Kapitel 1. Die Dokumenttypdefinition (DTD): Einführung

In den vorhergehenden Lerneinheiten haben wir grundlegende Eigenschaften von XML kennen gelernt. XML ist eine Markupsprache und dient der Auszeichnung von Informationen in zumeist textuellen Dokumenten. Die Syntax von XML ist standardisiert durch Regeln, wie die Auszeichnung vorgenommen werden muss (bestimmte Zeichen zur Markierung von Anfangs- und Schließtags, Attributmarkierung, hierarchische Dokumentstruktur etc.). Wir haben verschiedene in XML definierte Vokabulare wie VoiceXML oder WML betrachtet, die uns die große Bandbreite der XML basierten Anwendungen vor Augen führten.

Um die XML-Vokabulare sinnvoll nutzen zu können, muss überprüft werden können, ob ein Dokument hinsichtlich seiner Struktur und Namensgebung zum jeweiligen Vokabular konform ist oder nicht. In LE 2 wurde bereits angesprochen, was das Konzept dieser Konformitätsüberprüfung ausmacht, nämlich die Validierung des Dokuments gegenüber einem Dokumenttypen. Damit Dokumenttypen automatisch verarbeitet werden können, muss die Syntax der Dokumenttypen selbst, also ihr Vokabular und dessen Bedeutung eindeutig definiert werden. In dieser Lerneinheit beschäftigen wir uns mit Definitionen von Dokumenttypen, d. h. den Beschränkungen über Strukturen und Namensgebungen, die formuliert werden können, und der Syntax, mit deren Hilfe dies geschieht.

> **Anmerkung:** SGML, von dem wir bereits in den vorherigen Lektionen gehört haben, unterscheidet nicht zwischen wohlgeformten und validierbaren Dokumenten, da der Standard Dokumente ohne dazugehörigen Dokumenttyp nicht vorsieht. Deshalb spricht man bei SGML-Dokumenten oft von *Instanzen*, die mittels in der Dokumenttypdefinition enthaltener Regeln erzeugt werden. In der Welt von XML ist diese zentrale Rolle der Dokumenttypen abgeschwächt. Viele XML-Dokumente werden ohne Dokumenttypdefinition erzeugt, oder sie wird erst nach Erstellen eines oder mehrerer exemplarischer Dokumente geschrieben. Dieser Umstand führt dazu, dass bei XML-Dokumenten auch selten von Instanzen eines Dokumenttyps die Rede ist.

Die verbreiteste, in XML 1.0 definierte Syntax für Dokumenttypen bieten DTDs (engl. Dokument Type Definition), weshalb wir uns auf diese konzentrieren. Andere *Schemassprachen* zur Definition von Dokumenttypen, die nicht eine eigene Syntax verwenden, sondern XML-basierter Vokabulare verwenden, sind in den letzten Jahren entwickelt worden oder stehen noch im Prozess der Entwicklung oder Revision. Beispiele sind *Relax NG* oder *XML-Schema* vom W3C. Sie werden ausführlich in LE 11 behandelt.

Zunächst werden wir die grundlegenden Konzepte von DTDs vorstellen. Anhand des Editors Emacs, den wir in LE 1 kennen gelernt haben und auf den wir noch öfters stoßen werden, wird die Deklaration einer DTD in einem Dokument veranschaulicht. Den Schwerpunkt dieser Lerneinheit nimmt die Beschreibung der DTD-Syntax ein, mit der Elemente, Attribute, Entitäten und Notationskonventionen deklariert werden. Dabei gehen wir von bereits bekannten XML-Dokumenten aus LE 2 aus und verwenden die DTD-Syntax in einer Übung. Anschließend diskutieren wir Richtlinien, wie die Informationen bei der Erstellung von DTDs sinnvollerweise dargestellt werden sollen, z. B. als Attributwerte oder als Elementinhalte. Am Ende der Lerneinheit steht eine Zusammenfassung des Stoffes und eine Weiterführung zum nächsten Abschnitt.

Überblick

Eine DTD kann verschiedene Formen von Restriktionen über Dokumente ausdrücken. Zum einen wird ein Dokument durch die DTD in seiner logischen Struktur beschränkt, d. h. die lineare und hierarchische Beziehung der Elemente wird definiert. Zum anderen werden Informationen über Elemente, d. h. die Attribute benannt und der Datentyp bestimmt, dem ein Attribut genügen muss. Schließlich können DTDs dazu dienen, physikalische Einheiten festzulegen, aus denen ein Dokument gebildet wird. Diese Funktionen stellen wir nun anhand von Beispieldokumenten und über sie formulierbaren Beschränkungen vor.

> **Anmerkung:** Die in dieser Lerneinheit behandelten Eigenschaften für Element-, Attributs- und Entitätsdefinitionen gelten größtenteils nicht nur für DTDs, sondern auch für andere Schemasprachen. Deshalb werden wir die Thematik zunächst generell, d. h. unabhängig von bestimmten Schemasprachen angehen, um im Anschluss die konkrete Syntax von DTDs zu beschreiben.

Strukturierung von Elementen

Mittels DTDs wird zunächst die logische Struktur der Dokumente definiert. Das folgende, bereits aus LE 2 bekannte Beispiel zeigt, was damit gemeint ist:

Beispiel 1-1. Die logische Strukturierung von Dokumenten

```
<gedicht verfasser = "Kurt Tucholsky" pseudonym = "Theobald Tiger">
<titel>An die Berlinerin</titel>
  <strophe>
    <vers>Mädchen, kein Casanova</vers>
    <vers>hätte dir je imponiert</vers>
    <vers>Glaubst du vielleicht, was ein dofer</vers>
    <vers>Schwärmer von dir phantasiert</vers>
  </strophe>
</gedicht>
```

Zunächst können wir die Regeln informell verbalisieren, die eine Möglichkeit zur späteren DTD-Bildung sind. Ein Gedicht sollte auf jeden Fall einen Titel besitzen, jedoch nicht mehr. Es sollte zudem aus mindestens einer Strophe bestehen, wobei die Zahl der Strophen nicht begrenzt sein muss. Die Strophe ist dem Titel nebengeordnet und steht im Dokument hinter dem Titel, in Übereinstimmung mit etablierten Lesegewohnheiten. Der Strophe untergeordnet sind die einzelnen Verse, von denen - wie bei den Strophen - mindestens einer vorhanden sein muss und beliebig viele stehen können.

