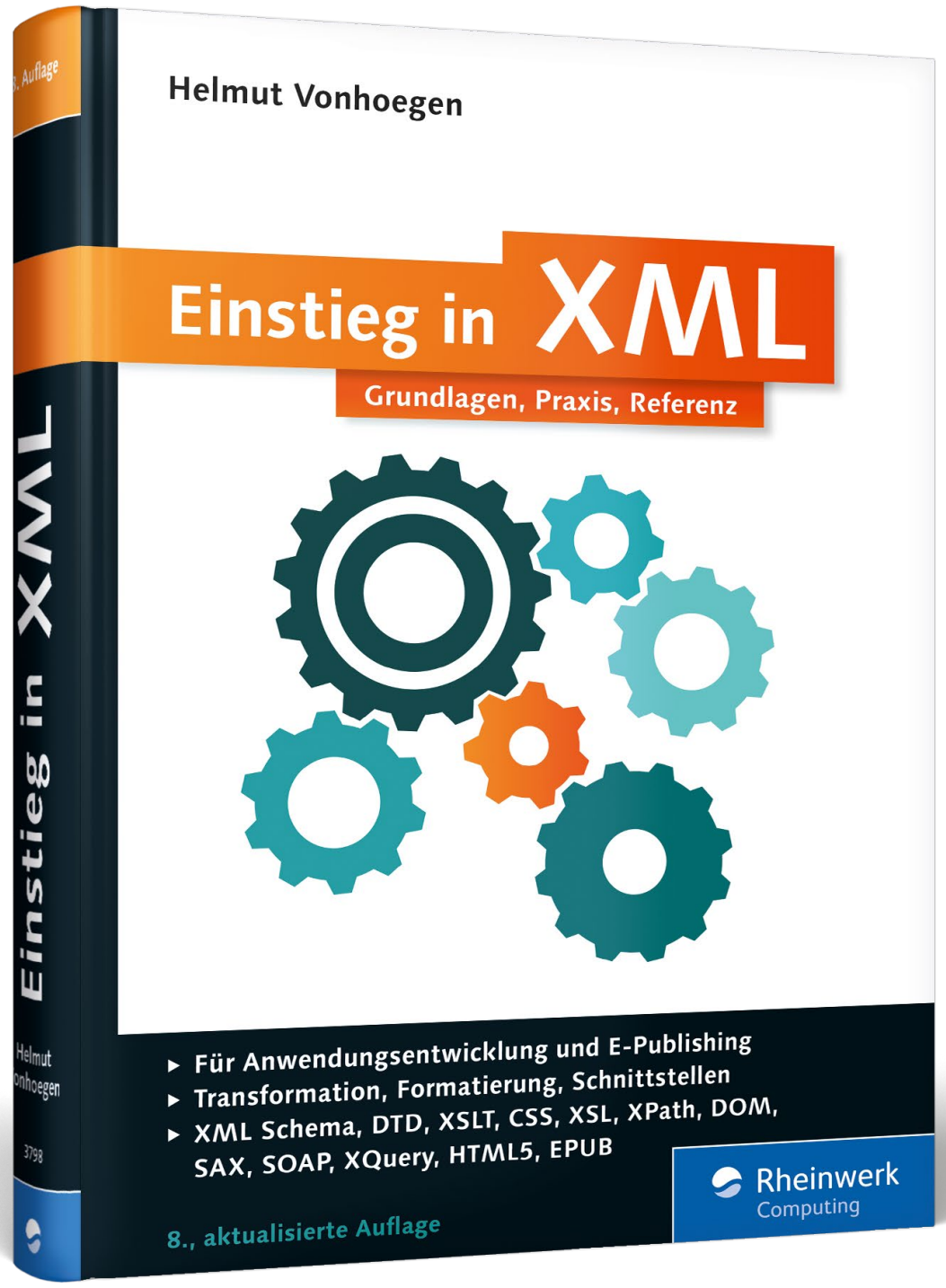
Einstieg in XML

Grundlagen, Praxis, Referenz

Bearbeitet von
Helmut Vonhoegen

8., überarbeitete Auflage 2015. Buch. 615 S. Gebunden
ISBN 978 3 8362 3798 7
Format (B x L): 16 x 24 cm



Leseprobe

Vom grundlegenden Aufbau über Validierung und Formatierung bis hin zu Transformation und Datenzugriff erfahren Sie alles über den Umgang mit XML-Dokumenten. In diesem Auszug stehen Ihnen die ersten drei Kapitel des Buches vollständig zur Verfügung. Außerdem enthält diese Leseprobe das Inhaltsverzeichnis und das gesamte Stichwortverzeichnis des Buches.

Helmut Vonhoegen

Einstieg in XML – Grundlagen, Praxis, Referenz

615 Seiten, gebunden, 8. Auflage 2015
39,90 Euro, ISBN 978-3-8362-3798-7

www.rheinwerk-verlag.de/3876

Kapitel 1
Einführung

Vor Babel: »Alle Welt hatte nur eine Sprache und dieselben Laute.«
(Genesis 11,1)

Damit Sie gleich wissen, wovon in diesem Buch die Rede ist, spielen wir zunächst ein kleines Beispiel durch. Die Daten einiger Buchprojekte werden in einer Liste zusammengestellt, um sie auf einer Seite im Web zu veröffentlichen.[1]

1.1 Kleines Einstiegsprojekt zum Kennenlernen

Der übliche Weg zu einer solchen Liste wäre vielleicht eine Datentabelle, die in einer Datenbankanwendung oder einem Kalkulationsprogramm realisiert wird. Sie würden sich einige Gedanken darüber machen, welche Feldnamen in der Kopfzeile dieser Tabelle auftauchen müssten, und später die einzelnen Datensätze unter dieser Kopfzeile eintragen.

1.1.1 Ein erstes XML-Dokument

Um das Problem mit den Mitteln von XML anzugehen, ist ein XML-Dokument erforderlich. Eine erste Lösung könnte in einer einfachen Variante so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<buchprojekte>
  <projekt name="Samsung Galaxy Tab">
    <teilnehmer>
      <autor>Helmut Vonhoegen</autor>
    </teilnehmer>
    <verlagsbereich>Vierfarben</verlagsbereich>
  </projekt>
  <projekt name="Einstieg in XML">
    <teilnehmer>
      <autor>Helmut Vonhoegen</autor>
```

1 Den Code der Beispiele können Sie von *www.rheinwerk-verlag.de/3876* herunterladen.

