragen werden (MINTERT, 1998).
3. **MIXED**
   Dieser Inhaltstyp wird meistens verwendet. Er kann beliebige gültige Elemente aufnehmen. Dies können Zeichenketten (#PCDATA) und andere Elemttypen (*children*) sein. Auch reine Zeichenketten werden laut Spezifikation durch einen mixed-Inhalt vereinbart (MARUGG, 2002).

   Durch folgende Operatoren kann der Inhalt näher bestimmt werden (x ist das Element):

   Wiederholungsoperatoren (Systax: xOP)
   (kein Operator) x kommt genau einmal vor. (1)
   + x kommt mindestens einmal vor. (1..*)
   * x kommt keinmal oder beliebig vor. (0..*)
   ? x kommt genau einmal oder keinmal vor. (0..1)
   Verkettungsoperatoren (Syntax: x1 OP x2)
   , Sequenz, die Elemente tauchen in dieser Reihenfolge auf.
   | Alternative, genau eines dieser Element muss zugewiesen werden (XOR).

   Durch Klammerung können komplexere Strukturen geschaffen werden (MARUGG, 2002).

   Bei gemischter Form von Zeichenkette und Element gelten die Einschränkungen
   - Die Zeichenkette muss zu anderen Elementen alternativ abgegrenzt werden. (|)
   - Der Inhaltsausdruck muss mit dem * - Operator enden.
   Um diese Einschränkungen zu umgehen, können Zeichenketten in eigenen Elementen gekapselt

1 Processing Instructions (PIs): Direkte Anweisungen für eine Anwendung die ein (XML-) Dokument verarbeitet als Teil des Dokuments selbst.

werden (MARUGG, 2002).

Einige Beispiele für Elementtyp-Deklarationen:

```
<!ELEMENT anyElement ANY> <!-- beliebiger Inhalt, keine Fehlerüberprüfung -->
<!ELEMENT img EMPTY>  <!-- kein Inhalt, vgl. IMG-tag von HTML -->
<!ELEMENT text (#PCDATA)>     <!-- Zeichenkette -->
<!ELEMENT slave (master)>          <!-- ein slave ist genau einem master zugeordnet -->
<!ELEMENT person (vorname, name, adresse)>  <!--... eine Person -->
<!ELEMENT person (anrede?,vorname, name, adresse)>  <!--... eine Person mit optionaler Anrede-->
<!ELEMENT gedicht (vers)+>  <!--... ein Gedicht, das aus mindestens einem Vers besteht-->
<!ELEMENT liste (eintrag)*>  <!--... eine Liste, die beliebig viele Einträge enthalten kann, auch keinen
                              -->
<!ELEMENT  ware(verkaufsware | ausstellungsstueck)>  <!--... eine Ware, die entweder zum Verkauf
                                                      aussteht oder als Ausstellungsstück dient
                                                      -->
<!ELEMENT kapitel (#PCDATA | abschnitt | titel)*>            <!-- gemischte Form mit Zeichenkette -->
<!ELEMENT  strasse  (bezeichnung?,(spur+  |  planung))>     <!-- Eine Strasse, die eine optionale
                                                             Bezeichnung hat und entweder aus
                                                             mindetens einer Spur besteht oder nur
                                                             geplant ist -->
```

Nach der Elementtyp-Deklaration folgt die **Attributlisten-Deklaration**. Sie bestimmt die Listen der Attribute für jeweils ein Element. Eine Deklaration hat die folgende Syntax :

```
<!ATTLIST elementname
  name     typ     zuweisung
   ...
>
```

elementname gibt an, auf welches logische Element sich die Attributliste bezieht. Das Element muss bereits deklariert sein (MINTERT, 1998). Danach folgt eine beliebig lange Liste von Attributen.
name bestimmt den den Namen des Attributs. In SGML muss der Name für jedes Element einmalig sein. In XML ist dagegen auch eine Mehrfachdefinition zulässig. Allerdings darf in jeder Definition jeder Attributname nur einmal vorkommen. Es gelten dieselben Regeln wie bei der Festlegung des Namens für ein Element bei dessen Deklaration. (MINTERT, 1998)
typ bestimmt den zulässigen Inhalt des Attributs. Folgende Schlüsselwörter stehen zur Verfügung:

CDATA: Beliebige Zeichenketten (analog zu #PCDATA bei Elementdeklarationen).

