trivadis
Part of Accenture

Update XSLT

Die interessantesten Features von XSLT/XPath 2.0

) Schulung)

) Beratung)

) Entwicklung)

) Artikel)

AUTOR

Matthias Born
Orientation in Objects GmbH

Veröffentlicht am: 10.3.2008

ABSTRAKT

Seit dem sehr weit verbreiteten XSLT 1.0 (1999) hat sich einiges getan. Mehr oder weniger offensichtliche Lücken und Wünsche der Anwender wurden als Erweiterungen in die XSLT Prozessoren eingebaut und fanden nun Eingang in die Spezifikation von XSLT/XPath 2.0 (Januar 2007). Dieser Artikel gibt einen kurzen Überblick über die neuen Features dieser Spezifikationen, insbesondere die konzeptionellen Änderungen sowie sprach- und inhaltliche Anpassungen von XSLT/XPath 2.0.

Trivadis Germany GmbH

Weinheimer Str. 68
D-68309 Mannheim

Tel. +49 (0) 6 21 - 7 18 39 - 0
Fax +49 (0) 6 21 - 7 18 39 - 50

www.oio.dekontakt@trivadis.com

EINLEITUNG

Obwohl weit verbreitet und in den verschiedensten Aufgabenbereichen im Einsatz, konnten XSLT/XPath 1.0 nicht wirklich als ausgereifte Technologien bezeichnet werden. Das beweisen die vielen unverzichtbaren Erweiterungen, welche die XSLT-Prozessoren in der Regel mit sich brachten (`<redirect:write />`, `<result-document />`), oder von EXSLT.org, eine "community initiative to provide extensions to XSLT" zur Verfügung gestellt wurden. Leider waren selbst diese Erweiterungen in der Realität häufig nicht ausreichend, weshalb jeder XSLT-Entwickler Kenntnisse über die spezifischen Erweiterungsmöglichkeiten des eingesetzten XSLT-Prozessor besitzen musste.

Alleine der Vergleich des Umfangs der Spezifikationen von XSLT/XPath zeigt, dass sich seit der Version 1.0 so einiges getan hat[1]. Ein umfassendes Update ist an dieser Stelle wohl unmöglich, deshalb wird sich dieser Artikel auf die interessantesten Features beschränken.

Die vollständigen Beispiele aus diesem Artikel finden Sie hier.

GLIEDERUNG

- Konzeptionelle Änderungen
- Neues Verhalten bei `<xsl:value-of />`
- Gruppieren
- Mehrere Zieldokumente generieren
- Definition eigener Funktionen
- Stringverarbeitung mit Regulären Ausdrücken
- Neuerungen in XPath 2.0
- XML Schema Support
- Die XSLT/XPath 2.0 Prozessoren

KONZEPTIONELLE ÄNDERUNGEN

Wie von 1.0 gewohnt, wurde auch XSLT 2.0 zur Verwendung zusammen mit XPath 2.0 entwickelt. Dazu kommt eine Technologie namens XQuery, quasi das SQL für XML-Dateien und -Datenbanken. Genau genommen ist XPath 2.0 eine Untermenge von XQuery 1.0, was zu einer Vereinheitlichung der Grundlagen aller drei Technologien geführt hat. Diese Spezifikationen sind jetzt viel stärker miteinander verbunden, als es XSLT/XPATH bisher waren.

Wenn Sie sich bisher schon mit XSLT 1.0 beschäftigt haben, müssen Sie sich auf zwei wesentliche Neuerungen einstellen: Statt wie bisher operiert XSLT nicht mehr auf den so genannten NodeSets, sondern auf Sequenzen. Und aus dem häufig kritisierten Result Tree Fragment (sieht aus wie ein XML-Baum, ist aber keiner) werden jetzt echte temporäre Bäume bzw. Sequenzen, die anschließend - im Gegensatz zum RTF - sofort weiter verarbeitet werden können.

DATENMODELL BASIEREND AUF SEQUENZEN

Wie bereits erwähnt, ändert sich mit XSLT/XPath 2.0 das grundlegende Datenmodell: XPath-Ausdrücke liefern keine NodeSets mehr zurück, sondern Sequenzen aus Knoten in Document-Order (Reihenfolge wie in der XML-Quelle).

