



## Editorial

Früher war alles einfacher! In den Anfangszeiten von XML konnte ich noch problemlos alle Spezifikationen lesen und verstehen. Heute bin ich schon froh, wenn ich die Existenz einer Spezifikation kenne. Es entstehen immer mehr und längere Spezifikationen. Die neue XPath-Spezifikation zum Beispiel ist ungefähr 5mal länger als die vorherige

und enthält 70% mehr Sprachelemente. Wer aber braucht XPath? Für jene, die XML programmieren, ist XPath inzwischen unverzichtbar. Die XPath-Sprache ist der „Pfadfinder“ im Knotenbaum eines XML-Dokumentes. Sie gehört zu den wenigen XML-Sprachen, die keine XML-Syntax haben. Schließlich muss sie auch in Attributen von XML-Sprachen wie XSLT verwendet werden können.

Wenn jemand keine Zeit für das Lesen von umfangreichen Spezifikationen hat, ist er auf gute Einführungen, wie zum Beispiel den Artikel von Herrn Becker über die alte und die neue XPath-Sprache, angewiesen.

Norbert Hranitzky, Kolumnist XMLspektrum

## Pfad-Finder

# Navigation mit XPath

Oliver Becker

Die Pfadsprache XPath ermöglicht den Zugriff auf die einzelnen Bestandteile eines XML-Dokumentes. XPath-Ausdrücke werden hauptsächlich innerhalb von XSLT-Stylesheets verwendet und dienen dazu, gezielt die Werte von Elementen oder Attributen verfügbar zu machen. Doch auch reine Java-Anwendungen können von XPath als einfacher XML-Anfragesprache profitieren. Dieser Artikel stellt die wichtigsten Bestandteile der aktuellen Version von XPath vor und gibt einen Ausblick auf die kommende Version 2.0.



## Ortsbestimmung

▶ XPath ist eine der zentralen Spezifikationen aus dem XML-Umfeld. Die aktuelle Version 1.0 von XPath [XPath, XPathd] stammt aus dem Jahre 1999. Gegenwärtig arbeitet das W3C an der Nachfolgeversion 2.0 [XPath2]. Mehrere andere W3C-Standards bauen auf XPath auf. Am bekanntesten dürfte wohl die XML-Transformationssprache XSLT sein. Weitere Beziehungen existieren zu

▼ XML Schema, das XPath benutzt, um den Gültigkeitsbereich von eindeutigen Werten zu definieren (s. `xs:key` und `xs:unique`),

▼ XPointer, das XPath erweitert, um Stellen und Bereiche in Dokumenten als Linkziele beschreiben zu können,

▼ der künftigen XQuery-Spezifikation, deren Abfragesprache auf XPath 2.0 aufbaut.

Was genau ist XPath? XPath ist eine Sprache, mit der sich Teile eines XML-Dokumentes adressieren lassen. Es handelt sich nicht um eine Programmiersprache, da keinerlei Anweisungen o. ä. enthalten sind. Vielmehr definiert die XPath-Spezifikation allein die Syntax und Semantik von Ausdrücken, die beispielsweise innerhalb von XSLT benutzt werden können (XSLT und XPath bilden zusammen durchaus eine vollwertige Programmiersprache). Da XPath-Ausdrücke vorrangig als Attributwerte oder im Zusammenhang mit XPointer als URIs vorkommen, ist XPath selbst nicht in XML ausgedrückt.

Das wichtigste Konstrukt in XPath stellen Pfadausdrücke dar, daher der Name. Beispielsweise greift

`/book/@date` auf das `date`-Attribut des Wurzelements `book` zu und `../header` auf die `header`-Elemente, die Kinder des aktuellen Elternelements sind. Pfade ermöglichen es, innerhalb eines XML-Doku-

Java	XPath
<code>+, -, *, &lt;, &gt;, &lt;=, &gt;=, !=</code>	wie in Java
<code>==</code>	<code>=</code>
<code>/</code>	<code>div</code>
<code>%</code>	<code>mod</code>
<code>&amp;&amp;</code>	<code>and</code>
<code>  </code>	<code>or</code>
<code>!</code>	<code>not(...)</code>
(not ist eine XPath-Funktion)	

Tabelle 1: XPath-Operatoren



ments zu navigieren, wie es Pfade im Dateisystem tun. Darüber hinaus erlaubt XPath arithmetische und logische Ausdrücke und enthält eine überschaubare Funktionsbibliothek von 50 Funktionen.