ID: Die ID (Identifier) beinhaltet einen eindeutigen Wert, mit welchem das zugehörige Element eindeutig referenziert werden kann (Primärschlüssel). Ein Elementtyp darf dabei nur genau ein Attribut vom Typ ID besitzen. Mögliche Zusätze sind #REQUIRED (ID muss gesetzt sein) oder #IMPLIED (ID ist optional). (MARUGG, 2002)

IDREF: Eine Referenz auf eine andere ID desselben Dokuments. (eine Primär-/Fremd-Schlüssel Beziehung) Dokumentübergreifende Referenzen führen zu Schwierigkeiten. Eindeutige Aussagen sind in der Spezifikation der XML 1.0 nicht zu finden. (MARUGG, 2002)

IDREFS: Entspricht IDREF, wobei jedoch mehrere Verweise - getrennt durch Leerzeichen oder Anführungszeichen - angegeben werden können. (MARUGG, 2002)

NMTOKEN: Beinhaltet einen Namenstoken. Dieses kann durch beliebige Kombinationen von Buchstaben, Ziffern, Punkten, Doppelpunkten, Unterstrichen und Bindestrichen entstehen. Doppelpunkte dürfen jedoch nicht an erster Stelle und nicht direkt hintereinander vorkommen. (MARUGG, 2002)

NMTOKENS: Beinhaltet mehrere NMTOKENs durch Leerzeichen oder Anführungszeichen getrennt. (MARUGG, 2002)

ENTITY: Der Attributwert muss dem Namen einer in der DTD definierten Entity entsprechen. (MARUGG, 2002)

ENTITIES: Beinhaltet mehrere Entitynamen der DTD durch Leerzeichen oder Anführungszeichen getrennt. (MARUGG, 2002)

Darüber hinaus ist auch noch ein *Aufzählungstyp* vorhanden. Dabei werden alle möglichen Werte in Klammern angegeben: (wert1 | wert2 | ...) Vor der Klammer kann auch das Schlüsselwort NOTATION

angegeben werden, dann sind die Vorgaben eine Liste von Notationen, welche in einer Notationsdeklaration (s.u.) deklariert werden müssen. (MARUGG, 2002)

Die Angabepflicht von Werten als Zuweisung in einem Dokument kann durch folgende Schlüsselwörter bestimmt werden:

#IMPLIED: Die Angabe ist optional und muss notfalls von der Anwendungssoftware ermittelt werden. (MINTERT, 1998)
#REQUIRED: Die Angabe muss auf jeden Fall in der Instanz gemacht werden.
#FIXED: Der Wert ist fest vorgegeben. In diesem Fall muss natürlich auch eine Vorgabe definiert werden. (MARUGG, 2002)

Im Anschluß an den Typ kann ein *Vorgabewert* angegeben werden, welcher bei einer nicht-Zuweisung automatisch gesetzt wird.

Einige Attributnamen sind in der XML *reserviert* und dürfen deshalb nicht neu deklariert werden:

xml:lang Attribut zur Sprachgebung. z.B. xml:lang = “de“
xml:space Attribut zur Behandlung von Leerräumen. Wird der Wert auf “preserve“ gesetzt, bleiben alle Leerzeichen, Zeilenvorschübe und Tabulatoren erhalten. Wird der Wert auf “default“ gesetzt, so kann die Anwendungssoftware ihre Standard-Behandlung solcher Leerräume durchführen. (MARUGG, 2002)
xml:link (außerhalb der eigentlichen W3C00-Spezifikation) Zeigt dem XML-Processor, dass es sich um ein Linkelement handelt. (MARUGG, 2002)
xml:attribute (außerhalb der eigentlichen W3C00-Spezifikation) Marugg gibt folgende Beschreibung: „Zusätzlich zum Attribut „xml:link“ unterstützt das Linking-Konzept XLink mit der Verwendung des Attributes xml:attribute auch Referenzen auf Attributebene.“ (MARUGG, 2002)

Einige Beispiele für Attributlisten-Deklarationen:

```
<!ATTLIST webadresse
  url           CDATA    #REQUIRED
  country       CDATA    #REQUIRED
  availble      CDATA    #IMPLIED
>
```