Dieses Beispiel stellt drei wichtige Parameter der Dokumentstrukturierung vor:

- Die Beschreibung von Reihenfolgen, also die Linearisierung von Informationseinheiten. Im Beispiel muss die Strophe nach dem Titel erscheinen. Bei anderen Einheiten ist die Position nicht so stark beschränkt, zugleich aber nicht völlig frei. Der Autor kann z. B. am Anfang oder am Ende stehen.

- Die Beschränkung der Häufigkeit von Informationseinheiten. Im Beispiel treten zwei Typen von Häufigkeitsbeschränkungen auf: Beim Titel obligatorisches, einfaches Vorkommen eines Elementes, bei der Strophe und beim Vers obligatorisches, mindestens einfaches Vorkommen eines Elementes.
- Die Beschreibung von hierarchischen Beziehungen zwischen Informationseinheiten. Im Beispiel sind die Elemente Titel und Strophe dem Element Gedicht unmittelbar untergeordnet, während die Verse als Untereinheit des Elementes Strophe stehen.

Ein wichtiger Aspekt von DTDs ist das Verhältnis von DTD zu Dokument: Viele zu viele. Wie angesprochen ist es möglich und sogar eine grundlegende Funktion von DTDs, aus einem Dokumenttyp beliebig viele Dokumente zu erzeugen. Genauso ist es aber möglich, anhand eines gegebenen Dokumentes verschiedene Dokumenttypen abzuleiten. Am Beispiel des Gedichts seien drei Variationen vorgestellt. Eine DTD kann die Beschränkung angeben, dass ein Gedicht aus genau einem Vers bestehen muss. Genauso ist es aber möglich, dass das Gedicht aus mindestens einem Vers bestehen soll oder - bei 'modernen' Gedichtformen - der Vers sogar fakultativ ist. Aus dem Dokument lässt sich auf Grund rein formaler Analysen also keine DTD generieren, welche für jedes beliebige, denkbare andere Dokument des Typs 'Gedicht' sinnvoll ist. Welche grundsätzlichen Richtlinien der DTD-Erstellung nützlich sind, unabhängig davon, welche Art von Information konkret zu modellieren ist, thematisieren wir in Abschnitt 4.3.

Definition von Attributnamen und -werten

Attribute dienen dazu, zusätzliche Informationen über die Elemente darzustellen, denen sie zugeordnet sind. Wie wir bereits gesehen haben, haben sie diese Funktion auch ohne eine DTD. Wie bei der Beschreibung der Elementdefinitionen wollen wir nun ein bereits bekanntes Beispiel aus LE 2 aufgreifen, um den Rahmen von Attributdefinitionen in DTDs vorzustellen:

Beispiel 1-2. Attributverwendung und -definition

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Rechteck</title>
  </head>
  <body>
    ...
    <rect stroke="black" fill="none"/>
    ...
  </body>
</html>
```

Von Interesse sind hier zum einen das Element <html> mit dem Attribut <xmlns>, zum anderen das Element <rect> mit seinen beiden Attributen <stroke> und <fill>. Das Attribut <xmlns>, das wir bereits aus LE 2 kennen, kann nahezu eine beliebige Zeichenkette als Wert annehmen, entsprechend ist es unnötig, mögliche Werte durch eine DTD stark zu beschränken[1]. Die Attribute <stroke> und <fill> hingegen müssen hinsichtlich potentieller Werte beschränkt werden, damit diese durch eine verarbeitende Anwendung genutzt werden können. <stroke> muss einen Wert aus einer Li-

ste von Farben besitzen, während <fill> aus den Werten <none> oder ebenfalls aus einem Farbwert bestehen kann.

Wie zuvor bei den Elementen können wir nun eine Liste für sinnvolle Funktionen der DTD aufstellen:

- Für Attributwerte können bestimmte Datentypen definiert werden. Im vorliegende Beispiel scheint für das Attribut <xmlns> ein Datentyp 'WWW-Adresse' sinnvoll.
- Attributwerte können auf eine Auswahlliste zurückgreifen. Dies erscheint im Beispiel für die beiden Attribute <stroke> und <fill> sinnvoll.
- Durch Auswahllisten können Mengen von Elementen gebildet werden, z. B. die Menge aller Elemente mit einem Attribut namens <fill>, das den Wert <none> hat[2]. Derartige Mengenbildungen sind Grundlage für viele Dokumentverarbeitungsprozesse, etwa die Suche nach bestimmten Einheiten - z. B. alle Rechtecke ohne Füllung -, Sortierung oder die Formatierung bei der Visualisierung der Daten etc.
- Attribute können als fakultativ oder obligatorisch deklariert werden. Im vorliegenden Beispiel scheint sowohl der Verweis auf den Namensraum im Element <html> als auch die Angaben über Strichfarbe und Füllung bei der Rechteckdeklaration weglassbar, z. B. wenn die verarbeitende Anwendung entsprechende Defaultwerte annimmt.
- Defaultwerte können auch vorgegeben werden. Z. B. wäre es möglich, dem Attribut <fill> den Wert <none> zuzuordnen, wenn kein anderer explizit ausgewählt wird.

Die Werte von Attributen können also im Gegensatz zu Daten in Elementen genauer spezifiziert werden. XML-DTDs bieten dazu einige vordefinierte Datentypen und mittels sogenannter *Notationen* auch die Möglichkeit, in einem Dokument andere Datentypen zu verwenden, zum Beispiel für nicht-textuelle Daten wie Grafikformate. Die genaue Syntax der Definition wird im Abschnitt zur DTD-Syntax vorgestellt.

Bestimmung der physikalischen Struktur von Dokumenten.

Während Elemente und Attribute zur logischen Strukturierung und Informationsanreicherung von Dokumenten verwendet werden, können in XML-DTDs mittels Entitäten Dokumente physikalisch segmentiert werden. Daten werden sozusagen 'gekapselt'. Sowohl Elemente als auch Attribute können Entitäten beinhalten. Es gibt verschiedene Formen von Entitäten, deren formale Definition im Abschnitt zu Entitätsdeklarationen behandelt wird.

Übung 1 - Erstellung von DTDs mittels dem Emacs

Wir wollen nun beispielhaft vorstellen, wie eine DTD mit einem Dokument verbunden werden kann. Hierzu gibt es zwei Möglichkeiten: Die DTD steht in einem separaten Dokument oder im Dokument selbst. Geregelt wird diese Frage durch die *Dokumenttyp-Deklaration*. Im folgenden Beispiel, das auf eine exemplarische DTD zurückgreift, stehen Muster für beide Vorgehensweisen:

Beispiel 1-3. Die Dokumenttyp-Deklaration