```
    </teilnehmer>
    <verlagsbereich>Rheinwerk Programmierung</verlagsbereich>
  </projekt>
</buchprojekte>
```

Listing 1.1 projektdaten.xml

Es wird sofort sichtbar, was bei XML-Dokumenten anders ist: Während Sie bei jedem Tabellenfeld nur über die Feldbezeichnung in der Kopfzeile erfahren, welche Information in dem einzelnen Feld enthalten ist, bringt in dem XML-Dokument jede Informationseinheit das Etikett mit der Beschreibung der Information gleich mit.

1.1.2 Standardausgabe im Webbrowser

Wenn Sie einen aktuellen Webbrowser verwenden, bei dem ein spezieller Prozessor, ein Parser für XML-Dokumente, integriert ist, lassen sich die XML-Daten darin ansehen. Gut geeignet dafür sind beispielsweise Firefox oder der Internet Explorer. Letzterer gibt das kleine XML-Dokument, das direkt vom lokalen Laufwerk geöffnet werden kann, wie in Abbildung 1.1, aus.

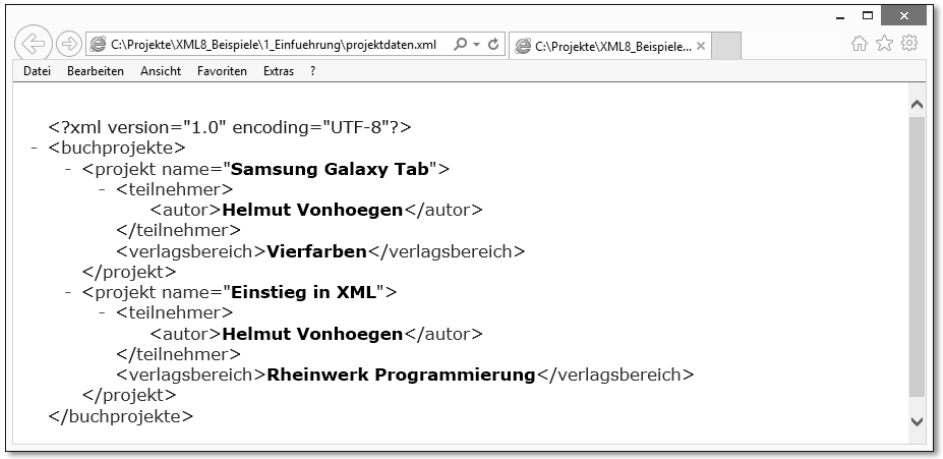


Abbildung 1.1 Wiedergabe einer XML-Datei im Internet Explorer

Der Internet Explorer verwendet zur Darstellung der Daten ein eingebautes Stylesheet. Die Darstellung entspricht ziemlich genau dem Quelltext, bis auf die vom Browser gewählte Hervorhebung der darin enthaltenen Textdaten und der teilweise vorgesetzten Minuszeichen.

Diese Minuszeichen sind Gliederungssymbole, mit deren Hilfe Sie Gruppen von Daten aus- oder einblenden können. Abbildung 1.2 zeigt zum Beispiel nur noch die Projektnamen.

Abbildung 1.2 Reduzierte Wiedergabe derselben XML-Datei

Das erste auffällige Merkmal an einem XML-Dokument ist, dass zu jeder einzelnen Information, die darin enthalten ist, an Ort und Stelle immer auch eine Benennung dieser Information beigefügt ist. Wenn XML als Datenformat gesehen wird, handelt es sich also um ein selbstbeschreibendes Format.

1.1.3 Wohlgeformtheit ist ein Muss

Aktuelle Webbrowser reagieren ziemlich streng, wenn auch nur eine dieser Beschreibungen in den spitzen Klammern, die *Tags* genannt werden, vergessen wird oder sich ein Schreibfehler darin einschleicht.

Abbildung 1.3 Reaktion des Chrome-Browsers auf das fehlerhafte End-Tag zu <teilnehmer>

In diesem Fall erkennt der im Browser integrierte XML-Parser, dass das XML-Dokument einen Formfehler hat – also nicht »wohlgeformt« ist, wie es im XML-Jargon heißt –, und bricht die Verarbeitung mit einer entsprechenden Fehlermeldung ab. Während der Webbrowser bei einer HTML-Seite allerlei Ungenauigkeiten durchgehen lässt und die Seite trotzdem anzeigt, herrscht also bei XML ein strenges Regime,

in der guten Absicht, Webanwendungen wenigstens in der Zukunft in einen solideren Zustand zu befördern.

1.1.4 Gültige Dokumente per DTD oder Schema

Die bloße formale Prüfung auf Wohlgeformtheit kann noch durch eine Prüfung auf Gültigkeit ergänzt werden. Dabei geht es um die Frage, ob das Dokument auch alle geforderten Informationen enthält, und zwar in der richtigen Reihenfolge. Ein Projekt in unserem Beispiel kann zwar mehr als einen Autor haben, aber kein Projekt kommt ohne Autor aus. Um die Elemente und Attribute und ihre Reihenfolge für ein Dokument genau festzulegen, kann dem XML-Dokument ein entsprechendes Datenmodell zugeordnet werden, an dem seine Gültigkeit dann zu messen ist.

Mit dem Werkzeug, das wir im Verlauf dieses Buchs noch häufiger verwenden werden – XMLSpy von Altova –, können Sie ein solches Datenmodell auch nachträglich aus einem schon vorliegenden XML-Dokument erzeugen. Dabei gibt es hauptsächlich zwei Möglichkeiten: Sie können eine DTD – eine Dokumenttyp-Definition – oder ein XML-Schema erzeugen. Die Rolle dieser beiden Datenmodelle wird ausführlich in Kapitel 3, »Dokumenttypen und Validierung«, behandelt. Eine DTD für unser Beispiel wird in Abbildung 1.4 dargestellt.

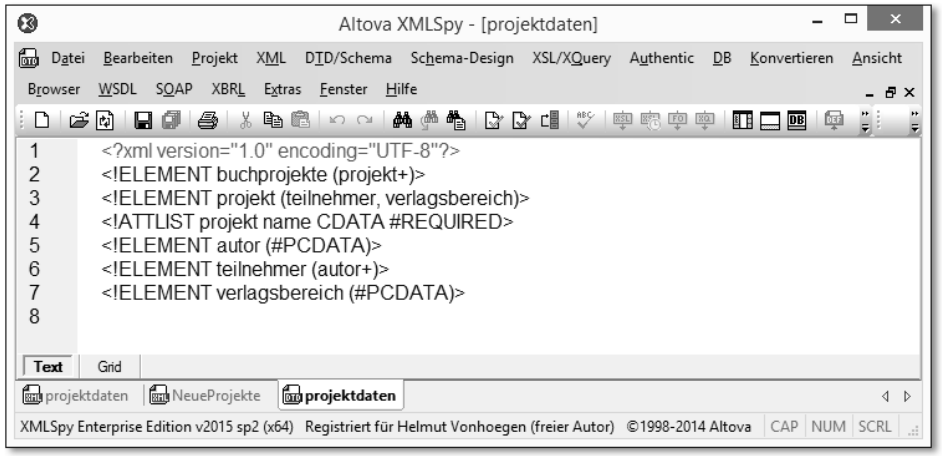


Abbildung 1.4 Eine DTD für die Projektdaten

Das gleiche Datenmodell in Form eines XML-Schemas zeigt Abbildung 1.5.

XMLSpy gibt dieses Datenmodell grafisch in einer entsprechenden Baumstruktur aus (siehe Abbildung 1.6). Der Modelldesigner kann sein Modell auch gleich in dieser grafischen Form entwerfen. Das erleichtert es, den Überblick über die Hierarchie der Informationseinheiten zu behalten, die er konstruiert.

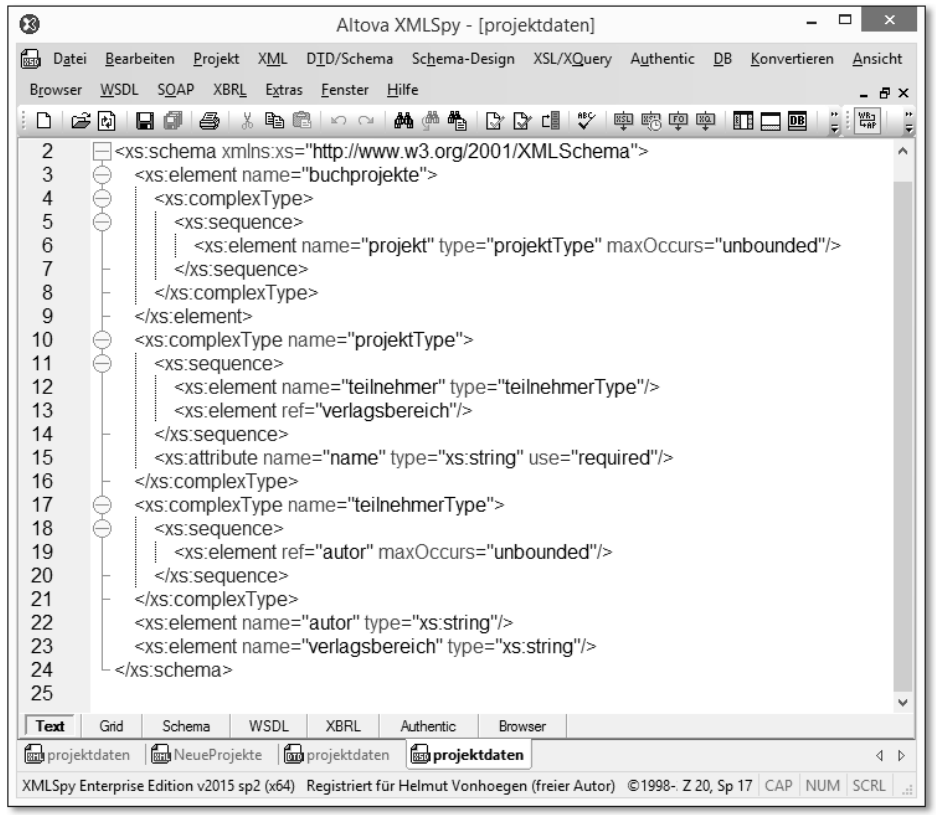


Abbildung 1.5 Das XML-Schema, das der DTD entspricht

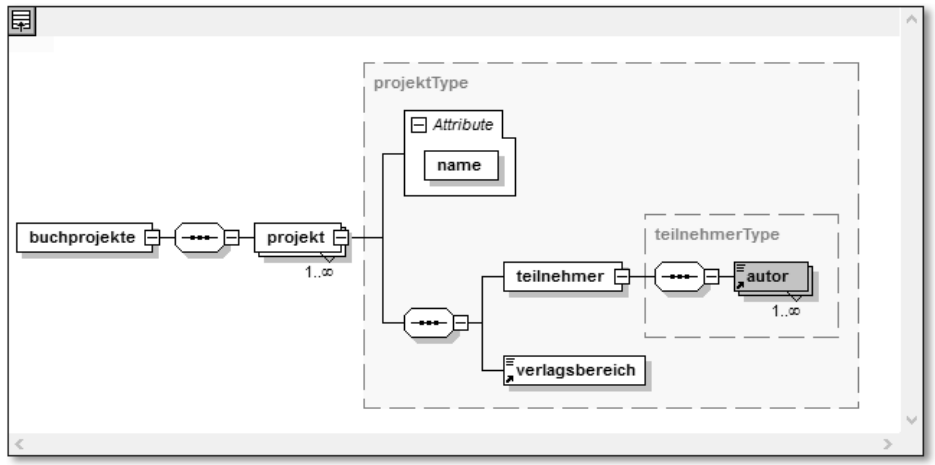


Abbildung 1.6 Das XML-Schema als Diagramm in XMLSpy

Die Gültigkeit eines Dokuments kann in XMLSpy immer sofort geprüft werden, weil ein validierender Parser integriert ist. Webbrowser ignorieren dagegen normalerweise die Frage der Gültigkeit, zeigen also auch ungültige Dokumente so lange an, wie sie wohlgeformt sind.

1.1.5 Formatierte Datenausgabe

Die Ausgabe von XML-Daten im Webbrowser in der oben abgebildeten Form ist natürlich nur ein Provisorium. Der wesentliche Unterschied zwischen einer XML- und einer HTML-Datei ist, dass das XML-Dokument eine reine Datensammlung ist, die zunächst noch keinerlei Hinweis darauf enthält, wie die Daten etwa in einem Browser zu präsentieren sind. Die oben abgebildete Darstellung kommt auch in dieser Form nur dadurch zustande, dass der Internet Explorer ein paar minimale Formatierungen vorgibt, etwa um die Tag-Namen farbig hervorzuheben. Andere Browser verhalten sich hier anders. Google Chrome zum Beispiel verwendet statt der Plus- oder Minus-Zeichen kleine Dreiecke, Safari zeigt nur die nackten Textdaten an.

Um die Daten in einer brauchbaren Form auszugeben, kann dem XML-Dokument ein Stylesheet beigegeben werden, entweder in Form von einfachen CSS-Stylesheets, wie sie auch für die Formatierung von HTML-Seiten verwendet werden, oder mit Stylesheets, die in der XML-Sprache XSLT oder XSL geschrieben sind.

Ein solches XSLT-Stylesheet könnte in diesem Fall beispielsweise so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="/">
    <html>
      <head>
        <title>Buchprojekte</title>
      </head>
      <xsl:apply-templates select="buchprojekte"/>
    </html>
  </xsl:template>
  <xsl:template match="buchprojekte">
    <body>
       <h1>Aktuelle Buchprojekte:</h1>
       <xsl:apply-templates select="projekt"/>
    </body>
  </xsl:template>
  <xsl:template match="projekt">
    <h3><xsl:value-of select="@name"></xsl:value-of></h3>
    <p>Autor(en):  <xsl:apply-templates select="teilnehmer"/></p>
```



```
    <p>Verlagsbereich: <xsl:value-of select="verlagsbereich">
    </xsl:value-of></p>
  </xsl:template>
  <xsl:template match="teilnehmer">
    <xsl:for-each select="autor">
      <p><xsl:value-of select="text()"></xsl:value-of></p>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Listing 1.2 projektdaten.xsl

Wird das XML-Dokument mit einer entsprechenden Anweisung mit dem Stylesheet verknüpft, zeigt beispielsweise der Firefox-Browser beim Öffnen des XML-Dokuments das in Abbildung 1.7 dargestellte Ergebnis.

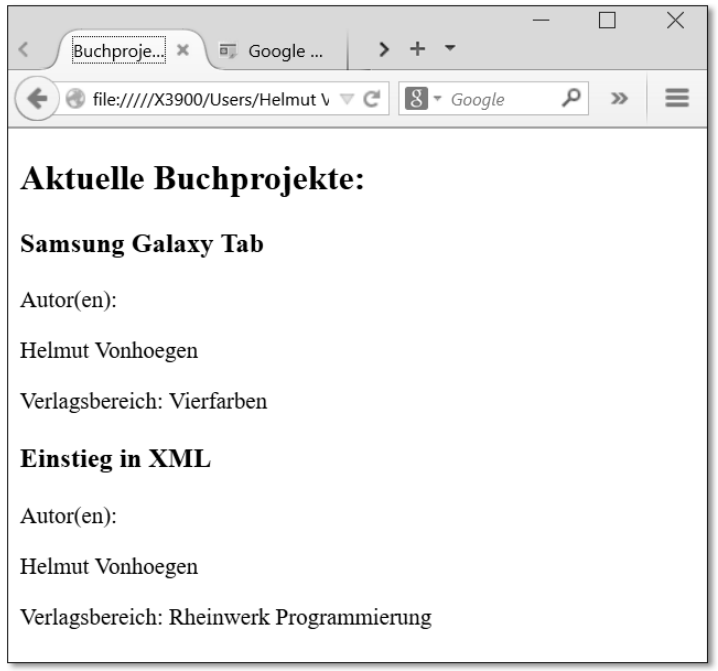


Abbildung 1.7 Die formatierte Ausgabe der Projektdaten in Firefox

1.2 XML – universale Metasprache und Datenaustauschformat

Der globale Erfolg des Internets wäre ohne offene Standards wie TCP/IP, HTTP und HTML nicht möglich gewesen. Derselben Philosophie ist XML verpflichtet, häufig

auch als »lingua franca« bezeichnet, in Anlehnung an die von den venezianischen Kaufleuten im Mittelmeerraum verwendete Verkehrssprache, einem mit arabischen Elementen vermischten Italienisch.

1.2.1 Unabhängigkeit von Anwendungen und Plattformen

Während die angesprochene Gruppe von offenen Standards dafür gesorgt hat, dass die Kommunikation zwischen Computern und die Präsentation von Daten weltweit möglich wurde, fehlte zunächst ein Standard, mit dessen Hilfe die Inhalte so strukturiert und beschrieben werden konnten, dass sie unabhängig von Anwendung und Plattform austauschbar sind und damit jederzeit wiederverwendet werden können.

Diesen Standard zu liefern, ist das Ziel der XML-Sprachfamilie, deren weltweite Geltung vom Word-Wide-Web-Konsortium – W3C – gewährleistet wird, das auch für die meisten anderen Standards verantwortlich ist, die den globalen Erfolg des Webs möglich gemacht haben. Die Hoffnungen, die mit XML verknüpft sind, gehen im Kern dahin, den Austausch von Informationen in ähnlicher Weise von Grenzen und Hindernissen zu befreien, wie der weltweite Handel durch den Abbau von Grenzbefestigungen und Zollschranken freier geworden ist.

1.2.2 SGML → HTML → XML

Es sind hauptsächlich zwei Motive, die die Entwicklung von XML befördert haben. Zum einen ergab sich, dass die bereits seit 1969 für die Beschreibung von Dokumentstrukturen hauptsächlich verwendete *Standard Generalized Markup Language (SGML)* für den Alltagsgebrauch zu komplex war, zum anderen galt HTML, die bisher einflussreichste aus SGML abgeleitete Auszeichnungssprache, aufgrund des eingeschränkten Satzes von Elementtypen für die weitere Entwicklung des Webs als zu inflexibel.

Zwar wurden häufiger neue HTML-Tags erfunden, aber diese Entwicklungen untergruben teilweise den Standard und führten für Entwickler und Webbesucher zu unangenehmen Browser-Inkompatibilitäten. Im Unterschied zu HTML gibt es bei XML keine festgelegte Liste von Tags. Der Entwickler kann seine eigenen Tags für die Strukturierung seiner Daten erfinden, oder er stützt sich auf öffentlich verfügbare Vokabulare, die für immer mehr Gegenstandsbereiche entwickelt werden.

Außerdem hat HTML den Nachteil, dass in vielen Fällen bei der Übersetzung von Dokumenten und Datenbeständen in der Darstellung von Webseiten Informationsgehalt verlorengeht. Nehmen Sie den Fall, dass Daten aus einer Datenbanktabelle einfach in eine HTML-Tabelle übernommen werden. Zwar sieht die abgebildete

Tabelle in etwa so aus, wie die entsprechende Tabelle im Fenster eines Datenbankprogramms. Während aber der Name eines Felds in der Datenbank Ihnen den Zugriff auf die Daten der entsprechenden Spalte ermöglicht, ist der entsprechende Name im Kopf der HTML-Tabelle nur ein einfacher Text.

Aus diesem Grund sind Suchmaschinen im Web im Wesentlichen auf die Volltextsuche angewiesen, weil die Webseiten zu wenige Informationen über die von ihnen dargebotenen Informationen anbieten, wenn man einmal von den üblichen `<meta>`-Tags absieht. Dass XML hier einen intelligenteren Zugriff eröffnet, war eine der Hoffnungen in Bezug auf die weitere Entwicklung des Webs.

Im Laufe der Zeit hat sich allerdings gezeigt, dass die Webentwickler eine viel stärkere Verhaftung an HTML entwickelt haben als zunächst angenommen. Mit HTML5 wurde deshalb eine Entwicklung in Gang gesetzt, die die Möglichkeiten von HTML wesentlich erweitert hat. Darauf wird in Kapitel 15 noch eingegangen.

1.2.3 Lob des Einfachen

Der Kern von XML ist einfach, und das ist sicherlich einer der Hauptgründe für seine universale Ausbreitung. XML definiert eine Syntax, um strukturierte Datenbestände jeder Art mit einfachen, verständlichen Auszeichnungen zu versehen, die zugleich von Anwendungen der unterschiedlichsten Art ausgewertet werden können. Diese Syntax ist aber zugleich streng. Der Anwender kann sich so seine eigenen Vokabulare zur Beschreibung der Dinge, mit denen er zu tun hat, aufbauen oder auf Vokabulare zurückgreifen, wie sie beispielsweise von Zusammenschlüssen einer Branche oder E-Commerce-Konsortien angeboten werden.

Daten und Auszeichnungen werden einfach in Textform abgelegt. Damit steht zugleich ein Datenaustauschformat zur Verfügung, das universal eingesetzt werden kann. Im Prinzip reicht für die Erstellung eines XML-Dokuments ein einfacher Texteditor, stattdessen gibt es aber auch komfortable XML-Editoren und komplette Entwicklungsumgebungen, die dem Entwickler eine Menge Arbeit abnehmen.

1.2.4 Inhaltsbeschreibungssprache

Die Extensible Markup Language ist keine Programmiersprache, da ihr Elemente für die Steuerung von Programmabläufen fehlen. Sie ist auch keine Seitenbeschreibungssprache wie etwa PostScript. Meist wird sie als Auszeichnungssprache bezeichnet, aber von einer einfachen Auszeichnungssprache wie HTML mit einer festgelegten Liste von Tags unterscheidet sie sich durch ihren generischen Charakter, also durch das X, durch die freie Erweiterbarkeit.

XML ist insofern eher eine Sprache zur Erzeugung von konkreten Auszeichnungssprachen, oder – etwas hoch gegriffen – eine Metasprache. »Inhaltsbeschreibungs-

sprache« schlagen Rothfuss/Ried in ihrem Buch »Content Management mit XML« (2001 im Springer Verlag erschienen) als passenderen Namen vor. Mit Hilfe von XML lassen sich bereichsspezifische Vokabulare festlegen, um die Elemente, aus denen sich Dokumente oder andere strukturierte Datenobjekte zusammensetzen, so zu beschreiben, dass Computer mit diesen Elementen umgehen können, als ob sie verstünden, was die den Elementen gegebenen Namen bedeuten.

Die große Stärke von XML ist dabei insbesondere, dass inhaltliche Strukturen in einer beliebigen Tiefe verschachtelt werden dürfen, so dass auch hoch komplexe Hierarchien jeder Art repräsentiert werden können.

1.2.5 Trennung von Inhalt und Form

Mit Hilfe von XML ist es möglich, die Struktur, den Inhalt und die Darstellung eines Dokuments streng zu trennen und entsprechend dann auch unabhängig voneinander zu be- und verarbeiten. Während die Tags in HTML in erster Linie festlegen, in welcher Form Inhalte in einem entsprechenden Medium ausgegeben werden sollen, wird mit XML versucht, die Bedeutung von Daten so festzuhalten, dass nicht nur Menschen, sondern auch Maschinen damit etwas anfangen können. Das erlaubt zum einen eine Prüfung der Gültigkeit von Dokumenten, ist zugleich aber auch die Basis für erweiterte Formen der Gestaltung und der Verknüpfung von Dokumenten.

Trotz des Einsatzes von Stylesheets ist eine klare Trennung von Inhalt und Form in HTML nicht möglich. Tags wie `<h1>`, `<h2>` etc. mischen inhaltliche Elemente – immer handelt es sich um Überschriften – unweigerlich mit Formatierungsanweisungen, auch wenn die Art der Formatierung variiert werden kann. Die Schwäche von HTML wird besonders bei Suchoperationen deutlich.

1.2.6 Vom Dokumentformat zum allgemeinen Datenformat

Zwar wurde XML zunächst als eine »digitale Repräsentation von Dokumenten« entworfen, entsprechend ihrer Entstehung als Light-Version der schwer zugänglichen Metasprache SGML – der SGML-Standard wurde auf etwa 500 Seiten definiert, für XML waren nur 40 erforderlich –, inzwischen aber hat XML längst die Aufmerksamkeit von Datenbankexperten, B2B- oder E-Commerce-Entwicklern auf sich gezogen.

Die Zahl der Programme, die ihre Daten automatisch in XML ausgeben oder importieren können, nimmt in allen Softwarebereichen ständig zu. Gleichzeitig stehen zahlreiche Möglichkeiten zur Verfügung, XML-Dokumente anzuzeigen, auszuwerten und weiterzuverarbeiten.

Die Übersetzung von Datenbeständen in XML ergibt natürlich nur Sinn, wenn die dabei entstehenden XML-Dokumente von entsprechenden XML-Prozessoren, von

Webbrowsern, Datenbanken, LDAP-Verzeichnisdiensten oder von Anwendungen auf mobilen Geräten auch ausgewertet und genutzt werden können. Dazu können die unterschiedlichsten Werkzeuge verwendet werden, angefangen bei einfachen JavaScripts bis hin zu komplexen Lösungen aus der Java-, COM- oder .NET-Welt.

1.2.7 Globale Sprache für den Datenaustausch

Seit der Einführung von XML im Jahre 1998 gab es kritische Kommentatoren, die XML als weitschweifige Variante des altertümlichen CSV-Datenformats abkanzelten, während häufig auf der Seite der Protagonisten die Inflation des Wortes »Revolution« vorangetrieben wurde. Die Bedeutung, die XML seitdem für die Entwicklung der Informationsverarbeitung gewonnen hat, hing aber nicht allein von den internen Eigenschaften des Produkts ab. Entscheidend war insbesondere, dass XML und die zugehörige Sprachfamilie tatsächlich von allen maßgeblichen Seiten als globaler Standard für den Datenaustausch akzeptiert wurden. Auch Esperanto ist von der Idee her nicht schlecht, aber die Entwicklung ist an dieser Idee vorbeigelaufen.

Zunehmend verwenden Anwendungen zudem XML als Vorzugsformat für alle Informationen, die die Anwendung selbst betreffen und steuern. Abbildung 1.8 zeigt ein typisches Beispiel, wie Anwendungseinstellungen in XML gespeichert werden. Apache benutzt beispielsweise XML als Format für die Konfigurationsdateien, die den Tomcat-Webserver steuern.

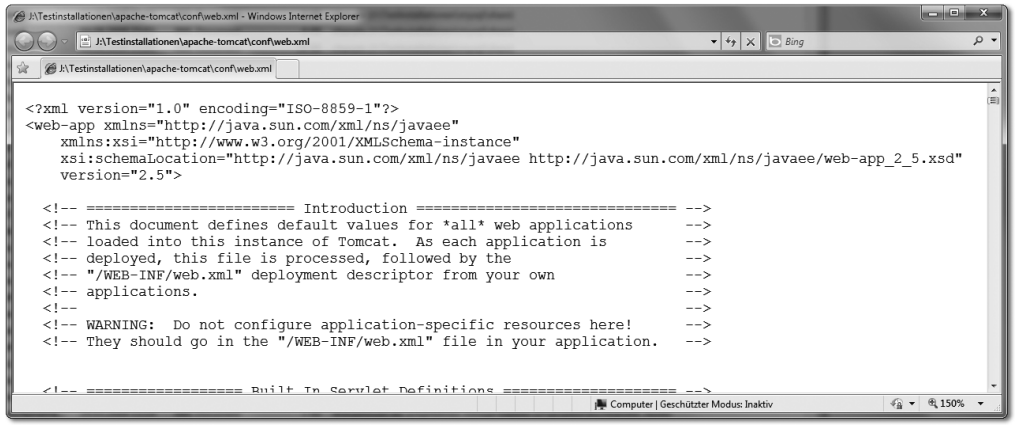


Abbildung 1.8 XML als Format für die Konfiguration des Tomcat-Servers

XML hat längst eine umfassende Unterstützung durch die großen Softwareanbieter wie IBM, Microsoft, Oracle etc. erreicht. Inzwischen werden die Dokumente der klassischen Büroanwendungen wie Textverarbeitung oder Tabellenkalkulation in erster Linie in XML-Dateiformaten gespeichert. Die Gefahr, dass der eine oder andere

Anbieter mit eigenen Produkten den Markt zu besetzen versucht, scheint im Wesentlichen abgewendet. Die schnelle Verbreitung des XML-Schema-Standards hat das Gewicht der XML-Technologie weiter befördert. Insofern kann von zukunftssicheren Investitionen gesprochen werden, wenn Energie in die Entwicklung von X-Lösungen gesteckt wird.

In jedem Fall hat XML die Art und Weise, wie Dokumente und Daten zwischen den unterschiedlichen Anwendungen, Systemen und Medien ausgetauscht werden, ganz entscheidend verbessert und mitgeholfen, die automatisierte Verarbeitung der immer schneller anwachsenden Informationsmassen zu befördern.

1.2.8 Interoperabilität

XML ist insbesondere auch der Titel für ein Bündel von Technologien, mit dem eine direkte Verknüpfung von Geschäftsprozessen, unabhängig von den jeweils verwendeten Anwendungen und Betriebssystemen, einfacher als bisher möglich geworden ist.

Dabei spielt das Konzept der Webdienste eine zentrale Rolle. Das von Microsoft, IBM und Ariba Ende 2000 in Gang gesetzte UDDI-Projekt – *Universal Description, Discovery and Integration*; ein weltweites öffentliches Unternehmensregister, in dem Firmen Webdienste sowohl registrieren lassen als auch suchen können, um Produkte und Dienstleistungen auszutauschen – basiert hauptsächlich auf XML und SOAP. Zwar ist das öffentliche UDDI-Verzeichnis nach einer mehrjährigen Testphase 2005 eingestellt worden, unternehmensinterne UDDI-Verzeichnisse auf der Basis des Standards UDDI Version 3 des *OASIS International Standards Consortiums* sind dagegen etablierte Bestandteile von serviceorientierten Architekturen in großen Unternehmen. Davon wird in Kapitel 11, »Kommunikation zwischen Anwendungen«, noch die Rede sein.

1.3 Übersicht über die Sprachfamilie XML

Wenn über die Errungenschaften oder Verheißungen von XML gesprochen wird, ist in der Regel nicht bloß der XML-Standard selbst gemeint, sondern ein ganzes Bündel von Standards, die die Sprachfamilie XML bilden. Zudem kommen immer wieder neue Mitglieder hinzu; bei einigen ist der Prozess der Standardisierung noch im Gange. Den schnellsten Zugang zum aktuellen Stand aller Standardisierungsprojekte des W3C finden Sie über die Adresse *www.w3.org/TR/*.

Abbildung 1.9 gibt einen kurzen Überblick über die wichtigsten Mitglieder der Sprachfamilie.

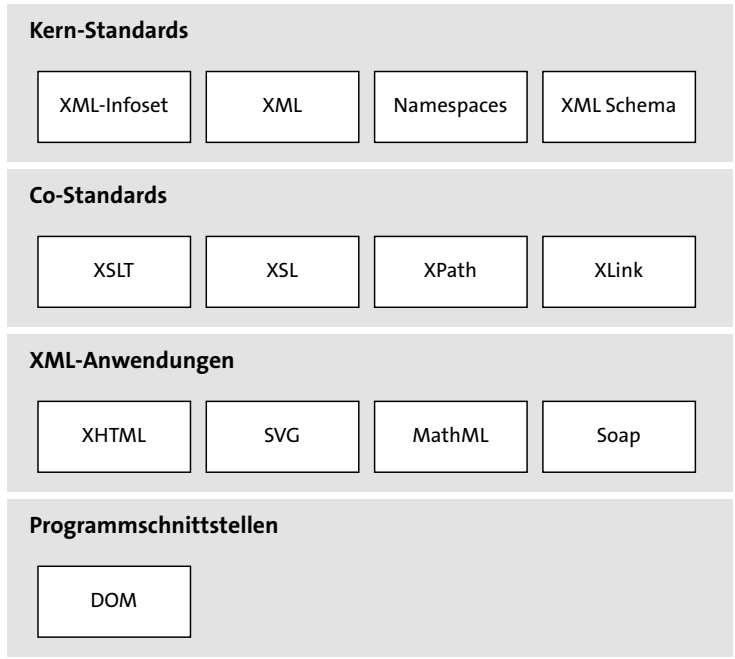


Abbildung 1.9 Überblick über die Sprachfamilie XML

1.3.1 Kernspezifikationen

Den Kern der Sprachfamilie bilden die *XML 1.0*-Spezifikation (seit Februar 1998), ihre Erweiterung mit *XML Namespaces* (seit Januar 1999) und seit Mai 2001 die Sprache zur Definition von Inhaltsmodellen, *XML-Schema*. Die theoretische Basis dieser vom W3C verabschiedeten Empfehlungen, die direkt den Inhalt von XML-Dokumenten betreffen, wurde im Oktober 2001 noch einmal separat als *XML Information Set* – kurz *Infoset* – formuliert, um einen konsistenten Satz von Definitionen für alle Spezifikationen rund um XML zur Verfügung zu stellen.

Seit 1998 ist die XML-Spezifikation – abgesehen von einigen nachträglichen Fehlerkorrekturen (die fünfte Edition erschien im November 2008) – unverändert geblieben, was ihre Verbreitung durchaus befördert hat. 2004 verabschiedete das W3C eine Version 1.1, die der Weiterentwicklung von Unicode zu Version 4.0 und darüber hinaus entsprechen sollte. 2006 erschien eine Second Edition der Version 1.1. Näheres dazu finden Sie im Abschnitt 2.10. Obwohl die Änderungen von Version 1.0 zu 1.1 insgesamt gering blieben, sind die entsprechenden XML-Dokumente nicht kompatibel. Eine XML 1.1-Datei ist für einen Version-1.0-Parser unter Umständen also nicht wohlgeformt. Praktisch folgt daraus, dass die Version 1.1 wohl über die nächsten Jahre in der täglichen Arbeit mit XML-Daten kaum eine Rolle spielen wird. Aus diesem Grunde wird in diesem Buch auch weiterhin von der Version 1.0 ausgegangen.

1.3.2 Ergänzende Spezifikationen

Die *Extensible Stylesheet Language – XSL –* ist für die Formatierung von XML-Dokumenten zuständig. Diese Aufgabe erwies sich als so umfangreich, dass XSL in drei Teile zerlegt wurde. Zunächst war es notwendig, eine Sprachregelung für die exakte Adressierung der einzelnen Einheiten in der Struktur eines XML-Dokuments zu schaffen. Dafür wurde im November 1999 ein Standard mit der *XML Path Language – XPath –* geschaffen.

Auf der Basis der Adressierungsregelung durch *XPath* wurden ebenfalls im November 1999 die *XSL Transformations – XSLT –* standardisiert, eine Sprache für die Umwandlung eines XML-Dokuments in ein anderes XML-Dokument oder auch ein anderes Format.

Eine neuere Version – *XPath 2.0* – liegt seit Januar 2007 als Empfehlung vor. Ebenfalls im Januar 2007 wurde die Empfehlung für *XSLT 2.0* veröffentlicht. Als Erweiterung von XPath wurde gleichzeitig die Empfehlung für die Abfragesprache *XQuery 1.0* verabschiedet. *XPath 3.0* wurde im April 2014 verabschiedet.

Die Formatierung von XML-Dokumenten mit XSL findet praktisch in einem zweistufigen Verfahren statt, bei dem das Dokument mit Hilfe von XSLT inhaltlich für die vorgesehene Präsentation präpariert wird, um das so gewonnene Material schließlich mit den vorgesehenen Formatierungen auszugestalten. XSLT kann aber auch unabhängig von Formatierungsaufgaben zur Umwandlung von XML-Dokumenten aus einem XML-Vokabular in ein anderes und insbesondere zur Erzeugung von HTML-Seiten verwendet werden.

Das eigentliche Vokabular für die Spezifizierung von Formatierungsregeln durch XSL sind die *XSL Formatting Objects*, ein Vokabular, das alle möglichen Gestaltungsmerkmale einer Publikation abdecken soll. XSL wurde im Oktober 2001 als Empfehlung verabschiedet. Die Version 1.1 folgte im Dezember 2006.

Dass auch Cascading StyleSheets für die Formatierung von XML-Dokumenten verwendet werden können, wurde in einer kurzen Empfehlung vom Juni 1999 geregelt.

1.3.3 Programmierschnittstellen

Schnittstellen für den Zugriff auf XML-Dokumente von anderen Anwendungen aus oder innerhalb von Webseiten hat das W3C in einer mehrteiligen Spezifikation für das *Document Object Model – DOM –* definiert, die seit 2004 Level 3 erreicht hat. Dabei handelt es sich um abstrakte Schnittstellen, die in konkreten Entwicklungsumgebungen implementiert werden können.

Eine andere Programmierschnittstelle mit dem Namen *Simple API for XML – SAX –* hat sich dagegen neben dem W3C als De-facto-Standard etabliert, entwickelt von einer Gruppe, die über die XML-DEV-Mailingliste kooperierte. Die Spezifikation kann über *www.saxproject.org* eingesehen werden.

In der Java-Welt werden zahlreiche Programmierschnittstellen unter dem Titel *Java API for XML Processing – JAXP –* zur Verfügung gestellt. Microsoft entwickelte zunächst eine DOM-Implementierung und APIs für SAX2 in Form der *XML Core Services* (*MSXML*) für die COM-Welt. Innerhalb des Microsoft .NET-Frameworks stehen über die *System.Xml*-Bibliothek Basisklassen für das Lesen und Schreiben von XML-Dokumenten zur Verfügung. Mehr zu alledem in Kapitel 10, »Programmierschnittstellen für XML«.

1.3.4 XML-Anwendungen

Die Bezeichnung *XML-Anwendung* wird für Auszeichnungssprachen verwendet, die mit Hilfe von XML definiert wurden. Es handelt sich also um XML-Vokabulare oder Dokumenttypen, die für bestimmte Bereiche fixiert wurden. In Abbildung 1.9 sind nur ein paar besonders wichtige aufgeführt: *XHTML*, die XML-kompatible Reformulierung von HTML 4.0, *Scalable Vector Graphics – SVG –*, eine Sprache für die Beschreibung von zweidimensionalen Grafikanwendungen und Bildern, *Mathematical Markup Language – MathML –*, eine Sprache für die Darstellung von mathematischen Formeln und komplexer Ausdrücke, und *Simple Object Access Protocol – SOAP –*, ein Vokabular für den Nachrichtenaustausch zwischen Anwendungen.

Während MathML in der ersten Version bereits 1999, XHTML seit Mai 2001 und SVG seit September 2001 als Empfehlungen vom W3C verabschiedet sind, wurde SOAP erst im Juni 2003 in mehreren Teilen vom W3C spezifiziert, basierend auf einem Vorschlag zu SOAP von einigen Firmen, der zunächst nur als Note des W3C veröffentlicht worden war.

1.4 XML-Editoren und Entwicklungsumgebungen

Aufgrund des Textformats lassen sich XML-Dokumente, DTDs und Schemas oder XSL- und XSLT-Dateien im Prinzip mit jedem Texteditor erstellen und bearbeiten. Das verspricht allerdings eine Menge < und > und wieder </ und >. Und der Name in jedem End-Tag muss haargenau mit dem im Start-Tag übereinstimmen. Außerdem muss die Groß-/Kleinschreibung immer beachtet werden. Das alleine schon spricht für den Einsatz spezieller Editoren.

1.4.1 Editoren für XML

Es gibt eine ganze Reihe von komfortablen XML- und Stylesheet-Editoren, die den Anwendern vieles abnehmen können, schon dadurch, dass End-Tags immer automatisch erzeugt werden, wenn das Start-Tag abgeschlossen ist.

Eine einfache Lösung ist die von Microsoft seit Anfang 2007 angebotene freie Version von XML Notepad, die eine übersichtliche Oberfläche für das Editieren und Ansehen von XML-Dokumenten zur Verfügung stellt. Der Editor beherrscht auch die direkte Überprüfung der Eingabe gegenüber einem vorhandenen Schema und kann die Daten auch mit einem eingebauten HTML-Viewer anzeigen.

Die Hierarchie der Elemente kann in einer Baumansicht angezeigt werden. Die aktuelle Version 2.6 wird über *www.codeplex.com*, das von Microsoft unterstützte Portal für Open Source Software angeboten.

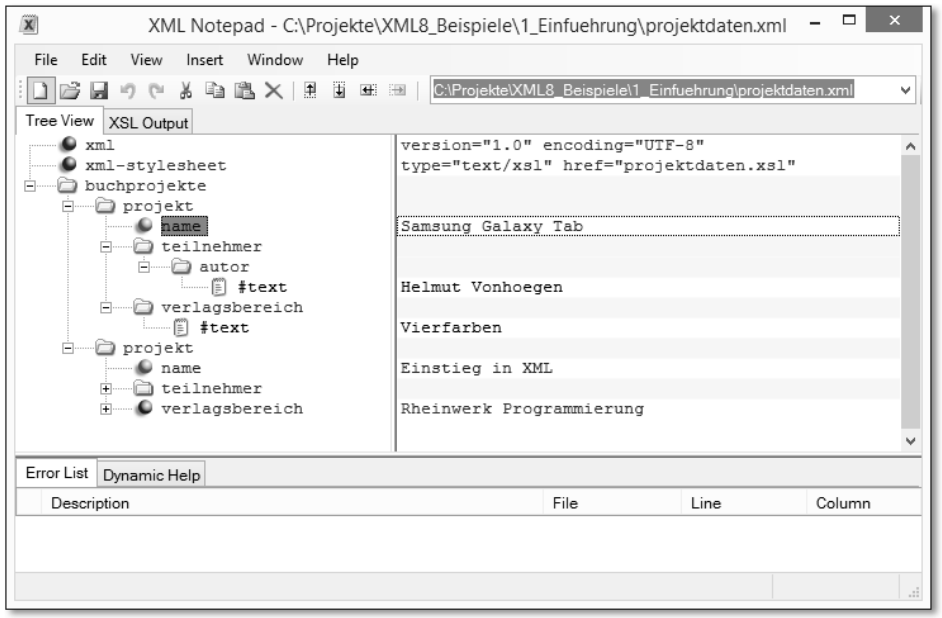


Abbildung 1.10 Baumansicht eines XML-Dokuments in XML Notepad

Komplette XML-Entwicklungswerkzeuge wie XMLSpy oder Stylus Studio, um nur zwei zu nennen, unterstützen Sie bei der Arbeit, zum Beispiel durch kontextsensitive Elementlisten oder Attribut-Inspektoren, und sorgen durch integrierte XML-Prozessoren dafür, dass die Wohlgeformtheit und auch die Gültigkeit der Dokumente schon während der Entwicklung jederzeit geprüft werden kann. (Testversionen finden Sie unter *www.altova.com/de/download-trial.html* bzw. *www.stylusstudio.com/xml_download.html*)

Tools wie XMLSpy unterstützen auch die Generierung von Eingabemasken für XML-Dokumente, die Ihnen die manuelle Eingabe von Tags gleich ganz abnehmen.

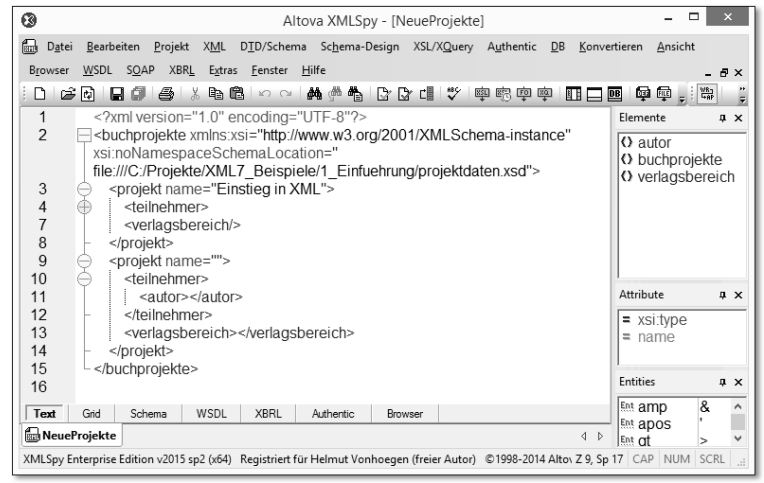


Abbildung 1.11 Vorgabe von Tags aufgrund eines Schemas in XMLSpy 2015

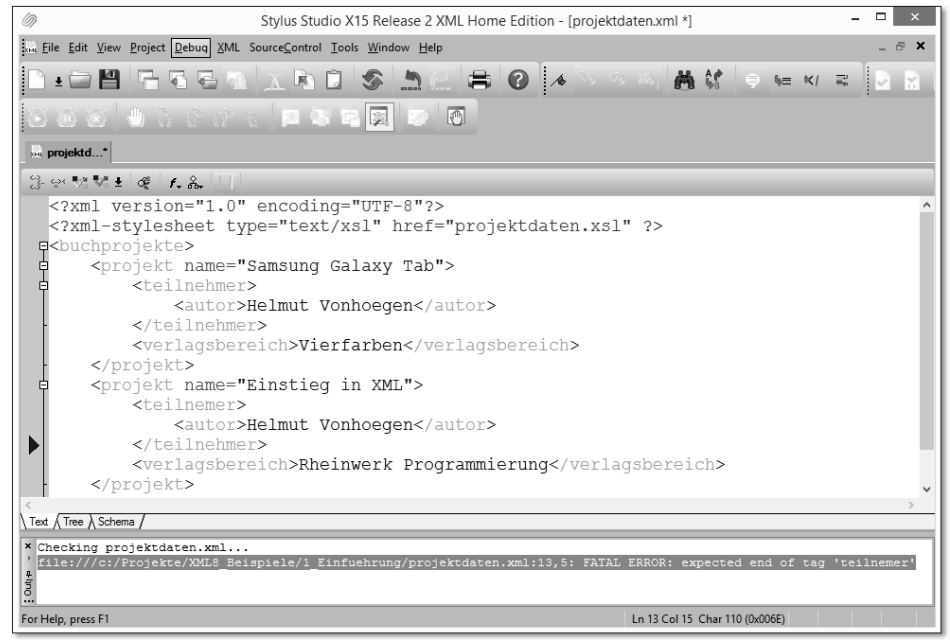


Abbildung 1.12 Fehlermeldung beim Prüfen eines Dokuments in Stylus Studio X15

1.4.2 Schema- und Stylesheet-Designer

Insbesondere der Entwurf von DTDs oder XML-Schemas wird sehr erleichtert durch grafische Designansichten, in denen die Baumstruktur eines XML-Dokuments aufgebaut und geprüft werden kann. Diese Programme sind außerdem in der Lage, DTDs

oder Schemas aus bestehenden XML-Dokumenten zu generieren oder DTDs in XML-Schemas zu übersetzen. Meist lassen sich die Ergebnisse wenigstens als Ausgangspunkt für ein endgültiges Dokumentmodell verwenden.

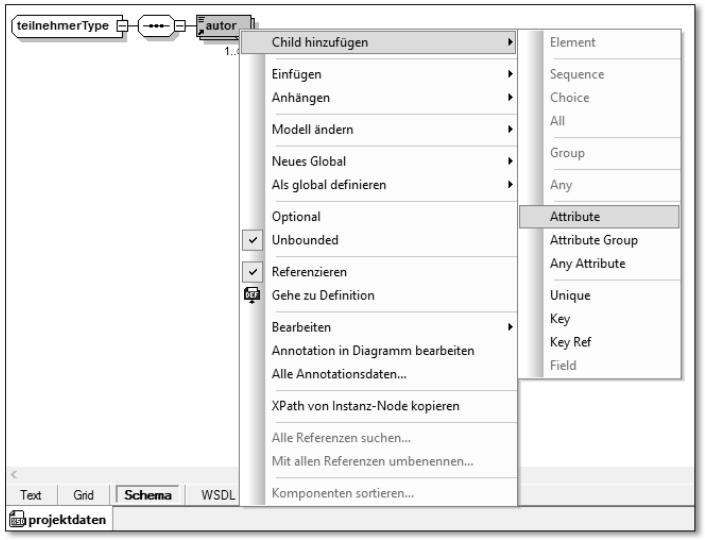


Abbildung 1.13 Grafischer Entwurf eines Schemas in XMLSpy 2015

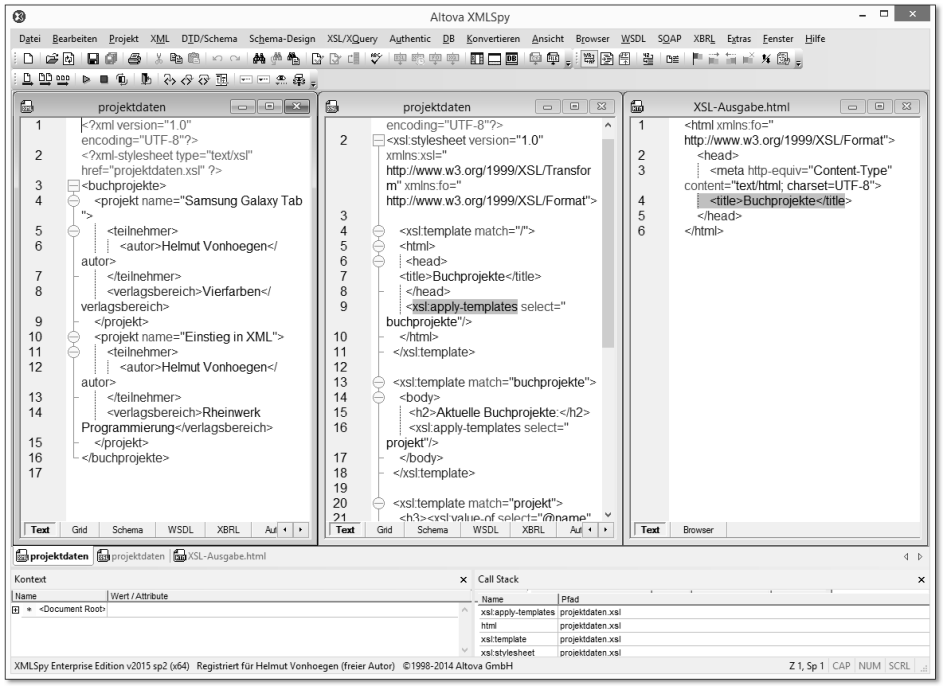


Abbildung 1.14 XSLT-Entwicklung mit XMLSpy 2015

Auch für die Entwicklung von Stylesheets sind komfortable Werkzeuge auf dem Markt. XMLSpy bietet beispielsweise einen leistungsfähigen Debugger für den XSLT-Code, mit dem der Aufbau der Ausgabe schrittweise verfolgt werden kann.

1.4.3 Entwicklungsumgebungen mit XML-Unterstützung

Neben solchen Spezialprogrammen für XML-Anwendungen finden Sie auch in allgemeinen Entwicklungsumgebungen wie Eclipse oder Visual Studio .NET spezielle Editoren für XML.

Sie werden im Verlauf des Buchs einige dieser Werkzeuge in Aktion sehen. Im Anhang finden Sie eine Liste von Links zu Werkzeugen und Tools rund um XML.

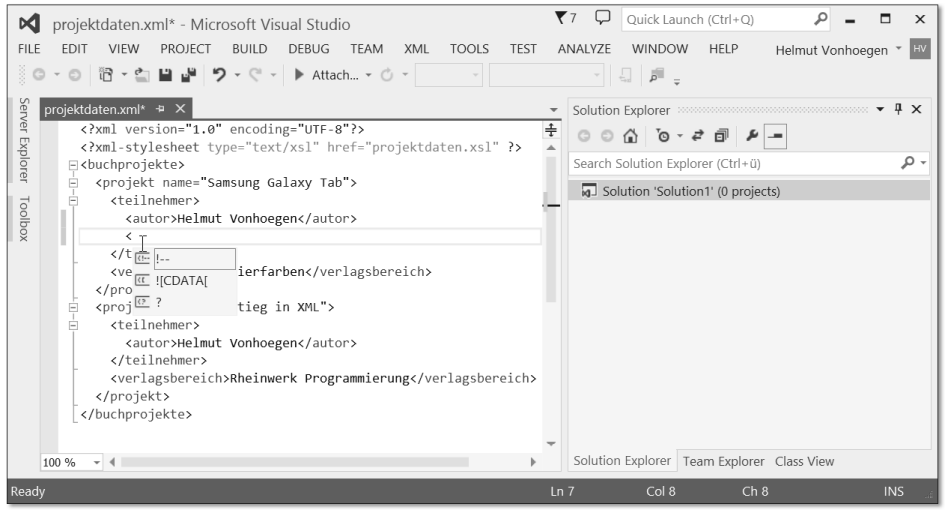


Abbildung 1.15 Editieren eines XML-Dokuments in Visual Studio Express

1.4.4 XML-Dokumente über Standardanwendungen

In vielen Situationen ist es gar nicht erforderlich, spezielle Editoren für die Erstellung von XML-Dokumenten zu verwenden, weil die Daten aus Standardanwendungen entweder direkt in XML ausgegeben oder in das XML-Format exportiert werden können, das dann als zusätzliches Datenformat angeboten wird. Auf XML als Standardformat für Office-Anwendungen wird in Kapitel 12, »XML in Office-Anwendungen«, eingegangen.

Dies gilt etwa für Textdateien oder Datenbanken. Auch viele Grafikprogramme unterstützen sowohl den Export als auch den Import im XML-Format. Vor allem Adobe hat seine Programme in dieser Hinsicht immer stärker erweitert, etwa Illustrator oder InDesign.

Besonders einfach ist die Erstellung von XML-Dokumenten aus Microsoft-Access-Tabellen. Über die Funktion Exportieren • XML-Datei im Kontextmenü zu einer Tabelle können nicht bloß reine XML-Dokumente aus Tabellen generiert werden, sondern gleich auch passende XML-Schemas und XSL-Stylesheets. Hier ein Beispiel für ein automatisch erzeugtes XML-Dokument aus einer Access-Datentabelle:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata">
  <adressdaten>
    <ID>1</ID>
    <Vorname>Jan</Vorname>
    <Nachname>Korn</Nachname>
    <Strasse>Poststr. 10</Strasse>
    <PLZ>50678</PLZ>
    <Ort>Köln</Ort>
  </adressdaten>
  <adressdaten>
    <ID>2</ID>
    <Vorname>Helga</Vorname>
    <Nachname>Cron</Nachname>
    <Strasse>Salierring 12</Strasse>
    <PLZ>50678</PLZ>
    <Ort>Köln</Ort>
  </adressdaten>
</dataroot>
```