Sequenzen unterscheiden sich gegenüber dem alten NodeSet in folgenden Punkten:

- Sequenzen haben eine definierte Reihenfolge. Dies kann von der Dokument Order abweichen
- Sequenzen dürfen Duplikate enthalten
- Sequenzen können heterogen sein, neben Knoten können sie auch beliebige atomare Werte enthalten
- Sequenzen können von Hand erstellt werden
- Sequenzen bestehen aus 0 bis n Items

Diese Änderungen lassen sich wie folgt zusammenfassen: Bisher hat XSLT quasi ausschließlich auf der Basis von Knoten gearbeitet. Durch die neuen Sequenzen ist das NodeSet aus XSLT/XPath 1.0 somit nur noch ein Spezialfall einer Sequenz.

Zwei einfache Beispiele:

Der erste XPath-Ausdruck liefert eine Sequenz aus allen Knoten "book", die Kinder des Root-Elements "books" sind.

```
//books/book
```

Beispiel 1: "Normaler" XPath-Ausdruck

Der zweite XPath-Ausdruck benutzt den Komma-Operator und die Klammern, um händisch eine Sequenz zu erstellen. Diese besteht aus den drei Integern 1,2 und 3, den Strings `eins` und `zwei`, dem Attributknoten `id`, den Elementknoten `foo` und - um die Möglichkeit von Duplikaten zu veranschaulichen - Attribut `id` und Element `foo` noch einmal.

```
( 1, 2, 3, 'eins', 'zwei', @id, foo, @id, foo )
```

Beispiel 2: Neuer XPath-Ausdruck

Wie sehr die Sequenzen die Arbeit mit XSLT/XPath 2.0 vereinfachen, soll folgendes Beispiel veranschaulichen:

```
<address>
  <street>Weinheimer Straße 68</street>
  <zip>68309</zip>
  <city>Mannheim</city>
</address>
```

Beispiel 3: Vorteil von Sequenzen: XML-Quelle

```
<xsl:template match="address">
  <xsl:value-of select="(street, 'in', zip, city)" separator=" "/>
</xsl:template>
```

Beispiel 4: Vorteil von Sequenzen: Stylesheet

Das Ergebnis: "Weinheimer Straße 68 in 68309 Mannheim".

Dabei werden zwei weitere Neuerungen benutzt, die `<xsl:value-of>` betreffen: Die Ausgabe aller Items einer Sequenz und deren Trennung durch einen Separator. Offensichtlich ist hier der Vorteil des Einzeilers gegenüber dem entsprechenden XSLT 1.0 Konstrukt mit mehreren `<value-of>`- und `<xsl:text>`-Anweisungen:

```
<xsl:template match="address">
  <xsl:value-of select="street"/>
    <xsl:text> in </xsl:text>
    <xsl:value-of select="zip"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="city"/>
</xsl:template>
```

Beispiel 5: Vorteil von Sequenzen: XSLT 1.0 Gegenbeispiel

RTF VS. TEMPORÄRE BÄUME

Eine weitere Neuerung in XSLT 2.0 betrifft das Result Tree Fragment (RTF) aus XSLT 1.0. Zur Erinnerung: Der gesamte Variablen- oder Template-Inhalt, der mittels XSLT 1.0 erzeugt/generiert wurde, war ein eigener Datentyp namens RTF. Für die Generierung von XML-Daten hieß das: Was aussieht wie ein XML-Baum muss noch lange keiner sein.

Tatsächlich war es so, dass von XSLT 1.0 generierte XML-Strukturen oder händisch erstellte Variablen mit XML-Elementen nicht weiter verarbeitet werden konnten. Es konnte noch nicht einmal darauf zugegriffen werden. Für XSLT 1.0 war das RTF einfach kein Baum.

Sollten solche "XML"-Fragmente weiter verarbeitet werden, musste auf Erweiterungen der XSLT-Prozessoren zurückgegriffen werden. Ein typisches Beispiel ist die Funktion `node-set()`, wie sie von den meisten XSLT-Prozessoren in der einen oder anderen Form implementiert wurde, um das RTF wieder in einen Baum zu wandeln. Das diese Erweiterung eine der ersten war, beweist zusätzlich, dass das RTF eine echte Schwachstelle in der Spezifikation von XSLT 1.0 darstellte.