```
Integration einer DTD in einem Dokument:
<!DOCTYPE gedicht [
<!ELEMENT gedicht (titel, strophe+,tonaufnahme?)>
...
]>

... oder außerhalb eines Dokumentes:
<!ELEMENT gedicht SYSTEM "dateiname"
[]
>
```

Im ersten Fall steht nach dem Schlüsselwort <DOCTYPE das oberste Element im Dokument. Anschließend beschreibt das *Declaration Subset* (der Inhalt der beiden eckigen Klammer) die Dokumenttypdefinition. Die nach rechts stehende, eckige Klammer schließt die Deklaration. Im zweiten Fall folgt nach dem Namen des obersten Elements das Schlüsselwort SYSTEM, da wir annehmen, dass sich die DTD auf unserem System bzw. Computer befindet. Es folgt der Pfad in Anführungszeichen, an dem sich die DTD befindet. Vor SYSTEM kann auch noch ein öffentlicher Bezeichner PUBLIC mit einem öffentlich bekannten Identifikator stehen, etwa bei HTML-Dokumenten der Version 4.01 PUBLIC "-//W3C//DTD HTML 4.01//EN". Das Declaration Subset ist hier weglassbar. Bei einer externen DTD kann es nützlich sein, um diese zu erweitern oder zu verändern. Darauf gehen wir im Abschnitt Regeln und "Best practices" beim Entwerfen von DTDs ein.

Die folgende Grafik zeigt, wie der Editor Emacs[3] auf die Einbindung einer DTD im Declaration Subset durch farbige Auszeichnung reagiert:

Beispiel 1-4. Einbindung einer DTD im Emacs

Im nächsten Abschnitt werden wir die syntaktischen Konstrukte von DTDs detailliert behandeln. Verwendet man einen Editor wie den Emacs, müssen diese Konstrukte nahezu 'von Hand' eingegeben werden. Es gibt allerdings auch graphische Editoren, die durch ihre Visualisierung den Prozess der DTD-Erstellung erleichtern. Der Emacs hat jedoch den Vorteil, die Komponenten eines XML-Systems - DTD-Editor, Dokument-Editor, XML-Parser - und ihre Funktionsweise klar vor Augen zu führen, so dass er für unsere ersten praktischen Schritte gut geeignet ist.

Syntax und Verwendung

Bei der detaillierten Beschreibung von DTD-Konstrukten werden wir wieder auf die Beispiele aus LE 2 zurückgreifen und eine passende DTD erstellen. Am Ende des Abschnittes steht eine Übung, wie wir selbst eine DTD verfassen und in ein Dokument einbinden können.

Aufbau von DTDs

Eine XML-DTD besteht aus folgenden Konstrukten:

- Mindestens eine Elementdeklaration.
- Deklarationen von Attributen in Form von Attributlisten.
- Deklarationen für generelle Entitäten.
- Deklarationen von Notationen für selbstdefinierte Datentypen.
- Deklarationen für Parameter-Entitäten und Marked Sections.

Wie diese Konstrukte für unser Beispiel verwendet werden können, sei schon einmal vorweggenommen in Form einer fertigen DTD[4]:

Beispiel 1-5. Beispiel-DTD