## Vertrautes Terrain

Arithmetische und logische Ausdrücke in XPath ähneln denen in anderen Programmiersprachen. Lediglich einige Operatoren werden in XPath anders notiert (s. Tabelle 1). Die folgenden Zeilen zeigen drei einfache Beispiele:

```
(1 + 2) * 3
$n mod 2 = 0
string-length($pass) < 6 or
contains($pass, "xml")
```

Das Ergebnis des Ausdrucks in der ersten Zeile ist die Zahl 9. Die zweite Zeile liefert einen booleschen Wert *wahr* oder *falsch*, je nachdem, ob der Wert der Variablen *n* bei der Division durch 2 den Rest 0 lässt oder nicht. Wie hier zu sehen ist, werden Variablenreferenzen mit einem Dollarzeichen (\$) eingeleitet. Die dritte Zeile schließlich ist eine *oder*-Verknüpfung zweier logischer Ausdrücke, die dann *wahr* ergibt, wenn die Länge des Wertes von *pass* kleiner als 6 ist oder in dieser Variablen die Zeichenkette "xml" irgendwo auftaucht.

XPath selbst sieht keinerlei Möglichkeiten vor, Variablen zu deklarieren. Dies muss in der jeweiligen Umgebung erfolgen, beispielsweise in XSLT per `xsl:variable` oder `xsl:param`. Werden XPath-Ausdrücke innerhalb einer XML-Datei verwendet (was typischerweise der Fall ist), muss selbstverständlich das Zeichen < geeignet maskiert werden, etwa als `&lt;`;

XPath 1.0 ist eine schwach-typisierte Sprache. Es existieren vier Datentypen: boolesche Werte, Zahlen (mit dem Umfang von *double* in Java), Zeichenketten und Knotenmengen. Innerhalb von Ausdrücken werden Werte bei Bedarf in den notwendigen Typ konvertiert. Der Vergleich `string-length(100) = "3"` ergibt beispielsweise den Wert *wahr*. Beim Rechnen in XPath können keine Laufzeitfehler auftreten, da alle Werte, die keine Zahlen sind, als der spezielle Wert NaN (not a number) repräsentiert werden.

## Aufbruch zu neuen Ufern

Der im letzten Absatz genannte Datentyp *Knotenmenge* stellt das wichtigste Konzept in XPath dar.

Ein Knoten stammt immer aus einem XML-Dokument. Tatsächlich ist der Hauptzweck von XPath, den Zugriff auf die Informationen eines XML-Dokumentes zu ermöglichen, und weniger, ausgefallene arithmetische Berechnungen ausführen zu können.

Für das Folgende ist es notwendig, sich ein XML-Dokument als Baumstruktur vorzustellen. Leser, die bereits mit dem Document Object Model (DOM) vertraut sind, werden im XPath-Datenmodell die Konzepte Baum und Knoten wiederfinden. Leider gibt es einige kleine Unterschiede, auf die unten kurz hingewiesen wird.

An der Spitze des Baumes steht die Dokumentwurzel als Repräsentation des gesamten Dokuments (in XPath 1.0 spricht man vom Wurzelknoten, in XPath 2.0 wird zukünftig der Begriff Dokumentknoten verwendet). Unterhalb dieses obersten Knotens befinden sich genau ein Elementknoten (für das Dokumentelement) und gegebenenfalls weitere Knoten für außerhalb des Dokumentelementes vorkommende Kommentare und Verarbeitungsanweisungen. Unterhalb des Dokumentelementes befindet sich dann jeglicher weitere Inhalt des Dokumentes in Form von Unterelementen, Textinhalt, etc.

