Helmut Vonhoegen

Einstieg in XML

Galileo Computing

Inhalt

Vorwort

Werden die Technologien, die sich unter dem Namen **XML** gruppieren lassen, mit einem Gebäudekomplex verglichen, so hat es sich dabei einige Jahre lang eher um eine Baustelle gehandelt. Die Fundamente waren schon gelegt und zwei, drei erste Aufbauten bezugsfertig. Inzwischen aber sind die Hauptgebäude hochgezogen, der Kern des Komplexes steht und es geht in Zukunft nur noch um diese oder jene Anbauten. Allerdings werden auch schon Stimmen laut, wie etwa die von einem der Mitarchitekten der XML-Welt, James Clark, der auf der IDEAlliance XML 2001 Konferenz in Orlando bereits eine Grundrenovierung des Ganzen ins Gespräch gebracht hat. Dabei geht es darum, die im Zentrum von XML stehenden Spezifikationen, die ja zeitversetzt entstanden sind, besser aufeinander abzustimmen.

Einige Zeit war auch nicht so ganz sicher, ob sich XML tatsächlich in der IT-Industrie etablieren würde. Einige Szene-Päpste und -Evangelisten schrieben dicke Wälzer, um die Branche von den Vorzügen des Projekts zu überzeugen. Heute ist die Frage entschieden. XML ist eine anerkannte Basistechnologie und es geht nur noch um das Wie und Wie viel, was den Einsatz betrifft. Die Nebel haben sich also gelichtet. Das Objekt kann mit nüchternem Blick inspiziert werden.

Das Buch richtet sich in erster Linie an alle, die mit XML und den damit verbundenen Sprachen und Werkzeugen arbeiten oder sie erlernen wollen, um damit entsprechende Anwendungen zu realisieren. Es gibt ihnen eine fundierte Basis für ihre Aktivitäten rund um XML. Die Darstellung wird dabei jeweils durch nachvollziehbare praktische Beispiele vertieft und erprobt.

Anstelle einer Batterie von oft weitschweifigen Büchern im XXL-Format zu den verschiedenen Teilstandards der Sprachfamilie XML bietet das Buch in konzentrierter Form das, was zur Entwicklung eigener XML-Lösungen benötigt wird. Ein bei Büchern über XML nicht leicht zu vermeidendes Handicap ist die Rasanz der Entwicklung in diesem Feld. Die wichtigsten Mitglieder der Sprachfamilie sind inzwischen akzeptierte Standards, andere sind aber noch in Bearbeitung. Das Buch konzentriert sich im Wesentlichen auf die fertigen Standards, zumal mit der Verabschiedung der XML Schema-Empfehlung der Kern der XML-Welt wohl fixiert ist. Was nachkommt, wird darauf aufbauen.

Helmut Vonhoegen
Köln, im September 2002

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/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 einen Blick wert. Wir wollen hier zunächst einen kurzen Überblick über die Syntax geben, verknüpft mit einem ersten 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 Dokument-Entität. 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 Dokument-Entität, die alle anderen Entitäten in sich einschließt. Für einen XML-Parser ist die Dokument-Entität, die alle anderen Entitäten umschließt, immer der Startpunkt. Liegt ein XML-Dokument als Datei vor, ist die Dokument-Entität eben diese Datei. Wenn Sie dagegen ein XML-Dokument über einen URL einfließen lassen, ist die Dokument-Entität 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 Textmakros, 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« bestehen 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.

»Unparsed data« sind 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 je 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 der logischen Sicht in Elemente zerlegt. Im Anschluss daran sind noch Kommentare oder Verarbeitungsanweisungen erlaubt, was aber in der Praxis nicht unbedingt zu empfehlen ist.

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

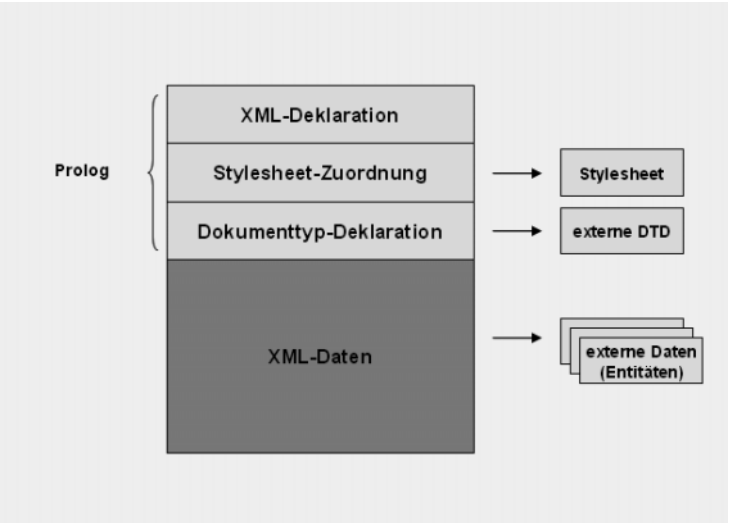

Abbildung 2.1 Aufbauschema eines XML-Dokuments

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 so genannte 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 der im Augenblick einzig gültigen Spezifikation 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 muss verwendet werden.

Neben dem Versionsattribut können in der XML-Deklaration noch zwei weitere Attribute verwendet werden, und zwar `encoding` und `standalone`. Im folgenden Beispiel wird angegeben, dass das Dokument die Zeichensatzkodierung UTF-16 verwendet und dass keine externen Markupdeklarationen 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 die 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 der **Universal Character Set** – **UCS** – als Basis für die Zeichenkodierung in XML-Dokumenten bestimmt, der 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 noch 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 so genannten 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 verwendet 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 andere 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 gegeben wird.)

Da aber Unicode noch nicht sehr verbreitet ist, werden auch andere Kodierungen unterstützt. Um sicherzustellen, dass die deutschen Umlaute korrekt dargestellt werden, kann ISO-8859-1 (ISO Latin-1) verwendet werden. Diese Einstellung wird deshalb in den meisten Beispielen in diesem Buch verwendet.

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« Element-Attributen, 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.

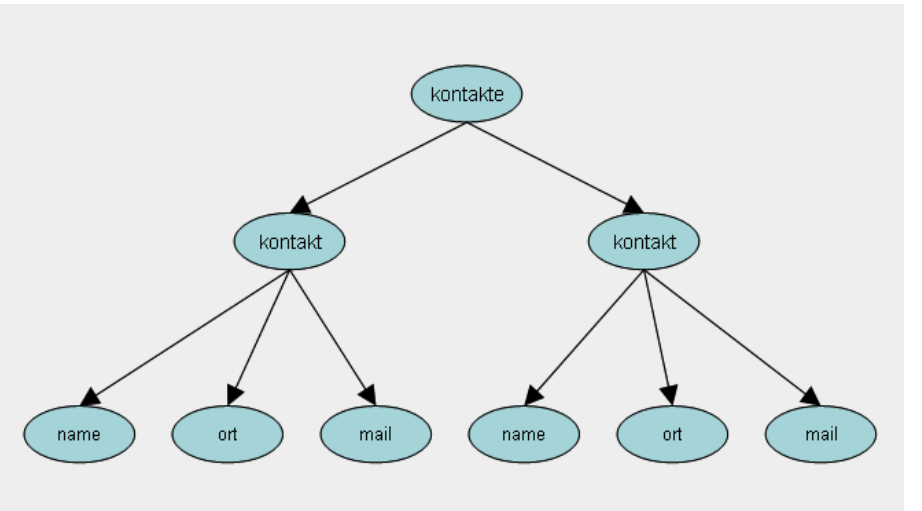

Abbildung 2.2 Baumstruktur eines Dokuments

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. 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.