Die Daten der Tabelle werden in ein XML-Dokument mit dem Wurzelelement `<dataroot>` eingefügt, dessen Namensraumattribut die Herkunft aus einem Office-Programm verrät.

1.4.5 Parser und andere Prozessoren

Wie schon angesprochen sind für die Verwertung von XML-Dokumenten spezielle XML-Prozessoren notwendig, die die Daten parsen und bei Bedarf auch auf Gültigkeit prüfen können. Zusätzlich werden für die Ausgabe mit Hilfe von Stylesheets spezielle XSLT-Prozessoren und XPath-Prozessoren benötigt. Für Abfragen mit XQuery werden ebenfalls spezielle XQuery-Prozessoren genutzt.

In diesem Buch werden Sie zunächst Anwendungen auf der Basis der *Microsoft XML Core Services (MSXML*; *msdn.microsoft.com/xml/default.aspx*) und der *Java API for XML Processing (JAXP)* kennenlernen. In Kapitel 10, »Programmierschnittstellen für XML«, und Kapitel 11, »Kommunikation zwischen Anwendungen«, wird außerdem

mit XML-Klassen aus dem .NET Framework von Microsoft gearbeitet. Für einen Einstieg können die frei verfügbaren Editionen von Visual Studio verwendet werden.

Da XML bekanntlich plattform- und sprachunabhängig ist, wird es in der Regel kein Problem sein, die in diesem Buch gezeigten Beispiele auch mit anderen Werkzeugen nachzuvollziehen, die die entsprechenden Standards unterstützen.

1.5 Anwendungsbereiche

Dieser Abschnitt ist dazu gedacht, Ihnen einige Hinweise zu wichtigen Implementierungen von XML zu geben. Dabei geht es insbesondere darum, einen Eindruck von der Vielfalt der Anwendungsmöglichkeiten zu vermitteln.

Während bei der Entstehung von XML vor allem die Verarbeitung von Textdokumenten, ihre Publikation im Web und ihre »intelligente« Nutzung im Fokus standen, rückte XML später insbesondere auch im Bereich der Datenbanktechnologien und der Verarbeitung strukturierter Daten als Basistechnologie in den Blick.

1.5.1 XML-Vokabulare

Dass sich XML inzwischen als Metasprache für benutzerdefinierte Markup-Sprachen etabliert hat, belegt die ständig wachsende Zahl von XML-Sprachen, die sich in bestimmten Bereichen oder für bestimmte Anwendungsformen etabliert haben. Wir wollen hier ein paar Hinweise zu einigen dieser Sprachen geben.

Dokumentauszeichnung und Content Management

Wichtige XML-Vokabulare wurden für Bereiche entwickelt, in denen ein besonderer Bedarf für spezielle Formen der Dokumentauszeichnung besteht. Dazu gehört die *Text Encoding Initiative – TEI –*, die DTDs und Schemas für jede Art von literarischen und linguistischen Texten anbietet. Sie helfen, Online-Recherchen und -Unterricht für Bibliotheken, Museen, Publizisten etc. zu vereinfachen. Inzwischen ist auch eine *TEIlite*-Version verfügbar. Quelle: *www.tei-c.org*.

Die umfangreiche DTD *DocBook* wurde für Dokumentationen zu Hard- und Software entwickelt. Sie wird vom DocBook Technical Committee gepflegt. Quelle: *www.oasis-open.org/docbook*. Eine vereinfachte Version – *Simplified DocBook* – wurde 2005 veröffentlicht. Mehr dazu finden Sie in Abschnitt 3.11.1, »Die Auszeichnungssprache DocBook«.

Auch das inzwischen bei elektronischen Büchern weit verbreitete EPUB-Format basiert hauptsächlich auf XML. Die Spezifikation wurde vom *International Digital Publishing Forum – IDPF –* veröffentlicht, die Quelle ist *www.openbook.org*. Mehr dazu finden Sie in Kapitel 14.

NITF, das *News Industry Text Format*, liegt seit 2012 in der Version 3.6 als DTD oder XML-Schema vor und soll den Nachrichtenaustausch im Medienbereich durch Hinzufügen von Metadaten so aufbereiten, dass Recherchen erleichtert werden. NITF wird vom International Press Telecommunications Council gepflegt. Quelle: *www.iptc.org*.

Die amerikanischen Behörden benutzen unter dem Namen NIEM – National Information Exchange Model – Pakete von XML-Schemas, um die Belange in zahlreichen Bereichen konsistent abzubilden, siehe *www.niem.gov*.

Eine große Rolle im Web spielen inzwischen auch XML-basierte Formate für die Verbreitung von Inhalten. Dazu gehören insbesondere die *RSS*-Formate, die Inhalte in Form abonnierbarer Feeds anbieten, siehe *web.resource.org/rss* und *www.rssboard.org*. Die Abkürzung hat versionsweise ihre Bedeutung gewechselt: für *RSS 1.0* gilt *RDF Site Summary*, für *RSS 2.0* wird *Really Simple Syndication* angegeben. Das sehr übersichtliche Vokabular basiert seit der Version 1.0 auf *RDF*, dem *Resource Description Framework*, einer Sprache zur Beschreibung von Ressourcen im Web, für die ebenfalls eine XML-basierte Syntax zur Verfügung steht, siehe *www.w3.org/TR/rdf-syntax-grammar*.

Bei der Darstellung von mathematischen oder chemischen Formeln helfen Vokabulare wie *MathML* – Mathematical Markup Language – und *CML* – Chemical Markup Language. Quellen sind *www.w3.org/math* und *www.sourceforge.net/projects/cml*.

XML bietet sich insbesondere auch als Basistechnologie für das Content Management an, also dort, wo es um einen effektiven Umgang mit umfangreichen Informationsbeständen geht. Es erlaubt nicht nur eine beliebig fein strukturierte Speicherung der Informationen selbst, sondern hilft auch bei der Pflege der notwendigen Meta-Informationen.

Finanztransaktionen und E-Business

Andere Sprachen wurden für spezielle Gebiete wie den Austausch von Finanzdaten entwickelt, z. B. *IFX* – Interactive Financial EXchange (*www.ifxforum.org*) – oder *FIX* – Financial Information eXchange Protocol (*www.fixtradingcommunity.org*).

Besonderes Gewicht haben die Anstrengungen, verbindliche Vokabulare für die Abwicklung von Geschäftsprozessen – branchenübergreifend oder auch branchenspezifisch – zu entwickeln, um die Geschäftsbeziehungen zwischen verschiedenen Firmen (B2B) oder von Firmen zu Kunden (B2C) zu vereinfachen und auch in größerem Umfang durch automatisierte Prozesse kostengünstiger zu gestalten. Allerdings sind diese Versuche teilweise sehr aufwendig konzipiert und überschneiden sich häufig, so dass alternativ dazu die Idee aufkam, einen etwas flexibleren, übergreifenden Versuch mit einer Universal Business Language – UBL – zu unternehmen, der Standards für typische Geschäftsdokumente schaffen soll, die auch von kleineren Unternehmen ohne großen Aufwand verwendet werden können. UBL 2.0

wurde im Dezember 2006 als Standard ratifiziert. Mehr dazu über *www.oasis-open.org/committees/ubl/*. Im Bereich der Finanzberichterstattung spielt die *eXtensible Business Reporting Language – XBRL* – eine wichtige Rolle, die von XBRL International entwickelt wurde (*www.xbrl.org*).

Repositorien für DTDs oder Schemas werden von verschiedenen Organisationen angeboten und gepflegt. Eine der wichtigsten ist RosettaNet, ein weltweit operierendes Konsortium, das Standards für eine allgemeingültige E-Business-Sprache fördert – *www.rosettanet.org*.

Grafik und Multimedia

Eine der praktischen Anwendungen von XML, die immer mehr an Gewicht gewinnen, ist das Grafikformat *SVG* – die Abkürzung steht für »Scalable Vector Graphics«. SVG beschreibt Vektorgrafiken im XML-Code und wird von Grafik- oder Webdesign-Programmen wie Illustrator, CorelDRAW oder von OpenOffice als Datenformat unterstützt. HTML5 kann inzwischen SVG-Fragmente direkt integrieren, wovon in Kapitel 15, insbesondere in Abschnitt 15.8.1, noch die Rede sein wird.

Wenig Einfluss hatte dagegen die XML-Anwendung *SMIL* (sprich engl. »smile«); die Synchronized Multimedia Integration Language wurde entwickelt, um mit Hilfe einfacher Textdateien multimediale Komponenten ganz unterschiedlicher Formate in einer Timeline zu koordinieren. 2012 hat das W3C die Pflege von SMIL eingestellt.

1.5.2 Datenaustausch zwischen Anwendungen

Einer der Gründe für die Popularität von XML ist zweifellos, dass es ein Datenformat für den Austausch von Daten zwischen unterschiedlichen Anwendungen und Plattformen bietet, das einfach zu handhaben ist und gegenüber so simplen Formaten wie CSV zugleich den Vorteil mit sich bringt, dass nicht bloß einfache Tabellenstrukturen gehandhabt werden können, sondern auch komplexe hierarchische Inhaltsmodelle, die das jeweilige Sachgebiet wesentlich gehaltvoller abbilden können.

Der Austausch zwischen Anwendungen, die eigene binäre Formate verwenden, ist durch die zunehmende Komplexität dieser Formate immer mehr zu einem Problem geworden, das nur durch ganze Batterien von Filtern und Konvertern gelöst werden konnte. Sehen Sie sich nur die zahllosen Grafikfilter an, die in Office-Anwendungen benötigt werden, um Dokumente mit Bildmaterial anzureichern.

In relativ kurzer Zeit hat sich deshalb XML als bevorzugtes Dateiformat vieler Standardprogramme etabliert. Anwendungen wie Access oder Excel sind zudem in der Lage, tabellarische Daten direkt als XML-Dokumente auszugeben und auch wiederum einzulesen. Auch ein Grafikprogramm wie Visio von Microsoft kann seine Zeichnungen und Diagramme als XML-Dokument abspeichern, mit Hilfe eines entsprechenden

Schemas, das ein XML-Vokabular für grafische Elemente definiert. Adobe Acrobat erlaubt es, die Inhalte von PDF-Dateien in XML abzulegen, so dass sie von anderen Anwendungen weiterverwendet werden können.

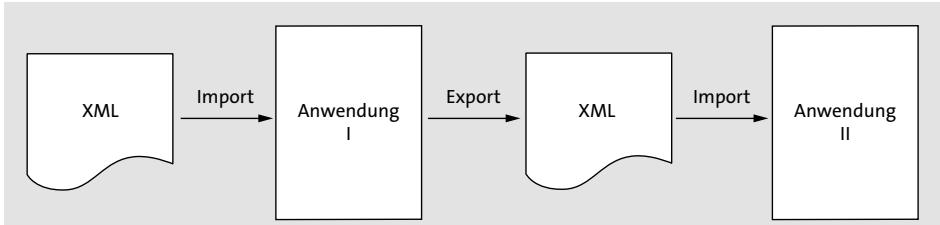


Abbildung 1.16 XML als Austauschformat zwischen unterschiedlichen Anwendungen

1.5.3 Verteilte Anwendungen und Webdienste

Die anspruchsvollsten Versuche, die Möglichkeiten von XML zu nutzen, betreffen zweifellos die Frage, ob sich mit XML ein besseres Zusammenspiel von verteilten Anwendungskomponenten erreichen lässt, als es mit bisherigen Komponentensystemen wie Java oder COM möglich ist. Da XML von allen relevanten Herstellern unterstützt wird, plattformunabhängig ist und auch nicht an bestimmte Sprachen gebunden, ergeben sich auf jeden Fall Vorteile, wenn es um den Austausch von Informationen zwischen verteilten Anwendungen geht.

Webdienste gehören zu den Anwendungsbereichen, bei denen XML als Basistechnologie in diesem Sinne zum Zuge kommt. In Microsofts .NET-Framework spielen solche Webdienste eine zentrale Rolle, und auch IBM und Oracle machen große Anstrengungen, taugliche Werkzeuge für solche Lösungen anzubieten.

Da es dabei immer um Kommunikation zwischen ganz unterschiedlichen Systemen geht, wurde mit SOAP ein spezielles Format für die XML-Meldungen entwickelt, die für den Austausch im Internet oder auch im firmeneigenen Intranet verwendet werden. Mehr dazu finden Sie in Kapitel 11, »Kommunikation zwischen Anwendungen«.

Kapitel 2
XML – Bausteine und Regeln

Die Regeln, nach denen XML-Dokumente gebildet werden, sind einfach, aber streng. Die eine Gruppe von Regeln sorgt für die Wohlgeformtheit, die andere für die Gültigkeit eines Dokuments.

Die Basis der Sprachfamilie XML ist der XML-Standard 1.0. Sie finden den Text unter *www.w3.org/TR/xml*. Der launige Kommentar von Tim Bray, selbst Mitglied der W3C XML Working Group, ist unter *www.xml.com/axml/axml.html* zu finden und immer noch einen Blick wert. Wir wollen hier zunächst einen kurzen Überblick über die Syntax geben, verknüpft mit einem einfachen Beispiel, und dann schrittweise die einzelnen Aspekte näher beleuchten.

2.1 Aufbau eines XML-Dokuments

Laut der Empfehlung des W3C beschreibt XML eine Klasse von Datenobjekten, die XML-Dokumente genannt werden. Das entscheidende Kriterium, ob ein Dokument als XML-Dokument genutzt werden kann, ist, dass es im Sinne des Standards »wohlgeformt« ist. Um wohlgeformt zu sein, muss es die syntaktischen Regeln der XML-Grammatik erfüllen, die in den folgenden Abschnitten beschrieben wird.

Entspricht ein solches Dokument außerdem weiteren Einschränkungen, die in Form eines Dokumentschemas in der einen oder anderen Weise festgelegt sind, wird es als »gültig« bezeichnet. Dabei ist entscheidend, dass die Wohlgeformtheit und Gültigkeit maschinell geprüft werden können.

2.1.1 Entitäten und Informationseinheiten

Der vage Begriff »Datenobjekt« bezieht sich darauf, dass ein XML-Dokument nicht unbedingt eine Datei sein muss, sondern auch ein Teil einer Datenbank oder eines Datenstromes sein kann, der im Netz »fließt«. In einem bestimmten Umfang werden dabei zugleich die Regeln festgelegt, die für den Zugriff von Computerprogrammen auf solche Dokumente gelten.

Die XML-Spezifikation behandelt sowohl die physikalischen als auch die logischen Strukturen eines XML-Dokuments. Physikalisch bestehen XML-Dokumente aus

Speichereinheiten. Zunächst ist ein XML-Dokument nichts anderes als eine Kette von Zeichen. Ein XML-Prozessor startet seine Arbeit mit dem ersten Zeichen und arbeitet sich bis zum letzten Zeichen durch. XML liefert dabei Mechanismen, um diese Zeichenkette in verwertbare Stücke zu zerlegen. Diese Textstücke werden *Entitäten* genannt. Auch das Dokument insgesamt wird als Entität bezeichnet, als Dokumententität. Im Minimalfall kann eine Entität auch aus nur einem einzigen Zeichen bestehen.

Jede dieser Entitäten enthält Inhalt und ist über einen Namen identifiziert, mit Ausnahme der Dokumententität, die alle anderen Entitäten in sich einschließt. Für einen XML-Parser ist die Dokumententität, der Container für alle anderen Entitäten, immer der Startpunkt. Liegt ein XML-Dokument als Datei vor, ist die Dokumententität eben diese Datei. Wenn Sie dagegen ein XML-Dokument über einen URL einfließen lassen, ist die Dokumententität der Bytestream, den Sie über einen Funktionsaufruf erhalten.

XML erlaubt, Bezüge auf bestimmte Entitäten in das Dokument einzufügen. Solche Entitätsreferenzen werden vom XML-Prozessor durch die Entität, auf die sie sich beziehen, ersetzt, wenn das Dokument eingelesen wird. Deshalb werden solche Entitäten auch *Ersetzungstext* genannt. Das ist ähnlich den Textbausteinen, die von Textprogrammen verwendet werden.

Der Ersetzungstext, den eine aufgelöste Entitätsreferenz liefert, wird als Bestandteil des Dokuments behandelt. Mit Hilfe solcher Referenzen können Entitäten in einem Dokument mehrfach verwendet werden. Durch solche Referenzen kann ein XML-Dokument auch aus Teilen zusammengesetzt werden, die in verschiedenen Dateien oder an unterschiedlichen Plätzen im Web abgelegt sind.

2.1.2 Parsed und unparsed

Eine weitere Unterscheidung ist hier von Bedeutung. Entitäten können laut Spezifikation »parsed or unparsed data« enthalten. *Parsed data* besteht in jedem Fall aus Zeichen. Diese Zeichenfolgen stellen entweder Markups oder Zeichendaten dar. Als Markups sind zu verstehen die Tags, Entitätsreferenzen, Kommentare, die Begrenzer von CDATA-Blöcken, Dokumenttyp-Deklarationen und Verarbeitungsanweisungen – also alles, was mit einer spitzen Klammer oder einem Ampersand-Zeichen beginnt. Die Bezeichnung »parsed« ist leicht irritierend, weil sie erst zutrifft, wenn ein XML-Prozessor das Dokument verarbeitet hat. Gemeint sind also die Teile des Dokuments, die ein XML-Parser auszuwerten hat.

Als *unparsed data* bezeichnet man dagegen Entitäten, die der Parser überhaupt nicht parsen soll und auch nicht kann, weil sie keine Markups enthalten, mit denen der Parser etwas anfangen könnte. Diese Teile müssen nicht unbedingt Text enthalten.

Sie werden zum Beispiel verwendet, um Bilder oder sonstige Nicht-Text-Objekte wie Sounds oder Videos in das Dokument einzubeziehen.

2.1.3 Die logische Sicht auf die Daten

Während die physikalische Struktur eines XML-Dokuments durch die Entitäten bestimmt wird, besteht seine logische Struktur aus einem Baum von Informationseinheiten, der seit der erst 2001 nachgereichten Empfehlung *XML Information Set* als *Infoset* bezeichnet wird. (Darin ist auch die Verwendung von Namensräumen aufgenommen, die für XML erst nach der Spezifikation für XML 1.0 eingeführt wurden.) Die Empfehlung definiert insgesamt elf Typen von Informationseinheiten mit jeweils speziellen Eigenschaften:

- Dokument
- Element
- Attribut
- Verarbeitungsanweisung
- nicht expandierte Entitätsreferenz
- Zeichen
- Kommentar
- Dokumenttyp-Deklaration
- ungeparste Entität
- Notation
- Namensraum

Die wichtigsten Komponenten, in die sich der Inhalt des Dokuments teilen lässt, werden in XML als *Elemente* bezeichnet. Der Baum der Elemente hat seine Wurzel im Dokumentelement, das alle anderen Elemente umschließt. Das XML-Dokument besteht also logisch aus Elementen, die jeweils in einer bestimmten Baumstruktur geordnet sind. Welche Bedeutung die Elemente jeweils haben, beschreibt das Dokument selbst durch seine Tags. Neben den Elementen enthält das Dokument noch Deklarationen, Kommentare, Zeichenreferenzen und Verarbeitungsanweisungen.

Die Grammatik von XML legt fest, wie ein wohlgeformtes XML-Dokument erzeugt werden kann. Sie ist in nicht weniger als 81 Produktionsregeln fixiert. Der harte Kern dessen, was XML ausmacht, findet sich aber konzentriert in den folgenden sechs Regeln, die wir hier zunächst in der Schreibweise der Empfehlung wiedergeben.

```
[1]  document  ::= prolog element Misc*
[39] element   ::= EmptyElemTag
     | STag content ETag
     [
       WFC: Element Type Match ]
     [
       VC: Element Valid ]
[40] STag      ::= '<' Name (S Attribute)* S? '>'
       WFC: Unique Att Spec ]
[41] Attribute ::= Name Eq AttValue
       VC: Attribute Value Type ]
       WFC: No External Entity References ]
       WFC: No < in Attribute Values ]
[42] ETag      ::= '</' Name S? '>'
[43] content   ::= (element | CharData | Reference | CDSect |
                   PI | Comment)*
```

Diese Regeln, die auch *Produktionen* genannt werden, sind in einer einfachen Extended-Backus-Naur-Form notiert, einer Erweiterung der von Backus und Naur für die Beschreibung von Grammatiken zuerst bei der Niederschrift von Algol 60 verwendeten Notation. Jede Regel in einer solchen Grammatik definiert jeweils ein Symbol in der Form:

```
symbol ::= ausdruck
```

Zusätzlich werden in bestimmten Fällen Einschränkungen in eckigen Klammern angehängt, und zwar entweder mit der Abkürzung `WFC`: für *well-formedness constraint* – also Einschränkungen, die beachtet werden müssen, damit das Dokument von einem Parser als »wohlgeformt« akzeptiert wird, oder `VC`: für *validity constraint*, also Einschränkungen, die die Gültigkeit des Dokuments betreffen.

Was besagen diese Regeln für den Aufbau eines XML-Dokuments? Zunächst ist festgelegt, dass jedes wohlgeformte XML-Dokument einen Prolog haben kann und aus mindestens einem Element bestehen muss. Der Inhalt eines XML-Dokuments wird also aus logischer Sicht in Elemente zerlegt. Im Anschluss daran sind noch Kommentare oder Verarbeitungsanweisungen erlaubt, was aber in der Praxis nicht unbedingt zu empfehlen ist.

Abbildung 2.1 zeigt den gesamten Aufbau des XML-Dokuments und die möglichen Bezüge auf externe Komponenten.

