rationen oder Vergleichen. Hier gibt es jeweils spezielle Regeln, die eine »vernünftige« Typumwandlung von diesen unbestimmten Typen in »geeignete« Typen für die jeweilige Operation vorschreiben.

xdt:anyAtomicType

XQuery führt außerdem den abstrakten Typ `xdt:anyAtomicType` ein, der von `xs:anySimpleType` abgeleitet ist. Von diesem sind wiederum `xdt:untypedAtomic` und alle anderen einfachen Typen (wie zum Beispiel `xs:integer`, `xs:string`) abgeleitet. Dieser Typ ist insbesondere bei der Deklaration von Funktionen hilfreich (Abschnitt 7.4).

xdt:yearMonthDuration
xdt:dayTimeDuration

In Analogie zu SQL:1992 führt XQuery außerdem zwei Untertypen von `xs:duration` ein, die beide (im Gegensatz zu `xs:duration`[1]) vollständig geordnet sind, nämlich `xdt:yearMonthDuration` (Zeiträume, die nur in Jahren und Monaten angegeben sind) und `xdt:dayTimeDuration` (Zeiträume, die nur in Tagen, Stunden, Minuten und Sekunden angegeben sind). Es ist möglich, Werte vom Typ `xs:duration` in diese Typen umzuwandeln. Dabei entfallen dann die Anteile des Zeitraums, die in dem entsprechenden Typ nicht vorgesehen sind. Zum Beispiel ergibt

```
xdt:dayTimeDuration(xs:duration("P2Y2M2DT8H3OM12.5S"))
```

denselben Wert wie

```
xdt:dayTimeDuration("P2DT8H3OM12.5S")
```

Abbildung 3–1 (nach [W3C-18]) zeigt die Einordnung von XQuery-Datentypen in die Typhierarchie, wie sie XML Schema einführt. Ovale stehen dabei für abstrakte Typen, Rechtecke für konkrete Typen. Die von XQuery eingeführten Typen sind grau hinterlegt. Die Werte komplexer Typen und die Werte von Listen- oder Vereinigungstypen sind keine Werte des XQuery-Datenmodells.

3.5 Knoten

In den vorhergehenden Abschnitten haben wir gesehen, wie XQuery die einfachen Datentypen von XML Schema integriert und einerseits in der Verwendung einschränkt (weil Listen- und Vereinigungstypen ausgeschlossen werden), aber andererseits auch um eigene Typen erweitert, um die fehlende totale Ordnung auf `xs:duration` auszugleichen und um Dokumente mit fehlender Typinformation zu behandeln.

Das so entstehende Typsystem reicht aber für die Zwecke von XQuery noch nicht aus. In einem XML-Dokument gibt es Anteile, die

1. xs:duration ist nicht vollständig geordnet, weil Zeiträume, die nur aus Monaten bestehen, nicht mit allen Zeiträumen aus Tagen vergleichbar sind. Es ist nicht möglich, zu sagen, ob P1M größer, gleich oder kleiner als P30D ist.

