2 XML-Basiskonzepte

Im Rahmen dieses Buches können wir keine vollständige Einführung in alle XML-Konzepte geben. Diejenigen Konzepte von XML, die für das Verständnis von XQuery nötig sind, sollen aber dennoch in kompakter Form vorgestellt werden. Wir beginnen mit einer kurzen Besprechung der Struktur eines XML-Dokumentes (und damit der XML-1.0-Spezifikation) und einer kurzen Einführung in Unicode. Als weiteres grundlegendes Konzept folgen die XML-Namensräume, die in XML Schema und XQuery intensiv genutzt werden. XML Schema schließt sich an, wobei wir uns allerdings auf die Konzepte beschränken, die im weiteren Verlauf des Buches eine Rolle spielen. Verweise können in XML auf verschiedene Arten realisiert werden. Die im Kontext von XQuery relevanten Arten stellen wir kurz vor. Als kleinen Vorgeschmack auf XQuery zeigen wir einige beispielhafte XQuery-Anfragen, die schon wichtige Konzepte illustrieren, die in späteren Kapiteln ausführlich erklärt werden.

2.1 XML-Dokumente

XML ist eine Metasprache, also eine Sprache zur Definition von Sprachen (Vokabularen) für Informationseinheiten, die so genannten XML-Dokumente. Das Wort »Dokument« weckt hier Assoziationen zu Dokumenten im herkömmlichen Sinn, wie etwa Verträgen oder Urkunden. Tatsächlich können diese mit XML dargestellt werden – man spricht dann von *textorientierten XML-Dokumenten* [Schö03]. Allerdings bietet XML auch die Möglichkeit, Daten darzustellen, die mit lesbaren Texten wenig gemein haben, wie z. B. die Inhalte relationaler Datenbanken. In diesen Fällen spricht man von *datenorientierten XML-Dokumenten*. Dank der Flexibilität von XML sind beliebige Zwischenstufen möglich.

Textorientierte und datenorientierte Dokumente

Wohlgeformte XML-Dokumente

XML gibt Syntaxregeln vor, nach denen ein XML-Dokument strukturiert sein muss. Wenn ein Dokument[1] den Syntaxregeln genügt, heißt es *wohlgeformt* (»well-formed«). Wie bereits erwähnt, lässt sich mit XML eine Sprache definieren, nämlich eine Grammatik für die erlaubte Struktur des Dokumentes. Wenn ein Dokument der jeweils erlaubten Struktur genügt, heißt es *gültig* (*valid*). Eine solche Strukturvorgabe kann entweder mit einer in der XML-1.0-Spezifikation vorgegebenen *Document Type Definition* (*DTD*) geschehen oder mit einer anderen XML-Schemabeschreibungssprache, wie zum Beispiel dem beim W3C standardisierten *XML Schema*. Den Vorgang der Prüfung, ob ein XML-Dokument gültig ist, nennt man Validierung. Wir werden sehen, dass im Rahmen einer solchen Validierung auch der Informationsgehalt eines XML-Dokumentes zunehmen kann.

Gültige XML-Dokumente

Validierung

XML ist aus SGML (»Standard Generalized Markup Language« [ISO8879]) entstanden, welches ebenfalls eine Metasprache ist. Die bekannteste Sprache, die mit SGML definiert wurde, ist HTML. So erstaunt es nicht, dass XML-Dokumente Strukturelemente enthalten, die aus HTML-Dokumenten bekannt sind. Es gibt sogar unter dem Namen XHTML eine Redefinition von HTML als XML-Sprache. Alle XHTML-Dokumente sind auch HTML-Dokumente. Umgekehrt gilt dies nicht, da HTML in vielen Beziehungen freizügiger ist. So unterscheidet XML im Gegensatz zu HTML Groß- und Kleinschreibung. An folgendem wohlgeformten XML-Dokument sieht man die Strukturähnlichkeit zu HTML.

```
<Klinik>
  <Name>Hochwaldklinik</Name>
  <Stationen>
    <Station Leitung="Pfleger_01">
      <Name>Notaufnahme</Name>
      <Standort>Vorort</Standort>
    </Station>
  </Stationen>
</Klinik>
```

2.1.1 Die Struktur eines XML-Dokumentes

Das *Element* ist das grundlegende Konstrukt eines XML-Dokuments. Ein Element besteht aus folgenden Komponenten:

1. Ab hier werden wir die Worte »Dokument« und »XML-Dokument« synonym verwenden.

- *Start-Tag*
  Ein Start-Tag[2] beginnt mit einer öffnenden spitzen Klammer, die vom Namen des Elementes, optional von Attributen und jedenfalls von einer schließenden spitzen Klammer gefolgt wird.
- *Inhalt*
  Der Inhalt eines Elementes kann aus Text und verschachtelten Elementen bestehen.
- *End-Tag*
  Ein End-Tag beginnt mit einer öffnenden spitzen Klammer und einem Schrägstrich, gefolgt vom Namen des Elementes und einer schließenden spitzen Klammer.