```
<!ELEMENT gedicht (titel?,strophe+,tonaufnahme*)>
<!ATTLIST gedicht
          katalognummer ID #REQUIRED
          sammlungszugehoerigkeit IDREFS #IMPLIED
          verfasser CDATA #REQUIRED
          pseudonym CDATA #IMPLIED>
<!ELEMENT strophe (vers+)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT vers (#PCDATA)>
<!ELEMENT tonaufnahme EMPTY>
<!ATTLIST tonaufnahme
          quelle ENTITY #REQUIRED>
```

Anmerkung: Die Reihenfolge der Konstrukte, d. h. ihre physikalische Anordnung in der DTD ist mit Ausnahme der Definition von Entitäten unerheblich. Bei der Verarbeitung der DTD, sei es als Grundlage zur Dokumenterstellung oder zur Validierung der Dokumente, entscheidet die Dokumenttyp-Deklaration in der XML-Instanz darüber, welches das oberste Element in der Dokumentstruktur sein soll. Im Beispiel des Gedichts ist es das Element gedicht; denkbar wäre aber auch, dass die DTD ein weiteres Element namens gedichtsammlung enthält oder hierarchisch noch höher stehende Elementdeklarationen wie prosa, literatur etc. Je nachdem welches Element in der Dokumenttyp-Deklaration zum Wurzelelement erklärt wird, kann auf verschiedene Bereiche der DTD zurückgegriffen werden. Die Dokumenttyp-Deklaration parametrisiert sozusagen die DTD für das jeweilige Dokument. Weitere Formen der Parametrisierung lernen wir im Abschnitt zu Parameter-Entitäten und Marked Sectionskennen.

Die Konstrukte im Einzelnen

Elementdeklarationen

Eine typische Elementdeklaration, wie sie sich auch in der Beispiel-DTD findet, sieht folgendermaßen aus:

Tabelle 1-1. Muster für Elementdeklarationen

<!ELEMENT	Name	Inhaltsmodell oder Schlüsselwort	>

<!ELEMENT leitet die Deklaration ein. Es folgt der Name des Elements, für den verschiedene Regeln gelten, die wir bereits in LE 2 kennen gelernt haben (keine Leerzeichen, Relevanz von Groß- und Kleinschreibung etc.). Das *Inhaltsmodell* legt die Struktur derjenigen Elemente fest, die in dem definierten Element vorkommen dürfen, d. h. ihm in der Dokumentstruktur unmittelbar untergeordnet sind. In unserer Beispiel-DTD darf ein Gedicht aus Titel, Strophe und Verfasser bestehen. Um die Häufigkeit der Elemente im Inhaltsmodell festzulegen, werden *Statusbezeichnungen* verwendet. Sie verleihen den Elementen einen der folgenden Status:

- Fakultatives Vorkommen, bezeichnet durch '?'.
- Null, einfach oder mehrfaches Vorkommen, bezeichnet durch '*'.
- Ein oder mehrfaches Vorkommen, bezeichnet durch '+'.

Fehlt die Statusmarkierung, muss das Element genau einmal im Inhaltsmodell vorkommen. Das Gedicht benötigt so genau einen Titel und mindestens eine Strophe, gekennzeichnet durch '+'.

Neben der Statusmarkierung legt das Inhaltsmodell die Abfolge der Elemente fest. Dafür stehen folgende *Konnektoren* zur Verfügung:

- Ein exklusives Oder, bezeichnet durch '|'.
- Eine Sequenz, bezeichnet durch ','.

Zudem können durch Klammerausdrücke '( )' sogenannte *Untermodelle*, d. h. tiefergehende Verschachtelungen im Inhaltsmodell gebildet werden. Untermodelle können wieder Untermodelle enthalten, die ebenfalls Untermodelle enthalten etc. Die Statusbezeichnungen können auch auf Untermodelle angewendet werden. Im folgenden Beispiel definieren wir zwei Formen von Gedichten: Ein Gedicht mit genau einem Vers durch das Element strophe-kurz, und ein Gedicht mit mindestens einem und beliebig vielen Versen, das Element strophe-lang:

Beispiel 1-6. Alternativ definierte Untermodelle

```
<!ELEMENT gedicht (titel,(strophe-kurz|strophe-lang))>
<!ELEMENT strophe-kurz (vers)>
<!ELEMENT strophe-lang (vers+)>
...
```

Textdaten in Elementen und 'gemischter Inhalt'

Soll ein Element nur textuelle Daten enthalten, wird das *Schlüsselwort* #PCDATA im Inhaltsmodell ohne weitere Elemente verwendet. Daten und Elemente können auch gemeinsam in einem Inhaltsmodell vorkommen. Dann steht das Schlüsselwort am Anfang des Inhaltsmodells und dahinter weitere Elemente, getrennt durch den Konnektor |. Man spricht in diesem Fall von *mixed content*

'gemischtem Inhalt': Die Abfolge der textuellen Daten und der Elemente lässt sich in XML-DTDs nicht regulieren. Möchte man z. B. in einem Vers das erste Wort als betont markieren, kann man dies durch die Definition <!ELEMENT vers (#PCDATA | emphase)*> erreichen. Dadurch werden jedoch nicht nur Dokumentfragmente wie <vers><emphase>Mädchen</emphase>, kein Casanova</vers> validierbar, sondern auch nicht unbedingt erwünschtes Markup wie <vers>Mädchen<emphase>, </emphase>kein Casanova</vers>.

Weitere Schlüsselwörter

Ein Inhaltsmodell kann alternativ zum obigen Muster auch eines der Schlüsselwörter EMPTY oder ANY enthalten. EMPTY dient der Definition leerer Elemente, die im Dokument keine anderen Elemente oder Dateninhalte enthalten dürfen. ANY definiert ein Inhaltsmodell, in welchem jedes andere Element oder Text vorkommen kann, also ein beliebiges Inhaltsmodell.

Ambiguität in Inhaltsmodellen

Eine selten beachtete Beschränkung, der Inhaltsmodelle unterliegen, ist das Verbot von *Ambiguität*. Ambiguität ist dann gegeben, wenn für die Verarbeitung eines Elementes im Inhaltsmodell Alternativen berücksichtigt werden müssten. Möchte man z. B. eine DTD für Gedichte mit mindestens einer, aber höchstens zwei Strophen schreiben, bietet sich scheinbar folgendes Inhaltsmodell an:

Beispiel 1-7. Ein mehrdeutiges Inhaltsmodell

```
<!ELEMENT gedicht (titel,strophe?,strophe)>
```

Dieses Inhaltsmodell ist jedoch mehrdeutig, da für ein Element strophe in einem zu validierenden Dokument nicht entschieden werden kann, ob noch eine Strophe im Dokument vorkommen muss oder nicht. Oft lassen sich Ambiguitäten allerdings durch die Umformulierung des Inhaltsmodells auflösen, in diesem Fall recht einfach, indem man die Statusbezeichnungen austauscht: Wird die erste Strophe als obligatorisch definiert und die zweite als fakultativ, ist ein entsprechendes Dokument wieder validierbar.

Deklarationen von Attributen, Notationen und Entitäten

Das generelle Muster für Attributsdeklarationen sieht folgendermaßen aus:

Tabelle 1-2. Muster für Attributsdeklarationen

<!ATTLIST	Element-Name	Attribut-Definition (mindestens einmal)	>

<!ATTLIST leitet die Deklaration ein. Es folgt der Name des Elements, dem das Attri-

but beziehungsweise die Attribute zugeordnet werden sollen. Hier gelten in Entsprechung zu den Elementnamen verschiedene Einschränkungen für den verwendbaren Zeichenvorrat. Eine Attributliste muss aus mindestens einem Attribut bestehen und besitzt das in der folgenden Tabelle zusammengefasste Muster. Die erste Spalte enthält den Attributnamen, die zweite Spalte durch Kommata getrennte, alternativ zu verwendende Charakterisierungen des Datentyps, die dritte Spalte ebenfalls alternativ zu verwendende Statusbestimmungen und Defaultwerte.

Tabelle 1-3. Muster für Attributlisten

Attributname	NMTOKEN-Gruppe, CDATA, ENTITIY, ENTITIES, ID, IDREF(S), NMTOKEN(S), NOTATION NMTOKEN-gruppe	"Wert", #FIXED "Wert", #REQUIRED, #IMPLIED	>

Diese Attributvarianten werden wir nun im Einzelnen betrachten.

Auswahllisten und Defaultwerte von Attributen

Mittels Auswahllisten, sogenannten *NMTOKEN-Gruppen*[5], können einem Attribut alternativ verschiedene Werte zugewiesen werden. Wir hatten bereits im Überblicksabschnitt zu dieser Lerneinheit am Beispiel der Farbmarkierung durch das Attribut stroke gesehen, wozu dies nützlich sein kann: Eine Anwendung, die das validierte Dokument weiterverarbeitet, kann auf eine bestimmte Wertemenge zurückgreifen. Zusätzlich können mit alternativen Werten Gruppen derjenigen Elemente gebildet werden, die hinsichtlich des Attributwertes übereinstimmen. Das folgende Beispiel definiert eine Auswahlliste für ein Attribut am Element Gedicht, an dem verschiedene Reimmuster ausgewählt werden können:

Beispiel 1-8. Auswahllisten und Defaultwerte von Attributen