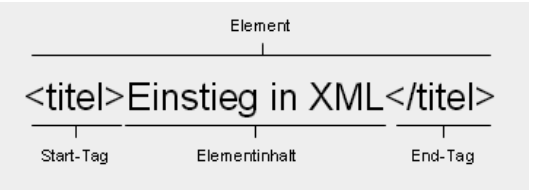

Abbildung 2.3 Ein Element in XML

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 Teil-Dokumenten zu einem Gesamtdokument.

2.1.9 Elementtypen und ihre Namen

Der 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 müssen. Die Wiederholung des Elementtypnamens im End-Tag ist notwendig, damit der Parser die Schachtelung der Elemente sofort korrekt erkennen kann.

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 einem Großbuchstaben am Anfang zu schreiben.

Ein Verstoß gegen die Regel der Wohlgeformtheit führt zu einem fatalen Fehler, d.h., ein XML-Prozessor wird die Verarbeitung des nicht wohlgeformten Dokuments 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 Webfehlern 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: statt `<leer></leer>` kann `<leer/>` verwendet werden. 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 Bin-

destrich 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 2.6 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.
- Die Länge der Namen ist nicht begrenzt, es sollte aber beachtet werden, dass möglicherweise Anwendungen, die auf die Daten zugreifen, die Länge einschränken.
- XML-Namen sind fallsensitiv. `<Name>...</name>` ist also beispielsweise nicht zulässig.

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 festgelegt 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 in sich 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 dann 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 aber 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 das folgende Beispiel zeigt:

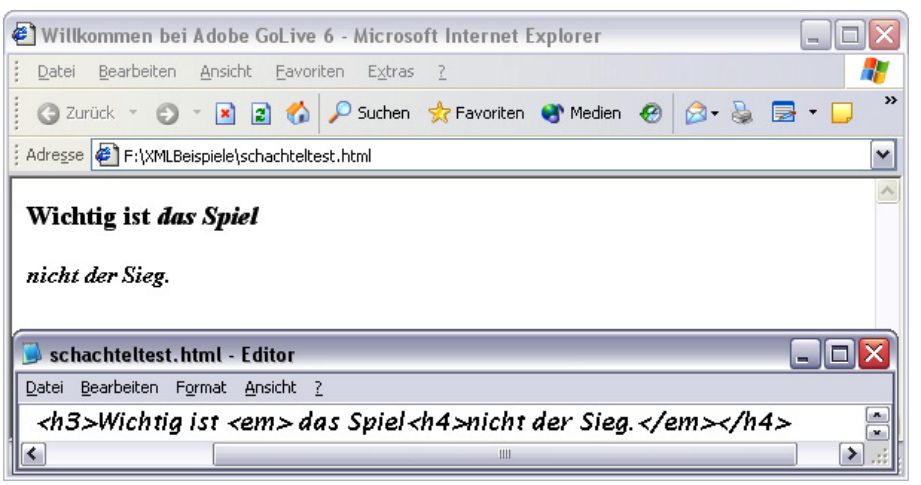

Abbildung 2.4 Der Internet Explorer verdaut die fehlerhaft geschachtelten HTML-Elemente.

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 Wohlgeformtheitskriterium verweigern.

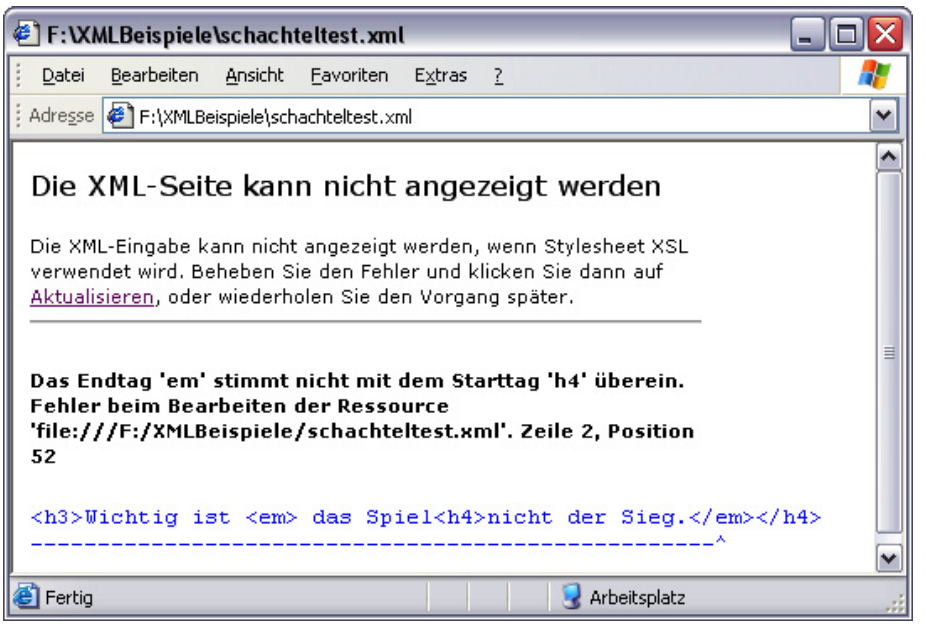

Abbildung 2.5 Reaktion des Internet Explorers auf nicht wohlgeformtes XML

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

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. Diese Attribute können verwendet werden, 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.

```
<haus nr="22“ nr=“33“>
```

wird also nicht zugelassen, macht 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 verwenden, 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, doppelte Anführungszeichen oder mit dem Apostroph eingeschlossen werden. Die Werte selbst dürfen nur Literale sein. Innerhalb des Literals dürfen die Markupzeichen »<", ">« und »&« nicht verwendet werden. Sie sind durch `<`, `>` und `&`, also durch so genannte Entitätsreferenzen zu maskieren. Dazu weiter unten gleich mehr.

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.

```
<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 zunächst ungeprüften Daten korrekt verarbeitet werden können. 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, das 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.

```
<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 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 separierbare Einheiten in einem Gegenstandsbereich geht, sollte allerdings normalerweise mit Elementen gearbeitet werden, zumal dadurch der Aufbau von Links auf diese Einheiten ermöglicht wird, etwa durch ID- und IDREF-Attribute in einem XSLT-Stylesheet.

2.4 Reservierte Attribute

Durch die XML-Spezifikation werden zwei Attribute vorgegeben, die in jedem Element verwendet werden können. Das eine – **xml:lang** – wird zur Identifikation der Sprache verwendet, 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 die zweistelligen Ländercodes verwendet, die durch **ISO 639** genormt sind. Auch Subcodes für regionale Sprachen sind möglich, etwa `"en-US"` oder `"en-GB"`.

Die zweite Möglichkeit sind Identifizierer, die bei der Internet Assigned Numbers Authority – **IANA** – registriert sind. Diese beginnen immer mit `i-` oder `I-`.

Auch eigene Codes können verwendet werden, wenn ein »x-« davor gesetzt wird.