XPath unterscheidet sieben Knotentypen. Neben den bereits erwähnten vier Typen Wurzelknoten, Elementknoten, Kommentarknoten und Verarbeitungsanweisungsknoten existieren Textknoten für den eigentlichen Textinhalt eines XML-Dokumentes, Attributknoten, die Werte in den Attributen des Starttags eines Elementes repräsentieren, und schließlich Namensraumknoten, die die deklarierten Namensräume repräsentieren. Da Namensraumknoten ab Version 2.0 von XPath nicht mehr direkt zugänglich sein werden, wird an dieser Stelle nicht genauer auf sie eingegangen.

Nur der Wurzel- und die Elementknoten können weitere Knoten als Kinder enthalten. Attribut- und Namensraumknoten sind immer einem Element zugeordnet. Dabei liegt hier die unsymmetrische Situation vor, dass diese beiden zwar keine Kinder ihrer Elementknoten sind, das jeweilige Element aber als Elternknoten seiner Attribut- und Namensraumknoten angesehen wird.

Es ist sehr wichtig zu verstehen, welche Daten eines Dokuments durch XPath tatsächlich modelliert werden. Nur auf diese Daten lässt sich demzu-

folge zugreifen. Im XPath-Datenmodell fehlen beispielsweise die XML-Deklaration `<?xml version="1.0" ...?>`, eine eventuelle Dokumenttyp-Deklaration (im Gegensatz zum DOM), die Information über verwendete Entities, da diese sich vollständig expandiert im Baum wiederfinden, die Information über die Grenzen von CDATA-Abschnitten (im Gegensatz zum DOM) und schließlich solche Feinheiten, wie die Reihenfolge der Attribute im Starttag und die Art der Begrenzer für den Attributwert (doppelte oder einfache Anführungszeichen). Namensraumdeklarationen werden in XPath nicht als Attributknoten dargestellt (ebenfalls im Gegensatz zum DOM).

Pfade in XPath dienen nun dazu, innerhalb dieses Baumes zu navigieren und so den Zugriff auf einzelne Knoten zu ermöglichen. Absolute Pfade gehen von der Dokumentwurzel aus; relative Pfade vom aktuellen Knoten, der auch Kontextknoten genannt wird.

Ein Name innerhalb eines Pfades adressiert immer einen Elementknoten. So wurde bereits `/book` als Beispiel genannt. Der Pfad `/book/title` greift ausgehend von der Wurzel auf deren Kind `book` (das damit das Dokumentelement sein muss) und dann auf das oder die Kindelemente mit dem Namen `title` zu. Das Ergebnis des Pfades `/book/title` besteht damit aus allen `title`-Elementen, die Kinder des Wurzelementes `book` sind, aber ohne das `book`-Element selbst. Ein Pfad wählt in der Regel mehrere Knoten, d. h. eine Knotenmenge, aus. Ein beliebiges Element lässt sich durch das Zeichen \* adressieren. Der Pfad \* allein wählt alle Kindelemente des Kontextknotens aus, ein Pfad `*/*` wählt alle Enkelelemente des Kontextknotens aus. Die letzten beiden Pfade beginnen ohne den Schrägstrich, es handelt sich um relative Pfade. Ein absoluter Pfad, der an der Dokumentwurzel beginnt, enthält als erstes Zeichen den Schrägstrich. Das Zeichen / allein wählt die Dokumentwurzel aus.

Enthält das Eingabedokument Elemente aus einem bestimmten Namensraum, muss man für XPath ein (beliebiges) Präfix für diesen Namensraum deklarieren und dann unter Angabe eines qualifizierten Namens auf das gewünschte Element zugreifen, beispielsweise `/d:book/d:title`. Ein voreingestellter Namensraum wirkt sich dabei nicht auf



XPath-Ausdrücke aus. Ein häufiger Fehler ist, dass die Deklaration eines voreingestellten Namensraums im XML-Eingabedokument übersehen wird und dann Pfade ohne explizite Verwendung eines Präfix ins Leere zeigen und somit nichts auswählen.