```
<!ATTLIST Gedicht
          versmass (trochäus|jambus|andere) "jambus">
          texttyp (belletristik) #FIXED "belletristik">
```

Das Attribut versmass ist mit einem Defaultwert belegt, d. h. wenn beim Erstellen eines Dokumentes kein anderer Wert angegeben wird, setzt der validierende Parser den Wert "jambus" ein. Der Defaultwert muss nicht vorgegeben werden, ein Attribut mit Auswahlliste kann auch über entsprechende Schlüsselwörter (siehe nächsten Abschnitt) als obligatorisch oder fakultativ eingestuft werden. Zudem ist es möglich, einen Defaultwert anzugeben, der nicht verändert werden kann. Im Beispiel wird der texttyp als "belletristik" bestimmt. So ist es möglich, verschiedene, zur Unterteilung literarischer Gattungen verwendete Elemente (Gedicht, Roman, ...) in einer Obermenge "belletristik" zusammenzufassen und diese gegenüber anderen Texttypen wie "sachtext" abzugrenzen.

Anmerkung: Defaultwerte sind eine besondere Konstruktion in DTDs, weil sie vom Prinzip der reinen Validierung einer Instanz gegen eine DTD abweichen. Zugleich verändern sie den Infoset, den wir in LE 2 behandelt haben: Der Parser setzt die Defaultwerte in das Infoset ein und verändert so das Dokument.

Textuelle Daten: Das Schlüsselwort CDATA

Das Schlüsselwort CDATA haben wir bereits in LE 2 beim Thema "CDATA-Abschnitte" kennen gelernt. Bei Attributen wird es verwendet, um den Datentyp 'textuelle Daten' zu bestimmen: Der Attributwert kann - anders als beim Datentyp NMTOKEN - auch Leerzeichen enthalten. Eine Definition mit CDATA sieht so aus:

Beispiel 1-9. Attributdefinition mit CDATA

```
verfasser CDATA #IMPLIED
pseudonym CDATA #REQUIRED
```

Die beiden Attribute aus der Beispiel-DTD unterscheiden sich durch die Schlüsselwörter #REQUIRED (Obligatorik des Attributs) und #IMPLIED (Fakultativität).

Identifikatoren und Verweise: Die Schlüsselwörter ID und IDREF(S)

Durch das Schlüsselwort ID kann einem Element ein eindeutiger Identifikator zugewiesen werden, der im gesamten Dokument genau einmal verwendet werden darf. Der Attributwert muss dem Datentyp NAME entsprechen, d. h. er darf keine Leerzeichen enthalten und muss mit einem Buchstaben beginnen. In unserer Beispiel-DTD haben wir ein Attribut katalognummer für das Element Gedicht definiert, das den ID-Datentyp verwendet.

Möchte man auf diesen Identifikator von einer anderen Stelle des Dokuments aus verweisen, kann ein Attribut vom Typ IDREF (ebenfalls als NAME zu definieren) eingesetzt werden. Beim Validieren überprüft dann der Parser, ob für jedes IDREF-Attribut ein ID-Attribut im Dokument vorhanden ist, dass den gleichen Wert enthält. In der Beispiel-DTD haben wir das Attribut sammlungszugehoerigkeit definiert, mit dem das Gedicht einer Sammlung zugeordnet werden kann. In diesem Fall verwenden wir die Varianten IDREFS: Das Attribut kann nicht nur eine Referenz enthalten, sondern mehrere, durch Leerzeichen getrennte Verweise.

Gekapselte Daten und Notationsdefinitionen: Die Schlüsselwörter ENTITIY(S) und die Definition von Notationen

Durch die Definition von Entitäten in der DTD kann man andere Datentypen als diejenigen verwenden, welche standardmäßig zur Verfügung stehen. Dies ist zum Beispiel für die Einbindung von audiovisuellen Daten sinnvoll, deren Typisierung Voraussetzung ist für eine Weiterverarbeitung des Dokuments. Angenommen, unser Gedicht soll mit einem Bild des Autoren versehen werden, könnte man auf dieses durch ein Attribut am Element Gedicht verweisen:

Beispiel 1-10. Einbindung von Entitäten in Attribute