Gemischter Inhalt

Wenn ein Element sowohl weitere Elemente als auch Text enthält, spricht man von gemischtem Inhalt, wie in folgendem Beispiel, in dem im Text Formatierungsanweisungen vorliegen:

```
<Befund>Mattheit, Fieber. Verdacht auf <em>schwere Grippe</em>
</Befund>
```

Für leere Elemente gibt es eine verkürzte Schreibweise. Folgende beiden Notationen sind äquivalent[3]:

```
<Bett ID="Bett_reha_25_001" Zimmernummer="025"></Bett>
<Bett ID="Bett_reha_25_001" Zimmernummer="025"/>
```

Attribute

Attribute bestehen aus Name und Wert, jeweils durch ein Gleichheitszeichen verbunden (im Beispiel `Zimmernummer="025"`). Der Wert muss in einfachen oder doppelten Anführungszeichen eingeschlossen sein. Ein Attributname darf pro Element nur einmal vorkommen. Während die Reihenfolge von Elementen Semantik trägt und nicht geändert werden darf, ohne dass sich das XML-Dokument ändert, ist die Reihenfolge der Attribute innerhalb eines Elementes beliebig vertauschbar.

Zum oben gezeigten Beispieldokument ist keine Strukturbeschreibung bekannt. Es ist also zwar wohlgeformt, aber nicht gültig. XML lässt dies ausdrücklich zu. Damit ein Dokument wohlgeformt ist, muss es nur wenige Syntaxanforderungen erfüllen. Die wichtigsten sind:

- *XML-Deklaration*
  Ein XML-Dokument darf mit einer XML-Deklaration beginnen, die beispielsweise folgendermaßen aussieht:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

2. *Tag*: englisch »Etikett«
3. Die XML-Spezifikation [W3C-3] gibt hier allerdings den – nicht bindenden – Ratschlag: »For interoperability, the empty-element tag should be used, and should only be used, for elements which are declared EMPTY.«

encoding

Dabei gibt das optionale `encoding` an, welcher Zeichensatz für das Dokument verwendet wurde, und die ebenfalls optionale `standalone`-Direktive, ob das Dokument externe Deklarationen enthält.

standalone

- *DTD*
  Nach der XML-Deklaration kann die Sprachdefinition in Form der *Document Type Definition* (DTD) folgen. Auf diese wird im folgenden Abschnitt näher eingegangen.
- *Wurzelelement*
  Ein XML-Dokument muss genau ein Element (auf oberster Ebene) enthalten (das normalerweise weitere Elemente enthält). Dieses wird als Wurzelelement bezeichnet.
- *Kommentare*
  An jeder Stelle außerhalb des Markup, also auch vor und nach dem Wurzelelement dürfen Kommentare stehen. Ein Kommentar beginnt mit `<!--` und endet mit `-->`.

processing instruction

- *Verarbeitungsanweisungen*
  Eine Verarbeitungsanweisung (»processing instruction«) gibt einer Applikation Hinweise, wie sie mit einem XML-Dokument umgehen soll. Bekannt ist die Verarbeitungsanweisung, die ein Stylesheet an ein XML-Dokument bindet:

```
<?xml:stylesheet type="text/xsl" href="stylesheets/print.xsl" ?>
```

  Eine Verarbeitungsanweisung beginnt immer mit »`<?`« gefolgt vom Ziel (*target*) — hier `xml:stylesheet` —, anhand dessen eine Anwendung entscheidet, ob sie die Verarbeitungsanweisung interpretieren kann. Danach folgt der eigentliche Inhalt der Verarbeitungsanweisung und die abschließende Zeichenfolge »`?>`«.