Die Knoten der anderen Knotentypen lassen sich über spezielle Knotentests adressieren. So werden Textknoten über `text()` angesprochen, Verarbeitungsanweisungsknoten über `processing-instruction()` und Kommentarknoten über `comment()`. Der Knotentest `node()` passt auf jeden Knoten. Auch wenn die Schreibweise dies nahe legt, handelt es sich hier nicht um Funktionsaufrufe. Der Pfad `/*/text()` adressiert beispielsweise alle Textknoten, die direkte Kinder des Wurzelelementes sind.

## Auf verschlungenen Wegen

Bisher konnte der Eindruck entstehen, dass die über Schrägstriche miteinander verbundenen Bestandteile eines Pfades immer in einer Eltern-Kind-Beziehung zueinander stehen. Dies ist nicht so. Tatsächlich kann in jedem Schritt (die Bestandteile eines Pfades zwischen den Schrägstrichen heißen Schritte) in eine beliebige Richtung im Dokument navigiert werden. Diese Richtungen werden als Achsen bezeichnet. Fehlt die Angabe der Achse, wird die so genannte Kind-Achse angenommen, so wie es in den bisherigen Beispielen der Fall war. Ein Schritt mit Achsenspezifikation besitzt die Form *Achse::Knotentest*. Der Bestandteil hinter den Doppelpunkten, der Knotentest, wurde bereits besprochen.

Tabelle 2 listet die 13 verfügbaren Achsen auf. Abbildung 1 veranschaulicht 9 von ihnen.

Während die ersten 11 dieser 13 Achsen auf „normale“ Knoten innerhalb des Baumes zugreifen, also auf alle Knoten, die in einer Eltern-Kind-Beziehung zueinander stehen können, werden mit den letzten beiden nur bestimmte Knotentypen angesprochen. Die `attribute`-Achse adressiert ausschließlich Attributknoten; jede der anderen Achsen (von `self` abgesehen) enthält niemals ein Attribut. Entsprechendes gilt für die `namespace`-Achse, die jedoch in XPath 2.0 als *deprecated* erklärt wird.

Mit Hilfe dieser Achsen lassen sich nun kompliziertere Wege durch das XML-Dokument finden. Beispielsweise wählt

```
parent::*/*following-sibling::chapter
```

alle `chapter`-Elemente aus, die dem Elternknoten als Geschwister im Dokument folgen, sozusagen die folgenden `chapter`-Tanten bzw. -Onkel. Entsprechend finden sich über

```
preceding-sibling::*/*descendant::comment()
```

alle Kommentarknoten, die Nachkommen eines vorangehenden Geschwisteres sind, quasi alle vorher im Dokument auftauchenden Kommentarnichten bzw. -Nichten inklusive aller Großnichten, Urgroßnichten usw. Alles klar soweit? Gut.

Da das Ergebnis eines Pfades immer eine Menge von Knoten ist, existieren in dieser Menge weder Duplikate noch besitzen die ausgewählten Knoten irgendeine änderbare Reihenfolge. Mit dem Operator `|` lassen sich solche Mengen vereinigen. Zum Beispiel liefern die Ausdrücke `image | table` und `table | image` die gleiche Menge aller `image`- und `table`-Kindelemente.

## Abkürzungen

Für bestimmte Pfade gibt es in XPath eine abgekürzte Syntax. So wird immer die `child`-Achse angenommen, wenn keine explizite Achse angegeben wurde (vgl. obige Beispiele). Darüber hinaus kann die häufig benötigte `attribute`-Achse durch das Zeichen `@` abgekürzt werden. So greifen z. B. `@id` auf das `id`-Attribut des Kontextknotens und `section/@id` auf die `id`-Attribute aller `section`-Kindelemente zu.