```
Dokumentfragment:
...
<vers bild="tucholsky-jung"></vers>
...
DTD-Fragment:
...
<!ATTLIST verfasser
          bild ENTITY #IMPLIED>
<!ENTITY tucholsky-jung SYSTEM "tucholsky.bmp" NDATA BMP>
<!ENTITY tucholsky-alt SYSTEM "tucholsky-alt.gif" NDATA GIF89a>
<!NOTATION EPS PUBLIC
"+//ISBN 0-201-18127-4::Adobe//NOTATION PostScript Language
Ref. Manual//EN">
<!NOTATION GIF89a PUBLIC
"-//CompuServe//NOTATION Graphics Interchange Format 89a//EN">
<!NOTATION BMP PUBLIC
"+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION Microsoft Windows
bitmap//EN">
...
```

In der DTD muss das Attribut bild durch das Schlüsselwort ENTITY als 'Datentyp Entität' deklariert werden. Bei den Entitäten tucholsky-jung und tucholsky-alt zeigt das Schlüsselwort *SYSTEM* an, dass die von den Entitäten gekapselten Daten auf unserem System, z. B. auf unserer Festplatte auffindbar sind. Der genaue Ort, hier der Dateiname, wird in Anführungszeichen wiedergegeben. Zusätzlich zur Entitätsdefinition benötigen wir für die verschiedenen Bild-Datentypen Notationsdefinitionen. Auf diese Definitionen wird bei den Entitäten durch das Schlüsselwort NDATA und den Namen der Notation referiert. In unserem Beispiel haben wir für die drei Grafikformate EPS, GIF und BMPB Notationen eingefügt. Bei den Notationen gibt das Schlüsselwort *PUBLIC* an, dass sie mit Identifikatoren versehen sind, die nicht auf unser System beschränkt sind.

Entitätsdeklarationen

Wir haben im Abschnitt über Attributsdefinitionen bereits die Definition von Entitäten in Attributen behandelt. Hier wollen wir weitere Aspekte der Entitätsdefinition darstellen und verschiedene Typen von Entitäten vorstellen.

Generelle Entitäten

Generelle Entitäten kennen wir aus LE 2, als wir z. B. die Zeichenreferenz behandelten. Sie werden im Dokument durch das "Ampersand" oder "Kaufmanns-Und" ("&") einerseits sowie das Semikolon andererseits umrahmt. Die Definition einer generellen Entität sieht folgendermaßen aus:

Tabelle 1-4. Definition einer generellen Entität

<!ENTITIY>	Entitätsname	SYSTEM	"Dateiname"	nichts	>

PUBLIC "Bezeichner" NDATA Notationsname

"Dateiname"

"Zeichenkette"

Die Vielzahl der Möglichkeiten mag abschreckend erscheinen, ist aber glücklicherweise 'fast' nur eine Zusammenfassung von Mustern, die wir schon kennen gelernt haben. Wird das Schlüsselwort SYSTEM mit einem Dateinamen oder nur der Dateiname verwendet, ist die Entität in einem Dokument auf dem Computer enthalten. Das Muster mit der Notationsdefinition haben wir bereits im Abschnitt über Notationensdefinitionen beschrieben. Neu hinzu kommt hier die Variante, eine Zeichenkette als Entität zu definieren. So können wir z. B. auch ein komplettes XML-Dokument durch eine Entität wiedergeben :

Beispiel 1-11. Kapselung textueller Daten in Entitäten

```
DTD-Fragment:
...
<!ENTITY tucholsky-gedicht
"<titel>An die Berlinerin</titel>
  <strophe>
    <vers>Mädchen, kein Casanova</vers>
    <vers>hätte dir je imponiert</vers>
    <vers>Glaubst du vielleicht, was ein dofer</vers>
    <vers>Schwärmer von dir phantasiert</vers>
  </strophe>
">
...
Dokument:
<gedicht>
&tucholsky-gedicht;
</gedicht>
```

In diesem Beispiel steht unser Gedicht, genauer gesagt der Abschnitt vom titel bis einschließlich der Strophe, in der Entität &tucholsky-gedicht als Teil der Dokumenttypendefinition, während das Dokument nur aus dem Wurzelelement gedicht und dem Aufruf der Entität besteht. Beim Validieren wird die Entität zunächst aufgelöst, d. h. die physikalisch vom Dokument getrennten Daten werden in dieses eingesetzt. Die Daten unterliegen dann den XML- und DTD-spezifischen Syntaxregeln.

Parameter-Entitäten und Marked Sections

Mit Parameter-Entitäten und Marked Sections können DTDs modularisiert werden und sind so leicht auf verschiedene Zwecke hin anpassbar[6]. Die allgemeinen Muster für ihre Definition sehen folgendermaßen aus:

Tabelle 1-5. Definition von Parameter-Entitäten

<!ENTITIY %	Entitätsname	"Zeichenkette"		>
		SYSTEM	Dateiname	
		PUBLIC	Public-Bezeichner	
		PUBLIC	"Public-Bezeicher" "Dateiname"	

Tabelle 1-6. Definition von Marked Sections

<![	INCLUDE	[Inhalt]]>
	IGNORE	
	CDATA	
	Referenz auf eine Parameter-Entität	

Marked Sections dienen dazu, einen Abschnitt in der DTD zu markieren und durch die Schlüsselworter INCLUDE und IGNORE anzugeben, ob der Bereich bei der Verarbeitung berücksichtigt werden soll oder nicht. Um z. B. eine zusätzliche Definition für unser Gedichtbeispiel bereit stellen zu können, die wir aber nicht in jedem Fall nutzen, definieren wir einen entsprechenden Abschnitt:

Beispiel 1-12. Marked Sections und Parameter-Entitäten

```
<![ %gedicht-generell; [
<!ELEMENT gedicht (titel, strophe+,tonaufnahme?)>
]>
<![ %gedicht-speziell; [
<!ELEMENT gedicht
(titel, strophe, strophe, strophe, anmerkung?, tonaufnahme?)>
]>
<!ENTITY %gedicht-generell "INCLUDE">
<!ENTITY %gedicht-speziell "IGNORE">
```

Mit Parameter-Entitäten kann nun der Status von Marked Sections reguliert werden. Hier haben wir eine Parameter-Entität gedicht-generell definiert und sie auf den Wert "INCLUDE" gesetzt, damit unsere häufig verwendete Gedichtform ohne weitere Veränderungen verfügbar ist. Die Parameter-Entität für das spezielle Gedicht ist auf den Wert "IGNORE" gesetzt. Soll das spezielle Inhaltsmodell für das Gedicht aktiviert werden, muss die Dokumenttypdeklaration mit dem Declaration Subset folgendermaßen aufgebaut sein:

Beispiel 1-13. Parameterisierung im Declaration Subset