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 genau so erscheinen, wie eingegeben.

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

Es hängt allerdings von der Anwendung ab, ob sie diesem Hinweis folgt oder in diesem Fall die Leerzeilen einfach entfernt.

2.5 Entitäten und Verweise darauf

Es ist schon angesprochen worden, dass die speziellen Zeichen, die in XML für Markups verwendet werden, nicht ohne weiteres innerhalb des Inhalts eines Elements oder innerhalb eines Attributwerts auftauchen können. 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:

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

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

Zusätzlich zu den eingebauten Entitäten können eigene Entitäten definiert werden, ähnlich wie es auch in HTML möglich ist. Dies kann innerhalb einer Dokumenttyp-Definition geschehen. Wie dies gemacht wird, ist in Abschnitt 3.4 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 auch 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 in Abschnitt 3.4.

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 so genannten 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 den 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 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 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 7, Umwandlungen mit XSLT.

2.9 Namensräume

Eine der ersten 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 gibt, wirft überall dort ein Problem auf, wo gleiche Namen mit unterschiedlichen Bedeutungen verwendet werden. Da XML-Dokumente auch dazu entworfen werden, dass sie von unterschiedlichen Programmen ausgewertet und bearbeitet werden können, können Mehrdeutigkeiten schnell zu unerwünschten Ergebnissen führen.

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 kann dies in vielen Fällen zu eher künstlichen Bezeichnungen führen 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 wieder verwendbar zu sein, lag es nahe, dafür eine andere Lösung zu suchen. Hier 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 dabei schlicht als Ansammlungen von Namen vorgestellt, die zu einem bestimmten Gegenstandsbereich gehören. Diese 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.

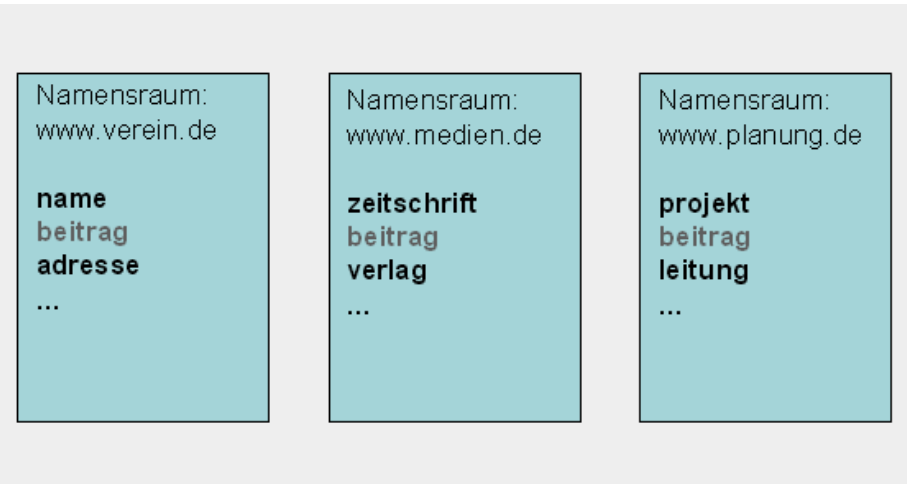

Abbildung 2.6 Gleichlautender Name in verschiedenen Namensräumen

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 wird eine solche Referenz nur als identisch betrachtet, wenn sie Zeichen für Zei-

chen gleich ist, 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.

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 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 ihn 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.

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.

Innerhalb eines XML-Dokuments werden die mit Präfix versehenen Namen allerdings 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 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. Trotzdem handelt es sich bei diesen Elementnamen um qualifizierte Namen, auch wenn das sonst verwendete Präfix gleichsam unsichtbar geworden ist. Darauf wird später 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.

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.

3 Dokumenttypen und Validierung

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

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 für die 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 Datenmodellì 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.

Da 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 bisher hauptsächlich verwendet werden, ist diese Sprache zunächst von SGML bereitgestellt worden. Für XML wurde die entsprechende Syntax übernommen. Die Spezifikation von DTD ist direkt in die Spezifikation von XML eingearbeitet worden.

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.

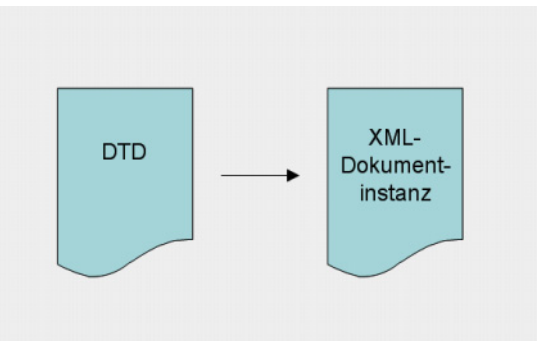

Abbildung 3.1 Ein XML-Dokument, das einer DTD entspricht, ist eine Dokumentinstanz dieser DTD

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

Inzwischen kann das Schema für ein Datenmodell aber auch in einer XML-Syntax beschrieben werden, mit Hilfe eines XML Schemas. Dieses spezielle Vokabular zur Beschreibung einer Datenstruktur wurde inzwischen vom W3C als Standard verabschiedet, sodass davon ausgegangen werden kann, dass die XML Schemas in Zukunft den Platz der DTDs übernehmen werden. Mit Hilfe von XML Schemas können präzise Angaben zu den Datentypen gemacht werden, die für die einzelnen Elemente oder Attributwerte erlaubt werden sollen.

Da aber DTDs inzwischen in zahlreichen gewichtigen Varianten für die verschiedensten Sachgebiete bereits im Einsatz sind – zwei Beispiele werden wir im Anschluss an dieses Kapitel kurz vorstellen: SVG und SMIL –, werden wir in diesem Buch beide Formen der Strukturbeschreibung von Dokumenten nacheinander behandeln, allerdings bereits mit einem deutlichen Schwerpunkt auf dem endlich verabschiedeten XML Schema-Standard.

Wer gewohnt ist, Schemas für Datenbanktabellen zu entwerfen, dem wird die Aufgabe der Datenmodellierung mit XML Schema sicher 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 MathML, WML, SVG oder SMIL, 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 die Anwendungen in dem entsprechenden Bereich mit Hilfe dieser XML-Sprache in einer Weise unterstützt werden, die die in dem Bereich tätigen Anwender nicht nur zufrieden stellt, 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 zufrieden gibt, 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 schon voraus, dass die Prüfung auf Wohlgeformtheit 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 Betreffangabe 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 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 verletzt, 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 sind, 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 heute 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 3.4.25 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="iso-8859-1"?>
<!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 ja 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="iso-8859-1"?>
<!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, d.h. 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.

Man muss 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 vier mögliche Komponenten, aus denen sich ein XML-Dokument zusammensetzen kann, festzulegen:

- 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 Element-Deklaration 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 ja im XML-Dokument der Doppelpunkt für die Trennung des Namensraumpräfix vom lokalen Namen selbst verwendet wird.

Die Länge der Namen ist nicht begrenzt. Beachtet werden muss auch wieder, dass die Namen wie bei XML üblich »fallsensitiv« verwendet werden: `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 später 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="ISO-8859-1">
<!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.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 Kommata. 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 vordefi-

niertes 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 als Inhalt 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, wie in Abschnitt 3.6 beschrieben.

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 Baums.