Weitere Abkürzungen sind `.` für `self::node()`, d. h. ein Pfad zum Kontextknoten selbst, `..` für `pa-`

`rent::node()`, d. h. ein Pfad zum Elternknoten, und `//` für `/descendant-or-self::node()`, d. h. eine Abkürzung für den Zugriff auf Knoten, die Nachkommen des vorherigen Schrittes sind. Diese letzte Abkürzung wird häufig zusammen mit `.` verwendet, wenn die Nachkommen des Kontextknotens gefragt sind: `./image` findet im aktuellen Unterbaum alle `image`-Elemente, unabhängig von ihrer Tiefe im Baum; `//image` würde dagegen den gesamten Baum durchsuchen. Vorsicht: die Auswertung solcher Ausdrücke ist in der Regel sehr rechenaufwändig. Wenn möglich, sollte die Nachkommen-Achse daher vermieden werden.

## Die Wahl des richtigen Weges

Nun möchte man in den meisten Fällen nur bestimmte Knoten mit gewissen Eigenschaften auswählen. Zu diesem Zweck stellt XPath Prädikate bereit. Ein Prädikat besteht aus einem oder mehreren Ausdrücken, die in eckigen Klammern hinter dem Knotentest eines Schrittes notiert werden. Liefert die Berechnung des Ausdrucks den Wert *wahr*, so wird der betrachtete Knoten in das Ergebnis aufgenommen, anderenfalls wird er aus der Knotenmenge ausgeschlossen. Typischerweise handelt es sich bei den Ausdrücken innerhalb eines Prädikates um logische Ausdrücke, obwohl dort auch jede andere Art von Ausdruck erlaubt ist, dessen Wert gegebenenfalls automatisch in einen booleschen Wert konvertiert wird.

<code>child</code>	Kinder
<code>parent</code>	Eltern
<code>descendant</code>	Nachkommen (Kinder, Enkel, ...)
<code>ancestor</code>	Vorfahren (Eltern, Großeltern, ...)
<code>preceding</code>	Knoten, die dem Kontextknoten vorangehen
<code>following</code>	Knoten, die dem Kontextknoten folgen
<code>preceding-sibling</code>	vorherige Geschwister
<code>following-sibling</code>	nachfolgende Geschwister
<code>descendant-or-self</code>	Nachkommen plus Kontextknoten
<code>ancestor-or-self</code>	Vorfahren plus Kontextknoten
<code>self</code>	der Kontextknoten selbst
<code>attribute</code>	Attribute
<code>namespace</code>	Namensräume

Tabelle 2: XPath-Achsen

Zur Veranschaulichung folgen wieder einige Beispiele: `task[@level>5]` wählt nur die `task`-Kindelemente aus, deren `level`-Attribut einen Wert größer als 5 besitzt; `task[@level]` wählt diejenigen `task`-Kindelemente aus, die überhaupt ein `level`-Attribut besitzen, `task[@level>5][1]` wählt aus denjenigen mit `level` größer als 5 nur noch das erste. Dieses letzte Beispiel zeigt zum einen die Verwendung mehrerer Prädikate hintereinander, zum anderen, dass eine Zahl innerhalb eines Prädikates die Position des ausgewählten Knotens innerhalb der zuvor ausgewählten Knotenmenge angibt und daher nur diesen Knoten auswählt.

Die Position eines Knotens hängt dabei von der Richtung der verwendeten Achse ab. Alle Achsen, die Knoten auswählen, die dem Kontextknoten folgen, sind vorwärts gerichtet, alle anderen rückwärts gerichtet. In rückwärts gerichteten Achsen werden die ausgewählten Knoten in umgekehrter Reihenfolge nummeriert. Das bedeutet, dass `precedingsibling::*[1]` den unmittelbaren Vorgänger und `ancestor::*[1]` den unmittelbaren Vorfahren auswählen. Die Position im Prädikat bezieht sich hier immer auf den verwendeten Schritt. Wird ein solches Prädikat auf eine ganze Knotenmenge angewendet, gilt wieder die Originalreihenfolge der Knoten im Dokument. Der Ausdruck `(ancestor::*)[1]` liefert also das Wurzelement.