```
<!DOCTYPE gedicht SYSTEM "gedicht.dtd"[
<!ENTITY % gedicht-generell "IGNORE">
<!ENTITY % gedicht-speziell "INCLUDE">
]>
```

Durch die Dokumenttypdeklaration wird die externe DTD aufgerufen und gedicht zum obersten Element erklärt. Im Declaration Subset wird nun die Standarddefinition für das Gedicht durch den Ausdruck <ENTITY % gedicht-generell "IGNORE"> ausgeschaltet, während die spezielle Definition durch den Ausdruck <!ENTITY % gedicht-speziell "INCLUDE"> aktiviert wird.

In Parameter-Entitäten können nicht nur die beschriebenen Schlüsselwörter zur Steuerung von Marked Sections stehen; sie können auch Teile von Inhaltsmodellen oder Attributlisten enthalten, um so häufig verwendete Teile übersichtlicher zu strukturieren. Diese Verfahren werden wir im Abschnitt Regeln und "Best practices" beim Entwerfen von DTDs vorstellen.

Übung

Verfassen Sie eine DTD zu folgendem XML-Dokument

Beispiel 1-14. XML-Dokument als Grundlage zur DTD-Erstellung

```
<adressenliste version="februar 02">
  <adresse id="adresse1" typ="amerikanisch">
 <name>Karl Meier</name>
 <strasse>Arlington Road</strasse>
 <stadt>Los Angeles</stadt>
 <staat>Kalifornien</staat>
 <zip-code>90952</zip-code>
 <zusatzinfos bezugsperson="adresse2">Bruder von Hans</zusatzinfos>
  </adresse>
 <adresse typ="deutsch">
  </adresse>
 <adresse id="adresse2" typ="deutsch">
 <name>Hans Meier</name>
 <strasse>Luisenstrasse</strasse>
 <postleitzahl>33615</postleitzahl>
 <stadt>Bielefeld</stadt>
 <zusatzinfos bezugsperson="adresse1">Bruder von Karl</zusatzinfos>
  </adresse>
 <adresse typ="deutsch">
  </adresse>
</adressenliste>
```

Versuchen Sie dabei, auf folgende Aspekte zu achten:

- Status von Elementen
- Wiederverwendbarkeit von Inhaltsmodellen
- Datentypen von Attributen

Regeln und "Best practices" beim Entwerfen von DTDs

Wir haben im letzten Abschnitt eine Vielzahl von Konstruktionen kennen gelernt, die wir in DTDs verwenden können. Die Frage lautet nun: Wie gehen wir damit um? Wie geben wir die Daten wieder, als Elemente oder Attribute? Sollen wir für jede Art von Dokument eine eigene DTD schreiben oder nicht? Dieser Abschnitt gibt einen kurzen Leitfaden, wie man am besten bei der DTD-Erstellung vorgeht.

Attribute vs. Elemente?

Eine wichtige Entscheidung, die man treffen muss, betrifft die Wiedergabe der Daten. Wir hatten diese Problematik bereits in LE2 angeschnitten und verschiedene Ansätze diskutiert. Das folgende Beispiel zeigt, wie man es nicht machen sollte:

Beispiel 1-15. Schlechteste Möglichkeit zur Wiedergabe von Daten

```
<gedicht verfasser="tucholsky"
pseudonym="Theobald Tiger" titel="An die Berlinerin"
strophe-vers1="Mädchen, kein Casanova" ...>
```

Hier stehen alle Informationen in Attributen, so dass die Struktur der Daten verloren geht und sie somit schlecht zu lesen sind, sowohl für den menschlichen Benutzer als auch für verarbeitende Programme. Wichtige Punkte bei der Wahl 'Attribut versus Element' lassen sich so zusammenfassen:

- Attribute können nicht geordnete Werte enhalten, Elemente hingegen geordnete Unterelemente. Im Gedicht kann nur durch Elemente die Reihenfolge von Titel und Strophe und die Unterordnung der Verse bestimmt werden.
- Attribute sind schwer zu erweitern. Die Struktur der Gedichtsdefinition, die wir durch Modifikationen des Inhaltsmodells im Verlauf dieser Lerneinheit verändert haben, ist nicht gegeben, wenn die Daten nur als Attributwert dargestellt werden.
- Elemente können nicht hinsichtlich eines Datentyps spezifiziert werden. Attribute bieten sich z. B. an, wenn Daten nur einmal im Dokument vorkommen sollen wie bei Identifikatoren[7].

Eine gute Faustregel, die vom menschlichen Benutzer ausgeht, lautet, alle lesbaren Informationen in Elemente unterzubringen, so wie wir es im Gedicht-Beispiel gemacht haben. Alle Informationen über den eigentlichen Text, die *Metainformationen*, stehen dann in Attributen.

Verwendung von Marked Sections und Parameter-Entitäten

Die beste Methode, eine DTD zu erstellen, liegt in der Übernahme und Modifikation bereits vorhandener DTDs. Für XML-Vokabulare wie XHTML oder SVG steht auf Grund ihrer Standardisierung eine große Menge verarbeitender Software zur Verfügung, auf die auch ein eigenes, von den Standards abgeleitetes Vokabular zurückgreifen kann. Das Stichwort hierzu lautet "Modularisierung". Wir haben bereits im Abschnitt Parameter-Entitäten und Marked Sections gesehen, wie mit diesen beiden Konstruktionen alternative Inhaltsmodelle definiert und im Dokument aktiviert bzw. deaktiviert werden. Die Parameter-Entitäten können aber auch dazu verwendet werden, um oft verwendete Attributlisten oder Erweiterungen der DTD aufzunehmen. Da Entitäten ebenfalls Entitäten enthalten dürfen, lassen sich so ineinander geschachtelte Module erstellen. Wir wollen dieses Prinzip am Gedicht-Beispiel und der exemplarischen DTD vorstellen:

Beispiel 1-16. Parameter-Entitäten als Mittel zur Modularisierung von DTDs