2.1.2 Die DTD

In der *Document Type Definition* (DTD) können die im Dokument zulässigen Elementtypen mit ihren Inhaltsmodellen und Attributen definiert werden. Eine DTD ist somit eine schematische Beschreibung des XML-Dokumentes, die allerdings optional ist. Ein Schema für ein Dokument kann auch mit einer anderen Schemasprache, beispielsweise XML Schema (Abschnitt 2.5), beschrieben sein. Ist keine Validierung notwendig, so ist es auch erlaubt, überhaupt keine schematische Beschreibung zu haben.

DOCTYPE

Wenn eine DTD vorhanden ist, wird sie durch `<!DOCTYPE` eingeleitet. Es folgen der Name des Dokumenttyps und, von eckigen Klammern umschlossen, Elementtyp- und Attributtypdefinitionen.

```
<!DOCTYPE Klinik [ ... ]>
```

Für jeden Elementtyp werden, durch `<!ELEMENT` eingeleitet, der Name und das Inhaltsmodell definiert. Als Auswahl für das Inhaltsmodell stehen neben dem leeren Inhalt eine Kombination aus Kindelementen oder ein textueller Inhalt oder eine Mischung daraus zur Verfügung. In diesen Fällen ist das Inhaltsmodell von runden Klammern umschlossen. Ein rein textueller Inhalt wird durch (`#PCDATA`) spezifiziert. Dabei ist eine nähere Typangabe (beispielsweise numerische Daten) nicht möglich:

```
<!ELEMENT Name (#PCDATA)>
```

Kardinalitätseinschränkungen

Ein leeres Element wird durch das Inhaltsmodell `EMPTY` in der Typdefinition gekennzeichnet. Elemente können wiederum Elemente enthalten (komplexer Elementinhalt). In diesem Fall kann man die erlaubten Elemente und Bedingungen an deren Kombination spezifizieren. Wenn mehrere Subelemente in einer bestimmten Reihenfolge auftreten sollen, werden diese durch ein Komma getrennt (»sequence«). Eine Auswahl (»choice«) aus mehreren möglichen Elementen wird durch einen senkrechten Strich spezifiziert.

```
<!ELEMENT Bett EMPTY>
<!ELEMENT Klinik (Name, Stationen)>
<!ELEMENT Adresse (Stadt, (Strasse | Postfach))>
```

Das erste Beispiel zeigt die Definition eines leeren Elementes, das zweite Beispiel die Definition eines Elementtyps `Klinik`, wobei ein Element dieses Typs als Kinder je ein Element `Name` und `Station` in dieser Reihenfolge enthalten muss, und das dritte Beispiel die Definition eines Elementtyps `Adresse`, bei dem Elemente zunächst ein Kindelement `Stadt`, und dann entweder ein Element `Strasse` oder ein Element `Postfach` enthalten müssen.

Wird nichts explizit spezifiziert, ist die Häufigkeit auf genau 1 beschränkt, d. h., ein entsprechendes Element muss genau einmal in einem Element des beschriebenen Typs vorkommen. Ein »`?`« kennzeichnet, dass ein solches Element höchstens einmal auftritt (also optional ist), ein »`+`« bedeutet, dass ein Element mindestens einmal erscheint, und ein »`*`« bedeutet, dass ein Element beliebig oft vorkommen (gegebenenfalls auch ganz fehlen) kann. Im folgenden Beispiel muss mindestens ein Vorname auftreten, während eine beliebige Anzahl von Telefonnummern und Faxnummern in beliebiger Reihenfolge erlaubt ist:

```
<!ELEMENT Person (Name, Vorname+, (Telefon | Fax)*)>
```

Gemischter Inhalt

Gemischter Inhalt wird dadurch beschrieben, dass an `#PCDATA` alle erlaubten Elementtypen mit einem senkrechten Strich angeschlossen werden. Eine Einschränkung der Kardinalität ist dabei nicht möglich – als Kardinalität des Inhaltsmodells muss »*« angegeben werden:

```
<!ELEMENT Befund (#PCDATA | b)*>
```

In manchen Fällen, so beispielsweise bei XHTML, können Elemente fast aller definierten Elementtypen als Inhalt eines Elementes vorkommen. Für eine kompakte Notation in diesem Fall wurde das Schlüsselwort `ANY` eingeführt. Es erlaubt alle in der DTD spezifizierten Elementtypen.