<!-- Eine Attributliste zu einem Webadresse-Element, welche die Informationen des links und des Staates in dem sich der Server befindet bereitstellt. Die Information, ob der Server erreichbar ist, muss von der Anwendungssoftware selbst ermittelt werden. -->

```
<!ATTLIST person
  kontakt      (privat | geschäftlich) privat
>
```

<!-- Eine Attributliste zu einem der obigen Personenelemente. Die Person kann entweder ein privater oder ein geschäftlicher Kontakt sein. Wird das Attribut nicht gesetzt, so ist die Vorgabe ein privater Kontakt. -->

```
<!ATTLIST film
  urheber       CDATA                 #REQUIRED
  serNoFilm     ID                    #REQUIRED
  genre         CDATA                 #IMPLIED     “Horror“
  medium        (dvd | vcd | vhs | ld) #IMPLIED     vhs
  firma         CDATA                 #FIXED       “Mein Videoversand“
>
```

<!-- Eine Attributliste zu einem Film-Element eines XML-basierten Informationsbestandes eines Videoversands. -->

```
<!ATTLIST viergewinntSpielstein
  farbe (rot | gelb) rot
>
<!-- die Attributliste eines Vier-Gewinnt Spielsteins. -->
```

Wie bereits zu Anfang erwähnt lassen sich auch Informationen zentral in einer DTD auslagern. Der Zugriff auf diese geschieht mit Hilfe von **Entity-Referenzen**, die wie in einem XML Dokument definiert werden:
<!ENTITY *name* "*inhalt*">

name bestimmt den den Namen des Entitys. In einer Instanz kann der Referenzwert dann mittels &name*;* benutzt werden. Es gelten dieselben Regeln wie bei der Festlegung des Namens für ein Element bei dessen Deklaration. (MINTERT, 1998)
inhalt beschreibt den Wert des Entitys. Es handelt sich um eine Zeichenkette.

Das Beispiel des Webprojekts demonstriert den Einsatz einer externen Entity-Referenz:

```
<!-- in der DTD definiert, hier weblink.dtd … -->
<!ENTITY gmxLink "http://www.gmx.net">
<!ELEMENT text (#PCDATA)>

<!-- es folgt das XML Dokument …  -->
<?xml version="1.0"?>
<!DOCTYPE weblink SYSTEM "weblink.dtd">
<weblink>
        <text>
                Zu GMX gelangen wir durch &gmxLink;
        </text>
</weblink>
```

Um wiederkehrende Inhaltsmodelle für Elemente zu referenzieren besteht ein ähnlicher Mechanismus wie die Entity-Referenzen: **Parameter Entity References**. (MINTERT, 1998)
<!ENTITY % *name* "*inhalt*">

Anstelle des vorangestellten & wird auf die Referenz mit einem vorangestelltem % zugegriffen. Die folgende Beispiele von Mintert demonstrieren den Einsatz:

```
<!ENTITY % block "absatz | listing | abbildung | kasten">
<!ELEMENT kapitel       (ueberschrift, (%block;)*, abschnitt+)>

<!-- Einbinden einer extern definierte Refrenz …  -->
<!ENTITY % tab-modell SYSTEM "/usr/local/sgml/tabelle.dtd">
%tab-modell;
<!ENTITY % block "absatz | listing | abbildung | kasten | tabelle">
<!ELEMENT kapitel       (ueberschrift, (%block;)*, abschnitt+)>
```

(MINTERT, 1998)

In der DTD hat man die Möglichkeit eine **Notations-Deklaration** zu definieren:
<!NOTATION *name* SYSTEM "*verarbeitendes Programm*">
<!NOTATION *name* PUBLIC "*Information zum Datentyp*">

Notationsdeklarationen spezifizieren ein Datenformat, welches von XML verschieden ist, oder identifizieren ein Programm, das ein bestimmtes Format verarbeitet. (MARUGG, 2002) Die folgenden Beispiele von Marugg demonstrieren den Einsatz:
Sie tauchen in der Verbindung mit Entities

```
        <!ENTITY bild SYTEM "xyz.gif" NDATA GIF>
        <!NOTATION GIF SYSTEM "Iexplore.exe">
```

und Attributdeklarationen auf

```
        <!NOTATION gif SYSTEM "photoshop.exe">
        <!NOTATION jpg SYSTEM "paintpro.exe">
        <!ELEMENT image (#PCDATA)>
        <!ATTLIST image type NOTATION (gif | jpg) "gif">
```

und werden folgendermaßen verwendet: <image type="gif"/>
(MARUGG, 2002)

Es ist wichtig, zu bedenken, dass die XML der PUBLIC bzw. SYSTEM Information der Deklaration keine Aufgabe zuweist. (MARUGG, 2002) Die Anwendungssoftware muss eigenständig mit diesen Informationen agieren.