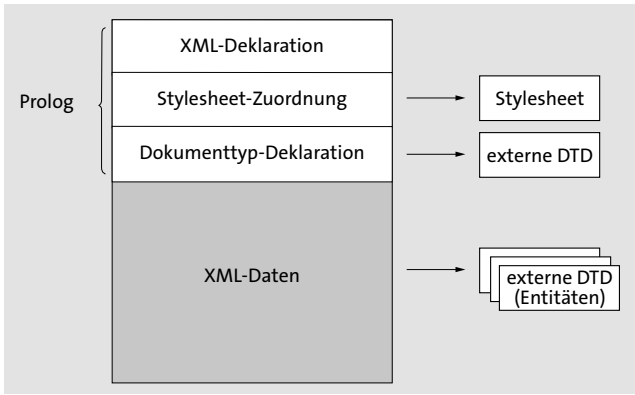


Abbildung 2.1 Aufbauschema eines XML-Dokuments

2.1.4 Der Prolog

Wenn Sie ein XML-Dokument erstellen wollen, beginnen Sie in der Regel mit dem Prolog. Der Prolog ist zwar nicht zwingend vorgeschrieben, aber unbedingt zu empfehlen, weil damit das Dokument sofort als XML-Dokument identifiziert werden kann. Die erste Zeile des Prologs ist meist die sogenannte XML-Deklaration, die zunächst die verwendete XML-Version angibt. In der Minimalform sieht sie so aus:

```
<?xml version="1.0"?>
```

Damit wird die Übereinstimmung des Dokuments mit einer der im Augenblick gültigen Spezifikationen von XML deklariert. Wenn die XML-Deklaration verwendet wird, muss sie in der ersten Zeile des Dokuments stehen, und es dürfen auch keine Leerzeichen davor auftauchen. Das Versionsattribut ist erforderlich.

Neben dem Versionsattribut können in der XML-Deklaration noch zwei weitere Attribute benutzt werden, und zwar `encoding` und `standalone`. Im folgenden Beispiel wird angegeben, dass das Dokument die Zeichensatzkodierung UTF-16 verwendet und dass keine externen Markup-Deklarationen vorhanden sind, auf die für die Verarbeitung des Dokuments zugegriffen werden müsste, also etwa eine externe Dokumententyp-Definition oder ein XML-Schema.

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

2.1.5 Zeichenkodierung

Da es bei XML-Dokumenten um Textdaten geht, muss eine Entscheidung getroffen werden, wie Zeichen in Bits und Bytes dargestellt, also kodiert werden sollen, und welche Zeichen, also welcher Zeichensatz, in einem bestimmten Dokument maßgeblich ist.

Um XML von vornherein für den internationalen Einsatz zu präparieren, wurde vom W3C genauso wie für die HTML 4.01-Empfehlung das *Universal Character Set – UCS –* als Basis für die Zeichenkodierung in XML-Dokumenten bestimmt, das im Standard *ISO/IEC 10646* festgelegt ist. Dieser Zeichensatz ist, was die verwendeten Zeichencodes betrifft, mit dem Zeichensatz Unicode synchronisiert, dem Standard, der vom Unicode-Konsortium – *www.unicode.org* – gepflegt wird. Dieser Standard enthält über ISO 10646 hinaus eine Reihe von Einschränkungen für die Implementierung, die gewährleisten sollen, dass Zeichen unabhängig von Anwendung und Plattform einheitlich verwendet werden. Unicode ist also eine Implementierung der ISO-Norm.

Unicode gibt jedem Zeichen eine eigene Nummer, die unabhängig von Plattformen, Programmen oder Sprachen ist. Text kann mit Unicode weltweit ausgetauscht werden, ohne dass es zu Informationsverlusten kommt.

Zunächst konnte mit einer 16-Bit-Kodierung eine Menge von mehr als 65.000 Zeichen abgedeckt werden. Es stellte sich aber schnell heraus, dass diese Menge nicht ausreichen würde, um alle weltweit in Vergangenheit und Gegenwart verwendeten Zeichen zu kodieren. Deshalb wurde Unicode um einen sogenannten Ersatzblock erweitert, der über eine Million Zeichen zusätzlich erlaubt. Allerdings sind diese Zeichen keine gültigen XML-Zeichen.

Inzwischen werden drei unterschiedliche Unicode-Kodierungen eingesetzt, mit 8, 16 oder 32 Bit pro Zeichen. Sie werden als *UTF-8*, *UTF-16* und *UTF-32* bezeichnet, wobei *UTF* eine Abkürzung für *Unicode* (oder *UCS*) *Transformation Format* ist.

Die `encoding`-Deklaration legt fest, welche Zeichenkodierung das Dokument verwendet, damit der XML-Prozessor diese Kodierung seinerseits ebenfalls benutzt. Wenn Ihr XML-Editor mit dem ASCII-Code arbeitet, ist diese Angabe nicht unbedingt nötig; der Prozessor wird den Code als Teil des Unicodes UTF-8 auswerten.

XML verwendet UTF-8 als Vorgabe. Diese Kodierung wird hauptsächlich für HTML und ähnliche Protokolle verwendet. Dabei werden alle Zeichen in variabel lange Kodierungen (1 bis 4 Bytes) umgesetzt. Das hat den Vorteil, dass sich bei den ersten 128 Zeichen der Unicode mit dem 7-Bit-ASCII-Code deckt. Außerdem können einfache Texteditoren so für XML-Dokumente eingesetzt werden.

Die andere Unicode-Kodierung, die XML-Prozessoren unterstützen müssen, ist UTF-16. Diese Kodierung ist unkomplizierter als UTF-8. Die am häufigsten verwendeten Zeichen werden jeweils mit 16-Bit-Einheiten kodiert, alle anderen Zeichen durch Paare von 16-Bit-Codeeinheiten.

UTF-32 hat zwar den Vorteil, dass es fast unendlich viele Zeichen darstellen kann. Dieser Vorzug wird aber damit erkauft, dass der Speicherbedarf pro Zeichen doppelt so hoch ist wie bei UTF-16.

Wenn ein anderer Zeichensatz als UTF-8 verwendet werden soll, muss er in der Deklaration angegeben werden. Der Wert für das Attribut `encoding` ist ausnahmsweise nicht fallsensitiv – `UTF-16` ist also ebenso erlaubt wie `utf-16`. (Jede externe Entität kann übrigens eine eigene Zeichenkodierung verwenden, wenn eine entsprechende Deklaration angegeben wird.)

Da aber Unicode noch nicht überall verbreitet ist, werden auch andere Kodierungen unterstützt. Für Westeuropa kann beispielsweise die verbreitete Kodierung *ISO-8859-1 (ISO Latin-1)* verwendet werden.

2.1.6 Standalone or not

Das Attribut `standalone` kann nur einen logischen Wert annehmen. Wird `standalone="no"` verwendet, ist das eine Anweisung für den XML-Prozessor, nach externen Markup-Definitionen Ausschau zu halten, um Referenzen auf externe Entitäten aufzulösen und die Gültigkeit des Dokuments prüfen zu können. Diese Einstellung muss allerdings nicht extra angegeben werden, weil sie Vorgabe ist. Der Wert `"yes"` dagegen bedeutet, dass das Dokument alle Informationen in sich selbst enthält, die für die Verarbeitung benötigt werden.

Beachtet werden muss, dass die Attribute `encoding` und `standalone` zwar optional sind; wenn sie verwendet werden, muss aber die gerade vorgeführte Reihenfolge eingehalten werden, im Unterschied zu »normalen« Elementattributen, bei denen die Reihenfolge keine Bedeutung hat.

Der Prolog kann nach der XML-Deklaration weitere Verarbeitungsanweisungen enthalten, wie zum Beispiel die Verknüpfung mit einem Stylesheet oder eine Dokumenttyp-Deklaration, etwa:

```
<?xml-stylesheet type="text/css" href="formate.css"?>
<!DOCTYPE kontaktdaten SYSTEM "kontakte.dtd">
```

2.1.7 XML-Daten – der Baum der Elemente

Erst hinter dem Prolog beginnen die eigentlichen XML-Daten in Form eines Baums aus Elementen und Attributen. Das erste Element im Dokument ist immer das Wurzelelement, das alle anderen möglichen Elemente in sich einschließt. Mit anderen Worten: Das Dokument hat die Struktur eines Baums aus ineinander verschachtelten Elementen.

Außer für das Wurzelelement gibt es folglich für jedes andere Element genau ein Elternelement, während das Wurzelelement und jedes seiner Kindelemente wieder weitere Kindelemente zum Inhalt haben können. Es sind beliebig tiefe Verschachtelungen erlaubt.

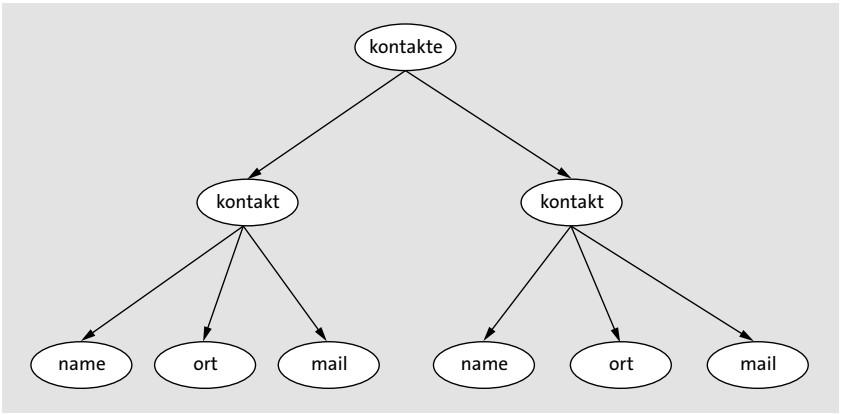


Abbildung 2.2 Baumstruktur eines Dokuments

Diese hierarchische Baumstruktur kann durch einen Graphen dargestellt werden, dessen Knoten durch gerichtete Kanten verbunden sind und dessen Wurzelknoten für keine dieser Kanten der Endknoten ist. Dieser Graph enthält folglich auch keine Zyklen.

Der Baum ist durch die sequenzielle Abfolge der Elemente im XML-Dokument implizit geordnet, die als *Dokumentreihenfolge* bezeichnet wird; darauf wird im Kapitel 5, »Navigation und Verknüpfung«, näher eingegangen. Natürlich kann auch eine ganz flache Struktur wie eine relationale Datenbanktabelle in XML ausgedrückt werden, aber die besonderen Stärken des Modells kommen dann zur Geltung, wenn es um tiefgestaffelte Datenstrukturen geht.

2.1.8 Start-Tags und End-Tags

Jedes Element wird jeweils durch ein Start-Tag und ein End-Tag begrenzt. Das XML-Dokument vermischt also die darin enthaltenen Inhalte mit Informationen über diese Inhalte, oder man kann auch sagen, es mischt Informationen und Informationen über diese Informationen. Die Meta-Information befindet sich in den Tags, die die Inhalte einschließen. Damit zwischen Inhalt und Markup unterschieden werden kann, werden spezielle Zeichen verwendet, die Beginn und Ende des Markups kennzeichnen und so das Markup vom Inhalt trennen.

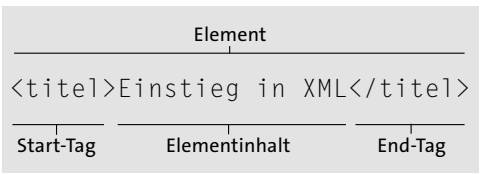


Abbildung 2.3 Ein Element in XML

Vergleicht man eine Gruppe von Datensätzen im CSV-Format oder in einer Datentabelle mit einer Feldnamenzeile mit denselben Datensätzen im XML-Format, wird sofort deutlich, dass das XML-Format zwangsläufig aufwendiger ist. Bei jedem Datensatz werden alle »Feldnamen« erneut angegeben. Dieses Format wäre in den Zeiten, in denen Speicherressourcen noch sehr knapp und die Bandbreiten in den Netzen sehr eng waren, wahrscheinlich wenig attraktiv gewesen.

Der Vorteil von XML ist aber, dass auch jeder einzelne Zweig im Baum der Elemente darüber Auskunft gibt, was die einzelnen Elemente bedeuten. Das erleichtert auch die Zerlegung eines großen XML-Dokuments in kleinere Teile oder umgekehrt das Zusammenfügen von Teildokumenten zu einem Gesamtdokument.

2.1.9 Elementtypen und ihre Namen

Das Start-Tag enthält den Namen des Elementtyps, eingeschlossen in spitze Klammern, beim End-Tag kommt vor den Namen des Elementtyps noch ein Schrägstrich. Als strenge Einschränkung für das Kriterium der Wohlgeformtheit ist festgelegt, dass der Elementtypname im Start-Tag und im End-Tag exakt übereinstimmen muss. Die Wiederholung des Elementtypnamens im End-Tag ist notwendig, damit der Parser die Schachtelung der Elemente sofort korrekt erkennen kann. Nur bei leeren Elementen ist eine vereinfachte Schreibweise erlaubt, bei der auf die Wiederholung des Elementnamens verzichtet wird.

Anders als bei HTML-Tags ist dabei unbedingt auf die Groß- und Kleinschreibung zu achten, weshalb es sinnvoll ist, sich von vornherein für eine einheitliche Schreibweise zu entscheiden, also alle Tags klein oder groß oder mit Groß- und Kleinschreibung zu schreiben.

Ein Verstoß gegen die Regel der Wohlgeformtheit führt zu einem fatalen Fehler, das heißt, ein XML-Prozessor wird die Verarbeitung des nicht wohlgeformten Dokuments ab der Stelle, wo der Fehler auftritt, verweigern. Diese harsche Reaktion unterscheidet XML wiederum stark von HTML, das für seine eher tolerante Reaktion auf viele Fehler bekannt ist, die dafür sorgt, dass der Webbesucher auch bei Seiten mit kleinen Fehlern nicht leer ausgeht. Diese »Nachgiebigkeit« der Browser bei der Verarbeitung von »unsauberem« HTML-Code sollte für XML mit Vorsatz nicht wiederholt werden, weil diese Situation letztlich dazu geführt hat, dass viel Wildwuchs auf Webseiten zugelassen worden ist. Der aber muss von den Browsern mit immer mehr Ausnahmeregelungen aufgefangen werden, was den Code enorm aufbläht.

Zwischen dem Start- und dem End-Tag befindet sich der Inhalt des Elements. Dass die Tags nicht einfach die Namen der Elemente enthalten, sondern die Namen der Elementtypen, weist schon darauf hin, dass Elemente desselben Typs in einem Dokument mehrfach verwendet werden können. Jedes einzelne Element ist also ein Exemplar oder eine Instanz eines bestimmten Elementtyps.

```
<team>
  <person>Hanna Karl</person>
  <person>Kurt Vondel</person>
</team>
```

Im Unterschied zu HTML und auch zu SGML darf das End-Tag nicht fehlen, sonst kann das Dokument eine Prüfung auf Wohlgeformtheit nicht bestehen. Nur bei einem leeren Element ist es erlaubt, eine verkürzte Schreibweise zu verwenden, also `<leer />` statt `<leer></leer>`. Leere Elemente werden zum Beispiel für das Einbinden von Bildern in ein XML-Dokument verwendet.

2.1.10 Regeln für die Namensgebung

Die in den Tags verwendeten Namen für die Typen der Elemente sind frei wählbar, solange nur die Wohlgeformtheit des gesamten XML-Dokuments interessiert. Ein Dokument kann also auch beliebig viele Elementtypen enthalten. Allerdings müssen bei der Wahl des Namens einige Einschränkungen beachtet werden:

- Ein Name muss mit einem Buchstaben oder mit Unterstrich oder Doppelpunkt beginnen.
- Danach dürfen alle Zeichen verwendet werden, die als Namenszeichen zugelassen sind: Neben den Zeichen für die erste Stelle sind das die Zahlen, der Bindestrich und der Punkt. Auch Umlaute, Akzente etc. sind erlaubt. Allerdings sollte der Doppelpunkt möglichst vermieden werden, weil er als Trennzeichen verwendet wird, wenn mit Namensräumen gearbeitet wird, wovon in Abschnitt 2.9, »Namensräume«, noch die Rede sein wird.
- Die Zeichenfolge »xml« darf in keiner der möglichen Schreibweisen am Beginn eines Namens stehen; diese Zeichenfolge ist für XML reserviert.
- XML-Namen sind »case-sensitive«, es wird also zwischen Groß- und Kleinschreibung unterschieden, so dass `<Name>...</name>` beispielsweise nicht zulässig ist.

Die Länge der Namen ist nicht begrenzt, es sollte aber beachtet werden, dass möglicherweise Anwendungen, die auf die Daten zugreifen, die Länge einschränken. Wenn Sie Namen aus mehreren Wörtern zusammensetzen, haben Sie die Möglichkeit, die einzelnen Wörter mit Bindestrichen oder Unterstrichen zu verbinden. Eine häufig genutzte Variante ist auch die Verwendung von Großbuchstaben am Wortbeginn, entweder bei jedem Wort wie in `<VorName>` – das wird auch *Pascal-casing* genannt, oder erst beim zweiten Wort wie in `<vorName>` – also im *Camel-casing*-Verfahren.

Es ist immer wieder die Rede davon, dass XML Tags erlaubt, die den Inhalt der eingeschlossenen Daten beschreiben, also semantische Tags wie `<Postleitzahl>` oder `<Titel>`. Der Standard legt aber nur fest, dass ganz beliebige Elementtypen bestimmt werden können; `<tag1></tag1>`, `<tag2></tag2>` würde die Wohlgeformtheitsprüfung

ebenfalls überstehen. Es kommt also hier darauf an, was die Entwickler mit der durch die Spezifikation angebotenen Freiheit anfangen.

Das Ziel, das der Entwicklung von XML die Richtung gibt, ist jedenfalls eine Identifizierung der Komponenten, aus denen eine Datensammlung oder ein Dokument besteht, mit Hilfe von bedeutungsvollen – also semantischen – Namen, die ein Computerprogramm zwar nicht wie ein Mensch versteht, die dem Programm aber erlauben, sich so zu verhalten, also ob es verstehen würde, was die verwendeten Namen bedeuten. Die XML-Tags haben insofern durchaus Ähnlichkeiten mit den Feldnamen, die bei der Strukturierung von Datenbanken verwendet werden.

Die Tags gehören zu den Auszeichnungen, den Markups, die die Struktur des Dokuments festlegen und im Idealfall zugleich den Inhalt des Dokuments beschreiben. Da XML-Dokumente ein schlichtes Textformat verwenden, bleiben sie für den menschlichen Betrachter im Prinzip lesbar.

2.1.11 Elementinhalt

Abgesehen von den schon angesprochenen leeren Elementen haben Elemente in der Regel einen Inhalt, der aus ganz unterschiedlichen Dingen bestehen kann. In diesem Sinne werden die Elemente auch als *Container* betrachtet. Zunächst kann ein Element wiederum untergeordnete Elemente enthalten. Man spricht dann von *element content*. Das folgende Element `<kontakt>` hat zum Beispiel drei Kindelemente zum Inhalt:

```
<kontakt>
  <name>Hans Maier</name>
  <ort>Hamburg</ort>
  <mail>hmaier@nonet.de</mail>
</kontakt>
```

In diesem Fall beginnt mit dem Element `<kontakt>` also ein Teilbaum innerhalb der gesamten Baumstruktur. Die Art der Anordnung der Kindelemente kann über ein Inhaltsmodell geregelt werden, entweder per DTD oder per XML-Schema. Dieses Modell legt beispielsweise fest, dass die Elemente unbedingt in einer bestimmten Reihenfolge auftreten müssen.

Den Blättern des Baums entsprechen dagegen die Elemente, die dann nur noch Zeichendaten enthalten, also Zeicheninhalt oder *character content*.

XML lässt auch zu, Elemente und Zeichendaten zu mischen, gemischte Inhalte oder *mixed content* also. Im Unterschied zu dem ersten Fall, bei dem Elemente selbst wiederum nur Kindelemente enthalten, bleibt bei einer solchen Mischung von Zeichendaten und Elementen die Reihenfolge weitgehend ungeregelt – ein Grund, solche Mischcontainer meistens zu vermeiden.

2.1.12 Korrekte Schachtelung

So schlicht die Kernfestlegungen auf den ersten Blick erscheinen, können doch schon bei den ersten Schritten zur Strukturierung eines Gegenstandsbereichs Probleme auftreten. Eine erste Falle ist eine falsche Schachtelung von Elementen. HTML reagiert auf eine falsche Schachtelung von Tags relativ harmlos, wie Abbildung 2.4 zeigt.

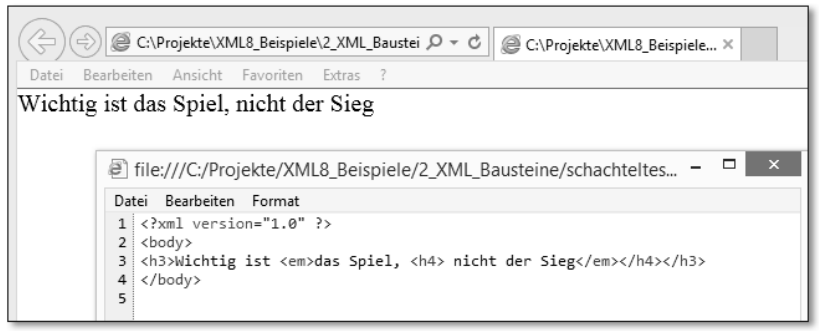


Abbildung 2.4 Der Internet Explorer verdaut die fehlerhaft geschachtelten HTML-Elemente.

Wenn Sie dieselben Tags als XML-Dokument speichern, ist die Reaktion eines XML-Prozessors wiederum sehr kategorisch. Er wird einen fatalen Fehler melden und das Prädikat der Wohlgeformtheit verweigern.

Schachtelungsfehler können sich auch dadurch einschleichen, dass einfach das End-Tag für ein übergeordnetes Element vergessen wird.

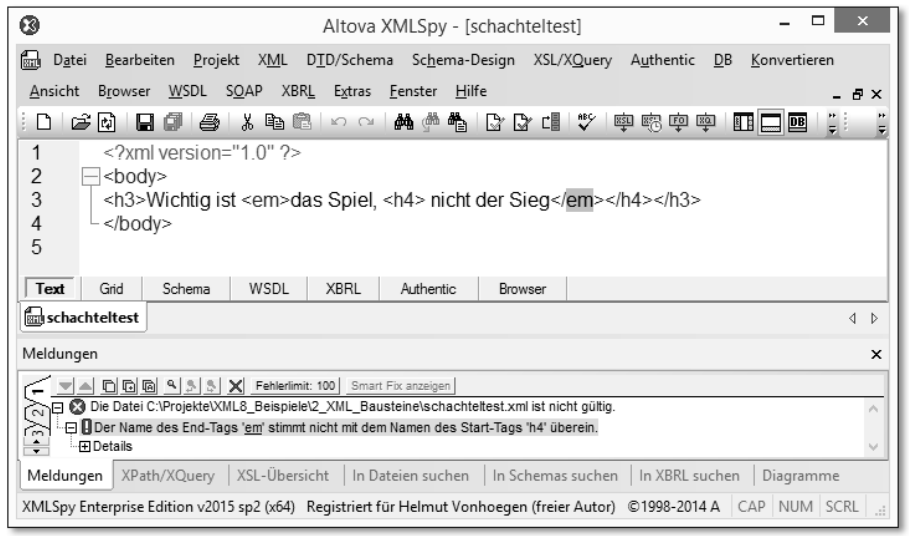


Abbildung 2.5 Reaktion des XMLSpy-Editors auf eine fehlerhafte Schachtelung

2.1.13 Attribute

Alle Elemente, die innerhalb der Struktur auftauchen, können mit beliebig vielen Attributen versehen werden, die jeweils als Paare von Attributnamen und Attributwerten auftreten. Attribute werden verwendet, um bestimmte Eigenschaften oder Besonderheiten eines Elements festzuhalten. Manchmal werden deshalb auch die Attribute mit den Adjektiven verglichen, die Elemente mit den Subjekten. Allerdings ist dieser Vergleich nicht streng anwendbar, denn Attribute können auch ähnlich wie Unterelemente verwendet werden.

Die Attribute werden jeweils im Start-Tag eines Elements platziert. Sie müssen eindeutig sein, es darf also nicht mehrfach derselbe Attributname in einem einzigen Start-Tag verwendet werden. Ein Ausdruck wie

```
<haus nr="22" nr="33">
```

ist also nicht zulässig – er ergibt ja auch wenig Sinn.

Dagegen darf durchaus derselbe Attributname bei unterschiedlichen Elementen verwendet werden. Obwohl die Attribute denselben Namen verwenden, bleiben sie für einen XML-Prozessor unterscheidbar, weil sie zu unterschiedlichen Elementen gehören.

```
<einrichtung>
  <tisch farbe="blau">Esstisch rund</tisch>
  <stuhl farbe="beige">Vierbeinstuhl</stuhl>
</einrichtung>
```

Werden mehrere Paare von Attributnamen und Attributwerten verwendet, werden sie durch ein Leerzeichen getrennt. Für die Attributnamen gelten dieselben Vorschriften wie für die Elementnamen. Während Elemente andere Elemente enthalten können, ist dies bei Attributen nicht erlaubt. Die Attributwerte wiederum müssen jeweils mit den oberen doppelten Anführungszeichen oder mit dem Apostroph eingeschlossen werden. Die Werte selbst dürfen nur Literale sein. Innerhalb des Literals dürfen die Markup-Zeichen `<`, `>` und `&` nicht verwendet werden. Sie sind durch `<`, `>` und `&`, also durch sogenannte *Entitätsreferenzen*, zu maskieren. Mehr dazu in Abschnitt 2.5.

Wenn Sie eine Art von Begrenzungszeichen verwenden, um einen Attributwert einzuschließen, können Sie die anderen Begrenzungszeichen innerhalb des Literals verwenden:

```
<antwort text="Er sagte: ´So geht es nicht´">
```

Auch leere Elemente können mit Attributen versehen werden. Ein Ausdruck wie

```
<leer name="leeres Element"/>
```

ist also erlaubt. Solche leeren Elemente werden zum Beispiel verwendet, um im Dokument Stellen für Daten in einem Nicht-XML-Format zu reservieren, etwa für Bilder, Videos oder Sounds. Dabei werden die Informationen zu diesen Datenquellen über Attribute des leeren Elements festgehalten.

2.2 Die Regeln der Wohlgeformtheit

Bei der Prüfung von XML-Dokumenten wird grundsätzlich zwischen der Prüfung der Wohlgeformtheit und der Gültigkeit unterschieden. Auch wenn ein XML-Dokument mit einer DTD oder einem XML-Schema verknüpft ist, kann es ein XML-Prozessor dabei bewenden lassen, nur die Wohlgeformtheit zu überprüfen.

Wenn auf eine Gültigkeitsprüfung verzichtet wird, besteht allerdings keinerlei Gewähr, dass Daten zum Beispiel in einem bestimmten Datenformat vorliegen. Werden solche Daten von Anwendungsprogrammen weiterverarbeitet, müssen diese Programme folglich selbst dafür sorgen, dass die ungeprüft übernommenen Daten korrekt verarbeitet werden. Das muss aber, beispielsweise bei Lösungen innerhalb eines Intranets, kein Problem sein.

Dass mit Wohlgeformtheit nicht unbedingt viel erreicht ist, wird schlagartig durch die folgenden Zeilen deutlich:

```
<tierprodukte>
  <milchprodukt>Käse</milchprodukt>
  <getreidesorte>Weizen</getreidesorte>
</tierprodukte>
```

Hier ist zwar kein syntaktischer Fehler zu finden, aber auf der Ebene der Bedeutungen ist gleich offensichtlich, dass etwas nicht stimmen kann.