In XSLT 2.0 wird das RTF durch echte temporäre Bäume ersetzt. Diese verhalten sich wie jeder andere XML-Baum, so dass jetzt XPath-Abfragen auf diesen abgesetzt oder nach belieben manipuliert werden können. Dadurch können jetzt Transformationen, für die bisher mehrere Verarbeitungsschritte/Stylesheets benötigt wurden, innerhalb einer einzigen Transformation abgearbeitet werden.

NEUES VERHALTEN BEI <XSL:VALUE-OF />

Dieses Element ist wohl das markanteste Beispiel für Verbesserungen gegenüber der 1.0-Version. Haben Sie sich auch schon immer gefragt, warum `<xsl:value-of />` nur den Wert des ersten Knotens ausgibt, obwohl der XPath in der `select`-Anweisung mehr als ein Element zurückliefert?

Damit ist jetzt Schluss. In XSLT 2.0 verhält sich `<xsl:value-of />` endlich so, wie man es erwarten würde: Aus einer Menge von Knoten, deren Anzahl größer als 1 ist, werden jetzt alle Werte per Default separiert durch ein Leerzeichen ausgegeben. Neu ist auch das Attribut `separator`, mit dem das Leerzeichen durch beliebige andere Trenner ersetzt werden kann. Typisches Beispiel: Das Erstellen einer Komma-separierten Liste:

```
<teilnehmer>
  <name>Peter Mustermann</name>
  <name>Helga Demonstration</name>
  <name>Foo Bar</name>
</teilnehmer>
```

Beispiel 6: Komma-separierte Liste: XML-Quelle

```
<xsl:template match="teilnehmer">
  <xsl:for-each select="name">
    <xsl:value-of select="."/>
    <xsl:if test="position() != last()">, </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Beispiel 7: Komma-separierte Liste: XSLT 1.0

```
<xsl:template match="teilnehmer">
  <xsl:value-of select="name" separator=", "/>
</xsl:template>
```

Beispiel 8: Komma-separierte Liste: XSLT 2.0

Gleiches gilt im Übrigen auch für das Attribute Value Template (AVT, die geschweiften Klammern {}) mit denen das Ergebnis von XPath-Ausdrücken Attributen zugewiesen wird. Das Leerzeichen als Separator ist fix, ein Attribut `xsl:separator` gibt es an dieser Stelle leider nicht.

GRUPPIEREN

In XSLT 1.0 konnte man durch das mehrfache Durchsuchen des gesamten Dokumentes jeden XSLT-Prozessor in die Knie zwingen. Verbesserung brachte das Gruppieren nach Steve Muench[2], ein Algorithmus, der durch die Anwendung von `<xsl:key>`, der `key()`- und der `generate-id()`-Funktion Verbesserung der Performance versprach.

In XSLT 2.0 gibt es endlich ein eigenes Element für diese Aufgabenstellung: `<xsl:for-each-group>`. Im einfachsten Fall bestimmt man mit dem `select`-Attribut die zu gruppierenden Elemente und mittels `group-by` wird das Kriterium bestimmt, nach dem gruppiert werden soll. Alle Elemente, für die XPath ein identisches Ergebnis zurück liefert, werden einer Gruppe zugeordnet.

Für den einfachen Zugriff auf die Gruppierungs-Informationen stehen zwei Funktionen zur Verfügung: `current-group()` enthält alle Elemente, die zur aktuellen Gruppe gehören und `current-grouping-key()` enthält den Wert des aktuellen Gruppierung-Schlüssels.

Nachfolgend ein Beispiel für die "normale" Gruppierung mittels `group-by`:

```
<entries>
  <entry genre="classic">c1</entry>
  <entry genre="folk">f1</entry>
  <entry genre="rock">r1</entry>
  <entry genre="classic">c2</entry>
  <entry genre="folk">f2</entry>
  <entry genre="rock">r2</entry>
  <entry genre="classic">c3</entry>
  <!-- ... -->
</entries>
```

Beispiel 9: Gruppieren: XML-Quelle

Zum Vergleich hier noch einmal eine mögliche Lösung mit XSLT 1.0