Ein nützliches Feature der XML DTD ist außerdem die Möglichkeit **bedingter Abschnitte**:
In der Testphase kann es manchmal nützlich sein, bestimmte Teile der DTD aus- und andere einzublenden. (MINTERT, 1998) Die Schlüsselwörter hierzu lauten IGNORE und INCLUDE.
Die Anwendung ist denkbar einfach:

```
<![IGNORE [
        ausgeblendeter Bereich
]]>
```

```
<![INCLUDE [
        eingeblendeter Bereich
]]>
```

In dem ein- bzw ausgeblendetem Bereich werden die betroffenen Deklarationen vorgenommen. Um diese Technik effizienter zu nutzen bietet es sich auch oftmals an, Entity-Referenzen für IGNORE bzw. INCLUDE zu verwenden. Dann ist der Austausch schneller möglich. (MINTERT, 1998)

Die Documenttype-Definition ermöglicht die Erstellung von *Datenstrukturen* in einer gültigen Form. Es ist theoretisch sogar möglich recht komplexe Datenstrukturen zu entwerfen.
Als Beispiel soll eine Adressverwaltung dienen:
Eine Adresseverwaltung verwaltet verschiedene Personen.
Eine Person besitzt einen Vor- und Nachnamen, eine optionale Anrede, ein oder mehrere Adressen und ein oder mehrere Telefonnummern, so wie eine optionale e-mail Adresse.
Eine Adresse besteht aus einer Strasse (mit der zugehörigen Hausnummer), einer Postleitzahl und einer Stadt.

Es ergibt sich folgende Backus-Naur-Form (die leere Menge ∅ lässt auch Auslassungen zu und kennt nur Textzeichen):

```
Person ::=  Vorname, Nachname, Adresse*, Telefonnummer* |
            Vorname, Nachname, Adresse*, Telefonnummer*, e-mail |
            Anrede, Vorname, Nachname, Adresse*, Telefonnummer* |
            Anrede, Vorname, Nachname, Adresse *, Telefonnummer*, e-mail
Vorname ::= char* | ∅
Name ::= char* | ∅
Anrede ::= char* | ∅
Adresse ::= Strasse, Postleitzahl, Stadt
Telefonnummer ::= char* | ∅
e-mail ::= char* | ∅
Strasse ::= char* | ∅
Postleitzahl ::= char* | ∅
Stadt ::= char* | ∅
```

Diese formale Beschreibung lässt sich ohne Aufwand in eine gültige DTD:

```
<!ELEMENT  Person (Anrede?,Vorname, Nachname, (Adresse)+,(Telefonnummer)+, e-mail?)>
<!ELEMENT  Adresse (Strasse, Postleitzahl, Stadt)>
<!ELEMENT  Anrede (#PCDATA)>
<!ELEMENT  Vorname (#PCDATA)>
<!ELEMENT  Nachname (#PCDATA)>
<!ELEMENT  Telefonnummer (#PCDATA)>
<!ELEMENT  e-mail (#PCDATA)>
<!ELEMENT  Strasse (#PCDATA)>
<!ELEMENT  Postleitzahl (#PCDATA)>
<!ELEMENT  Stadt (#PCDATA)>
```

Im Grunde genommen lässt sich jede BNF, die als Literale nur Textzeichen kennt und auch die leere Menge zulässt (oder man definiert die leere Zeichenkette als ein bestimmtes Textzeichen), in einer DTD realisieren.
Selbst Rekursionen sind möglich, wie das folgende Beispiel beweist:

BNF:

```
wort ::= char*
Wortreihe ::= wort |  wortreihe
```

DTD:

```
<!ELEMENT  wort (#PCDATA)>
<!ELEMENT  wortreihe (wort | wortreihe)>
```