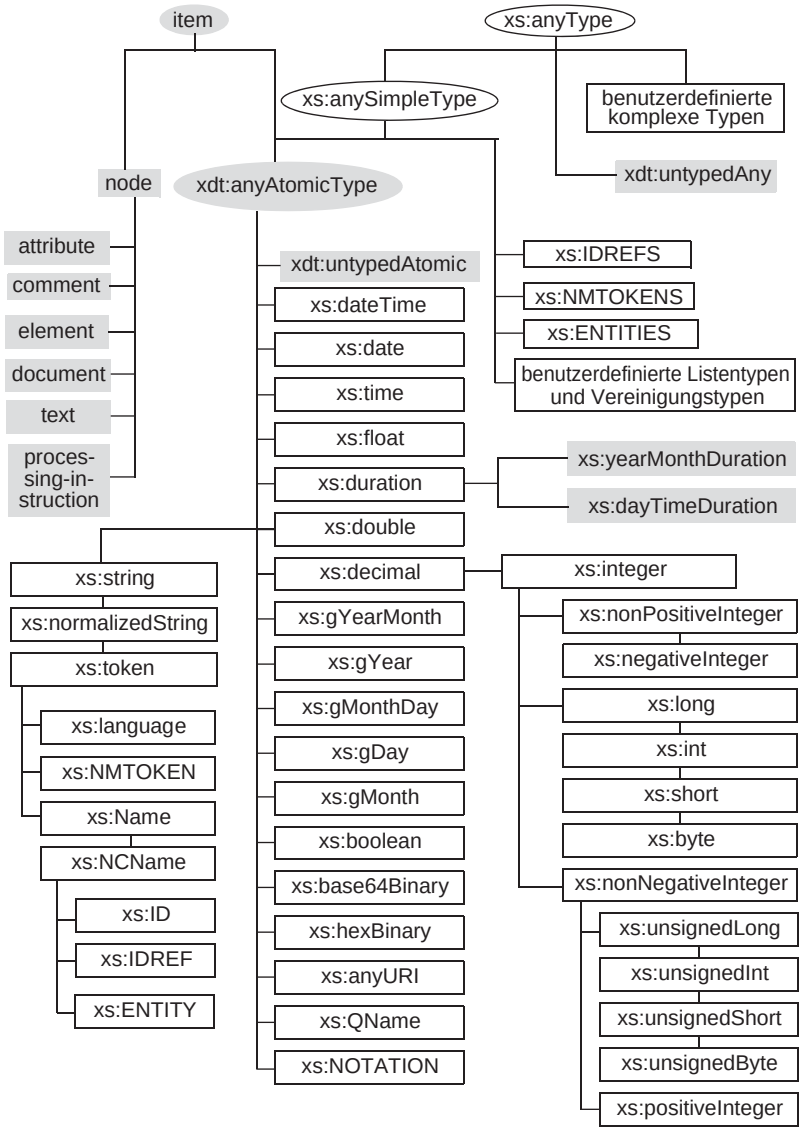

Abb. 3–1 *XQuery-Datentypen in der Typhierarchie von XML Schema*

von XML Schema nicht beschrieben werden können, wie z. B. Kommentare oder Verarbeitungsanweisungen oder auch das Dokument als eigene Einheit. Außerdem lässt sich mit XML Schema nur die mögliche Struktur, nicht aber die tatsächliche Struktur eines XML-Dokumentes beschreiben.

Knotenarten

Zur Beschreibung dieser tatsächlichen Struktur gibt es Datenmodelle wie XML Information Set [W3C-5], das DOM Structure Model

[W3C-2] oder das XPath-1.0-Datenmodell [W3C-7]. Da XQuery eine möglichst große Kompatibilität zu XPath 1.0 herstellen möchte (Abschnitt 9.7), baut es auf dessen Datenmodell zur Darstellung der aktuellen Dokumentstruktur auf. XPath 1.0 kennt sieben Arten von Knoten: Elementknoten, Dokumentknoten, Attributknoten, Namensraumknoten, Knoten für Verarbeitungsanweisungen, Kommentarknoten und Textknoten. XQuery übernimmt diese Knotenarten, lässt aber eine genauere Bestimmung der Knoten zu, wie wir sehen werden.

Jeder Knoten hat einen textuellen Wert und einen getypten Wert. Damit kommen wieder die atomaren Typen ins Spiel, die wir in den vorigen Abschnitten behandelt hatten.

3.5.1 Elementknoten

Ein Elementknoten repräsentiert ein Element aus XML 1.0. Der textuelle Wert eines Elementknotens ergibt sich aus der Verkettung aller Textknoten, die Nachfolger dieses Elementknotens sind. Darüber hinaus hat ein Element einen getypten Wert, falls es keinen komplexen Inhalt hat, wenn es also nicht ausschließlich Kindelemente besitzt. Dieser ergibt sich aus der Validierung des textuellen Wertes gegen die Typdefinition gemäß XML Schema (der getypte Wert hat also zum Beispiel den Typ `xs:integer`, wenn laut Schemadefinition das Element den einfachen Typ `xs:integer` hat). Hat das Element den Typ `xdt:untypedAny` (also einen unbestimmten komplexen Typ), so ist der getypte Wert vom Typ `xdt:untypedAtomic` (dem unbestimmten atomaren Typ).