2.3 Elemente oder Attribute?

In der Regel sollten Attribute verwendet werden, um Zusatzinformationen zu der Information darzustellen, die das Element selbst enthält. Es liegt aber nicht immer auf der Hand, welche Aspekte eines Dokuments oder einer Datenstruktur besser als Element oder besser als Attribut behandelt werden sollten. Ein Ausdruck wie

```
<an name="Clara Donna" email="cd@clarad.de"/>
```


ist ebenso akzeptabel wie

```
<an>
  <name>Clara Donna</name>
  <email>cd@clarad.de </email>
</an>
```

Prinzipiell sind Attribute immer einem Element zugeordnet; anstelle eines Attributs kann aber häufig auch ein Unterelement verwendet werden. Im Unterschied zu Elementen können Attribute keine Unterelemente oder Unterattribute enthalten. Ein Nachteil von Attributen ist auch, dass der Zugriff auf die Attributwerte über Anwendungen, etwa mit Hilfe der Schnittstellen des in Kapitel 10, »Programmierschnittstellen für XML«, vorgestellten *Document Object Models* (*DOM*), umständlicher ist als der auf den Inhalt von Elementen. Auch wenn Sie die Daten mit Cascading Stylesheets anzeigen wollen, sind Elemente vorzuziehen.

Ein gewisser Vorteil von Attributen dagegen ist, dass ihre Reihenfolge nicht wie bei Elementen fixiert werden muss.

Wenn es um tatsächlich getrennte oder begrifflich sinnvoll auftrennbare Einheiten in einem Gegenstandsbereich geht, sollte in der Regel mit Elementen gearbeitet werden, zumal dadurch auch der Aufbau von Links auf diese Einheiten ermöglicht wird, etwa durch ID- und IDREF-Attribute in einem XSLT-Stylesheet.

2.4 Reservierte Attribute

Die XML-Spezifikation gibt zwei Attribute vor, die in jedem Element verwendet werden können. Das eine – `xml:lang` – dient zur Identifikation der Sprache, die für die Inhalte eines Dokuments oder einzelner Elemente gilt, das andere Attribut – `xml:space` – erlaubt Festlegungen zur Behandlung von Leerraum im Inhalt eines Elements. Obwohl die Attribute vorgegeben sind, müssen sie innerhalb einer DTD explizit deklariert werden, wenn sie zum Einsatz kommen sollen.

2.4.1 Sprachidentifikation

In den folgenden Elementen

```
<zeile xml:lang="en">to be or not to be</zeile>
<zeile xml:lang="de">Sein oder Nichtsein</zeile>
```

wird durch das `xml:lang`-Attribut erkennbar, welche Sprache für den Inhalt des Elements verwendet wird. Auf diese Weise kann zum Beispiel ein Stylesheet entwickelt werden, das dem Wunsch eines Anwenders entsprechend die Zeile einmal in der

einen, einmal in der anderen Sprache ausgibt. Die Einstellung gilt jeweils auch für die Unterelemente.

Als Werte des Attributs werden zwei- oder dreistellige Ländercodes verwendet, die durch *ISO 639* genormt sind. Auch Subcodes für regionale Sprachen sind möglich, etwa "en-US" oder "en-GB".

2.4.2 Leerraumbehandlung

Mit dem Attribut xml:space können Sie versuchen, Einfluss darauf zu nehmen, wie ein XML-Prozessor mit Leerräumen im Inhalt von Elementen umgeht. Leerräume sind Leerzeichen, Tabulatoren und Leerzeilen. Auch diese Einstellung gilt jeweils für das Element und die Unterelemente. Es gibt zwei mögliche Werte:

- preserve meldet der Anwendung den Wunsch, dass alle Leerräume, so wie sie vorhanden sind, erhalten bleiben.
- default stellt der Anwendung, die die Daten verarbeitet, anheim, deren Vorgabe für den Umgang mit Leerraum zu verwenden.

Der Inhalt des folgenden Elements sollte also in einer Anwendung genauso erscheinen wie eingegeben:

```
<zeile xml:space="preserve">
T o o o r !
</zeile>
```

Es hängt allerdings von der Anwendung ab, ob sie dem Hinweis folgt oder die Leerzeichen einfach entfernt.

2.5 Entitäten und Verweise darauf

Es ist schon angesprochen worden, dass die speziellen Zeichen, die XML für Markups verwendet, nicht ohne weiteres im Inhalt eines Elements oder innerhalb eines Attributwerts auftauchen dürfen. Es gibt nun mehrere Möglichkeiten, Zeichen davor zu schützen, von einem XML-Prozessor als Markup-Zeichen ausgewertet zu werden, wenn sie als Inhalt eines Elements oder innerhalb eines Attributwerts benötigt werden.

2.5.1 Eingebaute und eigene Entitäten

Der eine Weg ist die Verwendung von Entitäten und Referenzen auf diese Entitäten. XML bringt fünf solcher Entitäten schon mit (siehe Tabelle 2.1).

Entität	Entitätsreferenz	Bedeutung
lt	<	< (kleiner als)
gt	>	> (größer als)
amp	&	& (Ampersand)
apos	'	' (Apostroph oder einfache Anführungszeichen)
quot	"	" (doppelte Anführungszeichen)

Tabelle 2.1 Entitäten von XML

Ein Firmenname wie »B&B« kann auf diese Weise als B&B eingegeben werden.

Zusätzlich zu den eingebauten Entitäten können Sie eigene Entitäten definieren, ähnlich wie es auch in HTML möglich ist. Dies geschieht innerhalb einer Dokumenttyp-Definition. Wie dies gemacht wird, ist in Abschnitt 3.9, »Verwendung von Entitäten«, beschrieben.

Um solche Entitäten zu verwenden, wird dieselbe Schreibweise verwendet wie bei den eingebauten Entitäten. Wird beispielsweise in der DTD ein Kürzel GB für Geschäftsbedingungen abgelegt, kann der Ersetzungstext mit &GB; referenziert werden.

2.5.2 Zeichenentitäten

Eine weitere Methode, Zeichen indirekt einzubringen – etwa wenn das Eingabegerät bestimmte Zeichen nicht zur Verfügung stellt –, sind Zeichenreferenzen, die mit Hilfe von dezimalen oder hexadezimalen Zahlen arbeiten, die auf den Unicode-Zeichensatz verweisen. Die Schreibweise ist ähnlich wie bei den Entitätsreferenzen, nur wird dem &-Zeichen noch ein #-Zeichen nachgestellt:

```
&#169;
&#xA9;
```

liefern beide das Copyright-Zeichen.

```
&#x20AC;
```

oder

```
&#8364;
```

liefern das Euro-Symbol.

Werden Entitätsreferenzen verwendet, muss das Dokument nach der Auflösung der Referenzen – also nachdem das Kürzel gegen den Ersetzungstext getauscht worden ist – weiterhin ein wohlgeformtes Dokument sein.

Solche Entitätsreferenzen sind deshalb nur erlaubt, wenn sie Bezüge auf *parsed data* enthalten; direkte Bezüge auf *unparsed data* sind nicht erlaubt. Solche Bezüge müssen auf einem Umweg über Attributwerte vom Typ `ENTITY` oder `ENTITIES` hergestellt werden. Mehr dazu finden Sie in Abschnitt 3.9.

2.6 CDATA-Sections

Wenn es sich ergibt, dass größere inhaltliche Teile eines XML-Dokuments sehr häufig reservierte Markup-Zeichen benötigen, etwa ein Dokument, das selbst wiederum ein anderes XML- oder HTML-Dokument beschreibt oder Skript-Code enthält, kann es mühsam werden, jedes Mal mit Zeichen- oder Entitätsreferenzen zu arbeiten. In diesem Fall ist es praktischer, CDATA-Blöcke zu verwenden. Hier ein kleines Beispiel:

```
<![CDATA[
  Tags in XML werden immer mit < und > begrenzt.
]]>
```

Ein solcher Block von *Character Data* beginnt mit dem String `<![CDATA[` und endet mit dem sogenannten CDEnd-String `]]>`. Alle Zeichen, die sich dazwischen befinden, werden von einem XML-Prozessor nicht als Markup interpretiert. Sie können also ungehindert die spitzen Klammern oder das Ampersand (`&`) verwenden. Nur die Zeichenfolge `]]>` selbst darf nicht innerhalb des Textes der CDATA-Section vorkommen.

2.7 Kommentare

Über die Nützlichkeit von Kommentaren muss man Entwicklern keine Vorträge halten, obwohl manchmal eher zu wenige als zu viele verwendet werden. Das gilt auch für XML, obwohl die »sprechenden« Tags das Dokument schon in einem bestimmten Umfang selbst dokumentieren.

Zur eigenen Erinnerung oder zur Information für andere lassen sich Kommentare überall in das XML-Dokument einbetten, sie sind nur nicht innerhalb von Tags oder Deklarationen erlaubt, im Unterschied übrigens zu SGML:

```
<!--
das Dokument muss noch mit einem Schema verknüpft werden
-->
```


Wie in HTML beginnt ein Kommentar mit `<!` und endet mit `>`. Die Zeichenfolge darf nicht innerhalb des Kommentars verwendet werden. Deshalb sind auch Kommentare innerhalb von Kommentaren verboten. Wenn Sie bei einem Test ein Element oder einen Teilbaum von Elementen vorübergehend herausnehmen wollen, können Sie die gesamte Gruppe auskommentieren:

```
<!--
<element>
  <unterelement>
  </unterelement>
</element>
-->
```

Streng beachtet werden muss, dass keine Kommentare vor der XML-Deklaration erscheinen dürfen, wenn diese verwendet wird. Dokumente ohne XML-Deklaration dürfen dagegen mit einem Kommentar beginnen!

2.8 Verarbeitungsanweisungen

Es ist möglich, in das XML-Dokument Verarbeitungsanweisungen – *processing instructions* – für den XML-Prozessor einzubetten. Ihr Inhalt gehört nicht zu den Zeichendaten, aus denen das Dokument besteht.

Die Anweisungen beginnen immer mit `<?` und enden mit `?>`. Sie geben zunächst ein Ziel an, das die Anwendung identifizieren soll, an die sich die Anweisung richtet, dann folgen die Daten der Anweisung selbst.

Welche Anweisungen möglich sind, hängt davon ab, was der verwendete Prozessor versteht und was nicht. Sie sind also nicht durch die XML-Spezifikation vorgegeben. Zweck einer solchen Anweisung kann zum Beispiel die Verknüpfung des Dokuments mit einem Stylesheet sein:

```
<?xml-stylesheet type="text/css" href="praesentation.css"
media="screen"?>
```

Da diese Instruktion in der ursprünglichen Spezifikation von XML noch nicht vorgesehen ist – das Wort Stylesheet taucht darin nicht auf –, wurde dafür im Sommer 1999 extra eine kleine Empfehlung *Associating Style Sheets with XML documents* über die Zuordnung von Stylesheets zu XML-Dokumenten herausgegeben, in der die `xml-stylesheet`-Anweisung definiert ist.

XML-Prozessoren, wie sie zum Beispiel im Internet Explorer ab Version 5 integriert sind, verstehen eine solche Anweisung und geben ein XML-Dokument in dem zugewiesenen Format aus. Mehr dazu in Kapitel 10, »Programmierschnittstellen für XML«.

2.9 Namensräume

Eine der ersten wichtigen Erweiterungen des XML-Standards war die Empfehlung *Namespaces in XML*, die bereits ein Jahr nach der Verabschiedung von XML 1.0 im Februar 1998 veröffentlicht wurde. Die Freiheit, die XML bei der Wahl von Element- und Attributnamen gewährt, wirft überall dort ein Problem auf, wo gleiche Namen mit unterschiedlichen Bedeutungen verwendet werden. Da XML-Dokumente häufig darauf ausgelegt sind, sie mit unterschiedlichen Programmen auszuwerten und zu bearbeiten, führen Mehrdeutigkeiten schnell zu unerwünschten Ergebnissen.

2.9.1 Das Problem der Mehrdeutigkeit

Ein einfaches Beispiel ist etwa ein Elementname wie `<beitrag>`, der in dem einen Kontext den Mitgliedsbeitrag in einem Verein benennt, in einem anderen Kontext einen Artikel in einer Zeitschrift und in einem dritten Kontext eine Arbeitsleistung, etwa innerhalb eines Projekts. Während bei dem ersten Element ein Programm beispielsweise prüfen soll, ob es Beitragsrückstände gibt, könnte im letzten Fall eine Berechnung der Arbeitszeit angestoßen werden.

Zwar wäre es jedes Mal möglich, den Namen entsprechend zu spezifizieren und die entsprechenden Elemente als `<mitgliedsbeitrag>`, `<textbeitrag>` und `<projektbeitrag>` zu differenzieren, aber erstens führt dies in vielen Fällen zu eher künstlichen Bezeichnungen, und zweitens kann man niemals sicher sein, dass beim Design eines XML-Vokabulars diese Regel immer beachtet wird.

2.9.2 Eindeutigkeit durch URIs

Da XML-Vokabulare aber auch unter dem Gesichtspunkt entwickelt werden, möglichst wiederverwendbar zu sein, lag es nahe, dafür eine andere Lösung zu suchen. Deshalb hat das W3C ein relativ einfaches Verfahren eingeführt, um die Eindeutigkeit von Namen für Elemente und Attribute zu gewährleisten. Dabei werden die einzelnen Namen als Teile eines bestimmten Namensraums behandelt. Namensräume werden als Ansammlungen von Namen vorgestellt, die zu einem bestimmten Gegenstandsbereich gehören. Eine solche Ansammlung benötigt keine bestimmte Struktur, wie es etwa bei einer DTD der Fall ist. Es reicht eine einfache Zugehörigkeit: Der Name `a` gehört zum Namensraum `x`. Das schließt aber nicht aus, sich auch auf eine DTD als Namensraum zu beziehen.

Mit der Angabe eines Namensraums wird der Kontext angegeben, in dem ein bestimmter Name seine ganz spezielle Bedeutung erhält. XML-Namensräume werden über eine URI-Referenz, also in der Regel einen URL, identifiziert. (Dabei werden solche Referenzen nur als identisch betrachtet, wenn sie Zeichen für Zeichen gleich sind, also unter Beachtung der Groß-/Kleinschreibung.) Diese URI-Referenz wird aber

nur verwendet, um dafür zu sorgen, dass der Namensraum auch eindeutig ist. Man nutzt also die Tatsache, dass URI-Referenzen immer eindeutig sein müssen, um dem Prozessor einen eindeutigen Namen für den Namensraum zu liefern.

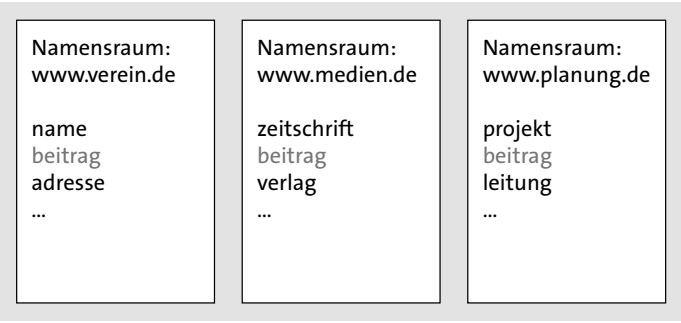


Abbildung 2.6 Gleichlautender Name in verschiedenen Namensräumen

Der Prozessor sieht keineswegs bei dem entsprechenden URL nach, ob dort ein Namensraum abgelegt ist. (Das W3C hinterlegt unter den URLs der von ihm selbst gepflegten Namensräume lediglich Hinweise, aber keine Listen der Namen.) Es handelt sich also im Grunde um eine Formalität, deren Zweck es ist, den Namensraum dauerhaft und eindeutig zu identifizieren. Solange der Schema-Autor einen URL verwendet, dessen Einmaligkeit er kontrollieren kann, weil er ja die Eindeutigkeit seines URLs kennt, gibt es keine Probleme für den XML-Prozessor.

2.9.3 Namensraumname und Präfix

Dass ein Name zu einem bestimmten Namensraum gehört, kann zum Zweck der Vereinfachung durch ein Präfix angegeben werden, das dem Namen vorgesetzt wird, wobei zur Trennung ein Doppelpunkt verwendet wird. Dieses Präfix kann bei der Deklaration eines Namensraums zugewiesen werden.

Das Präfix dient einfach nur als Abkürzung für den Namen des Namensraums; der XML-Prozessor wird diese Abkürzung immer durch den eigentlichen Namensraumnamen, also den URI, ersetzen.

2.9.4 Namensraumdeklaration und QNamen

Um Namensräume in einem XML-Dokument verwenden zu können, müssen sie zunächst deklariert werden. Dies geschieht in Form eines Elementattributs. Der Namensraum ist durch die Deklaration für das betreffende Element und alle seine Kindelemente gültig. Soll der Namensraum also im gesamten Dokument gültig sein, muss das Attribut dem Wurzelelement zugewiesen werden. Es ist aber auch möglich, erst auf einer tieferen Ebene ein Element mit einem Namensraum zu versehen.

Für die Deklaration werden reservierte Attribute verwendet. In dem folgenden Beispiel ordnet die Namensraumdeklaration dem Namensraumnamen `http://mitglieder.com/organisation` das Präfix `mtg` zu:

```
<mitglieder xmlns:mtg="http://mitglieder.com/organisation">
...
</mitglieder>
```

Dieses Präfix kann anschließend verwendet werden, um für ein Element oder Attribut anstelle eines einfachen Namens einen qualifizierten Namen – *QName* – zu verwenden:

```
<mtg:beitrag>100</mtg:beitrag>
```

In diesem Fall nennt das Element nicht nur seinen Namen, sondern gibt zugleich an, zu welchem Namensraum es gehört. Der qualifizierte Name beugt der Gefahr der Mehrdeutigkeit eines Namens vor, indem er diesen durch die Zuordnung zu einem Namensraum erweitert. Er besteht in diesem Fall aus dem Präfix, das die Bindung an den Namensraum herstellt, dem Doppelpunkt als Trennzeichen und dem lokalen Namen des Elements.

Auch bei Attributnamen kann so verfahren werden. Wenn der Attributname aber zum gleichen Namensraum gehört wie der Name des Elements, zu dem das Attribut gehört, darf der Attributname nicht mit dem entsprechenden Präfix versehen werden.

Das Präfix kann frei gewählt werden, nur die Zeichenfolge `xml` ist für den Namensraum `http://www.w3.org/XML/1998/namespace` und `xmlns` für die Einbindung von Namensräumen reserviert. Es ist an den Namensraum `http://www.w3.org/2000/xmlns` gebunden. Beide Namensräume müssen nicht deklariert werden.

Innerhalb eines XML-Dokuments werden die mit Präfix versehenen Namen ansonsten wie normale Namen behandelt oder wie Namen, die einen Doppelpunkt als eines der Zeichen enthalten.

2.9.5 Einsatz mehrerer Namensräume

Wenn erforderlich, kann auch mit mehreren Namensräumen gearbeitet werden:

```
<mtg:mitglieder xmlns:mtg="http://XMLbeisp.com/organisation"
                xmlns:pro="http://XMLbeisp.com/abrechnung"
                xmlns:mass="http://XMLbeisp.com/masse">
  <mtg:mitglied>
    <mtg:name>Hansen</name>
    <mtg:beitrag mass:waehrung="EUR">100</beitrag>
    <mtg:projekt>
      <pro:beschreibung>Haussanierung</pro:beschreibung>
```



```
      <pro:beitrag mass:einheit="Std" pro:status="ehrenamtlich">20
      </pro:beitrag>
    </mtg:projekt>
  </mtg:mitglied>
</mtg:mitglieder>
```

Mit Hilfe der Präfixe lassen sich die unterschiedlichen Elemente und Attribute exakt dem jeweils gültigen Namensraum zuordnen. Allerdings ist es beim Einsatz mehrerer Namensräume oft praktisch, einen dieser Namensräume als Default-Namensraum, also als Vorgabe, zu verwenden. Dies geschieht dadurch, dass bei der Deklaration kein Präfix zugeordnet wird. In der folgenden Variante des letzten Beispiels wird der erste angegebene Namensraum als Vorgabe verwendet:

```
<mitglieder xmlns="http://XMLbeisp.com/organisation"
            xmlns:pro="http://XMLbeisp.com/abrechnung">
  <mitglied>
    <name>Hansen</name>
    <beitrag waehrung="EUR">100</beitrag>
    <projekt>
      <pro:beschreibung>Haussanierung</pro:beschreibung>
      <pro:beitrag einheit="Std">20</pro:beitrag>
    </projekt>
  </mitglied>
</mitglieder>
```

Bei den Elementen, die zum vorgegebenen Namensraum gehören, kann dann auf ein Präfix verzichtet werden, was die Dokumente lesbarer macht und die Schreibarbeit verringert. Trotzdem handelt es sich bei diesen Elementnamen um qualifizierte Namen, auch wenn das sonst verwendete Präfix gleichsam unsichtbar geworden ist. Darauf wird in Kapitel 4, »Inhaltsmodelle mit XML-Schema«, noch einmal eingegangen werden, wenn es um die Validierung von Dokumenten geht, die Namensräume verwenden. Werden Attribute ohne Präfix benannt, gehören sie dagegen nicht zum vorgegebenen Namensraum. Sie erben also nicht den vorgegebenen Namensraum des Elements, zu dem sie gehören.

Die Voreinstellung auf einen bestimmten Namensraum, die beispielsweise innerhalb des Wurzelelements vorgenommen worden ist, kann auf einer tieferen Ebene auch wieder überschrieben werden, indem erneut das Attribut `xmlns` ohne Präfixzuordnung verwendet wird. Die neue Vorgabe gilt dann aber nur für diese Ebene und eventuelle Kindelemente dieser Ebene. Soll eine Vorgabe ganz aufgehoben werden, kann auch mit dem Wertepaar `xmlns=""` gearbeitet werden. Dieses Verfahren kann allerdings nicht für die Aufhebung von Namensraumzuordnungen werden, die mit einem Präfix arbeiten.

2.10 XML Version 1.1

Das W3C hat 2004 eine Version XML 1.1 verabschiedet, um der Weiterentwicklung von Unicode besser zu entsprechen. In der XML-Version 1.0 sind zwar für Elementinhalte und Attributwerte fast alle Unicode-Zeichen zugelassen, für Element- und Attributnamen, für Aufzählungen vorgegebener Attributwerte und für die Formulierung des Ziels von Verarbeitungsanweisungen sind aber nur die Zeichen zugelassen, die bereits in Unicode 2.0 enthalten sind. Um kommende Erweiterungen von Unicode zuzulassen, führt die Version XML 1.1 hier eine Lockerung ein und lässt auch in den Namen alle Zeichen zu, die nicht ausdrücklich verboten sind.

Die zweite Änderung betrifft die Behandlung von Leerraum. Um bisherige Umständlichkeiten beim Datenaustausch mit Großrechnern von IBM und damit kompatiblen Systemen zu vermeiden, wird den bisher für Leerraum verwendeten Zeichen – Space, Tab, CR, LF – noch ein NEL-Zeichen (Newline, U+0085 bzw. #x85) hinzugefügt. Zudem wird das Unicode-Zeilentrennzeichen U+2028 bzw. #x2028 unterstützt. Mit Ausnahme der erwähnten Leerraumzeichen dürfen Steuerzeichen im Codebereich #x1 bis #x1F und #x7F bis #x9F allerdings nur in Form von Zeichenreferenzen verwendet werden, damit XML-Dateien auch weiterhin von normalen Texteditoren angezeigt und bearbeitet werden können. Verboten in jeder Form bleibt dagegen das NUL-Zeichen (#x00).

Um den binären Vergleich von zwei Zeichenfolgen zu vereinfachen, soll außerdem eine Unicode-Normalisierung dafür sorgen, dass für jedes Zeichen eine einheitliche Schreibweise verwendet wird. Das ist bisher nicht unbedingt der Fall. Umlaute können beispielsweise durch ein einzelnes Zeichen oder als Kombination von zwei Zeichen dargestellt werden. Da diese Normalisierung aber in einem XML-Parser nicht einfach zu implementieren ist, kann die Umsetzung dieser Anforderung noch geraume Zeit dauern.

Obwohl die Änderungen von Version 1.0 zu 1.1 insgesamt gering bleiben, sind die entsprechenden XML-Dokumente nicht kompatibel. Eine XML 1.1-Datei ist für einen Version 1.0-Parser unter Umständen also nicht wohlgeformt.

Insgesamt folgt aus all dem, dass die Version 1.1 wohl über die nächsten Jahre in der praktischen Arbeit mit XML-Daten kaum eine Rolle spielen wird. Aus diesem Grund wird in diesem Buch auch weiterhin von der Version 1.0 ausgegangen. Dies gilt umso mehr, als in der 2008 erschienenen 5. Edition von XML 1.0 bereits einige der Lockerungen aus XML 1.1 in Bezug auf die erlaubten Element- und Attributnamen übernommen wurden.

Kapitel 3
Dokumenttypen und Validierung

Soll die Gültigkeit eines XML-Dokuments auf ein bestimmtes Vokabular eingeschränkt werden, ist der Entwurf von Inhaltsmodellen notwendig. Das zuerst dafür etablierte Verfahren ist die Definition von Dokumenttypen.

SGML – *the mother of XML* – ist zunächst hauptsächlich zur Bewältigung umfangreicher, strukturierter Textmengen entworfen worden. Die Erfahrungen beim Umgang mit solchen Texten haben zu dem Konzept des *Dokumenttyps* geführt. Texte, wie sie in einem Buch, einem Artikel oder einer Gebrauchsanweisung dargeboten werden, haben eine innere Struktur, die sich in abstrakte Informationseinheiten gliedern lässt.

Es erwies sich aber sofort, dass die Verfahren, die für die Strukturierung von Textdokumenten angewendet werden können, auch bei der Informationsmodellierung in ganz anderen Bereichen anwendbar sind.

3.1 Metasprache und Markup-Vokabulare

So wie sich ein Text in Einleitung, Hauptteil und Nachwort gliedern lässt, der Hauptteil wiederum in Kapitel und Unterkapitel, die Unterkapitel in Abschnitte etc., so kann beispielsweise ein Projekt in Vorbereitungsphase, Durchführungsphase und Nachbereitungsphase zerlegt werden.

Die Beschreibung eines Produkts kann gegliedert werden in die darin enthaltenen Teilprodukte bis hinunter zur Ebene der Grundbausteine, aus denen das Produkt zusammengesetzt ist. In allen Fällen liegt es nahe, solche hierarchischen Zusammenhänge durch eine grafische Darstellung zu verdeutlichen, deren Kern eine geordnete Baumstruktur ist, geordnet in dem Sinn, dass die Anordnung der Elemente nicht beliebig ist.

3.1.1 Datenmodelle

Die Arbeit, einen Gegenstandsbereich in einem abstrakten Datenmodell abzubilden, ist zunächst ganz unabhängig von technischen Vorschriften oder Einschränkungen. Es geht dabei darum, einen wie immer gearteten Bereich in sinnvoller Weise zu gliedern, also so, wie es seiner inneren Logik entspricht.

Wenn es aber nicht nur darum geht, dass ein solches Informationsmodell von Menschen verstanden wird, sondern dass auch Maschinen bzw. Programme mit diesen Modellen etwas anfangen können, sind formale Beschreibungswerkzeuge notwendig, die in der Lage sind, das Modell entsprechend zu repräsentieren.

3.1.2 Selbstbeschreibende Daten und Lesbarkeit

Damit ein Programm die Gültigkeit eines Dokuments in der oben schon angesprochenen Weise durch einen Abgleich mit einem zugeordneten Schema prüfen kann, muss die Beschreibung dieses Schemas selbst in einer auch von Maschinen lesbaren Sprache abgefasst sein. Das heißt, sie muss sich einer formalen Sprache bedienen.

Bei den Dokumenttyp-Definitionen, den DTDs, die dafür zunächst hauptsächlich zum Einsatz kamen, wurden entsprechende Sprachelemente von SGML übernommen. Sie wurden direkt in die Spezifikation von XML eingearbeitet.