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 Kommata 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 geben ihre Telefonnummer an, 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, sodass 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, d.h., wenn der Referent eine Telefonnummer und E-Mail-Adresse angibt, muss zuerst die Telefonnummer angegeben 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 können:

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.

Durch eine geschickte Kombination und Verschachtelung von Elementgruppen und Operatoren für die Kardinalität von Elementen oder Elementgruppen lassen sich auch 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 Element-Deklarationen 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 dieser Inhalt sein soll und darf.

Das Element kann also beispielsweise reine Textdaten enthalten oder weitere deklarierte Kindelemente, deren Anordnung aber frei gewählt werden kann, 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 damit angewiesen, dieses Element nicht weiter auf Gültigkeit zu kontrollieren. Nur die Wohlgeformtheit muss erhalten bleiben.

Das Inhaltsmodell ANY widerspricht in gewissen Sinne dem, was mit einer DTD erreicht werden soll, aber es kann in vielen Fällen nützlich sein, 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 dann 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 ist schon erwähnt worden, 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><hin-
weis>Eine Linkliste finden Sie unter:</hinweis>
<link>www.galileo-press.de</link></anhang>
```

Allerdings gelten für dieses Inhaltsmodell bestimmte Einschränkungen, die es meist ratsam erscheinen lassen, dieses Modell zu vermeiden. Es ist nicht möglich, die Reihenfolge der Kindelemente festzulegen, die innerhalb des gemischten Inhalts auftauchen können. 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:

Inhaltsmodell	Beschreibung
`EMPTY`	Das Element hat keinen Inhalt, kann aber Attribute haben.
`ANY`	Das Element kann beliebige Inhalte enthalten, 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.

Die Reihenfolge der Elemente in der DTD ist normalerweise nicht von Bedeutung, mit der schon erwähnten Ausnahme, dass Elemente doppelt deklariert werden. Nur wenn Parameter-Entitäten verwendet werden, was weiter unten behandelt wird, 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 Werkzeug wie XML Spy meldet einen entsprechenden Fehler, wenn bei der Prüfung einer DTD ein Element, auf das bereits in einem Inhaltsmodell Bezug genommen wurde, fehlt. Ein einmal deklariertes Element kann andererseits durchaus in mehreren Elementgruppen verwendet werden, wenn dies erforderlich ist.

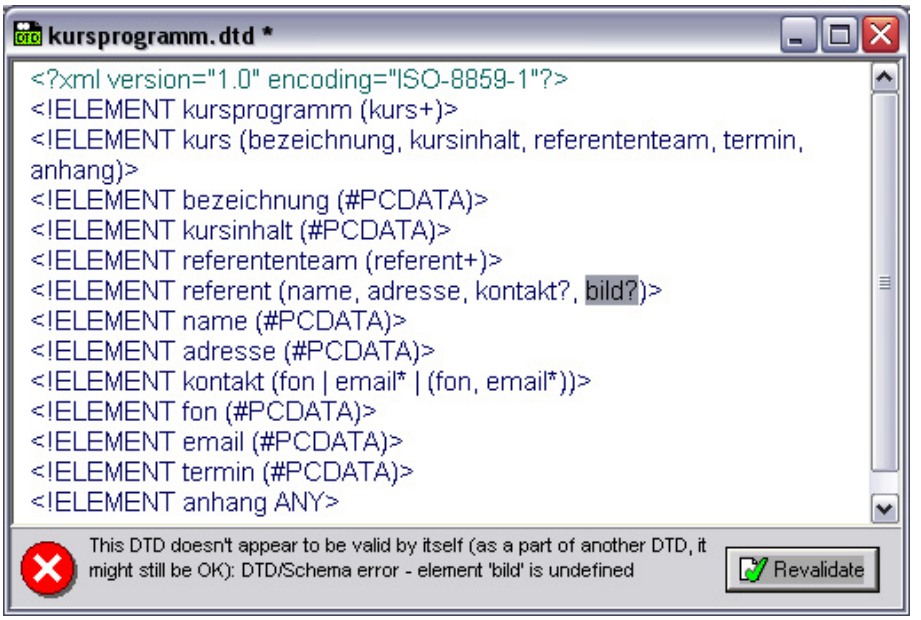

Abbildung 3.2 Meldung in XML Spy, wenn in einer DTD ein Element fehlt

Zur besseren Lesbarkeit von DTDs ist es allerdings ratsam, entweder den Elementebaum 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. Sie 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

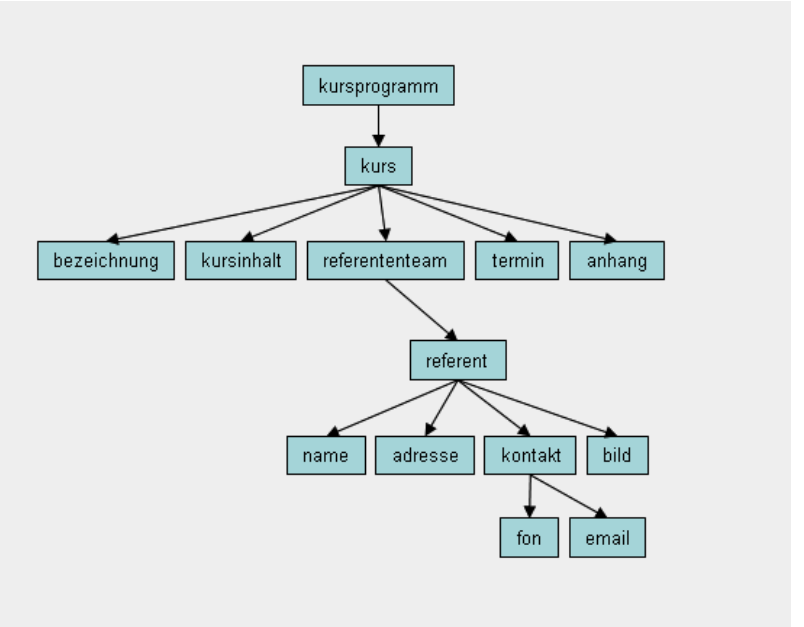

Abbildung 3.3 Die Baumstruktur des Beispiels

Beim Entwurf einer DTD ist es oft sehr hilfreich, sich die Hierarchie der Elemente in einer Baumstruktur zu vergegenwärtigen, um Fehler beim Design zu vermeiden. Die Abbildung 3.3 für unser Beispiel zeigt 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 zum Beispiel so aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!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>12.12.2001</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>6.12.2001</termin>
    <anhang>Das Seminar wird jeden Monat wiederholt.</anhang>
  </kurs>
</kursprogramm>
```

Listing 3.3 kursprogramm.xml

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 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 Kurzeinstufung 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 der besseren Lesbarkeit wegen die Attributliste direkt hinter dem Element einfügen, auf das sich die Liste bezieht. Es ist aber 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 Reihenfolge die Attribute in der Dokumentinstanz erscheinen müssen. Nur wenn ein Attributname in der Liste mehrfach verwendet wird, gilt 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 eingegeben wird.

Für die Werte von Attributen können zehn grundlegende Typen gewählt werden, 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
`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.
`Aufzählung`	In Klammern eingeschlossene Liste von Token-Werten, von denen jeweils einer als Attributwert verwendet werden kann und muss.
`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.