Dies zeigt, dass also auch recht komplexe Datenstrukturen mit Hilfe von DTDs erstellt werden können. Auch Datenstrukturdefinitionen (textbasierter) höherer Programmiersprachen lassen sich in einer DTD realisieren. Hierzu eignen sich besonders reine Datenelemente wie der Record von Pascal oder struct von C++. Es bleibt natürlich zu beachten, dass sämtliche Anwendungsbereiche (Fehlerüberprüfung, Berechnung von Werten) ausgeklammert werden müssen. Außerdem sollte man auf keinen Fall die fehlende Mächtigkeit von Auszeichnungssprachen vernachlässigen.

Java:
```
public class Anschrift implements
java.io.Serializable {
    protected String strasse=new String();
    protected String plz=new String();
    protected String ort=new String();
}
```

DTD:
```
<!ELEMENT  strasse (#PCDATA)>
<!ELEMENT  postleitzahl (#PCDATA)>
<!ELEMENT  stadt (#PCDATA)>
```

Mit Hilfe der Rekursion sind sogar einfache Baumstrukturen möglich.
Das folgende Beispiel könnte eine Person-ist-befreundet-mit Beziehung darstellen:
```
<!ELEMENT  person (name, person*)>
<!ELEMENT  name (#PCDATA)>
```

6 Entwicklung einer DTD

Eine für ein spezifisches Problem passende DTD zu entwickeln ist oft nicht einfach. Mintert beschreibt den folgenden Ablauf (MINTERT, 1998):
Am Anfang steht stets die Aufgabenstellung. Die Anforderungen müssen aus ihr heraus detailliert erarbeitet werden. Dabei ist es vielleicht nicht gerade einfach die notwendigen Abstraktionen zu leisten. Oftmals kann man die Datenstruktur nur über das zu-Hilfe-Nehmen von exemplarischen Beispieldaten erschließen. Der Schritt der Anforderungsfestlegung ist jedenfalls elementar wichtig, ohne ihn macht es kaum Sinn auch nur im Entferntesten an die Realisierung einer passenden DTD zu denken.
Besteht vielleicht die Möglichkeit eine öffentliche DTD zu benutzen? Sollte eine fremde DTD nicht in Frage kommen, bleibt nichts anderes übrig, als selbst Hand anzulegen.
Nachdem man das Problem analysiert hat, vielleicht auch nur einige aussagekräftigen Beispieldaten vorliegen hat, kann der Entwurf beginnen.

Dazu kann man folgende nützlichen Fragen durcharbeiten:
Welche Daten gibt es? Wie lassen sie sich sinnvoll zueinander abgrenzen? In welcher Beziehung stehen sie zueinander? Was ist ihre genaue Bedeutung im Hintergrund des zu bearbeitenden Problems? Wie könnten die jetzigen Erkenntnisse noch abgeändert werden? Welcher Grad von Flexibilität ist für die Daten und eventuell spätere Weiterentwicklungen notwendig?

Mintert zeigt zwei Ansätze auf, eine Struktur aufzustellen.
1. Die DTD sollte so klar wie möglich auf die Aufgabenstellung zugeschnitten sein.
2. Die DTD sollte so einfach wie möglich abänderbar sein.

Als Beispiel sei folgende Aufgabenstellung gegeben:
Eine Buchhandlung will Eigenveröffentlichungen elektronisch im Internet zur Verfügung stellen. Das Projekt soll in XML realisiert werden. Vorerst soll nur ein Buch erfasst werden. Es handelt sich um einen Kurzgeschichte umgeben von zwei Gedichten.

Lösung 1:
```
<!ELEMENT  Buch (Titel,Vorwort,GedichtA,Geschichte,GedichtB)>
<!ELEMENT  Titel (#PCDATA)>
<!ELEMENT  Vorwort (Abschnitt+)>
<!ELEMENT  GedichtA (SubTitel,(Vers+))>
<!ELEMENT  Geschichte (SubTitel,(GeschichteAbschnitt+))>
<!ELEMENT  GedichtB (SubTitel,(Vers+))>
<!ELEMENT  Abschnitt (#PCDATA)>
<!ELEMENT  GeschichteAbschnitt (#PCDATA)>
<!ELEMENT  SubTitel (#PCDATA)>
<!ELEMENT  Vers (#PCDATA)>
```