### Pfadfinderausrüstung

XPath definiert eine Basisfunktionsbibliothek, die alle XPath-Implementierungen (z. B. in XSLT) zur Verfügung stellen müssen. (XSLT fügt noch eigene Funktionen hinzu, sodass nicht sofort ersichtlich ist, in welcher Spezifikation man im Zweifelsfall nach einer Funktionsdefinition suchen muss.)

Zu den wichtigsten Funktionen gehören Zeichenkettenfunktionen, wie z. B. `contains`, `substring`, `substring-after` und `-before` und `translate`, des Weiteren einfache Zahlenfunktionen wie `round`, `floor` und `ceiling`, Funktionen zum expliziten Konvertieren in andere Datentypen und Funktionen für Knotenmengen wie z. B. `count`. Die genaue Beschreibung der jeweiligen Funktionalität kann in der Spezifikation nachgelesen werden.

Zwei Funktionen sollen jedoch noch kurz erwähnt werden: `position` und `last`. Die erste liefert die Position des Kontextknotens innerhalb der Kontextknotenmenge, die zweite liefert die Größe dieser Kontextknotenmenge (d. h. die Position des letzten Elementes). Innerhalb eines Prädikats lässt sich so das letzte Element einer Knotenmenge bestimmen: `section[position()=last()]` oder kürzer `section[last()]` wählen jeweils das letzte `section`-Kindelement aus.

### Erkundungsgebiet: XPath 2.0

Das W3C entwickelt derzeit die Nachfolgeversion 2.0 von XPath [XPath2], zusammen mit XSLT 2.0 und XQuery 1.0. In zunehmendem Maße modularisiert das W3C seine Spezifikationen, sodass sich die XPath-Syntax, die in XPath vorhandenen Funktionen, die formale Semantik und das Datenmodell in verschiedenen Dokumenten wiederfinden.

Die wichtigste Neuerung in XPath 2.0 ist der Sequenztyp als grundlegender Typ. Jede Operation liefert im Prinzip eine Sequenz, wobei bei einer ein-elementigen Sequenz diese nicht von dem enthaltenen Element unterschieden werden kann. Eine Sequenz kann Werte der bekannten einfachen Typen oder Knoten enthalten. Die Elemente einer Sequenz können durch Kommata getrennt aufgezählt werden; die leere Sequenz muss durch ein Paar leerer Klammern () notiert werden. Sequenzen sind immer flach, d. h. es existieren keine geschachtelten Sequenzen. So beschreiben die Notationen (`$s`, 3, 2, 1) und (1, (2, 3), ()), ((2), 1)) die gleiche Sequenz der Zahlen 1, 2, 3, 2, 1, vorausgesetzt, die Variable `s` enthält als Wert die Sequenz 1, 2.

Längere Sequenzen ganzer Zahlen lassen sich durch den Bereichs-Operator `to` angeben: `1 to 100` liefert die Zahlen von 1 bis 100 in einer Sequenz. Mit Hilfe des `for`-Operators können Ausdrücke gebildet werden, die direkt auf die einzelnen Sequenzelemente zugreifen und daraus eine neue Sequenz konstruieren. Beispielsweise liefert

```
for $x in (1 to 10) return $x*$x
```

die Sequenz der ersten 10 Quadratzahlen. Bedingte Ausdrücke (die Entsprechung des Java-Operators `?:`) lassen sich zukünftig auch in XPath darstellen, beispielsweise in

```
if (@level) then @level else 5
```

Eine größere Umstellung bedeutet die Einführung der starken Typisierung basierend auf dem in XML Schema definierten Typsystem. Viele der in XPath 1.0 implizit vorgenommenen Typumwandlungen müssen nun explizit hingeschrieben werden. Beispielsweise führt nun `concat("Nr. ", position())` zu einem Typfehler. Da die Funktion `position` eine Zahl liefert, muss diese nun explizit in eine Zeichenkette umgewandelt werden: `concat("Nr. ", string(position()))`. Die Einführung dieser strengen Regeln sorgt für viel Diskussion innerhalb der XSLT-Gemeinde, und es ist nicht auszuschließen, dass es hier bis zur endgültigen Verabschiedung der Spezifikation noch Änderungen geben wird.