Zur Erzeugung eines Elementknotens gibt es zwei verschiedene Konstruktoren: den direkten Elementkonstruktor und den berechneten Elementkonstruktor.

Der direkte Elementkonstruktor

Der direkte Elementkonstruktor benutzt die vertraute XML-Notation, also zum Beispiel:

```
<Arzt ID="Arzt_01">Hans Müller</Arzt>
```

In Attributwerten und im Elementinhalt sind Referenzen auf vordefinierte Entitäten (zum Beispiel `<`) und Zeichenreferenzen (zum Beispiel `&#x20;`) erlaubt, die bei der Konstruktion aufgelöst werden:

```
<Fähigkeit>Altenbetreuung &amp; Altenpflege</Fähigkeit>
```

Statt eines konstanten Inhalts kann auch ein Ausdruck angegeben werden, der bei der Konstruktion ausgewertet wird. Damit ein Ausdruck von einem literalen Inhalt unterschieden werden kann, wird er in geschweifte Klammern eingeschlossen (Die XQuery-Spezifikation spricht hier von einem *eingeschlossenen Ausdruck*. Dieser signalisiert einen neuen »Auswertungskontext«, also nicht eine literale Übernahme, sondern eine Ausdrucksauswertung). So entsteht durch den Konstruktor

Eingeschlossener Ausdruck

```
<Gehalt>{12*3000}</Gehalt>
```

das folgende Element.

```
<Gehalt>36000</Gehalt>
```

Soll eine geschweifte Klammer ausnahmsweise keinen Ausdruck kennzeichnen, so ist sie zu verdoppeln. Die Auswertung der Ausdrücke in geschweiften Klammern ergibt Sequenzen aus Knoten und atomaren Werten. Entstehen so Attributknoten (siehe unten), dann werden diese zu Attributen des umgebenden Elementes. Allerdings müssen diese Attributknoten vor anderen Knoten stehen, sonst wird ein Fehler gemeldet. Elementknoten werden zu Kindern des umgebenden Elementes.

Verschachtelte Konstruktoren

Weiter verschachtelte CDATA- und Elementkonstruktoren und Konstruktoren für Kommentare und Verarbeitungsanweisungen müssen nicht in geschweiften Klammern stehen, wie das folgende Beispiel zeigt.

```
<Freiwilliger ID="vol_01"><!-- eingefügt 1.1.2004-->
   <Name>
      <Vorname>Daniela</Vorname>
      <Nachname>Baumann</Nachname>
   </Name>
   <Adresse>
      <Straße>Potsdamer Straße</Straße>
      <Hausnr>62</Hausnr>
      <Stadt>Berlin</Stadt>
      <Staat>D</Staat>
      <PLZ>14145</PLZ>
   </Adresse>
   <Geburtsdatum> {xs:date("1982-07-23")} </Geburtsdatum>
   <Telefon>+49 30-234626</Telefon>
   <Nummer>07</Nummer>
   <Fähigkeit>Altenbetreuung</Fähigkeit>
   <Berufsklasse>Pfleger</Berufsklasse>
</Freiwilliger>
```

Die genaue Syntax des direkten Elementkonstruktors lautet:

```
DirElemConstructor ::= <QName AttributeList (/> | (> ElementContent* </QName>))

ElementContent     ::= ElementContentChar| {{ | }} | DirElemConstructor
                       | EnclosedExpr | CdataSection | CharRef
                       | PredefinedEntityRef | XmlComment | XmlPI
AttributeList      ::= ( QName = AttributeValue )*

EnclosedExpr       ::=  { Expr }
```

Attributwerte

Ein Ausdruck in geschweiften Klammern ist auch in einem Attributwert erlaubt. Das Ergebnis der Auswertung eines Ausdrucks in einem Attributwert wird durch Atomisierung (Abschnitt 3.3) zu einer Sequenz von atomaren Werten, die wiederum in Werte vom Typ `xs:string` gewandelt werden. Alle diese Zeichenketten werden verkettet und ergeben den Attributwert. Alle Leerzeichen werden dabei in gewöhnliche Leerräume (`U+0020`) umgewandelt. So wird zum Beispiel aus dem Konstruktor

```
<Arzt Gehalt="Jahresgehalt {<Berechnung>
{12*3000}</Berechnung>} Euro"/>
```