Für die Vorgabebehandlung bei Attributwerten stehen vier Einstellungen zur Verfügung, die Tabelle gibt einen Überblick:

Vorgabedeklaration	Beschreibung
`Attributwert`	In Anführungszeichen eingeschlossene Zeichendaten geben den vorgegebenen Wert an, der verwendet wird, wenn kein Wert angegeben wird.
`#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.

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, würde so aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kursprogramm SYSTEM "kursprogramm_attr.dtd">
<kursprogramm>
  <kurs  id="xmleinf" kurstyp="Wochenkurs"
    kurseinstufung="Einsteiger"
    sprache="EN"  >
    ...
  </kurs>
  <kurs  id="schemadtd"
    kurseinstufung="Profis"  >
    ...
```

```
  </kurs>
</kursprogramm>
```

Listing 3.4 kursprogramm_attr.xml+dtd

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 folgende Auszug zeigt.

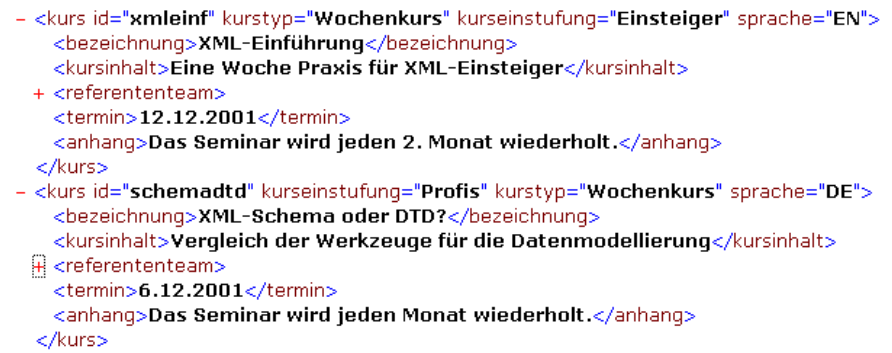

Abbildung 3.4 Die Ausgabe im Browser zeigt die vorgegebenen Werte mit an.

Beachten Sie, dass es sich bei den Werten des Typs `ID` um gültige XML-Namen handeln muss. Die vielleicht nahe liegende Idee, Zahlen als Schlüsselwerte zu verwenden, führt zu einem Fehler, weil ein XML-Name ja nicht mit einer Zahl beginnen darf. ID-Attribute sind ein wichtiges Mittel, um auch Elemente eindeutig ansprechen zu können, deren Inhalt sonst gleich ist. Davon wird in dem Abschnitt 5.4 zu XPath und XPointer noch die Rede sein.

3.8 Verweis auf andere Elemente

Mit dem Attributtyp `IDREF` können interne Verweise innerhalb eines Dokuments erstellt werden. Sie setzen voraus, dass Attribute vom Type `ID` vorhanden sind. Wenn in dem Kursprogramm zum Beispiel eingefügt werden soll, dass ein bestimmter Kurs einen anderen Kurs voraussetzt, kann folgende Deklaration helfen:

```
<!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 kommen wir noch einmal zu sprechen, nachdem gleich die Verwendung von Entitäten in der DTD behandelt worden ist.

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 bereits behandelt worden. In einer DTD können zusätzlich noch spezielle Parameter-Entitäten verwendet werden, die es erlauben, innerhalb einer DTD Verweise auf Teile einer internen oder externen DTD zu verwenden. Die Abbildung gibt einen Überblick über die verschiedenen Entitätstypen in einer DTD.

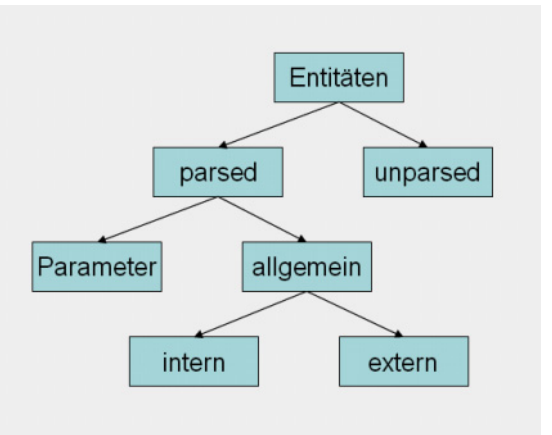

Abbildung 3.5 Typenbaum der Entitäten

3.9.1 Interne Entitäten

Am einfachsten ist die Deklaration von internen allgemeinen Entitäten. Sie können zum Beispiel verwendet werden, um Kürzel für längere Zeichenketten zu definieren, auf die dann im XML-Dokument verwiesen werden kann. Die Syntax ist:

```
<!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.

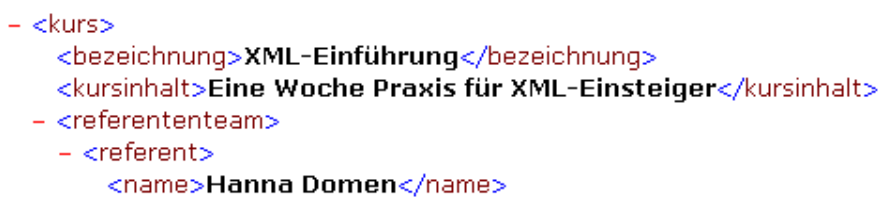

Abbildung 3.6 Der Browser zeigt den vollen Namen an, der über eine Entitätsreferenz eingegeben wurde.

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

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

3.9.2 Externe Entitäten

Wenn es sich um umfangreichere Ersetzungstexte handelt, 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.

```
<!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 der 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 dem vorhin verwendeten 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 3.7 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 aber 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 aber 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 Portraits 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 ja durchaus erlaubt ist.

3.9.4 Verwendung von Parameter-Entitäten

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

3.9.5 Interne Parameter-Entitäten

Innerhalb einer internen DTD werden Parameter-Entitäten 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 Parameter-Entität deklariert wird. Um Parameter-Entitäten von allgemeinen Entitäten zu unterscheiden, wird ein %-Zeichen vor den Entitätsnamen gesetzt:

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

Auch beim Verweis auf eine Parameter-Entität wird das %-Zeichen verwendet. Ein einfaches Beispiel ist eine Parameter-Entität 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 Parameter-Entitäten

In vielen Fällen ist es sinnvoll, DTDs in kleinere Module zu zerlegen, die dann je nach Bedarf kombiniert werden können. Wenn zum Beispiel in verschiedenen Dokumenten immer eine bestimmte Form der Aufbereitung von Adressdaten benötigt wird, kann ein solches DTD-Modul in einer separaten Datei abgelegt werden. Es ist sinnvoll, dafür den Dateityp .mod zu verwenden.

Wenn in einer Elementtyp-Deklaration Referenzen auf Parameter-Entitäten verwendet werden sollen, kann das in folgender Form geschehen:

```
<!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 kann diese Deklaration auf folgende Weise übernehmen:

```
<!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]>
```

Für eine interne DTD.

```
<!DOCTYPE wurzelelementname SYSTEM uri>
```

Für eine externe DTD, die als private DTD verwendet werden soll.

```
<!DOCTYPE wurzelelementname SYSTEM uri [DTD]>
```

Diese Deklaration ergänzt eine externe, private DTD durch eine interne DTD.