Lösung 2:
<!ELEMENT Buch (Element+)>
<!ELEMENT Element ((Element+) | #PCDATA)*>

Die erste Lösung ist ohne Zweifel zu spezifisch, nicht nur dass völlig überflüssige Elemente definiert werden, - was passiert wenn ein weiteres Buch ins Netz gestellt werden soll, das vielleicht ein reiner Gedichtsband ist?
Die zweite Lösung ist zu allgemein. Kein Benutzer wird im Nachhinein noch angenehm die so verarbeiteten Informationen zu einem umfangreichen Buch einsehen und entschlüsseln können.
Es gilt einen Kompromiss zwischen den beiden Ansätzen zu finden. Logische Konsequenz ist, dass einer der Ausgangspunkt für die Lösung sein muss.
Hier bietet sich nach Mintert eher die Möglichkeit 1 an. Durch immer mehr Beispiele wird man einen immer abstrakteren Aufbau erzielen, der allen Beispielen gerecht wird.
Der umgekehrte Weg ist schwieriger zu beenden, denn von einer generellen, allgemein-gültigen Struktur verzweigen sich die Spezialisierungen ins Unendliche. Der Überblick geht zu schnell verloren. (MINTERT, 1998)

Folgende Lösung ist akzeptabler, als die ersten beiden Versuche:
<!ELEMENT Buch (Buchtitel, Vorwort?, (Gedicht | Geschichte | Sachtext)+)>
<!ELEMENT Vorwort (Titel?, Absatz+)>
<!ELEMENT Gedicht (Titel?, Strophe+)>
<!ELEMENT Geschichte (Titel?,((Kapitel+)|(Absatz+)))>
<!ELEMENT Sachtext (Titel?,((Kapitel+)|(Absatz+)))>
<!ELEMENT Strophe (Vers+)>
<!ELEMENT Kapitel (Absatz+)>
<!ELEMENT Buchtitel (#PCDATA)>
<!ELEMENT Titel (#PCDATA)>
<!ELEMENT Vers (#PCDATA)>

Eine weitere Frage betrifft die Entscheidung ob Information als Attribut oder als Elementinhalt aufgenommen werden sollte. Mintert gibt einige Entscheidungshilfen.
In welchem Zusammenhang wird die Information benötigt? Ist sie von primärer Bedeutung (Element), eher „Mittel zum Zweck“ (Attribut) oder eine Beschreibung des Elements (Attribut)?
Attribute können nur begrenzte Strukturen aufnehmen, allerdings bieten sie die Möglichkeit eine Auswahl von festen Vorgabewerten zu liefern. Im Elementinhalt wird früher oder später auf ein PCDATA oder EMPTY-Element verzweigt werden. (MINTERT, 1998)
Es handelt sich also um ein sehr Aufgaben-spezifisches Problem und kann nicht immer eindeutig beantwortet werden. Die vorangegangenen Überlegungen helfen allerdings bei der Entscheidung.

7 Vergleich: feste DTD, individuelle DTD

Die Verwendung einer *festen* DTD würde eine Menge Arbeit abnehmen. Allerdings muss man auch hier die Einarbeitungszeit berücksichtigen. Außerdem wird eine feste DTD wohl nie ganz passend auf die spezifischen Anforderungen des Projekts zugeschnitten sein (MINTERT, 1998), schon allein die Benennung logischer Elemente mit passenden semantischen Namensbezeichnungen wird verhindert.
Es stellt sich auch die Frage, ob mit der Nutzung fester DTDs nicht ein großer Vorteil von Auszeichnungssprachen verloren geht: Die Neudefinition eines Dokumenttyps und dementsprechend eine angepasste automatisierbare Validierung der Dokumente. Ein Negativ-Beispiel stellt die HTML dar, die man mit Einschränkungen als Auszeichnungssprache auffassen könnte. Im Grunde stellt sie eine feste DTD zur Verfügung. Doch vielleicht möchte der Webdesigner einen Gedichttext in der Präsentation anders formatiert als eine Arbeit über verschiedene Literaturepochen darstellen. Mit HTML-Auszeichnungen wird ihm dies nicht oder zumindest nicht logisch gelingen.
Andererseits kann Standardisierung auch erwünscht sein, z.B. der erwähnte einheitliche Aufbau von Dokumentationen. (man vergleiche den JavaDoc-Standard zur Dokumentation von JAVA Klassen) DocBook und LinuxBook haben durchaus eine Existenzberechtigung.
Dagegen zeigt sich wiederum beim eLearning der Nachteil: Die derzeitige Lösung mit festen DTDs ähnelt einer Katalogisierung der eigentlichen Dokumente im Sinne des Dublin Core. Viel wünschenswerter wäre aber eine Lösung mit individuellen DTDs, so dass Lerninhalte einer logischen Struktur unterworfen sind. Man könnte natürlich versuchen, eine generelle alles-umfassende DTD zu entwerfen. Doch eine solche Zusammenführung führt zu Unübersichtlichkeit. Außerdem fällt es nicht schwer, sich klar zu machen, dass die Menge der möglichen Auszeichnungen eine unendliche Menge