das Element

```
<Arzt Gehalt="Jahresgehalt 36000 Euro"/>
```

Leerraum in Elementkonstruktoren

Begrenzender Leerraum

Die Behandlung von Leerraum in Elementkonstruktoren wird durch den Wert von `xmlspace` (Abschnitt 8.2.5) aus dem XQuery-Prolog gesteuert. Wenn nicht explizit definiert ist, dass Leerraum erhalten bleiben muss, wird begrenzender Leerraum (»boundary white space«) entfernt, also Leerraum, der lediglich Elementgrenzen (»start tag« und »end tag«) und/oder Auswertungskontexte voneinander trennt. Anderer Leerraum bleibt immer erhalten. Auch Leerraum, der in Form von Zeichenreferenzen (zum Beispiel `&#x020;`) angegeben ist, bleibt erhalten. Der oben gezeigte Elementkonstruktor ist dann äquivalent zu

```
<Freiwilliger ID="vol_01"><!-- eingefügt 1.1.2004--><Name>
<Vorname>Daniela</Vorname><Nachname>Baumann</Nachname></Name>
<Adresse><Straße>Potsdamer Straße</Straße><Hausnr>62</Hausnr>
<Stadt>Berlin</Stadt><Staat>D</Staat><PLZ>14145</PLZ></Adresse>
<Geburtsdatum>1982-07-23</Geburtsdatum>
<Telefon>+49 30-234626</Telefon><Nummer>07</Nummer>
<Fähigkeit>Altenbetreuung</Fähigkeit>
<Berufsklasse>Pfleger</Berufsklasse></Freiwilliger>
```

Die folgende Tabelle 3–4 zeigt einige Beispiele für Ergebnisse von Elementkonstruktoren mit und ohne Erhaltung von begrenzendem Leerraum:

Elementkonstruktor	Ergebnis der Auswertung	
	mit Leerraumerhaltung	ohne Leerraumerhaltung
`<X y="{1}  {2}"/>`	`<X y="1  2"/>`	`<X y="12"/>`
`<X>  </X>`	`<X>  </X>`	`<X></X>`
`<X y="{1}  2"/>`	`<X y="1  2"/>`	`<X y="1  2"/>`
`<X>  1  {"Text"}  </X>`	`<X>  1  "Text"  </X>`	`<X>  1  "Text"</X>`
`<X>&#20;  </X>`	`<X>  </X>`	`<X>  </X>`
`<X>{"  "}</X>`	`<X>  </X>`	`<X>  </X>`
`<X><![CDATA[  ]]></X>`	`<X>  </X>`	`<X>  </X>`

Tab. 3–4 *Beispiele zur Leerraumbehandlung in Elementkonstruktoren*

Im ersten und zweiten Beispiel stehen Leerzeichen zwischen Auswertungskontexten bzw. Tags. Diese gelten als begrenzender Leerraum. Im Gegensatz dazu handelt es sich beim dritten Beispiel nicht um begrenzenden Leerraum, weil kein Auswertungskontext, sondern ein einfaches Zeichen folgt. Das vierte Beispiel zeigt analog, dass auch Leerzeichen am Anfang eines Elementkonstruktors immer erhalten bleiben, wenn kein Ausdruck, sondern ein Wert folgt. Im fünften Beispiel sieht man, dass die Zeichenreferenz `&#x20;` für ein Leerzeichen ebenfalls bewirkt, dass kein begrenzender Leerraum angenommen wird. Im sechsten und siebten Beispiel steht der Leerraum innerhalb eines Ausdrucks bzw. CDATA-Konstruktors und bleibt daher erhalten.