```
<!ENTITY % a.global "
      id       ID        #IMPLIED
      revision CDATA     #IMPLIED
      lang     CDATA     #IMPLIED">
...
<!ATTLIST gedicht
          %a.global;
          verfasser CDATA #REQUIRED
          pseudonym CDATA #IMPLIED>
...
<!ATTLIST strophe
          %a.global;
          versmass (trochäus|jambus|andere) #IMPLIED>
...
```

Hier haben wir die zwei globalen Attribute revision und lang in einer Parameter-Entität %a.global zusammengefasst[8]. Dadurch werden die Attribute wiederverwertbar für die verschiedenen Module, die neben dem Aufruf der Parameter-Entität zusätzliche Attributdefinitionen enthalten.

Manche Standard-Vokabulare von XML wie XHTML liegen bereits in modularisierter Form vor. Auf Grund der Komplexität von XHTML ist es recht kompliziert, die notwendigen Erweiterungen vorzunehmen. Allerdings gibt es ausführliche Anleitungen zur XHTML-Modularisierung[9], auf die wir hier jedoch nicht weiter eingehen werden. Einige Modularisierungen selbst, z. B. die Kombination von XHTML, SVG und MAthML (Ein Vokabular zur Auszeichnung mathematischer Formeln), sind bereits selbst Gegenstand von Standardisierungsbemühungen.

Zusammenfassung

In dieser Lerneinheit haben wir uns mit folgenden Aspekten von Dokumenttypdefinitionen (DTD) beschäftigt:

- Wir haben gelernt, was für Funktionen DTDs haben. DTDs bestimmen durch Elements- und Attributsdefinitionen die logische Struktur von Dokumenten. Elemente werden spezifiziert hinsichtlich Abfolge und Status (z. B. fakultativ oder obligatorisch). Attribute erhalten ebenfalls einen Status und darüberhinaus einen Datentyp, z.B. NMTOKEN für eine Zeichenkette ohne Leerzeichen. Mittels Entitäten kann die physikalische Dokumentstruktur verändert werden.
- An einem Beispiel haben wir gesehen, wie eine DTD mit einem Dokument in Verbindung gesetzt wird und wie ein Standardeditor (Emacs) darauf reagiert.
- Die syntaktischen Konstrukte einer DTD haben den größten Teil dieser Lerneinheit eingenommen. Wir haben das Muster von Elementdefinitionen betrachtet: Ein Inhaltsmodell bestimmt die Abfolge der Elemente durch Konnektoren und den Status durch Statusmarkierer. Bestimmte Schlüsselwörter wie ANY oder EMPTY können anstelle des Inhaltsmodells stehen. Attribute erlauben die Datentypen 'Auswahlliste von Zeichenketten' (NMTOKEN-Gruppe), CDATA für textuelle Daten, ENTITY oder ENTITIES für Entitäten, ID und IDREF(S) für Verweise oder Referenzen und NMTOKEN für Zeichenketten. Mittels Notationen können eigene Datentypen definiert werden. Entitäten lassen sich unterscheiden in generelle Entitäten, z. B. für Zeichenreferenzen, und Parameterentitäten. Letztere dienen im Zusammenspiel mit markierten Bereichen innerhalb der DTD dazu, diese zu modularisieren und so spezielle DTDs aus einer generellen DTD zusammenzustellen.
- Eine Übung hat uns gezeigt, wie wir eine einfache DTD mit den wichtigsten Konstrukten selber erstellen.
- Zum Schluß sind wir der Frage nachgegangen, wie man am besten bei der DTD-Erstellung vorgeht. Elemente sollten für solche Daten verwendet werden, die die eigentlichen, für den Benutzer sichtbaren Informationen ausmachen. Attribute sollten Metainformationen, d. h. Informationen über diese Informationen enthalten, z. B. die Identifikatoren von Elementen, das Revisionsdatum etc. Am Beispiel der Modularisierung von DTDs mit Parameterentitäten und Marked Sections haben wir gesehen, wie eine DTD wiederverwendbar gemacht werden kann.

Im nächsten Abschnitt geht es um XML in der Praxis: Wir werden lernen, wie man Dokumente mit dem Emacs editiert und mit verschiedenen Parsern auf Wohlgeformtheit hin überprüft und gegenüber einer DTD validiert.

Fußnoten

1. Da durch dieses Attribut ein Namensraum definiert wird, wäre es eigentlich wünschenswert, das Muster von Webadressen als Datentyp zu definieren und zu validieren. Eine so weitreichende Datentypbestimmung lassen DTDs jedoch nicht zu.
2. Dies ist auch mit Attributen möglich, die als Typ 'textuelles Datum' definiert sind, allerdings ist dann die Konsistenz der Werte schwerer zu realisieren.
3. In LE 5.1 werden wir zeigen, welche Schritte nötig sind, um mit dem Emacs DTDs zu erstellen, Dokumente zu bearbeiten und zu validieren.
4. Diese DTD steht separat zum XML-Dokument und nicht im Deklaration-Subset. Es empfiehlt sich insbesondere bei umfangreichen und oft verwendeten DTDs, generell einsetzbare Deklarationen in der separaten DTD vorzunehmen und das Deklaration-Subset dokumentspezifischen Deklarationen vorzubehalten. Siehe

hierzu auch den Abschnitt Regeln und "Best practices" beim Entwerfen von DTDs.

5. Die Auswahllisten oder Defaultwerte werden als NMTOKEN-Gruppe bezeichnet, da ihr Datentyp NMTOKEN (eine Zeichenkette ohne Leerzeichen) lautet.
6. Am Beispiel der DocBook-DTD werden wir in LE 6.3 sehen, wie eine komplexe, modulare DTD aufgebaut ist.
7. Diese Einschränkung von Elementen betrifft insbesondere DTDs. Schemasprachen wie XML-Schema erweitern den Skopus von Datentypendefinitionen auf Elemente, siehe LE 11.
8. Das Präfix a... wird normalerweise für Entitäten zur Parametrisierung von Attributslisten verwendet; m... steht für Inhaltsmodelle, x... für Erweiterungen.
9. http://www.w3.org/TR/xhtml-modularization