```
<!DOCTYPE wurzelelementname PUBLIC fpi uri>
```

Für eine externe DTD, die als öffentlich zugängliche DTD verwendet werden soll.

```
<!DOCTYPE wurzelelementname PUBLIC fpi uri [DTD]>
```

Diese Deklaration ergänzt eine externe, öffentliche DTD durch eine interne DTD.

3.10.1 Öffentliche und private DTDs

Der Vorteil der Verwendung 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 noch 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 gibt die für die DTD verantwortliche Gruppe an. Das dritte Feld gibt die `public text class` an, in diesem Fall also immer DTD, und den eindeutigen Namen für den öffentlichen Text, die `public text description`. Das letzte Feld gibt die Sprache an, die die DTD verwendet. Dabei wird der Code ISO 639 verwendet, der die Sprachen mit zwei Groß-buchstaben kennzeichnet. 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, wird die interne Teilmenge zuerst ausgewertet. Das hat dann zur Folge, dass beispielsweise eine interne Elementtyp-Deklaration zu einem

Element <anhang> Vorrang vor dem Element <anhang> in der externen Teilmenge hat.

```
<!DOCTYPE kursprogramm SYSTEM "kursprogramm.dtd"
[
<!ELEMENT anhang (#PCDATA)>
]>
```

Während in der Datei **kursprogramm.dtd** das Element <anhang> mit dem Inhaltsmodell ANY deklariert ist, wird hier das Element als ein einfaches Element bestimmt, das Zeichendaten enthält.

Allerdings lassen nicht alle XML-Prozessoren die gleichzeitige Deklaration von Elementen und Attributlisten in der internen und externen Teilmenge 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, sodass 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 Parameter-Entitäten für die Schlüsselworte deklarieren, haben Sie eine einfache Möglichkeit, bedingte Abschnitte dadurch an- oder abzuschalten, dass 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 Parameter-Entität auflösen. Wenn Sie im XML-Dokument dann den Wert der Parameter-Entität innerhalb der internen Teilmenge der DTD überschreiben, kann der bedingte Abschnitt bei der Verarbeitung ausgeblendet werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kursprogramm SYSTEM "kursprogramm_attr.dtd"
[!ENTITY % optional "IGNORE">]>
```

Allerdings muss darauf geachtet werden, 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 Das grafische Format SVG

Eine der schon erwähnten praktischen Anwendungen von XML, das Grafikformat **SVG**, soll an dieser Stelle etwas ausführlicher vorgestellt werden. Dies nicht nur, weil dieses »grafische XML« die Leistungsfähigkeit von DTDs eindruckvoll demonstriert, sondern auch deshalb, weil diese Anwendung gerade dabei ist, zu einer vorrangigen Technik für grafische Inhalte auf dem Handy oder dem PDA zu werden.

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.

Seit September 2001 ist die Scalable Vector Graphics (SVG) 1.0 Specification vom W3C verabschiedet. SVG 1.1 wird vermutlich verabschiedet sein, wenn dieses Buch erscheint. Darin 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.

SVG ist eine per DTD festgelegte 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.

Kleines Animationsbeispiel

Die folgende Abbildung zeigt als Beispiel eine kleine Animation, die mit Hilfe von WebDraw, einem speziellen Editor für SVG, der von Jasc angeboten wird, erzeugt worden ist.

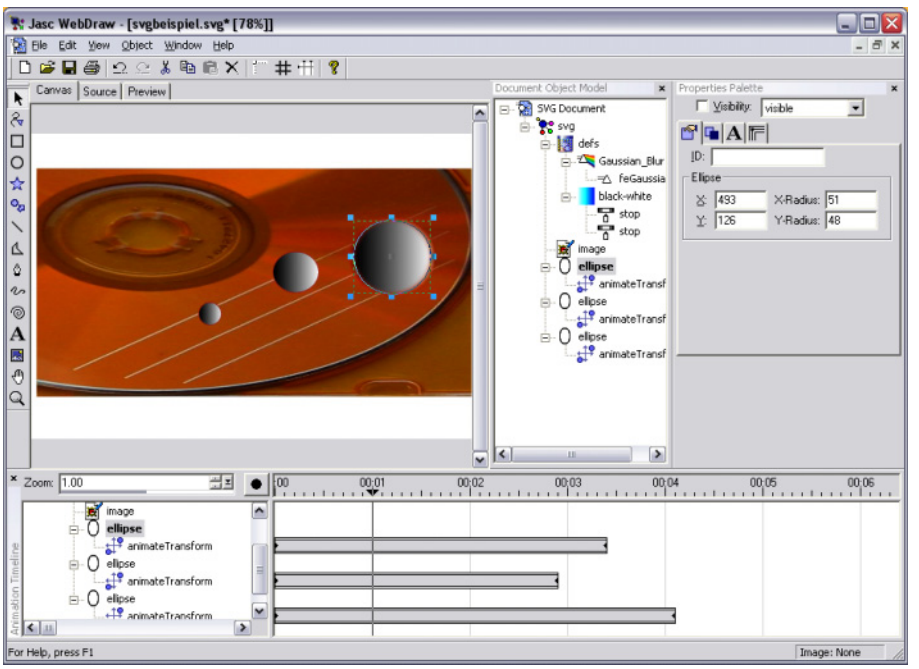

Abbildung 3.8 SVG-Entwicklung mit WebDraw

Dabei werden die Funktionen, die normalerweise ein Grafikprogramm zur Verfügung stellt, ergänzt durch die Möglichkeit, direkt in den SVG-Code einzugreifen. Animationen können sehr einfach über eine Timeline definiert werden. In einem Fenster steht immer das aktuelle Objektmodell zur Verfügung, das während des Editierens aufgebaut wird.

Das Programm kann als Testversion über **www.jasc.com** bezogen werden und befindet sich auch auf der beiliegenden CD.

Das SVG-Format lässt sich sowohl in Webseiten einbinden als auch direkt im Internet Explorer darstellen. Der Netscape Navigator verwendet ein Plug-in. In diesem Fall enthält das Diagramm als Hintergrund ein Bitmap.