Der berechnete Elementkonstruktor

element

Beim direkten Elementkonstruktor können Elementinhalt und Attributwerte durch Ausdrücke berechnet werden; der Elementname jedoch ist konstant. Wenn auch der Name eines Elementes berechnet werden soll, muss ein berechneter Elementkonstruktor eingesetzt werden. Dabei folgt dem Schlüsselwort `element` der Name des Elementes, und zwar als konstanter QName (Abschnitt 2.5.1) oder als Ausdruck in einem Auswertungskontext, dessen Ergebnis wieder der Atomisierung unterworfen wird und dann einen der Typen `xs:QName`, `xs:string` oder `xs:untypedAtomic` haben muss. Darauf folgt der Elementinhalt in einem Auswertungskontext, wieder mit der Regel, dass Attributknoten am Anfang der sich ergebenden Sequenz stehen müssen. Analoges gilt für Namensraumknoten, die sogar noch vor den Attributknoten plat-

Reihenfolge der Knoten

ziert werden müssen. Aus allen atomaren Werten im Elementinhalt werden Textknoten, wobei nebeneinander stehende Textknoten zusammengefasst werden. So wird beispielsweise aus

```
element Labortest {
  attribute ID {"Labortest_040782"},
  element Nummer {1},
  <Name>Röntgen</Name>,
  element Datum {"2002-05-10T10:30:00-05:00"},
  <Testgegenstand>linker Oberschenkel</Testgegenstand>,
  element Labor {
    namespace xlink {"http://www.w3.org/1999/xlink"},
    attribute
      xlink:href {'Hochwaldklinik.xml#xpointer(id("Radiologie"))'}
    }
  }
```

folgendes Element:

```
<Labortest ID="Labortest_040782">
  <Nummer>1</Nummer>
  <Name>Röntgen</Name>
  <Datum>2002-05-10T10:30:00-05:00</Datum>
  <Testgegenstand>linker Oberschenkel</Testgegenstand>
  <Labor xmlns:xlink="http://www.w3.org/1999/xlink"
      xlink:href='Hochwaldklinik.xml#xpointer(id("Radiologie"))'/>
</Labortest>
```

Wie man sieht, können innerhalb des Elementkonstruktors wieder beide Arten von Elementkonstruktoren verwendet werden. Im Element `Labor` muss ein Namensraumknoten für das Präfix `xlink` erzeugt werden, wenn dieses nicht schon in der Umgebung definiert ist. Man sieht auch, dass der Typ, der innerhalb eines Elementkonstruktors verwendet wird, nicht entscheidend ist, solange er sich nach `xs:string` konvertieren lässt (im Element `Nummer` wird ein Integer-Literal verwendet, in `Datum` eine Zeichenkette, obwohl das Element den Typ `xs:date` hat). Die Syntax des berechneten Elementkonstruktors ist:

```
CompElemConstructor::= element ( QName | { Expr } ) { Expr? }
```

Gemeinsame Regeln für beide Konstruktorarten

Werden Namen mit einem Namensraumpräfix in einem Elementkonstruktor verwendet, dann muss das Präfix definiert sein, und zwar entweder im XQuery-Prolog (Abschnitt 8.2.1) oder im Element selbst.

Validierung

Das entstehende Element wird unter Berücksichtigung des Validierungskontexts (Abschnitt 3.8.2) automatisch gegen die bekannten Schemadefinitionen validiert und bekommt damit gegebenenfalls einen Typ zugewiesen. Das impliziert auch, dass Attribute mit ihren

Ein solcher Konstruktor kann sowohl in direkten als auch in berechneten Elementkonstruktoren verwendet werden, um Attributknoten zu dem jeweiligen Element zu erzeugen. Zum Beispiel liefert

```
<Gehalt>{attribute Währung {" "}, "10000"}</Gehalt>
```

das Element `<Gehalt Währung=" ">10000</Gehalt>`. Äquivalent hätte man schreiben können

```
element Gehalt {attribute Währung {" "}, "10000"}
```

oder

```
<Gehalt Währung=" ">10000</Gehalt>
```

Attributknoten sind von Namensraumknoten verschieden. Daraus folgt, dass der Name eines Attributes nicht `xmlns` sein darf oder mit `xmlns:` beginnen darf.

3.5.4 Namensraumknoten

Namensraumknoten repräsentieren die im jeweiligen Kontext bekannten Namensräume. Der textuelle Wert eines Namensraumknotens ist die zugehörige URI. Der getypte Wert ergibt sich aus dem textuellen Wert durch Wandlung nach `xdt:untypedAtomic`.

namespace

Auch ein Namensraumknoten kann durch einen berechneten Konstruktor erzeugt werden. Dem Schlüsselwort `namespace` folgt das zugewiesene Präfix und ein Ausdruck in einem Auswertungskontext zur Berechnung der URI des Namensraumes. Ein Knoten für den »default namespace« (ohne Präfix) kann so nicht erzeugt werden.