3.1.3 Dokumenttyp-Definition – DTD

Eine DTD definiert eine bestimmte Klasse von Dokumenten, die alle vom gleichen Typ sind, indem sie verbindlich das Vokabular und die Grammatik für die Auszeichnungssprache festlegt, die bei der Erstellung des Dokuments verwendet werden soll und darf.

Das Dokument, das mit einem bestimmten Datenmodell übereinstimmt, wird deshalb auch als Instanz des Modells betrachtet, ähnlich wie in der objektorientierten Programmierung ein Objekt als Instanz einer Klasse behandelt wird.

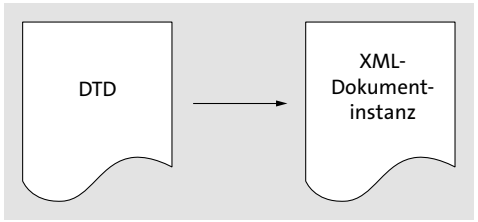


Abbildung 3.1 Ein XML-Dokument, das einer DTD entspricht, ist eine Dokumentinstanz dieser DTD.

Wie mächtig eine DTD sein kann, wird an der Tatsache deutlich, dass alle Webseiten im Grunde Instanzen einer einzigen Klasse von Dokumenten sind, die mit der DTD für HTML definiert ist.

Die XML 1.0-Spezifikation geht im Übrigen noch explizit davon aus, dass die Gültigkeit von XML-Dokumenten gegen eine entsprechende DTD geprüft wird, und enthält dazugehörige Regeln.

3.1.4 XML-Schema

Seit einigen Jahren kann das Schema für ein Datenmodell auch in einer XML-Syntax beschrieben werden, mit Hilfe von *XML-Schema*. Dieses spezielle Vokabular zur Beschreibung von Datenstrukturen wurde im Jahre 2001 vom W3C als Standard verabschiedet. In der Folge haben XML-Schemas in vielen Bereichen den Platz der DTDs übernommen. Mit Hilfe von XML-Schemas können vor allem präzise Angaben zu den Datentypen gemacht werden, die für die einzelnen Elemente oder Attributwerte erlaubt werden sollen.

DTDs sind immer noch in zahlreichen gewichtigen Varianten für die verschiedensten Sachgebiete im Einsatz – zwei Beispiele zum Schluss des Kapitels sind die DTDs für DocBook und SVG. Deshalb werden in diesem Buch beide Formen der Strukturbeschreibung von Dokumenten nacheinander behandelt, allerdings mit einem deutlichen Schwerpunkt auf dem XML-Schema-Standard.

Wer gewohnt ist, Schemas für Datenbanktabellen zu entwerfen, dem wird die Aufgabe der Datenmodellierung mit XML-Schema sicherlich nicht fremd sein, auch wenn XML dafür ein spezielles Vokabular zur Verfügung stellt, das erst einmal gelernt werden will.

3.1.5 Vokabulare

Durch die Verkoppelung eines XML-Dokuments mit einer DTD oder einem Schema entsteht jedes Mal eine bestimmte Variante der XML-Sprache, eine Art Dialekt der formalen Art. Es gibt also in gewissem Sinne so viele XML-Sprachen, wie es DTDs und Schemas gibt. Das Spektrum der Aufgaben, die mit Hilfe dieser Spezialsprachen gelöst werden können, ist längst nicht ausgeschöpft, aber so unterschiedliche Sprachvarianten wie XHTML, MathML oder SVG, um nur wenige zu nennen, deuten an, wie tragfähig XML als Metasprache für all diese Sprachen ist.

Ob DTDs oder Schemas ihren Zweck erfüllen, hängt natürlich hauptsächlich davon ab, wie sie den Gegenstandsbereich, auf den sie sich beziehen, repräsentieren. Dabei kommt es darauf an, dass eine entsprechende XML-Sprache Anwendungen in dem betreffenden Bereich in einer Weise unterstützt, die die Anwender nicht nur zufriedenstellt, sondern ihnen gegenüber bisherigen Lösungen einen zusätzlichen Nutzen verspricht.

3.2 Regeln der Gültigkeit

Die Arbeit der Datenmodellierung gewinnt ihren Wert insbesondere dadurch, dass auf der Basis einer einmal fixierten Struktur ein Programm automatisch prüfen kann, ob ein bestimmtes Dokument dieser Struktur tatsächlich entspricht und in

diesem Sinne regelgerecht erstellt worden ist. Während sich die oben schon behandelte Prüfung auf Wohlgeformtheit gewissermaßen mit Äußerlichkeiten zufriedengibt, geht es hierbei um eine wesentlich strengere Prüfung, der ein XML-Dokument unterworfen werden kann, nämlich um die Prüfung auf Gültigkeit.

Allerdings setzt die Prüfung auf Gültigkeit, die auch *Validierung* genannt wird, immer voraus, dass die Prüfung auf Wohlgeformtheit schon bestanden ist. Diese Prüfung kann stattfinden, wenn Regeln definiert werden, die den Betrachter oder eine Maschine zwischen gültigen und ungültigen Inhalten unterscheiden lassen. Existieren solche Regeln, besteht die Gültigkeit des Dokuments darin, dass es diesen Regeln entspricht. Nicht weniger, aber auch nicht mehr, denn in diesem Zusammenhang sagt Gültigkeit nichts über die Richtigkeit des Inhalts im XML-Dokument aus. Ein Element `<adresse>` kann korrekt aus den Unterelementen `<name>`, `<strasse>`, `<plz>` und `<ort>` zusammengesetzt sein, das hindert aber niemanden daran, eine falsche Postleitzahl einzugeben.

Die Validierung verlangt also ein schematisches Modell, das beschreibt, welche Elemente ein XML-Dokument eines bestimmten Typs enthalten muss und kann, welche Eigenschaften diese Elemente haben müssen oder können und wie Elemente und Attribute innerhalb des Dokuments verwendet werden.

Die Modellierung arbeitet dabei mit bestimmten Einschränkungen, die bei der Erstellung des Dokuments zu beachten sind. Ein Dokument vom Typ Geschäftsbrief muss zum Beispiel eine Betreff-Angabe enthalten, ein Datum und ein Kürzel des Bearbeiters. Folgt ein Dokument diesem Modell, wird es als *Instanz* des Modells bezeichnet. Die Beziehung gleicht, wie schon angesprochen, derjenigen zwischen einem Objekt und einer Klasse, die eine Menge von möglichen Objekten gleichen Typs vorgibt, wie es aus der objektorientierten Programmierung vertraut ist.

3.3 DTD oder Schema?

In welcher Situation ist überhaupt ein Dokumentmodell erforderlich, und wann ist es ratsam, eine DTD oder ein Schema zu verwenden? Die Antwort auf die erste Frage hängt in erster Linie davon ab, ob sich bestimmte Datenstrukturen häufig wiederholen oder nicht. Wird eine Datenstruktur nur einmal verwendet, ist es nicht nötig, ein Modell dafür zu entwerfen. Sobald bestimmte Datenmengen mehrfach verwendet werden müssen, lohnt sich ein Dokumentmodell zur Kontrolle der Gültigkeit, aber auch, weil aktuelle XML-Editoren aus solchen Modellen Eingabemasken generieren können, die die Datenpflege wesentlich erleichtern.

Wenn Sie auf ein Datenmodell zur Kontrolle der Gültigkeit von XML-Dokumenten verzichten, müssen Sie darauf eingestellt sein, dass ein normaler XML-Prozessor jeden Elementnamen akzeptieren wird, der die Namensregeln von XML nicht ver-

letzt, also keine ungültigen Zeichen enthält. Es gibt in diesem Fall auch keine grammatikalischen Einschränkungen. Ein Element kann einen beliebigen Inhalt haben: Kindelemente, Text, Mischungen aus Elementen und Text oder gar nichts.

Jedes Element kann auch beliebige Attribute enthalten, solange sie eindeutig sind. Allerdings sind nur Attribute vom Typ `CDATA`, also Zeichendaten, erlaubt; Verknüpfungen, wie sie durch `ID`- und `IDREF`-Attribute möglich sind, lassen sich ohne Datenmodell nicht realisieren. Das gilt auch für Vorgabewerte bei Attributen.

Die Entscheidung, ob DTD oder Schema günstiger ist, hängt in erster Linie davon ab, um welche Daten es geht. Anwendungen, die mit Textdokumenten zu tun haben, können mit DTDs gut zurechtkommen, solange nicht sehr spezielle Datenformate eine Rolle spielen, deren Korrektheit zu beachten ist. Anwendungen mit stark differenzierten Datenstrukturen, wie sie von Datenbanksystemen bekannt sind, sollten die Möglichkeiten von XML-Schema nutzen.

3.4 Definition eines Dokumentmodells

Wenn das XML-Dokument mit einer Dokumenttyp-Definition – also einer DTD – verknüpft werden soll, ist nach der XML-Deklaration eine Dokumenttyp-Deklaration notwendig. Die DTD kann dabei direkt in das Dokument eingebettet werden, und zwar vor dem ersten Element, oder es kann ein Bezug auf eine externe DTD hergestellt werden oder eine Kombination von beidem. Auf die Einzelheiten wird in Abschnitt 3.10, »Formen der DTD-Deklaration«, eingegangen.

3.4.1 Interne DTD

Hier zunächst ein kleines Beispiel für eine interne DTD und ein Beispiel für den Bezug auf eine externe DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Lager [
  <!ELEMENT Absatz (#PCDATA)>
  <!ELEMENT Artnr (#PCDATA)>
  <!ELEMENT Bestand (#PCDATA)>
  <!ELEMENT Bezeichnung (#PCDATA)>
  <!ELEMENT Farbe (#PCDATA)>
  <!ELEMENT Material (#PCDATA)>
  <!ELEMENT Preis (#PCDATA)>
  <!ELEMENT Umsatz_Vorjahr (#PCDATA)>
  <!ELEMENT Umsatz_lfd_Jahr (#PCDATA)>
  <!ELEMENT Warengruppe (#PCDATA)>
```

```
  <!ELEMENT Artikel (Artnr, Bezeichnung, Warengruppe, Material,
     Farbe, Bestand, Preis, Absatz, Umsatz_lfd_Jahr, Umsatz_Vorjahr)>
  <!ELEMENT Lager (Artikel+)>
]>
<Lager>
  <Artikel>
    <Artnr>7777</Artnr>
    <Bezeichnung>Jalousie Ccxs</Bezeichnung>
    <Warengruppe>Jalousie</Warengruppe>
    <Material>Kunststoff</Material>
    <Farbe>grau</Farbe>
    <Bestand>100</Bestand>
    <Preis>198</Preis>
    <Absatz>120</Absatz>
    <Umsatz_lfd_Jahr>23801</Umsatz_lfd_Jahr>
    <Umsatz_Vorjahr>67988</Umsatz_Vorjahr>
  </Artikel>
  <Artikel>
    <Artnr>7778</Artnr>
    <Bezeichnung>Jalousie Ccxx</Bezeichnung>
    <Warengruppe>Jalousie</Warengruppe>
    <Material>Metall</Material>
    <Farbe>rot</Farbe>
    <Bestand>200</Bestand>
    <Preis>174</Preis>
    <Absatz>330</Absatz>
    <Umsatz_lfd_Jahr>57600,8</Umsatz_lfd_Jahr>
    <Umsatz_Vorjahr>76800,88</Umsatz_Vorjahr>
  </Artikel>
</Lager>
```

Listing 3.1 lagerbestand.xml

Das XML-Dokument beginnt in diesem Fall mit einer Dokumenttyp-Deklaration, der die Dokumenttyp-Definition, die DTD, gleich folgt. Die gesamte DTD wird dabei in eckige Klammern gesetzt. Erst danach kommen die Daten, die den Regeln dieser DTD gehorchen sollen.

3.4.2 Externe DTD

Die im Beispiel vorhandene DTD kann auch aus dem XML-Dokument ausgegliedert werden. Die Datei, für die üblicherweise die Dateierweiterung *.dtd* verwendet wird, sieht dann so aus:


```
<!ELEMENT Absatz (#PCDATA)>
  <!ELEMENT Artnr (#PCDATA)>
  <!ELEMENT Bestand (#PCDATA)>
  <!ELEMENT Bezeichnung (#PCDATA)>
  <!ELEMENT Farbe (#PCDATA)>
  <!ELEMENT Material (#PCDATA)>
  <!ELEMENT Preis (#PCDATA)>
  <!ELEMENT Umsatz_Vorjahr (#PCDATA)>
  <!ELEMENT Umsatz_lfd_Jahr (#PCDATA)>
  <!ELEMENT Warengruppe (#PCDATA)>
  <!ELEMENT Artikel (Artnr, Bezeichnung, Warengruppe, Material,
     Farbe, Bestand, Preis, Absatz, Umsatz_lfd_Jahr,
     Umsatz_Vorjahr)>
  <!ELEMENT Lager (Artikel+)>
```

Für den Bezug auf diese externe DTD wird im XML-Dokument folgende Syntax verwendet:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Lager SYSTEM "Lager.dtd">
```

Wie Sie sehen, wird als Name der DTD immer der Name des Wurzelelements des entsprechenden Dokumenttyps verwendet.

Das Beispiel zeigt, dass DTDs direkt in ein XML-Dokument eingebettet oder mit dem XML-Dokument verknüpft werden können. Auch eine Mischung von internen und externen DTDs ist möglich, etwa um eine in einem Anwendungsbereich verwendete DTD für eine spezielle Situation zu erweitern oder auch einzuschränken.

3.5 Deklarationen für gültige Komponenten

Wir wollen hier zunächst an einem übersichtlichen Beispiel – einem Kursprogramm – die einzelnen Komponenten einer DTD anhand einer internen Verwendung beschreiben. Technisch besteht die Definition eines Dokumenttyps aus einer Zusammenstellung von Markup-Deklarationen, das heißt, in der DTD wird bestimmt, welche Markups oder Tags in einem Dokument verwendet werden dürfen und müssen und in welcher Weise dies zu geschehen hat. Die DTD legt also Einschränkungen fest, die die große Tag-Freiheit in einem nur wohlgeformten XML-Dokument ersetzt durch das harte Regime eines fixierten Vokabulars.

Sie sollten dabei im Auge behalten, dass ein XML-Dokument aus der Umzäunung durch eine DTD ausbrechen kann, ohne den Status eines wohlgeformten Dokuments zu verlieren. Wenn der XML-Prozessor auf eine Bewertung der Gültigkeit verzichtet, kann ein solches Dokument also durchaus verarbeitet werden.

Eine Dokumenttyp-Definition ist entweder innerhalb einer Dokumenttyp-Deklaration eingebettet oder in einer separaten Datei abgelegt, auf die dann eine Dokumenttyp-Deklaration in einem XML-Dokument Bezug nimmt.

3.5.1 Vokabular und Grammatik der Informationseinheiten

Die Dokumenttyp-Definition ist ein Vokabular und eine Grammatik für eine bestimmte Klasse von Dokumenten, und zwar in Form von Markup-Deklarationen. Markup-Deklarationen wiederum werden benutzt, um die vier möglichen Komponenten, aus denen sich ein XML-Dokument zusammensetzen kann, festzulegen. Dazu dienen:

- Elementtyp-Deklarationen
- Attributlistendeklarationen
- Entitätsdeklarationen
- Notationsdeklarationen

3.5.2 Syntax der Dokumenttyp-Deklaration

Die Dokumenttyp-Deklaration, die mit `<!DOCTYPE` startet, benennt den Dokumenttyp und legt damit zugleich den Namen des obersten Elements fest, das für Dokumente dieser Klasse verwendet wird. Wenn der hinter dem Schlüsselwort `DOCTYPE` verwendete Name nicht mit dem Namen des obersten Elements des Dokuments übereinstimmt, wird ein XML-Prozessor das Dokument nicht als gültig anerkennen.

Die Deklaration muss hinter der XML-Deklaration und vor dem ersten Element des Dokuments eingefügt werden. Kommentare sind in jeder Zeile erlaubt. Hier das Skelett eines XML-Dokuments mit einer eingebetteten DTD, die zunächst nur ein einziges Element deklariert:

```
<?xml version="1.0">
<!DOCTYPE  kursprogramm  [
<!ELEMENT kursprogramm (kurs+)>
]>
<kursprogramm>
...
</kursprogramm>
```

3.5.3 Syntax der Elementtyp-Deklaration

Die Definition des unter dem angegebenen Namen `kursprogramm` deklarierten Dokumenttyps beginnt mit der ersten Elementtyp-Deklaration. Wenn ein Element im XML-Dokument als gültig angesehen werden soll, muss in der DTD eine Typdeklaration für dieses Element aufgeführt sein. Diese Deklaration beginnt wiederum mit `<!ELEMENT`, gefolgt vom Namen des Elementtyps. Hinter dem Namen des Elements folgt eine verbindliche Angabe darüber, was der Inhalt des Elements sein soll und darf. Die allgemeine Syntax einer Elementdeklaration ist also:


```
<!ELEMENT Name Inhaltsmodell>
```

Über die Einschränkungen, die die Namen von Elementen betreffen, ist in Abschnitt 2.1.9 bereits gesprochen worden: Namen müssen mit einem Buchstaben oder einem Unterstrich beginnen. Auch ein Doppelpunkt ist erlaubt, sollte aber vermieden werden, weil im XML-Dokument der Doppelpunkt für die Trennung des Namensraumpräfixes vom lokalen Namen selbst verwendet wird.

Die Länge der Namen ist nicht begrenzt. Beachtet werden muss auch wieder, dass bei Namen, wie bei XML üblich, zwischen Groß- und Kleinschreibung unterschieden wird: `name` ist also nicht dasselbe Element wie `Name`.

Ansonsten kann der Name frei gewählt werden; es sollte aber darauf geachtet werden, dass er seinen beschreibenden Charakter behält. Der Name sollte also möglichst präzise angeben, welche Information das Element bereitstellt. Namen wie »element1«, »element2«, ... sind nicht verboten, aber damit wird der große Vorteil von XML verschenkt, der ja gerade darin besteht, dass sich bei diesem Datenformat die Daten selbst beschreiben.

Elementnamen sollten in einer DTD nicht mehrfach verwendet werden, sondern eindeutig sein, durchaus im Unterschied zu XML-Schema, wie Sie in Kapitel 4 noch sehen werden. Wird bei zwei Elementtyp-Deklarationen derselbe Name verwendet, ignoriert der Prozessor die zweite Deklaration.

3.5.4 Beispiel einer DTD für ein Kursprogramm

Das Kursprogramm soll nun aus mehreren Kursen bestehen. Pro Kurs wird eine Bezeichnung vergeben, ein Text über den Kursinhalt angelegt, eine Liste der Referenten erstellt und ein Termin festgelegt. Die DTD für das Kursprogramm könnte in einem ersten Entwurfsstadium folglich so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kursprogramm [
<!ELEMENT kursprogramm (kurs+)>
<!ELEMENT kurs (bezeichnung, kursinhalt, referententeam, termin, anhang)>
<!ELEMENT bezeichnung (#PCDATA)>
<!ELEMENT kursinhalt (#PCDATA)>
<!ELEMENT referententeam (referent+)>
<!ELEMENT referent (name, adresse, kontakt?, bild?)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT adresse (#PCDATA)>
<!ELEMENT kontakt (fon | email* | (fon, email*))>
<!ELEMENT fon (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT bild EMPTY>
<!ELEMENT termin (#PCDATA)>
<!ELEMENT anhang ANY>
]>
```

Listing 3.2 kursprogramm.xml und .dtd

Elemente, die weitere Elemente enthalten

Das Element `<kursprogramm>` darf als Inhalt nur Kindelemente vom Typ `<kurs>` enthalten. Da das Programm mehr als einen Kurs enthalten soll, wird ein Pluszeichen hinter den Elementnamen gesetzt. Das Pluszeichen wird als Operator für die *Kardinalität* verwendet, gibt also an, wie häufig ein Element an einer bestimmten Stelle vorkommen kann, darf oder muss. Dabei bedeutet +, dass das Element mindestens einmal oder mehrmals vorkommt. Wird dagegen kein Operator verwendet, muss das Element genau einmal vorkommen, was für das Kursprogramm wenig sinnvoll wäre.

Das Element `<kurs>` ist wiederum selbst aus mehreren Kindelementen zusammengesetzt, die in der Klammer hinter dem Elementnamen aufgereiht werden, getrennt durch Kommas. Damit wird festgelegt, dass die betreffenden Elemente im XML-Dokument auch in dieser Reihenfolge – als Sequenz – zu erscheinen haben. Werden die Daten in der falschen Reihenfolge eingegeben, ist das Dokument nicht gültig.

Entscheidend ist, dass für jedes Kindelement von `<kurs>` auch eine entsprechende Elementtyp-Deklaration in der DTD angelegt wird.

Elemente mit Zeichendaten

Für das erste und zweite Kindelement von `<kurs>` wird in der Elementtyp-Deklaration nur noch der Datentyp `#PCDATA` angegeben. Das Wort ist eine Abkürzung für *parsed character data*, womit gemeint ist, dass es sich um reinen Text ohne jedes Markup handelt. Das Doppelkreuz davor besagt, dass `PCDATA` ein vordefiniertes Schlüsselwort ist. Der Name weist zugleich darauf hin, dass der Parser eventuelle Entitätsreferenzen, die in den Zeichendaten enthalten sind, vor der weiteren Verarbeitung aufzulösen hat.

Der Datentyp `#PCDATA` ist zugleich auch schon der einzige Datentyp, der für den Inhalt dieses Elements angegeben werden kann. Sie haben also keine Möglichkeit festzulegen, dass ein Element nur Zahlen enthalten darf oder ein Datum oder eine Zeitangabe in einem entsprechenden Format. Auch über die Menge der Zeichendaten, die Länge eines Strings, kann hier nichts Einschränkendes festgelegt werden. Dieser Mangel wird erst durch XML-Schema beseitigt.

Damit ist zugleich klar, dass dieses Element selbst keine weiteren Kindelemente umschließt, sondern eben nur noch aus Zeichendaten besteht. Innerhalb der Baumstruktur, mit der das XML-Dokument dargestellt werden kann, sind das gewissermaßen die äußersten Enden, die Blätter des Baumes.

Anordnung der Elemente in einem Container-Element

Das Element `<referententeam>` dagegen enthält wieder Kindelemente, die teilweise selbst wiederum weitere Kindelemente enthalten. Zunächst wird auch bei der Elementdeklaration `<referententeam>` der Elementname `<referent>` durch ein Pluszeichen ergänzt. Der Kurs braucht ja mindestens einen Referenten, es können aber auch zwei oder drei sein.

Auch das Element `<referent>` besteht aus Kindelementen, diesmal aber aus Elementen unterschiedlichen Typs. Diese Unterelemente sind in der Klammer wieder durch Kommas getrennt. Als verbindliche Elemente erscheinen zunächst `<name>` und `<adresse>`, einfache Elemente, die wieder nur Zeichendaten enthalten dürfen. Das dritte Unterelement `<kontakt>` ist dagegen wieder ein Container weiterer Elemente. Es soll allerdings nur optional sein, es kann also auch fehlen, wenn der Referent dazu keine Daten hergeben will. Dazu wird der Operator ? an den Elementnamen angehängt. Dieses Element darf höchstens einmal vorkommen, aber es kann in einer Instanz dieses Dokumenttyps auch weggelassen werden.

Leere Elemente

Ebenfalls optional ist in unserem Beispiel das Element `<bild>`. Für dieses Element ist als Inhalt das Schlüsselwort `EMPTY` angegeben. Damit wird festgelegt, dass dieses Element leer ist, also weder Zeichendaten noch andere Elemente enthalten darf. Leere Elemente werden gerne verwendet, um über Attribute Verweise auf Dateien einzubinden, etwa Bilder, Sounds oder Video, oder auch, um bestimmte Kennzeichen zu setzen. Dazu später mehr.

3.5.5 Inhaltsalternativen

Welche Elemente soll das Element `<kontakt>` aber enthalten? Es gibt Leute, die ihre Telefonnummer angeben, andere wollen nur per E-Mail kontaktiert werden, wieder andere erlauben beide Formen der Kontaktaufnahme. Wer E-Mail zulässt, hat möglicherweise gleich mehrere E-Mail-Adressen. All diese Varianten soll die DTD berücksichtigen, so dass für die verschiedenen Referenten durchaus variable Sätze von Kontaktinformationen zusammengestellt werden können, ohne dass es zu Fehlern bei der Validierung des XML-Dokuments kommt.

Zunächst werden deshalb in der Klammer, die den Inhalt des Elements `<kontakt>` beschreibt, die drei Alternativen aufgelistet. Als Operator wird der senkrechte Strich

verwendet, der einem ausschließenden Oder entspricht. Es kann also nur eine der drei Alternativen verwendet werden.

Hinter den Elementnamen `email` wird diesmal ein * gesetzt. Damit wird festgelegt, dass beliebig viele E-Mail-Adressen an dieser Stelle vorkommen können. Die E-Mail-Adresse kann aber auch ganz fehlen. Die dritte Alternative ist die Angabe einer Telefonnummer und einer beliebigen Zahl von E-Mail-Adressen. Diese Abfolge ist noch einmal in Klammern gesetzt, das heißt, wenn der Referent eine Telefonnummer und E-Mail-Adresse angibt, muss zuerst die Telefonnummer eingegeben werden. Es wäre auch möglich, mehrere Kombinationen von Telefonnummern und E-Mail-Adressen zuzulassen:

```
<!ELEMENT kontakt (fon | email* | (fon, email*)*)>
```

In diesem Fall bezieht sich der dritte *-Operator auf die Sequenz der in Klammern aufgeführten Elementgruppe. In der Tabelle sind noch einmal alle Operatoren zusammengestellt, die bei der Beschreibung von Inhaltsmodellen in einer Elementtyp-Deklaration verwendet werden (siehe Tabelle 3.1).

Operator	Bedeutung
+	Das vorausgehende Element oder die Elementgruppe muss mindestens einmal, kann aber auch mehrfach vorkommen.
?	Das vorausgehende Element oder die Elementgruppe kann einmal vorkommen, kann aber auch fehlen.
*	Das vorausgehende Element oder die Elementgruppe kann beliebig oft vorkommen oder fehlen.
,	Trennzeichen innerhalb einer Sequenz von Elementen
|	Trennzeichen zwischen sich ausschließenden Alternativen
()	Bildung von Elementgruppen

Tabelle 3.1 Operatoren in einer DTD

Durch eine geschickte Kombination und Verschachtelung von Elementgruppen und Operatoren für die Kardinalität von Elementen oder Elementgruppen lassen sich bei Bedarf hoch komplexe Inhaltsmodelle in einer solchen Elementtyp-Deklaration realisieren.

3.5.6 Uneingeschränkte Inhaltsmodelle

Das letzte Kindelement von `<kurs>` ist ein Element `<anhang>`, dessen Inhalt mit dem Wort `ANY` spezifiziert wird. Während bei den bisher beschriebenen Elementdeklarationen alles relativ streng zuging, wird hier nun Tür und Tor für alles und jedes geöffnet. Die DTD legt zwar fest, dass zu jedem Kurs ein Anhang vorkommen soll, macht aber keine Vorschriften, was dessen Inhalt sein soll und darf.

Das Element kann also beispielsweise reine Textdaten enthalten oder weitere deklarierte Kindelemente, deren Anordnung aber frei gewählt werden darf, oder eine Mischung aus Text und Kindelementen oder auch ein leeres Element mit einem Bezug auf Daten, die keine XML-Daten sind. Oder auch gar nichts! Der Parser, der dieses Element in einer Dokumentinstanz überprüft, wird mit `ANY` angewiesen, dieses Element nicht weiter auf Gültigkeit zu kontrollieren. Nur die Wohlgeformtheit muss erhalten bleiben.

Das Inhaltsmodell `ANY` widerspricht in gewissem Sinne dem, was mit einer DTD erreicht werden soll, aber es ist in vielen Fällen nützlich, damit zu arbeiten. Das gilt zum Beispiel während der Entwicklung einer DTD, wenn bestimmte Bereiche eines Dokumentmodells noch in der Diskussion sind. Diese Bereiche können vorübergehend mit `ANY` gekennzeichnet werden, während die Inhaltsmodelle der anderen Bereiche schon fixiert werden. Die DTD kann dann in dieser Form bereits für die Validierung verwendet werden.

3.5.7 Gemischter Inhalt

Bei dem Inhaltsmodell `ANY` hatte ich schon erwähnt, dass innerhalb einer DTD auch ein Inhaltsmodell zugelassen wird, bei dem Textdaten und Elemente gemischt werden. Hier ein Beispiel:

```
<!ELEMENT anhang (#PCDATA | link | hinweis)*>
```

Der *-Operator hinter der Klammer sorgt dafür, dass das Element `<anhang>` in beliebiger Anzahl aus Zeichendatenteilen und den angegebenen Elementen gemischt werden kann. Ein gültiges Element in der Dokumentinstanz kann zum Beispiel so aussehen:

```
<anhang>Infos zu XML unter: <link>www.xml.org</link>
<hinweis>Eine Linkliste finden Sie unter:</hinweis>
<link>www.rheinwerk-verlag.de</link></anhang>
```

Allerdings gelten für dieses Inhaltsmodell bestimmte Einschränkungen, die es meist ratsam erscheinen lassen, ein solches Modell zu vermeiden. Es ist nicht möglich, die Reihenfolge der Kindelemente festzulegen, die innerhalb des gemischten Inhalts auftauchen dürfen. Außerdem lässt sich die Häufigkeit nicht mit den üblichen Operatoren *, + und ? bestimmen.

Dieses Inhaltsmodell kann allerdings helfen, wenn es darum geht, Textbestände, die bisher nicht im XML-Format vorliegen, schrittweise in XML zu konvertieren.

3.5.8 Inhaltsmodell und Reihenfolge

Was den Inhalt eines Elements betrifft, sind also insgesamt fünf verschiedene Inhaltsmodelle erlaubt (siehe Tabelle 3.2).

Inhaltsmodell	Beschreibung
EMPTY	Das Element hat keinen Inhalt, kann aber Attribute haben.
ANY	Das Element kann beliebige Inhalte haben, solange es sich um wohlgeformtes XML handelt.
#PCDATA	Das Element enthält nur Zeichendaten.
gemischter Inhalt	Das Element kann Zeichendaten und Unterelemente enthalten.
Elementinhalt	Das Element enthält ausschließlich Unterelemente.

Tabelle 3.2 Inhaltsmodelle in einer DTD

Die Reihenfolge der Elemente in der DTD ist normalerweise nicht von Bedeutung, mit der schon erwähnten Ausnahme, dass Elemente doppelt deklariert werden. Nur wenn Parameterentitäten verwendet werden, müssen die Deklarationen, auf die Bezug genommen werden soll, immer vorher erscheinen.

Wichtig ist nur, dass alle Elemente, die als Unterelemente eines anderen Elements erscheinen, auch tatsächlich deklariert werden. Ein einmal deklariertes Element kann andererseits durchaus in mehreren Elementgruppen verwendet werden, wenn dies erforderlich ist.

Zur besseren Lesbarkeit von DTDs ist es ratsam, den Elementebaum entweder von oben nach unten oder umgekehrt von unten nach oben abzuarbeiten. Im ersten Fall werden zunächst die Elternelemente und dann die Kindelemente deklariert, im zweiten Fall erst die Kindelemente und dann die zusammengesetzten Elternelemente. Es kann aber auch sinnvoll sein, die Elemente einfach alphabetisch nach den Namen zu sortieren, wie es einige Generatoren für DTDs tun.

3.5.9 Kommentare

Wie in den XML-Dokumenten selbst werden auch innerhalb einer DTD Kommentare mit den Trennzeichenfolgen `<!--` und `-->` verpackt. Diese können überall verwendet werden, nur nicht innerhalb einer Deklaration selbst. Bei komplexen DTDs ist es sehr sinnvoll, Gruppen von Elementen durch Kommentare einzuleiten, um die Struktur des Informationsmodells deutlich werden zu lassen.

3.5.10 Die Hierarchie der Elemente

Beim Entwurf einer DTD ist es oft hilfreich, sich die Hierarchie der Elemente in einer Baumstruktur zu vergegenwärtigen, um Fehler beim Design zu vermeiden.

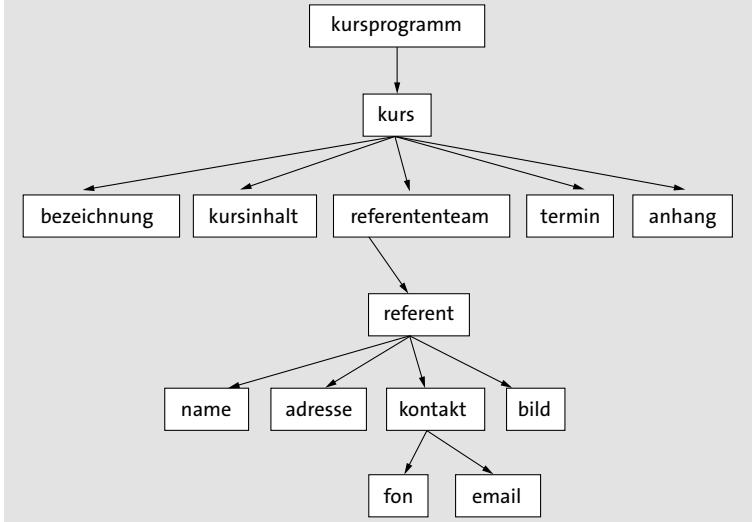


Abbildung 3.2 Die Baumstruktur des Beispiels

Abbildung 3.2 zeigt für unser Beispiel einen Baum mit sechs Ebenen. So kann leicht geprüft werden, ob alle Elemente, die benötigt werden, berücksichtigt worden sind und ob sie in der richtigen Reihenfolge zusammengestellt sind.

3.6 Dokumentinstanz

Um zu prüfen, ob eine DTD »funktioniert«, sollte sie zunächst mit einer ersten Dokumentinstanz getestet werden. In unserem Beispiel könnte dies so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kursprogramm SYSTEM "kursprogramm.dtd">
<kursprogramm>
  <kurs>
    <bezeichnung>XML-Einführung</bezeichnung>
    <kursinhalt>Eine Woche Praxis für XML-Einsteiger</kursinhalt>
    <referententeam>
      <referent>
        <name>Hanna Domen</name>
        <adresse>Tegolitstr. 10, 50678 Köln</adresse>
```

```
      <kontakt>
        <fon>0221998877</fon>
        <email>hannad@net.net</email>
      </kontakt>
      <bild/>
    </referent>
  </referententeam>
  <termin>12.12.2015</termin>
  <anhang>Das Seminar wird jeden 2. Monat wiederholt.</anhang>
</kurs>
<kurs>
  <bezeichnung>XML Schema oder DTD?</bezeichnung>
  <kursinhalt>Vergleich der Werkzeuge für die Datenmodellierung
  </kursinhalt>
  <referententeam>
    <referent>
      <name>Karl Frimm</name>
      <adresse>Herthastr. 12, 50679 Köln</adresse>
      <kontakt>
        <fon>0221998557</fon>
      </kontakt>
      <bild/>
    </referent>
  </referententeam>
  <termin>6.12.2015</termin>
  <anhang>Das Seminar wird jeden Monat wiederholt.</anhang>
</kurs>
</kursprogramm>
```

Listing 3.3 Dokumentinstanz für Beispiel-DTD

Wie zu sehen ist, sind die Kontaktinformationen für die beiden `<referent>`-Elemente unterschiedlich zusammengesetzt, was die Elementtyp-Deklaration von `<kontakt>` ja ausdrücklich zulässt.

3.7 Attributlisten-Deklaration

Ergänzend zur Deklaration aller Elemente, die in einem gültigen Dokument erlaubt sein sollen, müssen auch alle Attribute, die die Information, die das Element selbst mitbringt, ergänzen, in der DTD deklariert werden. Über die Frage, welche Information über das Element und welche über seine Attribute bereitgestellt werden sollte, ist bereits in Abschnitt 2.3, »Elemente oder Attribute?«, einiges gesagt worden.

Attribute werden nicht einzeln deklariert, sondern innerhalb von Attributlisten, die einem bestimmten Element zugeordnet werden.

3.7.1 Aufbau einer Attributliste

Die allgemeine Syntax einer Attributlisten-Deklaration ist:

```
<!ATTLIST Elementname
  Attributname Attributtyp Vorgabewertdeklaration
  Attributname Attributtyp Vorgabewertdeklaration
  ...
>
```

Soll in unserem Beispiel das Element `<kurs>` um zwei Attribute erweitert werden, die den Kurstyp und die Kurseinstufung enthalten, lässt sich die folgende Deklaration verwenden:

```
<!ELEMENT kurs (bezeichnung, kursinhalt,
<!ATTLIST kurs
  kurstyp CDATA #REQUIRED
  kurseinstufung CDATA #REQUIRED
>
```

Für Attributnamen gelten dieselben Regeln wie für Elementnamen, auch hier muss die Groß-/Kleinschreibung beachtet werden – `id` ist also nicht dasselbe Attribut wie `ID`.

Es steht Ihnen frei, alle Attribute für ein Element in einer Attributliste zu deklarieren oder mehrere Teillisten für dasselbe Element zu verwenden. Die Platzierung der Attributlisten-Deklaration innerhalb der DTD ist Ihnen genauso freigestellt wie die der Elemente. Da der Bezug auf das Element immer in die Deklaration mit hineingenommen wird, spielt es keine Rolle, ob die Attributliste vor oder hinter dem betreffenden Element deklariert wird. Sie können die Attributliste der besseren Lesbarkeit wegen direkt hinter dem Element einfügen, auf das sich die Liste bezieht. Es ist oft auch sinnvoll, alle Elemente und alle Attributlisten getrennt in geschlossenen Blöcken anzuordnen.

Die Anzahl der Attribute in der Attributliste ist nicht begrenzt, und die Reihenfolge ist ohne Bedeutung, sie schreibt also nicht vor, in welcher Anordnung die Attribute in der Dokumentinstanz erscheinen müssen. Nur wenn ein Attributname in der Liste mehrfach verwendet wird, spielt die Reihenfolge eine Rolle. Es gilt dann immer die erste Definition, die folgenden werden ignoriert.

3.7.2 Attributtypen und Vorgaberegelungen

Anders als bei den Elementen, die außer Inhaltsmodellen nur noch nicht weiter typisierte Zeichendaten enthalten können, lassen sich für die Werte, die einem Attribut zugewiesen werden können, etwas differenziertere Festlegungen treffen. Außerdem sind Vorgaben möglich, falls in der Dokumentinstanz kein Wert für ein bestimmtes Attribut angegeben wird.

Für die Werte von Attributen sind zehn grundlegende Typen wählbar: ein Typ ohne Struktur – `CDATA` –, sechs atomare Token-Typen – `NMTOKEN`, `ID`, `IDREF`, `NOTATION`, `ENTITY`, Aufzählung – und drei Listentypen – `NMTOKENS`, `IDREFS` und `ENTITIES`. Die folgende Tabelle gibt einen Überblick.

Attributtyp	Beschreibung
Aufzählung	In Klammern eingeschlossene Liste von Token-Werten, von denen jeweils einer als Attributwert verwendet werden kann und muss
`CDATA`	Einfache Zeichendaten, die kein Markup enthalten. Entitätsreferenzen sind aber erlaubt.
`ENTITY`	Name einer in der DTD deklarierten nicht geparsten Entität
`ENTITIES`	Durch Leerzeichen getrennte Liste von Entitäten
`ID`	Eindeutiger XML-Name, der als Identifizierer eines Elements verwendet werden kann; entspricht einem Schlüsselwert in einem Datensatz.
`IDREF`	Verweis auf den `ID`-Identifizierer eines Elements. Der Wert von `IDREF` muss mit dem `ID`-Wert eines anderen Elements im Dokument übereinstimmen.
`IDREFS`	Liste von Verweisen auf `ID`-Identifizierer, getrennt durch Leerzeichen
`NMTOKEN`	Namenssymbol aus beliebigen Zeichen, die in XML-Namen erlaubt sind, aber ohne Leerzeichen. Hier sind auch reine Zahlen-Token möglich, etwa für Jahreszahlen.
`NMTOKENS`	Liste von Namens-Token, getrennt durch Leerzeichen
`NOTATION`	Verweis auf eine Notation, zum Beispiel der Name für ein Nicht-XML-Format, etwa eine Grafikdatei

Tabelle 3.3 Attributtypen in DTD

Für die Vorgabebehandlung bei Attributwerten stehen vier Einstellungen zur Verfügung; die folgende Tabelle gibt einen Überblick.

Vorgabedeklaration	Beschreibung
Attributwert	In Anführungszeichen eingeschlossene Zeichendaten geben den vorgegebenen Wert an, der verwendet wird, wenn kein Wert angegeben ist.
`#IMPLIED`	Es gibt keine Vorgabe, und es ist auch kein Wert für das Attribut erforderlich.
`#REQUIRED`	Legt fest, dass kein Vorgabewert existiert, der Wert aber erforderlich ist. Der Wert kann aber auch eine leere Zeichenkette sein.
`#FIXED Wert`	Legt fest, dass in jedem Fall die mit `Wert` angegebene Konstante zu verwenden ist.

Tabelle 3.4 Werte für die Vorgabedeklaration

3.7.3 Verwendung der Attributlisten

Im Folgenden einige Beispiele, bezogen auf unsere Kursprogramm-DTD:

```
<!ATTLIST kurs
  id ID #REQUIRED
  kurstyp CDATA "Wochenkurs"
  kurseinstufung (Einsteiger | Profis) #REQUIRED
  sprache NMTOKEN "DE"
>
```

Diese Deklaration verlangt für das Element `<kurs>` einen eindeutigen Schlüssel, gibt den Kurstyp vor und bietet für die Kurseinstufung die Wahl zwischen zwei Werten. Für die Kurssprache sind Token erlaubt, wobei `"DE"` vorgegeben wird. Eine Instanz, die diese Bedingungen erfüllt, sähe beispielsweise so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kursprogramm SYSTEM "kursprogramm_attr.dtd">
<kursprogramm>
  <kurs  id="xmleinf" kurseinstufung="Einsteiger"
         kurstyp="Wochenkurs" sprache="EN">
    <bezeichnung>XML-Einführung</bezeichnung>
    ...
  </kurs>
```

```
  <kurs  id="schemadtd" kurseinstufung="Profis">
    <bezeichnung>XML-Schema oder DTD?</bezeichnung>
    ...
  </kurs>
</kursprogramm>
```

Listing 3.4 kursprogramm_attr.xml

Obwohl bei dem zweiten Kurselement die Attribute `kurstyp` und `sprache` im Quelldokument nicht angegeben worden sind, zeigt der Internet Explorer die vorgegebenen Werte für beide Attribute an, wie der Auszug in Abbildung 3.3 zeigt.

Abbildung 3.3 Die Ausgabe im Browser zeigt vorgegebene Werte mit an.

Beachten Sie, dass es sich bei den Werten des Typs `ID` um gültige XML-Namen handeln muss. Die vielleicht naheliegende Idee, Zahlen als Schlüsselwerte zu verwenden, führt zu einem Fehler, weil ein XML-Name nicht mit einer Zahl beginnen darf. ID-Attribute sind ein wichtiges Mittel, um auch Elemente eindeutig ansprechen zu können, deren Inhalt sonst gleich ist.

3.8 Verweis auf andere Elemente

Mit dem Attributtyp `IDREF` können Sie interne Verweise innerhalb eines Dokuments erstellen. Voraussetzung ist, dass Attribute vom Typ `ID` vorhanden sind. Soll in dem Kursprogramm zum Beispiel eingefügt werden, dass ein bestimmter Kurs einen anderen Kurs voraussetzt, hilft folgende Deklaration:

```
<!ATTLIST kurs
  id ID #REQUIRED
  voraussetzung IDREF #IMPLIED
  ...>
```


Diese Deklaration erlaubt die Verknüpfung mit einem anderen Kurs, verlangt sie aber nicht. Im Dokument kann das so aussehen:

```
<kursprogramm>
  <kurs  id="xmleinf" kurstyp="Wochenkurs"
    kurseinstufung="Einsteiger"
    sprache="EN">
    ...
  </kurs>
  <kurs  id="xmldtd"
    kurseinstufung="Profis" voraussetzung="xmleinf">
    ...
  </kurs>
```

Auf den Einsatz der Attributtypen `ENTITY`, `ENTITIES` und `NOTATION` komme ich noch einmal zu sprechen, nachdem ich die Verwendung von Entitäten in der DTD behandelt habe.

3.9 Verwendung von Entitäten

Die dritte Form von Deklarationen, die in einer DTD vorkommen können, sind die Entitätsdeklarationen. Die Rolle von Entitäten in XML-Dokumenten ist in Abschnitt 2.5, »Entitäten und Verweise darauf«, bereits behandelt worden.

In einer DTD können Sie zusätzlich noch spezielle Parameterentitäten verwenden. Sie erlauben es, innerhalb einer DTD Verweise auf Teile einer internen oder externen DTD zu nutzen. Abbildung 3.4 gibt einen Überblick über die verschiedenen Entitätstypen in einer DTD.

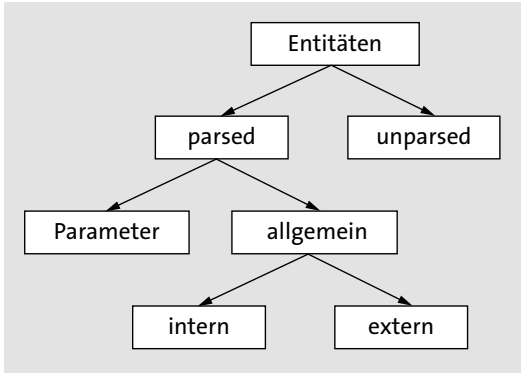


Abbildung 3.4 Typenbaum der Entitäten

3.9.1 Interne Entitäten

Am einfachsten ist die Deklaration von internen allgemeinen Entitäten. Sie lassen sich zum Beispiel verwenden, um Kürzel für längere Zeichenketten zu definieren, auf die dann im XML-Dokument verwiesen wird. Die Syntax ist sehr einfach:

```
<!ENTITY name "Ersetzungstext">
```

Zum Beispiel könnte in unserem Kursprogramm jeweils ein Kürzel für die Namen der Referenten festgelegt werden.

```
<!ENTITY hd "Hanna Domen">
<!ENTITY kf "Karl Frimm">
```

In der Dokumentinstanz lässt sich der Name des Referenten dann mit einer entsprechenden Entitätsreferenz eingeben:

```
<name>&hd;</name>
<name>&kf;</name>
```

Der XML-Parser wird bei der Verarbeitung der Dokumentinstanz diese Entitätsreferenzen auflösen – er erkennt sie an dem vorgestellten &-Zeichen und am abschließenden Semikolon – und die vollen Namen der Referenten in das Dokument einfügen. Deshalb wird in diesem Fall auch von geparsten Entitäten gesprochen.

Allgemeine Entitäten dürfen auch verschachtelt werden. Es ist also Folgendes erlaubt:

```
<!ENTITY hd "Hanna Domen">
<!ENTITY kf "Karl Frimm">
<!ENTITY duo "&hd; und &kf;">
```

Abbildung 3.5 Der Browser zeigt den vollen Namen an, der über eine Entitätsreferenz eingegeben wurde.

3.9.2 Externe Entitäten

Bei umfangreicheren Ersetzungstexten ist es meist sinnvoll, diese in separaten Dateien zu führen und mit Verweisen auf externe Entitäten zu arbeiten. Auf diese Weise kann zum Beispiel ein XML-Dokument aus mehreren Dokumenten zusammengebaut werden. Die Syntax der Deklaration


```
<!ENTITY name SYSTEM uri>
<!ENTITY name PUBLIC fpi uri>
```

verwendet einen URI für den Bezug auf die externe Entität und bei Verweisen auf öffentliche Ressourcen zusätzlich einen *Formal Public Identifier* (*FIP*), wie er auch für den Verweis auf öffentliche DTDs verwendet wird. Wie dieser zusammengesetzt ist, wird in Abschnitt 3.9.6, »Externe Parameterentitäten«, behandelt.

```
<!ENTITY kursdoc SYSTEM "kursbeschreibung1.xml">
```

ist ein Beispiel für eine Entitätsdeklaration, die einen Bezug auf ein externes XML-Dokument herstellt. Dieses Dokument muss so gestaltet sein, dass die Dokumentinstanz, in die eine Referenz auf dieses Dokument eingefügt wird, nach der Ersetzung der Referenz durch die angegebene Datei ein wohlgeformtes und im Sinne der DTD gültiges Dokument bleibt. Soll beispielsweise in das vorhin verwendete Element `<anhang>` der Inhalt eines externen Dokuments eingefügt werden, das in einem Element `<kursbeschreibung>` einen Text zum Kurs enthält, könnte die oben angeführte DTD zum Kursprogramm folgendermaßen geändert werden:

```
<!ELEMENT anhang (kursbeschreibung?)>
<!ELEMENT kursbeschreibung (#PCDATA)>
```

In der Dokumentinstanz kann dann das Element `<anhang>` so aussehen:

```
<anhang>&kursdoc;</anhang>
```

Der Browser zeigt das Element nach Auflösung der Entitätsreferenz mit der Kursbeschreibung an.

Abbildung 3.6 Der Browser zeigt den über eine Entitätsreferenz einbezogenen Text.

Es wäre auch möglich, die Daten über die einzelnen Kurse insgesamt jeweils in separaten Dateien bereitzustellen. Die DTDs könnten dann entsprechende Entitätsdeklarationen enthalten:

```
<!ENTITY kurs1 SYSTEM "kurs1.xml">
<!ENTITY kurs2 SYSTEM "kurs2.xml">
...
```

Die Dokumentinstanz dazu wäre:

```
<kursprogramm>
  &kurs1;
```

```
  &kurs2;
  ...
</kursprogramm>
```

3.9.3 Notationen und ungeparste Entitäten

Das Datenformat XML ist in erster Linie für Textdaten konzipiert. XML bietet überdies Möglichkeiten, innerhalb von XML-Dokumenten auch andere Datenformate einzubinden, etwa grafische Formate, Videos oder Sounds. Bei der Auflistung der verschiedenen Attributtypen wurde die Notation bereits als ein möglicher Typ aufgeführt. Dieser Typ kann aber erst verwendet werden, wenn eine dazugehörige Notationsdeklaration stattgefunden hat. Die allgemeine Syntax ist:

```
<!NOTATION name SYSTEM uri>
```

oder

```
<!NOTATION name PUBLIC fpi uri>
```

Diese Deklaration gibt dem Parser gewissermaßen Bescheid, dass es sich bei einem bestimmten Format um etwas anderes als XML-Daten handelt, benennt also das XML-fremde Format oder gibt Hinweise auf die Anwendung, die mit den Daten etwas anfangen kann. Zu diesem Zweck wird diesen Daten ein bestimmter Name zugeordnet, der anschließend in Entitäten oder Attributlisten verwendet werden kann.

Wie können nun Entitäten mit Verweisen auf externe Datenformate aussehen, die nicht XML-konform sind? Ungeparste Entitäten, also Entitäten, die der Prozessor nicht parsen soll, werden im Prinzip ähnlich deklariert wie allgemeine externe Entitäten. Allerdings kommt eine Ergänzung hinzu, die über das Schlüsselwort `NDATA` einen vorher deklarierten Notationstyp benennt.

Um beispielsweise in das Kursprogramm-Dokument Porträts der Referenten mit aufzunehmen, kann das Element `<bild>` so deklariert werden:

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ENTITY hanna SYSTEM "hanna.jpg" NDATA jpeg>
<!ELEMENT bild EMPTY>
<!ATTLIST bild quelle ENTITY #IMPLIED>
```

Im Dokument wird

```
<bild quelle="hanna"/>
```

eingetragen, um einen Verweis auf die ungeparste Entität zu setzen, und zwar als Wert des Attributs `quelle`. Das Element `<bild>` ist zwar ohne Inhalt, also leer, hat aber ein Attribut, was durchaus erlaubt ist.

3.9.4 Verwendung von Parameterentitäten

Im Unterschied zu den bisher behandelten allgemeinen Entitäten werden die sogenannten Parameterentitäten nur innerhalb einer DTD benutzt, sie spielen innerhalb der Dokumentinstanz also keine Rolle. Es gibt keine Verweise auf diese Entitäten im Dokument. Parameterentitäten können sowohl innerhalb der internen als auch der externen Teilmenge verwendet werden.

3.9.5 Interne Parameterentitäten

Innerhalb einer internen DTD werden Parameterentitäten verwendet, um mehrfach vorkommende Elementgruppen oder Attributlisten jeweils nur einmal definieren zu müssen. Das erspart nicht nur Schreibarbeit, sondern erleichtert auch die Pflege eines Dokumentmodells, weil notwendige Änderungen immer nur dort vorgenommen werden müssen, wo die Parameterentität deklariert wird. Um Parameterentitäten von allgemeinen Entitäten zu unterscheiden, wird ein %-Zeichen vor den Entitätsnamen gesetzt:

```
<!ENTITY  % name "Ersetzungstext">
```

Auch beim Verweis auf eine Parameterentität wird das %-Zeichen verwendet. Ein einfaches Beispiel ist eine Parameterentität für eine mehrfach benötigte Attributdefinition:

```
<!ENTITY % id "id ID #REQUIRED">
<!ATTLIST kurs
  %id;
...>
<!ATTLIST referent
  %id;
...>
```

Praktisch sind solche Referenzen auch, wenn mehrfach bestimmte Aufzählungslisten benötigt werden, etwa Monats- oder Tagesnamen.

3.9.6 Externe Parameterentitäten

In vielen Fällen ist es praktisch, DTDs in kleinere Module zu zerlegen, um sie dann nach Bedarf zu kombinieren. Wenn zum Beispiel in verschiedenen Dokumenten immer eine bestimmte Form der Aufbereitung von Adressdaten benötigt wird, sollten Sie ein solches DTD-Modul in einer separaten Datei ablegen. Es ist sinnvoll, dafür den Dateityp *.mod* zu verwenden.

Um in einer Elementtyp-Deklaration Referenzen auf Parameterentitäten zu nutzen, verwenden Sie folgende Syntax:

```
<!ENTITY % name SYSTEM uri>
<!ENTITY % name PUBLIC fip uri>
```

Unsere DTD zum Kursprogramm könnte beispielsweise einen Verweis auf eine Datei enthalten, die die Elemente und Attribute für den einzelnen Kurs beschreibt. Die DTD für das Kursprogramm übernimmt diese Deklaration auf folgende Weise:

```
<!ENTITY % kurs SYSTEM "kurs.mod">
<!ELEMENT kursprogramm(kurs+)>
%kurs;
```

3.10 Formen der DTD-Deklaration

Die Dokumenttyp-Deklaration, die im Prolog eines XML-Dokuments erscheinen muss, kann unterschiedliche Formen annehmen. Die Syntaxvarianten sehen so aus:

- `<!DOCTYPE wurzelelementname [DTD]>`
  gilt für eine interne DTD.
- `<!DOCTYPE wurzelelementname SYSTEM uri>`
  gilt für eine externe DTD, die als private DTD verwendet werden soll.
- `<!DOCTYPE wurzelelementname SYSTEM uri [DTD]>`
  ergänzt eine externe, private DTD durch eine interne DTD.
- `<!DOCTYPE wurzelelementname PUBLIC fpi uri>`
  gilt für eine externe DTD, die als öffentlich zugängliche DTD verwendet werden soll.
- `<!DOCTYPE wurzelelementname PUBLIC fpi uri [DTD]>`
  ergänzt eine externe, öffentliche DTD durch eine interne DTD.