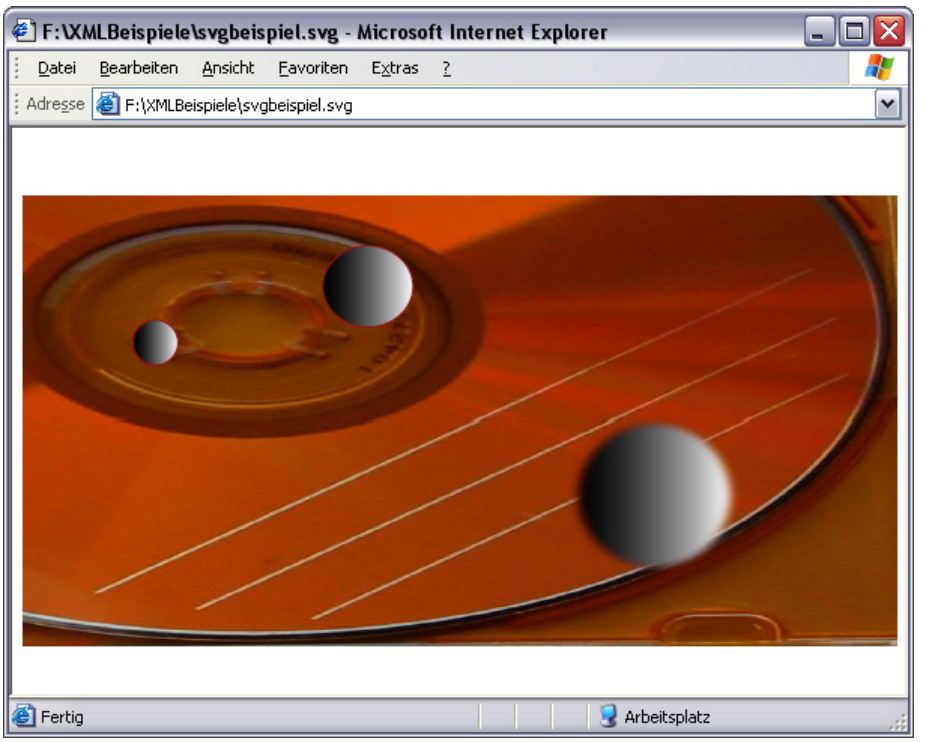

Abbildung 3.9 Beispiel für eine animierte SVG-Grafik

Aufbau des 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. 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>`, `<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> bzw. <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 width="600" height="400">
  <defs>
    <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 x="7" y="45" width="582" height="300" xlink:href="C:\
Dokumente und Einstellungen\Helmut Vonhoegen\Eigene Dateien\
Eigene Bilder\Cologne\102-0256_img.jpg"/>
  <ellipse 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 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 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

Adobe setzt sehr stark auf dieses neue Format. Über **www.adobe.com/SVG** können Sie mehr darüber erfahren. Illustrator kann ab Version 9 das Format exportieren.

3.11.2 SMIL

Bereits 1998 wurde vom W3C die Synchronized Multimedia Integration Language – **SMIL** – in einer ersten Version spezifiziert, SMIL 2.0 folgte im August 2001. Diese XML-Anwendung wurde entwickelt, um mit Hilfe einfacher Textdateien multimediale Komponenten ganz unterschiedlicher Formate in einer Timeline zu koordinieren. Auf diese Weise können interaktive Präsentationen aus Texten, Bildern, Audioklängen, Videos und auch kompletten Flash-Animationen zusammengestellt und vorgeführt werden.

Dabei können die Quelldateien über URLs eingebunden werden, sodass es ohne Umstände möglich ist, zum Beispiel zwei Videos in zwei Fenstern parallel abzuspielen, die sich auf unterschiedlichen Servern befinden. Dem Besucher können mit Hilfe eines `<switch>`-Tags zudem unterschiedliche Sprachaufzeichnungen zu einem Video oder verschiedene Bandbreiten zur Auswahl angeboten werden. Dabei ist es nicht notwendig, die einzelnen Mediendateien in einer großen Container-Datei zusammenzubinden.

Die SMIL-DTD

SMIL ist wie SVG ein XML-Vokabular, das nach den Regeln der Metasprache XML erzeugt worden ist. Der Sprachumfang ist ziemlich übersichtlich, die zugrunde liegende DTD ist nur ein paar Seiten lang. Hier ein etwas komprimierter Auszug der wichtigsten Elemente:

```
Auszug aus der DTD für SMIL
...
<!--== SMIL Document ====-->
<!-- The root element SMIL contains all other elements.
```

```
-->
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil
        %id-attr;>
<!--== The Document Head ====-->
<!ENTITY % layout-section "layout|switch">
<!ENTITY % head-element "(meta*,((%layout-section;), meta*))?">
<!ELEMENT head %head-element;>
<!ATTLIST head %id-attr;>
...
<!--== The Document Body =====-->
<!ENTITY % media-object "audio|video|text|img|animation|
textstream|ref">
<!ENTITY % schedule "par|seq|(%media-object;)">
<!ENTITY % inline-link "a">
<!ENTITY % assoc-link "anchor">
<!ENTITY % link "%inline-link;">
<!ENTITY % container-content "(%schedule;)|switch|(%link;)">
<!ENTITY % body-content "(%container-content;)">
<!ELEMENT body (%body-content;)*>
<!ATTLIST body %id-attr;>
...
<!--== The Parallel Element ====-->
<!ENTITY % par-content "%container-content;">
<!ELEMENT par    (%par-content;)*>
<!ATTLIST par
        %id-attr;
        %desc-attr;
        endsync CDATA           "last"
        dur     CDATA           #IMPLIED
        repeat  CDATA           "1"
        region  IDREF           #IMPLIED
        %sync-attributes;
        %system-attribute;>
<!--== The Sequential Element =====-->
<!ENTITY % seq-content "%container-content;">
<!ELEMENT seq    (%seq-content;)*>
<!ATTLIST seq
        %id-attr;
        %desc-attr;
```

```
        dur     CDATA           #IMPLIED
        repeat  CDATA           "1"
        region  IDREF           #IMPLIED
        %sync-attributes;
        %system-attribute;>
...
```

SMIL-Komponenten können ihrerseits in SVG- und XHTML-Dokumente integriert werden. Die Unterstützung für SMIL erlaubt es, Medien für die Wiedergabe mit dem von RealNetworks entwickelten RealOne-Player oder mit dem QuickTime-Player einzubinden.

Wir zeigen Ihnen hier ein kleines Beispiel für eine Animation mit drei Abbildungen, die mit einem Song unterlegt und mit einem Text kommentiert werden.

Allgemeine Dokumentstruktur

Das Wurzelelement eines SMIL-Dokuments ist immer `<smil>`. Innerhalb dieses Elements ist der Aufbau ähnlich wie bei einer HTML-Datei in einen Head- und einen Body-Bereich unterteilt. Der Head-Bereich ist optional und wird verwendet, um Informationen für die Präsentation mit Hilfe von `<meta>`-Tags weiterzureichen. Aber auch Layout-Anweisungen, die die Bereichsaufteilung betreffen, werden hier untergebracht.

```
<smil xmlns="http://www.w3.org/2000/SMIL20/CR/Language">
  <head>
    <layout>
      ...
    </layout>
    <meta name="title" content="Kleine Smile-Demo"/>
    <meta name="author" content="HV"/>
  </head>
  <body>
    <par begin="2s" dur="60s" >
  ...
    </par>
  </body>
</smil>
```

Abfolge der Medien

Wir kümmern uns hier zunächst um den Body-Bereich. Die beiden Tags, mit denen die Steuerung der Präsentation hauptsächlich geleistet wird, sind als Kinder oder Kindeskinder von `<body>` das Tag `<par>` für die parallele Vorführung

und <seq> für sequenzielle Anordnungen von Medienobjekten. Innerhalb dieser Tags werden dann die einzelnen Clips mit medienspezifischen Tags wie <audio>, <img>, <video>, <text> etc. eingefügt. Für jeden dieser Tags ist das Attribut src, das die Quelle des Mediums als URL angibt, notwendig.

Um die vorgesehenen Abbildungen und den gleichzeitig abzuspielenden Song einzufügen, kann das vorgegebene Element <par> als Ausgangspunkt genommen und editiert werden.

Zunächst lässt sich der Beginn und die Dauer der gesamten Vorführung als Eigenschaft des <par>-Elements festlegen. Die Eigenschaft begin wird verwendet, um den Start der Animation zu bestimmen. Als Wert kann beispielsweise "20s" angegeben werden. Entsprechend kann die Dauer mit Hilfe des Attributs dur oder mit End bestimmt werden.