Namensraumknoten werden auch implizit erzeugt. Jeder Elementknoten erhält für jedes `xmlns`-Attribut in seinem direkten Elementkonstruktor einen entsprechenden Namensraumknoten als Kind. Hinzu kommen Kinder für alle Namensraumknoten umgebender Elementkonstruktoren. Für jeden weiteren im Elementnamen oder seinen Attributen verwendeten bekannten Namensraum wird ebenfalls ein Kind erzeugt. Außerdem erhält jeder Elementknoten einen Namensraumknoten für den xml-Namensraum.

Auf Namensraumknoten kann man in XQuery allerdings nicht explizit zugreifen.

3.5.5 Knoten für Verarbeitungsanweisungen

Verarbeitungsanweisungen bestehen aus einem Ziel und einem Inhalt. Ihr textueller Wert entspricht dem Inhalt, der getypte Wert dem textuellen Wert.

Eine Verarbeitungsanweisung kann mit einem direkten Konstruktor erzeugt werden, der wiederum die XML-Syntax benutzt:

```
<?Beispiel dies ist der Inhalt?>
```

processing instruction

Alternativ kann ein berechneter Konstruktor verwendet werden. Dieser besteht aus dem Schlüsselwort `processing-instruction`, gefolgt vom Ziel der Verarbeitungsanweisung. Diese kann als ein konstanter QName oder ein Ausdruck in einem Auswertungskontext angegeben werden. Darauf folgt der Inhalt als Ausdruck in einem Auswertungskontext.

Diese Ausdrücke werden jeweils der Atomisierung unterworfen:

```
processing-instruction Beispiel {"dies ist der Inhalt"}
```

3.5.6 Kommentarknoten

Ein Kommentarknoten entspricht einem Kommentar in einem XML-Dokument. Der textuelle Wert eines Kommentarknotens besteht aus dem Inhalt des Kommentars, der getypte Wert ebenfalls (als Wert vom Typ `xs:string`). Auch für Kommentarknoten gibt es einen direkten Konstruktor, der die XML-Syntax benutzt:

```
<!-- Dies ist ein Kommentar -->
```

comment

Ebenso gibt es einen berechneten Konstruktor: das Schlüsselwort `comment`, gefolgt von einem Ausdruck in einem Auswertungskontext.

```
comment {"Dies ist ein Kommentar"}
```

Auch hier wird das Ergebnis der Ausdrucksauswertung der Atomisierung unterworfen. Anzumerken ist an dieser Stelle, dass ein Kommentarknoten nicht mit einem XQuery-Kommentar (Abschnitt 4.1) verwechselt werden darf.

3.5.7 Textknoten

Ein Textknoten entspricht dem einfachen Inhalt eines Elementes, und zwar unabhängig von seinem Typ. Für Textknoten gibt es einen direkten Konstruktor, der die CDATA-Syntax aus XML verwendet:

```
<![CDATA[Dies ist der Inhalt des Textknotens]]>
```

Vorbelegungswerten zum Element hinzukommen können. Dieser Validierungsvorgang berücksichtigt ein eventuell angegebenes `xsi:type`-Attribut. So wird der Inhalt des folgenden Elementes gegen `xs:integer` validiert, auch wenn für das Element keine Definition bekannt ist:

```
<unbekannt xsi:type="xs:integer">42</unbekannt>
```

3.5.2 Dokumentknoten

Ein Dokumentknoten repräsentiert ein XML-Dokument. XQuery hat einen etwas weiteren Begriff von einem XML-Dokument als die XML-Spezifikation. So ist nicht gefordert, dass es genau einen Elementknoten als Kind des Dokumentknotens gibt. Der textuelle Wert eines Dokumentknotens ergibt sich aus der Verkettung der textuellen Werte aller Textknoten, die Nachfolger des Dokumentknotens sind.