3.10.1 Öffentliche und private DTDs

Der Vorteil von externen DTDs liegt generell darin, dass sie für beliebig viele Dokumentinstanzen verwendet werden können. Private DTDs werden mit dem Schlüsselwort `SYSTEM` spezifiziert. Der URI gibt an, wo sich die DTD im Web oder lokal befindet.

Ist eine DTD für die allgemeine Verwendung freigegeben, wird innerhalb der Dokumenttyp-Deklaration das Schlüsselwort `PUBLIC` verwendet, bevor der URI angegeben wird, unter dem die DTD verfügbar ist. Zusätzlich muss ein *Formal Public Identifier* verwendet werden, der aus vier Feldern besteht, die durch // getrennt werden.


```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML Basic 1.0//EN"
"http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
```

ist zum Beispiel die Dokumenttyp-Deklaration für ein XHTML-Dokument.

Das erste Feld gibt mit – an, dass es sich um eine nichtregistrierte Organisation handelt; + gilt für registrierte Organisationen. Das zweite Feld bestimmt die für die DTD verantwortliche Gruppe (hier das W3C). Im dritten Feld steht die `public text class`, in diesem Fall also immer `DTD`, und der eindeutige Namen für den öffentlichen Text, die `public text description`. Das letzte Feld definiert die Sprache, die die DTD verwendet. Dabei wird der Code ISO 639 verwendet, der die Sprachen mit zwei Großbuchstaben kennzeichnet, in diesem Fall `EN` für Englisch. Solche DTDs werden auch *XML-Anwendungen* genannt.

3.10.2 Kombination von externen und internen DTDs

Wie die aufgeführten Dokumenttyp-Deklarationen zeigen, lassen sich externe und interne DTDs durchaus kombinieren. Das kann zum Beispiel sinnvoll sein, um eine externe DTD durch interne Deklarationen zu erweitern oder um Deklarationen in einer externen DTD für bestimmte Dokumente zu überschreiben.

Generell unterscheidet die XML-Spezifikation bei einer DTD zwischen einer internen und einer externen Teilmenge. Wird sowohl die interne als auch die externe Teilmenge benutzt, hat die interne Teilmenge Vorrang. In dem folgenden Beispiel wird zu dem in der externen DTD definierten Element `<kurs>` intern noch ein Attribut `typ` hinzugefügt.

```
<!DOCTYPE kursprogramm SYSTEM "kursprogramm.dtd"
[
<!ATTLIST kurs typ CDATA #REQUIRED>
]>
```

Während in der Datei *kursprogramm.dtd* das Element `<kurs>` ohne Attribut genutzt wird, fordert die interne Teilmenge für die aktuelle Dokumentinstanz ein zusätzliches Attribut.

Das einfache Überschreiben einer Elementdeklaration durch eine neue Elementdeklaration lassen die XML-Prozessoren allerdings in der Regel nicht zu.

3.10.3 Bedingte Abschnitte in externen DTDs

An dieser Stelle sei noch kurz auf einen einfachen Mechanismus hingewiesen, der es erlaubt, bestimmte Abschnitte einer DTD wahlweise ein- und auszuschalten. Das

kann sowohl in der Entwicklungsphase sinnvoll sein als auch generell bei großen, bausteinartig aufgebauten DTDs. Dies Verfahren kann allerdings nur bei externen DTDs angewandt werden.

Sie können einen bestimmten Abschnitt einer externen DTD mit einem speziellen Markup kennzeichnen, so dass dieser Abschnitt durch manuelles Einfügen des Schlüsselworts `INCLUDE` in Kraft gesetzt oder mit `IGNORE` außer Kraft gesetzt werden kann.

```
<![INCLUDE[
<!-- diese Elemente einfügen -->
<!ELEMENT kommentar ANY>
]]>
<![IGNORE[
<!-- diese Elemente ignorieren -->
<!ELEMENT bildkommentar (#PCDATA)>
]]>
```

Wenn Sie in der DTD vorweg Parameterentitäten für die Schlüsselwörter deklarieren, haben Sie eine einfache Möglichkeit, bedingte Abschnitte an- oder abzuschalten, indem Sie eine entsprechende Entitätsreferenz im Dokument verwenden.

```
<!ENTITY % optional "INCLUDE">
<![%optional;[
<!ELEMENT anhang ANY>
]]>
```

In diesem Fall wird der Prozessor erst die Referenz auf die Parameterentität auflösen. Wenn Sie im XML-Dokument dann den Wert der Parameterentität innerhalb der internen Teilmenge der DTD überschreiben, kann der bedingte Abschnitt bei der Verarbeitung ausgeblendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kursprogramm SYSTEM "kursprogramm_attr.dtd"
[!ENTITY % optional "IGNORE">]>
```

Allerdings müssen Sie darauf achten, dass das Dokument in beiden Fällen ein wohlgeformtes Dokument bleibt. Sie können daher nicht die Deklarationen für Elemente ausblenden, die als Kindelemente von anderen Elementen aufgelistet wurden.

3.11 Zwei DTDs in der Praxis

Zur Illustration der Rolle, die die DTDs in der XML-Welt spielen, werden in diesem Abschnitt zwei XML-Anwendungen etwas ausführlicher vorgestellt, die auf der Basis von DTDs arbeiten.

3.11.1 Die Auszeichnungssprache DocBook

Die eine der häufig eingesetzten praktischen Anwendungen von XML ist das Dokumentenformat *DocBook*. Die sehr mächtige Auszeichnungssprache wurde zunächst als DTD entwickelt, die erste Version bereits 1991 für SGML-Dokumente. Bis zur Version 4.5 blieb die DTD maßgeblich, Schemata wie XML-Schema, RELAX NG und Schematron wurden davon abgeleitet. Bei der Version 5 ist es umgekehrt, die DTD ist die abgeleitete Variante eines RELAX NG-Schemas.

Der Standard wird seit Jahren von OASIS gepflegt und zielt hauptsächlich auf die Erstellung von technischen Dokumentationen. Er ist aber im Prinzip für alle Formen von Veröffentlichungen geeignet, bei denen der Text vorrangig ist. Sie finden die aktuellen Spezifikationen unter *docbook.org/specs/*.

Die Spezifikation definiert inzwischen mehrere Hundert Elemente, sodass so ziemlich alles, was in einer Publikation vorkommen kann, abgedeckt ist.

Wenn Sie die DocBook-DTD in ein XML-Dokument einbinden wollen, benutzen Sie am Anfang eine entsprechende DOCTYPE-Deklaration, beispielsweise:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
  "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
```

Die Dokumenttypdefinition *docbookx.dtd* liest fünf Module ein, in denen Elemente, Attribute und Entitäten für verschiedene Bereiche definiert sind. Die Kernelemente finden Sie in der Datei *dbhierx.mod*. Das folgende Listing zeigt die Definition des Elements `<book>`.

```
<!-- Book and BookInfo ..................................................... -->

<!ENTITY % book.content.module "INCLUDE">
<![%book.content.module;[
<!ENTITY % book.module "INCLUDE">
<![%book.module;[

<!ENTITY % local.book.attrib "">
<!ENTITY % book.role.attrib "%role.attrib;">

<!ENTITY % book.element "INCLUDE">
<![%book.element;[
<!--doc:A book.-->
<!ELEMENT book %ho; ((%div.title.content;)?, bookinfo?,
    (dedication | toc | lot
    | glossary | bibliography | preface
    | %chapter.class; | reference | part
    | %article.class;
```

```
      | %appendix.class;
      | %index.class;
      | colophon)*)
     %ubiq.inclusion;>
<!--end of book.element-->]]>
```

Die Moduldatei *dbnotnx.mod* deklariert Notationen, *dbcentx.mod* deklariert Zeichenentitäten, *dbpoolx.mod* definiert Inline-Elemente; *dbgenent.mod* ist der Ort für zusätzliche allgemeine Entitäten, allgemeingültige Texte oder Verweise auf Bilder. Dieses Modul ist als Vorgabe leer.

Tools wie XMLSpy von Altova unterstützen Sie beim Editieren von DocBook-Dokumenten. Wenn Sie den Dialog DATEI • NEU öffnen, werden die Dokumenttypen XML DOCBOOK 4.5 BOOK und DOCBOOK 4.5 ARTICLE angeboten.

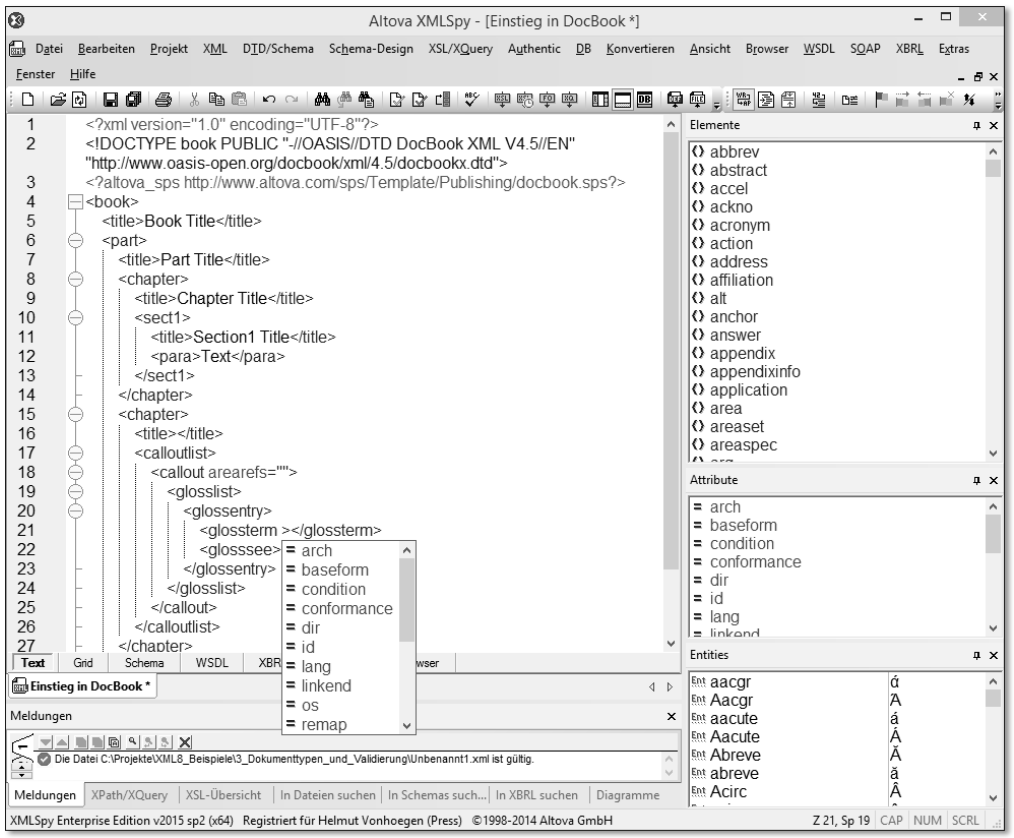


Abbildung 3.7 Skelett für ein neues DocBook-Dokument in XMLSpy

Die Abbildung zeigt die ersten Elemente für ein Buch mit dem DOCTYPE von OASIS in der Version 4.5. Das Wurzelelement ist in diesem Fall `<book>`, im anderen Fall

`<article>`. Nach einem `<title>`-Element für das gesamte Buch werden `<part>`-Elemente angeboten und darin jeweils `<chapter>` für die einzelnen Kapitel.

Die verschiedenen Abschnitte des Buches werden in `<sect1>`, `<sect2>`, `<sect3>` ... -Elemente eingefasst. Dabei wird jedes Mal ein `<title>`-Element für die Überschrift und `<para>`-Elemente für die Inhalte eines Abschnitts verwendet.

XMLSpy bietet auf der Basis der verwendeten DTD in dem Fenster ELEMENTE alle erlaubten Elemente zum Einfügen an. Wenn Sie beispielsweise ein neues Element für ein Kapitel einfügen, generiert XMLSpy bereits die vermuteten Kindelemente dazu.

Auch die Attribute zu den einzelnen Elementen werden Ihnen zur Auswahl angeboten, wenn Sie am Ende eines Startelements ein Leerzeichen eingeben.

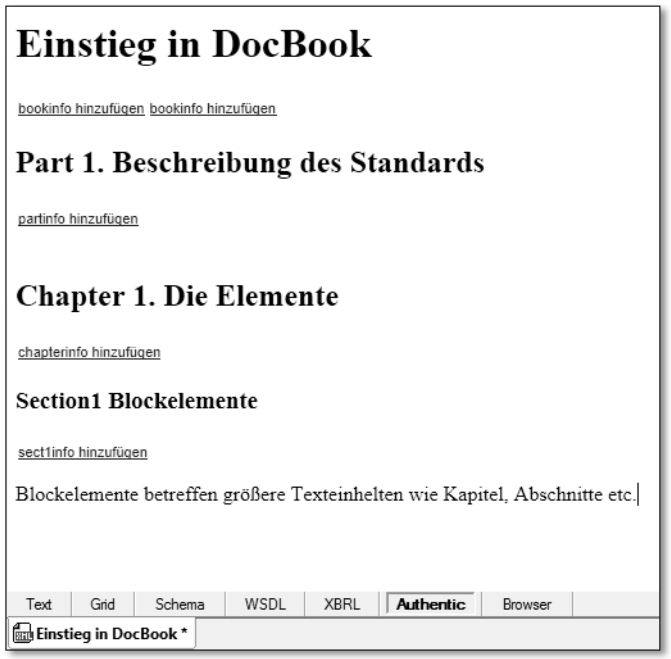


Abbildung 3.8 DocBook-Dokument in der Authentic-Ansicht

Um die Texte bequemer einzugeben, generiert XMLSpy automatisch eine entsprechende Eingabemaske auf dem Register AUTHENTIC, wo Sie dann die vorgegebenen Platzhalter überschreiben können.

3.11.2 Das grafische Format SVG

Die andere der schon erwähnten praktischen Anwendungen von XML ist das Grafikformat *SVG*. Dieses »grafische XML« demonstriert eindrucksvoll die Leistungsfähig-

keit von DTDs und wird als eine der bevorzugten Techniken für die Präsentation grafischer Inhalte auf dem Handy oder Smartphone oder für Grafikanwendungen im Web verwendet.

SVG beschreibt Vektorgrafiken in purem XML-Code, also als bandbreitenschonende Textdatei, und zwar nicht nur die grafischen Grundelemente, sondern auch alle komplizierteren Elemente wie Farbverläufe, Animationen und Filtereffekte. Da SVG außerdem mit dem Dokumentobjektmodell konform geht, kann über Skripts auf die einzelnen Elemente einer Grafik zugegriffen werden, so dass auch dynamische Grafiken möglich werden, etwa animierte Diagramme, um nur ein Beispiel zu nennen.

Seit September 2001 ist die *Scalable Vector Graphics* (*SVG*) *1.0 Specification* vom W3C verabschiedet. *SVG 1.1* folgte im Januar 2003, die Second Edition erschien im August 2011. In *SVG 1.1* ist eine Modularisierung des Standards ausgeführt, die es erlaubt, bestimmte Untermengen von SVG für die Darstellung von Grafiken auf einem Handy oder einem PDA zu verwenden. 2008 wurde noch die Empfehlung *SVG Tiny, Version 1.2* verabschiedet, die eine Untermenge von SVG 1.1 mit einigen Neuerungen kombiniert, gedacht für Anwendungen, die sich auf ganz unterschiedlichen Geräten implementieren lassen, vom Handy bis zum PC.

SVG 1.1 ist eine per DTD definierte Sprache zur Darstellung zweidimensionaler Grafiken, wobei innerhalb einer SVG-Grafik neben den Vektorgrafiken, die die Sprache hauptsächlich beschreibt, auch Bitmaps und Texte eingebunden werden können. Die Vektorgrafiken lassen sich frei skalieren. Die Grafiken lassen sich animieren und mit Hilfe von Skripts interaktiv gestalten.

Mit der Einführung von HTML5 hat SVG eine Art Wiederbelebung erfahren, unter anderem auch, weil SVG-Code jetzt direkt in den Code einer HTML-Seite integriert werden kann. Darauf komme ich in dem neuen Kapitel 15 noch einmal zu sprechen.

Kleines Animationsbeispiel

Abbildung 3.9 zeigt als Beispiel eine kleine Animation, die mit Hilfe von *Inkscape* erzeugt worden ist, einem freien Editor für SVG, der über *www.inkscape.org* angeboten wird. Für erste Versuche kann auch der Online SVG-Editor verwendet werden, der von Google über *code.google.com/p/svg-edit* angeboten wird.

In *Inkscape* werden die interaktiven Designfunktionen, die normalerweise ein Grafikprogramm zur Verfügung stellt, ergänzt durch die Möglichkeit, in einem eigenen Fenster direkt in den SVG-Code einzugreifen. In diesem Fenster steht immer das aktuelle Objektmodell zur Verfügung, das während des Editierens aufgebaut wird.

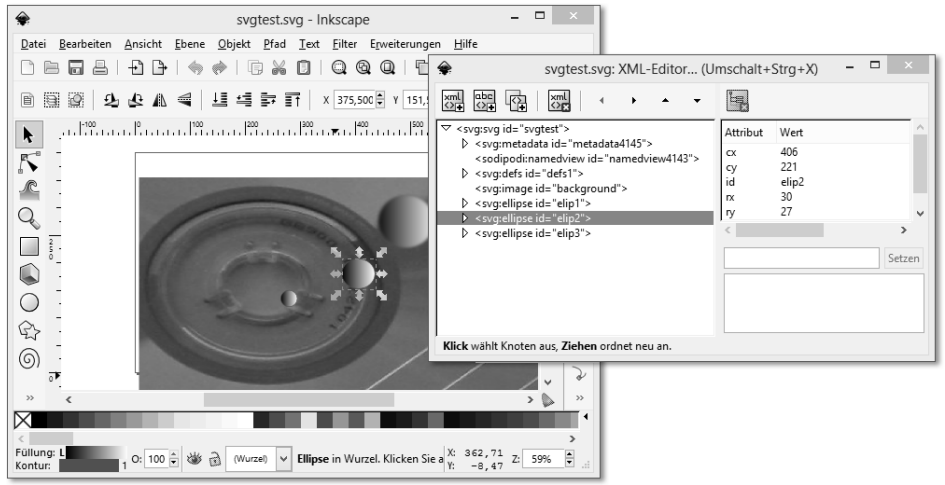


Abbildung 3.9 SVG-Entwicklung mit Inkscape

Das SVG-Format lässt sich sowohl in Webseiten einbinden – etwa mit einem `<object>`-Tag – als auch direkt im Browser darstellen. Opera verfügt über einen integrierten Viewer, der in Abbildung 3.10 verwendet wird. Firefox unterstützt seit Version 4.0 auch SVG-Animationen. Für den Internet Explorer musste in den älteren Versionen ein Viewer installiert sein wie der SVG-Viewer von Adobe; der IE ab Version 9 unterstützt SVG direkt.

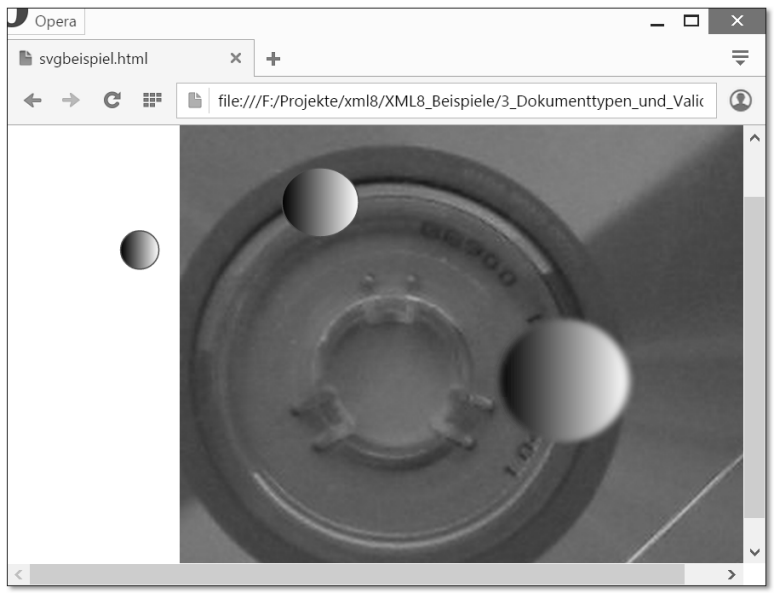


Abbildung 3.10 Anzeige der SVG-Animation im Opera-Browser

Adobe hat allerdings 2009 die Unterstützung des eigenen Viewers mit der Begründung eingestellt, dass die aktuellen Browser inzwischen eigene Implementierungen anbieten. Mehr darüber erfahren Sie unter *www.adobe.com/devnet/svg.html*. Adobes Illustrator-Programm unterstützt SVG als Export-Format, Adobe InDesign wiederum kann SVG-Grafiken importieren. Auch Microsoft Visio erlaubt den Export von Illustrationen in SVG.

Aufbau eines SVG-Dokuments

Das Wurzelelement ist immer das Element `<svg>`. Die Attribute `width` und `height` legen die Gesamtgröße der Grafik fest. Um Verweise auf grafische Elemente einbauen zu können, wird *XLink* benutzt, eine Erweiterung der Hyperlink-Technik von HTML, mehr dazu in Kapitel 5, »Navigation und Verknüpfung«. SVG erlaubt es so, eine Grafik aus verschiedenen grafischen Bausteinen zu montieren. Grafiken werden auf diese Weise automatisch aktualisiert, wenn sich eine über einen Link eingebundene Komponente ändert, was sehr flexible Lösungen erlaubt.

Über `<filter>`-Elemente lassen sich grafische Filter einbinden. Für die grafischen Grundformen stehen entsprechende Elemente wie `<ellipse>`, `<circle>`, `<rect>`, `<line>`, `<polyline>` und `<polygon>` zur Verfügung, wobei die Details über Attribute geregelt werden.

Um Objekte zu animieren, werden zu dem Element der Grundform Kindelemente vom Typ `<animate>` oder `<animateTransform>` eingebaut, die die Bewegung des Objekts und die Dauer der Bewegung wiederum über Attribute festlegen.

Über die `DOCTYPE`-Deklaration muss die DTD des W3C eingebunden werden.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg id="svgtest" width="600" height="400">
  <defs id="defs1">
    <filter id="Gaussian_Blur" filterUnits="objectBoundingBox"
     x="-10%" y="-10%" width="150%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3 2"/>
    </filter>
    <linearGradient id="black-white" x1="0%" y1="0%" x2="100%"
     y2="0%" spreadMethod="pad" gradientUnits="objectBoundingBox">
      <stop offset="0%" style="stop-color:rgb(0,0,0);
       stop-opacity:1"/>
      <stop offset="100%" style="stop-color:rgb(255,255,255);
       stop-opacity:1"/>
    </linearGradient>
  </defs>
```



```
<image id="background" x="7" y="45" width="582" height="300"
 xlink:href="CD_Background.jpg"/>
<ellipse id="elip1" cx="493" cy="126" rx="51" ry="48"
 style="fill:url(#black-white);stroke:rgb(170,42,42);
 stroke-width:1;filter:url(#Gaussian_Blur);">
  <animateTransform attributeName="transform" begin="0s"
   dur="3.4s" fill="freeze" calcMode="linear" from="0 0"
   to="-65.3846 119.231" type="translate" additive="sum"/>
</ellipse>
<ellipse id="elip2" cx="406" cy="221" rx="30" ry="27"
 style="fill:url(#black-white);stroke:rgb(170,42,42);
 stroke-width:1">
  <animateTransform attributeName="transform" begin="0s"
   dur="2.9s" fill="freeze" calcMode="linear" from="0 0"
   to="-169.231 -115.385" type="translate" additive="sum"/>
</ellipse>
<ellipse id="elip3" cx="280" cy="266" rx="15" ry="15"
 style="fill:url(#black-white);stroke:rgb(170,42,42);
 stroke-width:1">
  <animateTransform attributeName="transform" begin="0s"
   dur="4.1s" fill="freeze" calcMode="linear" from="0 0"
   to="-184.615 -123.077" type="translate" additive="sum"/>
  </ellipse>
</svg>
```

Listing 3.5 svgbeispiel.svg

Auf einen Blick

Inhalt

Index

E

F

G

H

I

J

K

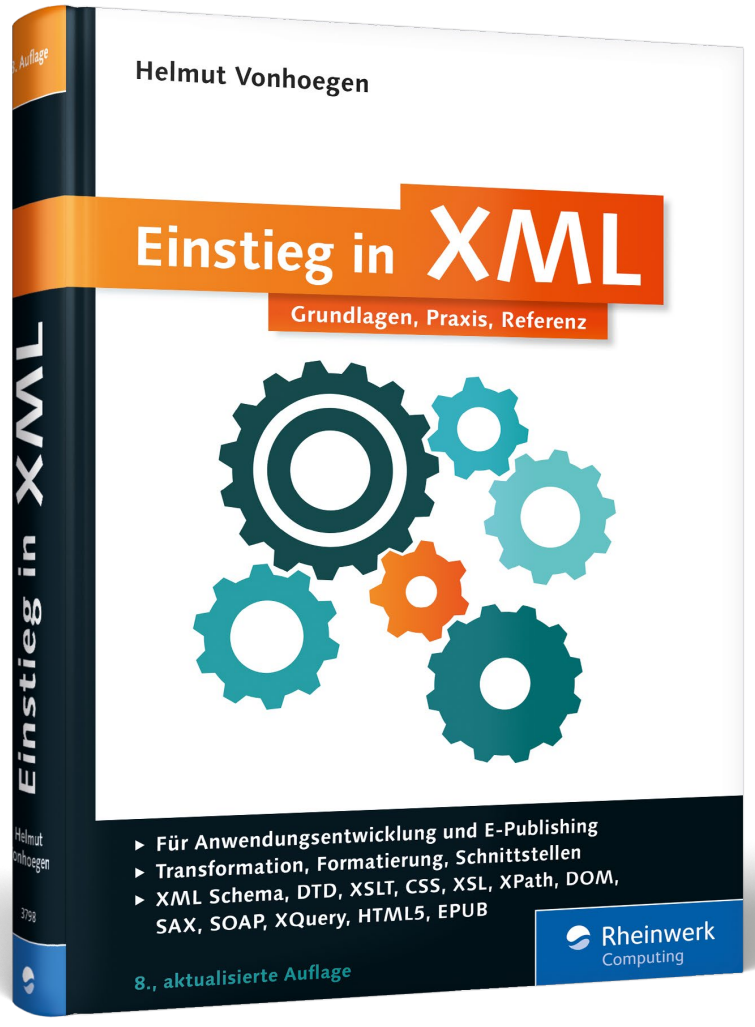
L

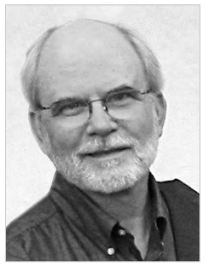
M

X

Y

Z





Helmut Vonhoegen arbeitet seit 1992 als freischaffender Autor und IT-Berater. Seine Spezialgebiete sind Windows, Office-Anwendungen, XML und Webprogrammierung.

Helmut Vonhoegen

Einstieg in XML – Grundlagen, Praxis, Referenz

615 Seiten, gebunden, 8. Auflage 2015
39,90 Euro, ISBN 978-3-8362-3798-7

www.rheinwerk-verlag.de/3876