Weitere Neuerungen sind erweiterte Pfadausdrücke, etwa `book/(chapter|section)/title`, Operatoren für die Differenz und den Durchschnitt von

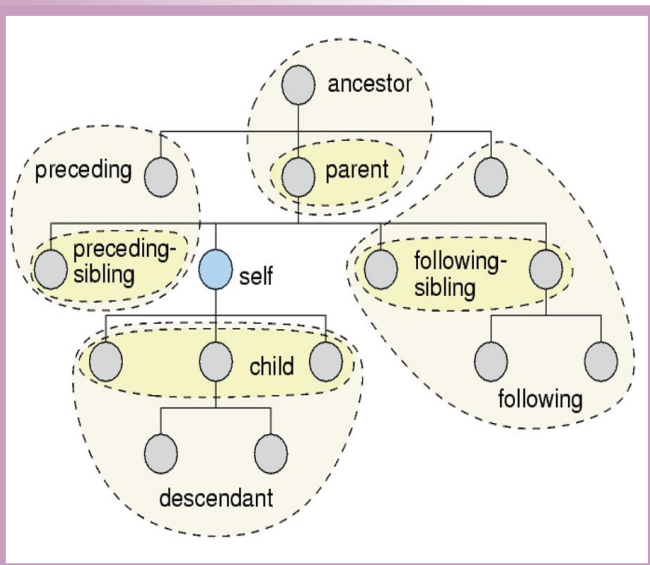


Abb. 1: Navigations-Achsen



Knotenmengen, die Quantoren **all** (für alle) und **exists** (es gibt ein) sowie mächtigere Vergleichsmöglichkeiten, z. B. bezüglich der Knotenreihenfolge oder der Knotenidentität.

Der Umfang der in XPath zur Verfügung stehenden Bibliothek von grundlegenden Funktionen ist sehr stark angewachsen und wird nun in einer separaten Spezifikation beschrieben. Hinzugekommen sind vor allem Funktionen für Datumswerte, für die Verarbeitung von Sequenzen sowie einige erweiterte Zeichenketten- und Knotenfunktionen.

## Den Kompass selbst in der Hand: XPath direkt

Doch zurück zur Version 1 von XPath. Haupteinsatzgebiet dieser Sprache ist wie erwähnt XSLT. Einige Java-APIs bieten jedoch die Möglichkeit an, XPath-Anfragen auf einer Java-Repräsentation eines XML-Dokuments auszuführen.

Das interessanteste API stellt die OpenSource-Entwicklung Jaxen [Jaxen] zur Verfügung. Es handelt sich hier um eine reine XPath-Implementierung, die auf verschiedenen Objektmodellen (namentlich W3C-DOM, dom4j, JDOM und EXML) benutzt werden kann. Das wichtigste Interface ist `org.jaxen.XPath`, für das konkrete, zum jeweiligen Objektmodell gehörende Implementierungsklassen bereitstehen (beispielsweise `org.jaxen.dom.DOMXPath` für W3C-DOM). Im Konstruktor der gewählten Klasse übergibt man den auszuwertenden XPath als String und kann anschließend über die Methode `evaluate` mit dem Kontextknoten als Parameter das erzeugte XPath-Objekt benutzen:

```
try {
    XPath xpath =
        new DOMXPath(
            "/book/chapter[position() < 5]");
    Document doc = retrieveDocument();
    List results = (List) xpath.evaluate(doc);
}
catch (JaxenException e)
{ e.printStackTrace(); }
```

Jaxen benutzt dabei als generischen Typ sowohl für den Parameter als auch für das Ergebnis `java.lang.Object`. Um auf den tatsächlichen Wert zugreifen zu können (dessen Typ vom XPath-Ausdruck abhängt), muss man das Ergebnis noch geeignet in den gewünschten Typ umwandeln. Hier helfen spezi-

elle Methoden, die unter Berücksichtigung der XPath-Typumwandlungsregeln sofort ein Objekt des gewünschten Typs zurückliefern. Spielen Namensräume innerhalb des XPath-Ausdrucks eine Rolle oder werden Variablen referenziert, müssen vor der Auswertung geeignete Objekte (`NamespaceContext` und `VariableContext`) konstruiert und übergeben werden. Insgesamt bietet Jaxen eine einfach zu verstehende Programmierschnittstelle, die universell für verschiedene Objektmodelle benutzt werden kann und zudem über eine gute Performance verfügt.