Um eine Folge von Bildern einzufügen, wird das Element <seq> eingefügt und darin nacheinander die gewünschten <img>-Elemente für die einzelnen Bilder.

Für jedes Bild muss über die Eigenschaft src die Bildquelle angegeben werden. Anschließend kann wieder die Dauer der Anzeige für jedes Bild bestimmt werden.

Hinter dem abschließenden </seq>-Tag ist ein Tag für die Sounddatei eingefügt. Dazu wird das Element <audio> benutzt. Die Attributzuweisung repeat= "indefinite" erlaubt, den Sound auch als Endlosschleife zu bestimmen.

Vorführregionen

Parallel zu der Bildfolge kann auch noch ein Text angezeigt werden, der die ganze Zeit unverändert bleibt. Der Text soll links von den Bildern erscheinen. Um dies zu erreichen, können im <head>-Bereich zwei unterschiedliche Regionen für die Website definiert werden. Das geschieht mit dem Element <layout> und dessen Kindelementen <root-layout> und <region>. Die Attribute von <root-layout> bestimmen zunächst die gesamte genutzte Fenstergröße. Über die Attribute der <region>-Elemente werden Lage, Höhe und Breite und, wenn gewünscht, über den z-Index auch die Überlappung geregelt. Um auf die Regionen im <body> Bezug nehmen zu können, werden IDs vergeben. Das <head>-Element sieht dann so aus:

```
<head>
  <layout>
    <root-layout height="700" width="1000"/>
    <region id="Textbereich" height="600" width="200"/>
    <region id="Bildbereich" height="600" width="800"/>
  </layout>
```

```
  <meta name="title" content="Kleine Smile-Demo"/>
  <meta name="author" content="HV"/>
</head>
```

Der Bezug auf die Regionen lässt sich bei den `<img>`-Elementen anschließend sehr einfach über das Attribut `region` herstellen.

Im Quelltext sieht das dann insgesamt so aus:

```
<?xml version="1.0"?>
<!--DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN" "http://
www.w3.org/tr/REC-smil/SMIL10.dtd"-->
<smil xmlns="http://www.w3.org/2000/SMIL20/CR/Language">
  <head>
    <layout>
  <root-layout height="700" width="1000"/>
  <region id="Textbereich" height="600" width="200"/>
  <region id="Bildbereich" height="600" width="800"/>
    </layout>
    <meta name="title" content="Kleine Smile-Demo"/>
    <meta name="author" content="HV"/>
  </head>
  <body>
    <par begin="2s" dur="60s" >
  <seq >
    <img id="Bild1" region="Bildbereich" src="bild1.jpg"
    dur="20s"/>
    <img id="Bild2" region="Bildbereich"src="bild2.jpg"
    dur="20s"/>
    <img id="Bild3" region="Bildbereich"src="bild3.jpg"
    dur="20s"/>
  </seq>
  <audio src="song1.mp3" repeat="indefinite"/>
  <text src="kommentar.txt" region="Textbereich"/>
    </par>
  </body>
</smil>
```

Listing 3.6 kleineBildfolge.smil

SMIL im Einsatz

SMIL-Dateien werden mit der Dateierweiterung .smil oder .smi abgespeichert. Auf der Seite **www.w3.org/AudioVideo/** finden Sie Hinweise zu SMIL-Playern, speziellen Editoren, Demos und Hintergrundinformationen. Webentwicklungsumgebungen wie GoLive 6 helfen bei der Eingabe des Codes. Der Internet Explorer 6.0 unterstützt die so genannten XHTML+SMIL-Profile, bei denen eine Untermenge von SMIL mit XHTML integriert ist. Der RealPlayer und auch der Quicktime-Player können SMIL-Dateien direkt abspielen.

Der SMIL-Code kann auch direkt in eine HTML-Datei integriert werden. Der Quellcode bei einer GoLive-Anwendung sieht so aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset
    =iso-8859-1">
  <meta name="generator" content="Adobe GoLive 6">
  <title>Kleine SMIL-Demo</title>
  </head>
  <body bgcolor="#ffffff">
    <p>
     <object classid="clsid:CFCDAA03-8BE4-11cf-B84B-
     0020AFBBCCFA" height="800" width="1000" align="middle">
  <param name="palette" value="background">
  <param name="controls" value="ImageWindow">
  <param name="autostart" value="true">
  <param name="src" value="kleinebildfolge.smil">
  <embed align="middle" height="800" palette="background"
src="kleinebildfolge.smil" type="audio/x-pn-realaudio-plugin"
width="1000" controls="ImageWindow" autostart="true">
  </object>
    </p>
  </body>
</html>
```

Wenn für den Webbrowser ein Real-Plug-in eingerichtet ist, kann die SMIL-Präsentation wiedergegeben werden.

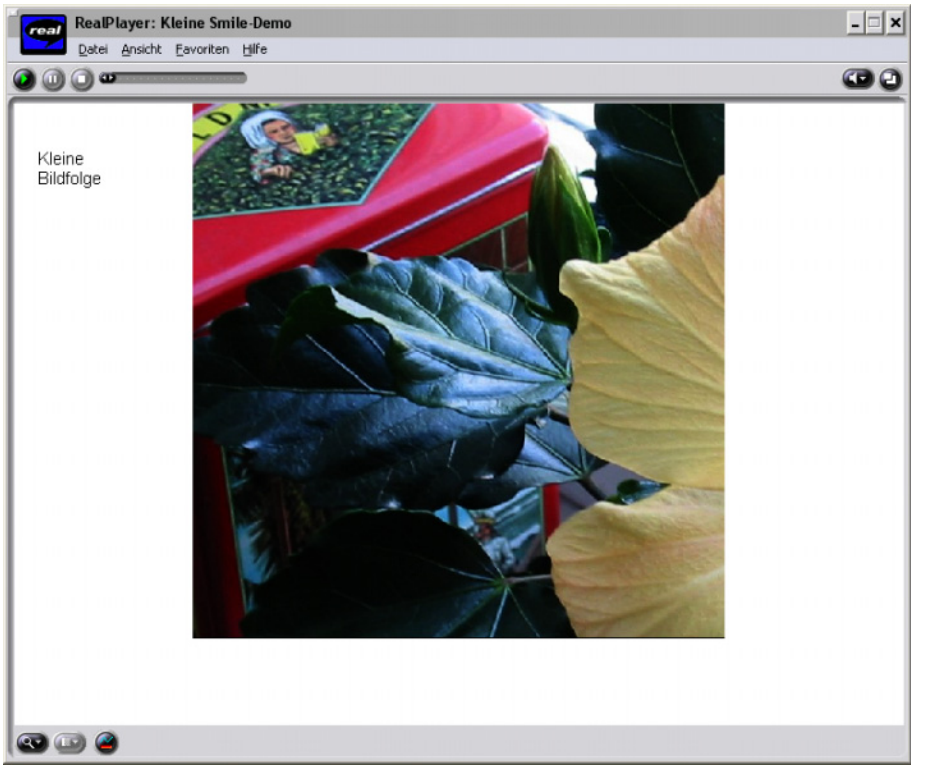

Abbildung 3.10 Vorführung des SMIL-Beispiels im RealPlayer