ist. Dies beinhaltet als Konsequenzen die Unmöglichkeit alles zu umfassen, so wohl für die Forderung an sich als auch für ihre technische Realisierung. (unendlicher Speicher)
Und um noch ein letztes Argument für feste DTDs dagegen halten zu können, muss man den Blick auf die technische Realisierung von Endgeräten wenden: Wenn jedes übertragene Bit teuer ist (beispielsweise für WAP Benutzer), können sich feste DTDs als echter Vorteil erweisen. In diesem Fall ist es nämlich nicht nötig, die Dokumenttyp-Definition an den Empfänger zu übermitteln. Die zu transferierende Datenmenge wird kleiner.

Es haben also auch feste DTDs ihre Daseinsberechtigung, allerdings sollten sie weniger verwendet werden. Eine Kommunikation zwischen Partnern mit verschiedenen DTDs ist stets durch Transformation möglich. So kann jeder eine logische Struktur benutzen, die seinen spezifischen Anforderungen genau gerecht wird.

Bibliographie

(ABERER, 2001)
Karl Aberer, Andreas Wombacher, *A language for information commerce processes*
veröffentlicht im Rahmen des "Third International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems San Jose, California, USA, June 21-22, 2001"
Download: http://lsirwww.epfl.ch/publications/tr/TR2001-008.PDF
In diesem Dokument wird eine mögliche Sprache vorgestellt, die die Automatisierung im eCommerce unterstützen soll. Dabei wird auf vorhandene Elemente wie XML und ACLs eingegangen. Fachwissen vorausgesetzt (z.B. Petri-Netze). Verfügt man über dieses, ist der Text als mittelschwer bis schwer einzustufen.

(FENSEL, 2001)
Dieter Fensel, Deborah L. Mc Guinness, Frank van Harmelen, Ian Horrocks, Peter F. Patel-Schneider, *OIL: An Ontology Infrastructure for the Semantic Web*
Beim IEEE verfügbar (http://www.ieee.org).
Dieser Artikel behandelt OIL als Realisierungsmöglichkeit des Semantic Web. Begriffe sind gut erklärt und es gibt zahlreiche Zusatzinformationen. Der Text ist auch ohne spezielle Vorkenntnisse relativ verständlich.

(FOSTER, 2001)
Iàn Foster, Carl Kasselman, Steven Tuecke, *The Anatomy of the Grid (Enable Scalable Virtual Organizations)*
erschienen in: Euro-Par 2001: Parallel Processing 7th International Euro Par Conference
Herausgeber: R. Sakellaniou, J. Keane, J. Gurd, L. Freeman
Springer Lecture Notes, 2001 (Vol 2150|2001)

(HEFLIN, 2000)
Jeff Heflin, James Hendler, *Dynamic Ontologies on the Web*
Download: http://www.cs.umd.edu/projects/plus/SHOE/pubs/aaai2000.pdf
Thema sind der Entwurf und die Zusammenarbeit unterschiedlicher Ontologien für das Web mit Hilfe von SHOE. Der Text ist relativ leicht verständlich und oft Praxis-orientiert, ohne dabei jedoch notwendige Tiefe zu vernachlässigen.