Zur zweiten Klasse von XPath-APIs gehören konkrete in Java geschriebene XSLT-Implementierungen, deren XPath-Bestandteil auch separat benutzt werden kann. Zu nennen wären unter anderem hier Xalan von Apache [Xalan], Saxon von Michael Kay [Saxon] oder `jd.xpath` von Johannes Döbler [jdxpath]. Während Xalan einfach mit Xerces zusammen arbeitet und auf einem DOM aufsetzt, muss man sich im Falle von Saxon und `jd.xpath` mit deren speziellen, aber effizienteren Datenstrukturen auseinander setzen.

Zu guter Letzt sei hier noch die kommende DOM-XPath-Spezifikation [DOMXPath] genannt, die Bestandteil von DOM Level 3 sein wird. Derzeit besitzt sie den Status eines Empfehlungskandidaten (candidate recommendation), ist also noch zwei Schritte von der endgültigen Verabschiedung entfernt. Mir sind derzeit keine Java-Implementierungen der dort spezifizierten Schnittstellen bekannt.

Die Vorgehensweise in DOM-XPath ähnelt der in anderen XPath-Implementierungen: Man erstellt ein Objekt, das einen XPath-Ausdruck repräsentiert

(hier wird ein DOM-Dokument als Factory genutzt), und kann anschließend dieses XPath-Objekt für verschiedene Kontextknoten auswerten. Da DOM sprachunabhängig spezifiziert ist, wirken auch hier die entsprechenden Java-Interfaces etwas schwerfällig. Die Entscheidung, ob man später die entsprechenden DOM-Interfaces oder lieber spezielle APIs wie Jaxen benutzt, hängt stark davon ab, ob allein in Java oder auch in anderen Programmiersprachen entwickelt wird. Im zweiten Fall macht sich die Sprachunabhängigkeit von DOM positiv bemerkbar. Aber das ist schon wieder eine andere Kolumne ...

## Links

**[DOMXPath]** Document Object Model (DOM) Level 3 XPath Specification,

<http://www.w3.org/TR/DOM-Level-3-XPath/>

**[Jaxen]** OpenSource-XPath-Implementierung Jaxen,

<http://jaxen.org/>

**[jdxpath]** XPath-Implementierung `jd.xpath` von Johannes Döbler,

<http://www.aztecrider.com/xpath/>

**[Saxon]** XSLT-Implementierung Saxon von Michael Kay, <http://saxon.sourceforge.net/>

**[Xalan]** XSLT-Implementierung Xalan von Apache,

<http://xml.apache.org/xalan-j/>

**[XPath]** XML Path Language (XPath) Version 1.0,

<http://www.w3.org/TR/xpath>

**[XPathd]** XPath 1.0, deutsche kommentierte Übersetzung, <http://www.edition-w3c.de/TR/xpath>

**[XPath2]** XML Path Language (XPath) 2.0,

<http://www.w3.org/TR/xpath2>



**Oliver Becker** ist seit 1998 wissenschaftlicher Mitarbeiter am Institut für Informatik der Humboldt-Universität zu Berlin. Er hat die XPath-Spezifikation ins Deutsche übersetzt und kommentiert und arbeitet derzeit an seinem Lieblingsprojekt, dem STX-Prozessor Joost. E-Mail: [obecker@informatik.hu-berlin.de](mailto:obecker@informatik.hu-berlin.de).



## Weiterführende Informationsquellen

Weiterführende Informationsquellen: <http://www.edition-w3c.de/TR/xpath>