Attribute

Schließlich können in der DTD auch die für ein Element zulässigen Attribute beschrieben werden. Während Elementnamen im ganzen Dokument eindeutig sein müssen, müssen Attributnamen nur innerhalb ihres Elementes eindeutig sein. Daher werden sie unter Angabe des zugehörigen Elementtyps definiert (als `ATTLIST`). Eine Attributliste kann mehrere Attributtypen für einen zugehörigen Elementtyp definieren – es kann aber zu einem Elementtyp auch mehrere Attributlisten geben. Das allgemeine Format lautet:

```
AttlistDecl      ::= <!ATTLIST Name AttDef*>
AttDef           ::= Name AttType DefaultDecl
DefaultDecl      ::= #REQUIRED | #IMPLIED | #FIXED AttValue | AttValue
```

Ein Beispiel für eine solche Attributdeklaration ist:

```
<!ATTLIST Station Leitung CDATA #REQUIRED>
```

Folgende Attributtypen kann man in einer DTD verwenden:

- *Zeichenkette (String)*
  Die Zeichenkette wird als `CDATA` bezeichnet.
- *Identifikatortyp* `ID`
  Die Werte aller `ID`-Attribute müssen im Dokument eindeutig sein (auch wenn es sich um verschiedene Attribute handelt) und unterliegen syntaktischen Beschränkungen.
- *Referenztyp auf Attribute des Typs* `ID`
  Je nachdem, ob es sich um eine einzelne Referenz oder eine Liste solcher Referenzen handelt, heißt der Typ `IDREF` oder `IDREFS`.
- *Einzelnes Token (*`NMTOKEN`*) bzw. Liste von Tokens (*`NMTOKENS`*)*
  Ein einzelnes Token (»name token«) besteht aus einer Folge von Buchstaben, Zahlen und bestimmten Sonderzeichen, aber ohne Leerzeichen.

- *Aufzählungstypen*
  Die einzelne Werte sind dabei durch einen senkrechten Strich getrennt.

Außerdem gibt es in der Attributdefinition noch Angaben zu Häufigkeit und Vorbelegungswerten (»default values«).

- `#REQUIRED`
  Das Attribut muss explizit im Element vorkommen.
- `#IMPLIED`
  Das Attribut darf fehlen und hat dann keinen Vorbelegungswert.
- *Angabe eines Vorbelegungswertes*
  Dieser Wert gilt, wenn das Attribut in einem Element nicht explizit einen anderen Wert trägt.
- `#FIXED`
  Zusammen mit Angabe eines Wertes bedeutet dies einen Fehler, wenn das Attribut im Dokument mit einem anderen Wert auftritt. Der angegebene Wert gilt in jedem Fall.

Normalisierung

Attributwerte werden bei der Verarbeitung durch einen XML-Prozessor einer Normalisierung unterzogen. Dazu gehört neben dem Auflösen von Zeichen- und Entity-Referenzen, die im Folgenden ausführlich beschrieben werden, auch die Umwandlung von Leerraum (*»white space«*): Alle Leerraumzeichen (U+0009, U+000A, U+000D) werden zunächst zu Leerzeichen (U+0020) umgewandelt.

Die Reihenfolge der Definitionen in der DTD spielt keine Rolle. Sie kann ohne Änderung der Semantik vertauscht werden. Alle Definitionen in der DTD sind global. Das bedeutet, dass alle Elementtypdefinitionen auf alle anderen definierten Elementtypen (und sich selbst) in der Inhaltsdefinition Bezug nehmen können. Es können somit für einen Elementtyp mehrere »Elterntypen« definiert werden. Außerdem wird so auf Typebene eine direkte oder indirekte Rekursion möglich, wie hier am Beispiel einer Definition für einen Baum gezeigt wird:

```
<!ELEMENT Knoten (Knoten*)>

<!ATTLIST Knoten Name CDATA #REQUIRED>
```

Umgekehrt bedeutet die Globalität der Definitionen auch, dass Elementtypen, die in verschiedenen Kontexten verwendet werden, kein kontextspezifisches Inhaltsmodell haben können. Dies ist nur möglich, wenn der Elementtyp anders benannt wird.

Wie man sieht, sind die Typspezifikationen für Attribute und noch mehr die für Elemente recht unspezifisch. Mit XML Schema (Abschnitt 2.5) existieren viel ausgefeiltere Möglichkeiten der Typdefinition.

Externe DTD

Eine DTD kann auch extern zum Dokument vorliegen. In diesem Fall enthält das Dokument nur eine Referenz, zum Beispiel:

```
<!DOCTYPE Klinik SYSTEM "http://www.xquery-buch.de/klinik.dtd">
```

Des Weiteren ist auch die Kombination aus interner und externer DTD möglich. In diesem Fall überschreiben die Definitionen der internen DTD die der externen DTD.