Ein Dokumentknoten kann in XQuery mit einem berechneten Konstruktor der Form `document {Expr}` erzeugt werden:

```
document {<Arzt>Emil Müller</Arzt>}
```

Die Knoten, die als Kinder dieses neuen Dokumentknotens angefügt werden, verlieren dabei jegliche Typinformation, d. h., ihnen werden die Typen `xs:anyType` (für Elementknoten) und `xs:anySimpleType` (für Attributknoten) zugewiesen. Es wird nicht erzwungen, dass ein Dokument gemäß XML 1.0 entsteht, welches zum Beispiel genau ein Wurzelelement besitzt.

3.5.3 Attributknoten

Attributknoten entsprechen den Attributen von XML-Elementen. Der textuelle Wert eines solchen Knotens ergibt sich aus dem Attributwert, gewandelt nach `xs:string`. Der getypte Wert des Attributknotens wird aus dem textuellen Wert durch Validierung gegen ein Schema (und damit Zuweisung des dort definierten Typs) gewonnen.

attribute

Einen Attributknoten kann man über einen berechneten Konstruktor erzeugen, bei dem auf das Schlüsselwort `attribute` der Name des Attributs (als konstanter QName oder als Ausdruck in einem Auswertungskontext) und sein Wert (in einem Auswertungskontext) folgen. Atomisierung wird jeweils auf die Werte in den Auswertungskontexten angewandt. Der Wert wird immer nach `xs:string` gewandelt, muss also nicht dem Typ des Attributs entsprechen. Der folgende Aufruf des berechneten Attributkonstruktors erzeugt ein Attribut, dessen Name sich aus dem Wert der Variablen `$a` ergibt und dessen Wert 42 ist:

```
attribute {$a} {6*7}
```

Dabei werden die Zeichen zwischen [und] nicht weiter ausgewertet. Das bedeutet zum Beispiel, dass spitze Klammern in diesem Teil nicht als XML-Markup verstanden werden. Die Notation CDATA bedeutet nicht, dass auch im Ergebnis ein CDATA-Abschnitt auftreten muss. Vielmehr wird hier ein Textknoten erzeugt, dessen spätere Darstellung (Serialisierung) jedoch nicht vorgegeben ist.

CDATA

Außerdem gibt es einen berechneten Konstruktor, bei dem ein Ausdruck in einem Auswertungskontext dem Schlüsselwort `text` folgt. Das Ergebnis dieses Ausdrucks wird atomisiert. Entsteht dabei eine leere Sequenz, so wird kein Textknoten erzeugt.

text

```
text {"Dies ist der Inhalt des Textknotens"}
```

3.6 Knoteneigenschaften

Wie bereits beschrieben, wird ein XML-Dokument im XQuery-Datenmodell als Baum aus Knoten abgebildet. Jeder Knoten hat dabei seine eigene Identität (nicht zu verwechseln mit einem ID-Attributwert). Selbst wenn zwei Knoten denselben Wert haben, sind sie nicht identisch. Ein Knoten ist nur identisch mit sich selbst.

3.6.1 Dokumentreihenfolge

Dokumentordnung

Die Ordnung im Baum definiert auch eine Ordnung auf den Knoten, nämlich die *Dokumentordnung* (Ordnung der Knoten in der Dokumentreihenfolge), die sich ergibt, wenn man den Baum folgendermaßen durchläuft:

- Der Elternknoten wird direkt vor seinen Kindern besucht.
- Namensraumknoten eines Elementes werden vor den Attributknoten desselben Elementes besucht.
- Attributknoten werden vor den Kindern eines Elementknotens besucht.
- Geschwisterknoten werden in derselben Reihenfolge besucht, die sie im XML-Dokument haben.

Somit ist der Dokumentknoten der erste Knoten in der Dokumentreihenfolge. Diese Dokumentordnung ist eine totale Ordnung. Obwohl XQuery keine Ordnung zwischen Attributknoten oder Namensraumknoten vorgibt, muss eine XQuery-Implementierung diese festlegen. Einige XQuery-Operationen sortieren eine Sequenz von Knoten nach der Dokumentordnung (zum Beispiel Pfadausdrücke). Das beinhaltet auch immer die Entfernung von Duplikaten, d. h. die Eliminierung identischer Knoten.