(JEFFERY, 1998)
Keith G Jeffery, *Metadata: An Overview and some Issues*
veröffentlicht in: ERCIM NEWS No.35, 1998
Download: http://www.ercim.org/publication/ws-proceedings/11th-EDRG/jefferey.pdf
Ein kurzer Überblick über den Sinn und die Anwendung von Metadaten. Dementsprechend nicht sehr tief gehend, aber sehr gut verständlich.

(LABROU, 1999)
Yannis Labrou, Tim Finin, Yun Peng, *Agent Communication Languages: The Current Landscape*
Beim IEEE verfügbar (http://www.ieee.org).
Das Dokument gibt einen Überblick über das Thema ACLs (Stand 1999), weitgehend leicht verständlich. Die Autoren konzentrieren sich hauptsächlich auf die Beschreibung von Standards, was aber auch nützlich zur Einarbeitung in das Thema sein kann.

(MARUGG, 2002)

Thomas Marugg, *Import und Export von DTDs und Daten in COMICS*

Diese Diplomarbeit wurde unter der Adresse http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/DA/DA_Arbeiten_2002/Marugg_Thomas.pdf gefunden. Der link war bei der Fertigstellung dieses Dokuments immer noch gültig.

Es handelt sich um eine wissenschaftliche Arbeit, die als einen Teil XML und DTDs ausführlich beschreibt. Details werden weitgehend behandelt. Man sollte über Grundkenntnisse verfügen.

(METZ-GÖCKEL, 1997)

Metz-Göckel, *Allgemeine Psychologie 1*

Download:
http://seminarserver.fb14.uni-dortmund.de/metz-goeckel/Allgemeine_Psychologie_1/manus97.pdf

Bei dem Dokument handelt es sich um ein Vorlesungsscript des Fachbereichs Psychologie der Universität Dortmund, das einen Überblick über die Kognitionspsychologie bietet. Der Text ist weitgehend leicht verständlich und durchgängig interessant geschrieben, dafür werden einige Gebiete recht oberflächlich behandelt. Keine Literaturangaben.

(MINTERT, 1998)

Henning Behme, Stefan Mintert, *XML in der Praxis*

Es wurde die alte Ausgabe von 1998 verwendet, welche unter http://www.mintert.com/xml/ zum kostenlosen Download bereit steht. Eine neuere Ausgabe ist beim Addison Wesley Verlag käuflich erwerblich.

Das Werk bietet einen weiten Überblick über das Thema XML mit dem Schwerpunkt Anwendung in der Praxis (Webdesign). Dafür geht es sehr in die Tiefe, wobei natürlich nicht alle Details ausführlich behandelt werden. Es ermöglicht dem Einsteiger einen leichten Einstieg, ist aber auch für Fortgeschrittene durchaus interessant, da variable Schwierigkeit und Komplexität.

(MÜNZ, 2001)

Stefan Münz, *SelfHTML 8.0*

Dieser Text ist eigentlich ein kostenloses Online-Projekt, kann aber auch beim Franzis Verlag in einer zweibändigen Ausgabe erworben werden. Die Adresse lautet http://selfhtml.teamone.de/.

Eigentlich geht es um HTML, aber auch CSS, XML, JavaScript, DOM werden angesprochen. Es handelt sich um eine umfangreiche Wissenssammlung für die Webdesign-Praxis. Für diese Arbeit konnte es nur als zusätzliche Informationsquelle dienen, denn XML wird recht oberflächlich behandelt. Der Schwerpunkt ist nun einmal Webdesign mit HTML (speziell für Einsteiger, aber nicht auf diese begrenzt) und dafür ist es ein wirklich großartiges Werk.

(SELIGMAN, 2001)

Len Seligman, Arnon Rosenthal, *XML's Impact on Databases and Data Sharing*

direkter Download: http://www.mitre.org/tech/itc/staffpages/arnie/pubs/XML-IEEE-Computer.pdf

Beim IEEE verfügbar (http://www.ieee.org).

Thematisiert werden die Vorteile von XML als Datenaustauschformat und für Datenbanken. Ohne Vorkenntnisse mittelschwer, recht detailliert.

(WAP FORUM, 2000)

Wireless Application Protocol Forum, *Wireless Markup Language Specification Version 1.3*

Die aktuellste Version steht unter http://www.wapforum.org/ zum Download bereit.

Es handelt sich um eine offizielle Spezifikation. Der Text ist nicht leicht verständlich, aber dafür natürlich vollständig.