Als Konsequenz können Dokumente, die alle auf dieselbe externe DTD verweisen, trotzdem unterschiedliche Typdefinitionen haben. Einige der eingeführten Konzepte zeigt das erweiterte Beispieldokument:

```
<?xml version="1.0"?>
<!DOCTYPE Klinik [
  <!ELEMENT Klinik (Name, Stationen)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Stationen (Station*)>
  <!ELEMENT Station (Name, Standort)>
  <!ATTLIST Station Leitung CDATA #REQUIRED>
]>
<!-- Dokument erstellt am 1.1.2004 -->
<?xml:stylesheet type="text/xsl" href="stylesheets/print.xsl" ?>
<Klinik>
  <Name>Hochwaldklinik</Name>
  <Stationen>
    <Station Leitung="Pfleger_01">
      <Name>Notaufnahme</Name>
      <Standort>Vorort</Standort>
    </Station>
  </Stationen>
</Klinik>
```

2.1.3 Textueller Inhalt eines Elementes

Wie bereits gezeigt wurde, besteht der Inhalt eines Elementes aus Text oder aus Kindelementen oder einer Mischung von beidem. Nun kann der Text in einem Element Zeichen enthalten, die normalerweise eine besondere Bedeutung in XML haben, wie z. B. das Zeichen »<«. Damit ein XML-Prozessor erkennen kann, dass es sich hier nicht um Markup, sondern um textuellen Inhalt eines Dokumentes handelt,

CDATA

kann man solch einen Text in einen `CDATA`-Abschnitt einbetten. Solche `CDATA`-Abschnitte sind in Elementen überall dort erlaubt, wo Zeichen stehen können. Sie beginnen mit `<![CDATA[` und enden mit `]]>` und lassen sich nicht verschachteln:

```
<Station Leitung="Pfleger_01">
  <Name><![CDATA[Notfall- & Durchgangsmedizin]]></Name>
</Station>
```

Parsed Entities

In einer DTD kann man so genannte »Parsed Entities« definieren, die wie die Makros in Programmiersprachen verwendet werden. Wenn sie vollständig in der internen DTD des Dokumentes definiert sind, heißen sie *intern*, sonst *extern*.

```
<!ENTITY hinweis "<Hinweis>Alle Angaben ohne Gewähr</Hinweis>">
<!ENTITY extern SYSTEM "http://www.xquery-buch.de/myentity">
```

Parsed Entities können Markup enthalten, wie das Beispiel zeigt (Entity `hinweis`). Der Wert einer Parsed Entity muss jedoch wohlgeformt sein.

Eine Entity wird durch das Symbol »&«, gefolgt vom Entity-Namen, referenziert und mit einem Semikolon abgeschlossen. Eine solche Referenz löst ein XML-Prozessor durch gegebenenfalls rekursives Einsetzen auf. Aus dem Dokumentfragment

```
<Kurs>&hinweis;20.0</Kurs>
```

wird mit obiger Entity-Definition:

```
<Kurs><Hinweis>Alle Angaben ohne Gewähr</Hinweis>20.0</Kurs>
```

Vordefinierte Entities

In XML sind fünf Entities vordefiniert, um die Verwendung von Zeichen, die in XML eine besondere Bedeutung haben, zu erleichtern. Es sind die Entities `lt` (für öffnende spitze Klammer), `gt` (für schließende spitze Klammer), `amp` (für `&`), `quot` (für `"`) und `apos` (für `'`). Für obiges XML-Fragment kann man also auch schreiben:

```
<Station Leitung="Pfleger_01">
  <Name>Notfall- &amp; Durchgangsmedizin</Name>
</Station>
```

Die vordefinierten Entities sind die einzigen Entities, die von XQuery unterstützt werden.

Zeichenreferenzen

Zeichenreferenzen haben die gleiche Syntax wie Entity-Referenzen. Das Unicode-Zeichen `U+00FF` kann z. B. entweder in hexadezimaler Form als `&#xFF;` oder in dezimaler Form als `&#255;` geschrieben werden. Zeichenreferenzen werden bevorzugt für Zeichen verwendet, die auf der jeweiligen Tastatur nicht vorhanden sind, sich in der Kodierung des Quellsystems darstellen lassen oder beim Transport des Dokumentes zu Seiteneffekten führen könnten. Zeichenreferenzen können auch in XQuery verwendet werden.