```
<xsl:key name="entries" match="entry" use="@genre"/>
<!-- -->
<xsl:template match="entries">
  <result>
    <xsl:for-each select="entry[generate-id(.) =
                          generate-id(key('entries', @genre))]">
      <xsl:element name="{@genre}">
        <xsl:apply-templates select="key('entries',current()/@genre)"/>
      </xsl:element>
    </xsl:for-each>
  </result>
</xsl:template>
<!-- -->
<xsl:template match="entry">
  <xsl:copy>
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>
```

Beispiel 10: Gruppieren: XSLT 1.0 (Gruppieren nach Muench)

Und hier die XSLT 2.0 Lösung mit `<xsl:for-each-group>`:

```
<xsl:template match="entries">
  <result>
    <xsl:for-each-group select="entry" group-by="@genre">
      <xsl:element name="{current-grouping-key()}">
        <xsl:apply-templates select="current-group()"/>
      </xsl:element>
    </xsl:for-each-group>
  </result>
</xsl:template>
<!-- -->
<xsl:template match="entry">
  <xsl:copy>
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>
```

Beispiel 11: Gruppieren: XSLT 2.0

```
<result>
  <classic>
    <entry>c1</entry>
    <!-- ... -->
  </classic>
  <folk>
    <entry>f1</entry>
    <!-- ... -->
  </folk>
  <rock>
    <entry>r1</entry>
    <!-- ... -->
  </rock>
</result>
```

Beispiel 12: Gruppieren: Ergebnis

Doch das ist noch längst nicht alles. Anstelle von `group-by` können folgende Attribute verwendet werden:

- **group-adjacent**

  Hierbei werden die Elemente solange zu einer Gruppe zusammengefasst, wie der XPath in `group-adjacent` den gleichen Wert zurück liefert. Ändert sich dieser, wird eine neue Gruppe eröffnet. Hierbei wird die Reihenfolge der Elemente nicht verändert, es kann somit auch der Fall sein, dass es mehrere Gruppen mit den gleichen Gruppierungs-Kriterium gibt.

- **group-starting-with**

  Jedes Mal, wenn das in `group-starting-with` formulierte Muster gefunden wird, wird eine neue Gruppe geöffnet.

  Ein typisches Beispiel ist die Umwandlung von (X)HTML in ein stärker strukturiertes Format wie Docbook. Die Anweisung `group-starting-with="H1"` bewirkt, das alles bis zur nächsten H1 zu einer Gruppe, sprich Kapitel zusammengefasst wird. Auch hier wird von der eigentlichen Reihenfolge im Dokument nicht abgewichen.

- **group-ending-with**

  Vergleichbar mit `group-starting-with`, jedoch wird die Gruppe geschlossen, wenn das passende Muster gefunden wird.

ERZEUGEN MEHRERER ZIELDOKUMENTE UND -FORMATE

Eine große Schwäche zeigte XSLT 1.0 bei der Erzeugung mehrerer Zieldokumente. Ein Abweichen vom 1:1 Mapping (eine Quelle, ein Zieldokument) war nicht vorgesehen. Demzufolge reagierten auch die XSLT-Prozessoren-Hersteller mit den entsprechenden Erweiterungen: `<redirect:write>` bei XALAN oder `<saxon:output>` bei Saxon von Michael Kay.

Diesen Mangel behebt XSLT 2.0 mit dem Element `<xsl:result-document href="[Dateiname]">`, mit dem während einer einzigen Transformation gleichzeitig mehrere Zieldokumente erzeugt werden können.

Typisches Beispiel ist das Erzeugen mehrerer Webseiten aus einem einzigen XML-Dokument:

```
<documents>
  <document>Dokument 1</document>
  <document>Dokument2</document>
  <document>Dokument 3</document>
  <!-- ... -->
</documents>
```

Beispiel 13: Mehrere Zieldokumente: XML-Quelle

```
<xsl:template match="document">
  <xsl:variable name="path" select="concat(generate-id(),'.htm')"/>
  <xsl:result-document href="{$path}">
    <html>
      <head>
        <title>
          <xsl:apply-templates/>
        </title>
      </head>
      <body>
        <h1>
          <xsl:apply-templates/>
        </h1>
      </body>
    </html>
  </xsl:result-document>
</xsl:template>
```

Beispiel 14: Mehrere Zieldokumente: Stylesheet

Das Ergebnis sind n Dokumente mit folgendem Aufbau:

```
<html>
  <head>
    <title>Dokument 3</title>
  </head>
  <body>
    <h1>Dokument 3</h1>
  </body>
</html>
```

Beispiel 15: Mehrere Zieldokumente: Ergebnis

Das primäre Zieldokument (das nicht über `<xsl:result-document>` erzeugte) kann dabei als Linkliste o.ä. dienen.

Gleichzeitig mit der Möglichkeit, mehrere Zieldokumente während einer Transformation zu erzeugen, wurde auch das Element `<xsl:output>` noch einmal überarbeitet. So ist es jetzt möglich, mehrere Ausgabeformate zu definieren. Das wird erreicht, in dem `<xsl:output>` mehrmals angegeben und die verschiedenen Ausgabeformate durch das Attribut `name` unterschieden werden. Dieses benannte Ausgabeformat kann anschließend bei der Ausgabe sekundärer Zieldokumente über das Attribut `format` des Elementes `<xsl:result-document>` verwendet werden. Dadurch ist es XSLT 2.0 möglich, beliebig viele Zieldokumente in beliebig vielen Zielformaten gleichzeitig zu generieren.

Zusätzlich gibt es noch einige neue Eigenschaften für das Element `<xsl:output>`, u.a. die zusätzliche Ausgabemethode `method="XHTML"` und Attribut `use-character-maps`. Character Maps können verwendet werden, um Sonderzeichen zu HTML-Entitäten oder bspw. deutschen Umlaute und "ß" in die ASCII-Entsprechungen umzumappen. Außerdem kann man damit auch Platzhalter in der XML-Quelle bzw. im Stylesheet in XML-ungeeignete Zeichenkombinationen ummappen, um z.B. Formate wie JSP-Dateien aus XML-Dokumenten zu generieren.

```
<!-- Character Map fuer Umlaute -->
<xsl:character-map name="umlaut">
  <xsl:output-character character="Ä" string="Ae" />
  <!-- ... -->
</xsl:character-map>
<!-- Character Map fuer HTML-Entities -->
<xsl:character-map name="entities">
  <xsl:output-character character="&amp;#x20AC;" string="&amp;amp;euro;" />
  <!-- ... -->
</xsl:character-map>
<!-- Character Map fuer JSP-Platzhalter -->
<xsl:character-map name="jsp">
  <xsl:output-character character="#@# " string="&amp;lt;%@" />
  <xsl:output-character character="#?# " string="%&amp;gt;" />
  <!-- ... -->
</xsl:character-map>
<!-- Alle drei Character Maps zu einer kombinieren -->
<xsl:character-map name="combined-CM" use-character-maps="umlaute entities jsp"/>
<!-- Verwendung der kombinierten Character Map fuer die Ausgabe-->
<xsl:output method="text" use-character-maps="combined-CM" encoding="UTF-8" />
```

Beispiel 16: Kombiniertes Beispiel für Character Maps

Wie am Beispiel zu erkennen, ist es auch möglich, die Character Maps zu modularisieren oder mehrere zu einer zusammenzufassen.

DEFINITIONEN EIGENER FUNKTIONEN

Ein weiteres Feature von XSLT 2.0 ist die Definition von eigenen Funktionen. Wer jetzt denkt, diesen Mechanismus gab es auch in 1.0 über die so genannten Named Templates hat nur zum Teil recht.

Stylesheet Functions unterscheiden sich wesentlich von Named Templates. Der wichtigste Punkt ist der Aufruf: Stylesheet Functions können aus jedem beliebigen XPath-Ausdruck heraus aufgerufen werden. Anders ausgedrückt: Stylesheet Functions erweitern die XPath-Funktionsbibliothek.

Es gibt jedoch noch mehr Unterschiede: Für die Verwendung von Named Templates gilt die Empfehlung, alle benötigten Informationen mittels Parameter zu übergeben und sich nicht auf den möglicherweise wechselnden Kontext zu verlassen.

Stylesheet Functions hingegen verlangen die Übergabe aller Daten mittels Parameter, auf den Aufrufkontext haben XSLT-Anweisungen innerhalb der Funktion keinen Zugriff. Ebenso ist es nicht erlaubt, die Parameter mit Default-Werten zu belegen.

Ferner ist zu beachten: Um Namenskonflikte mit evt. bereits vorhanden XPath-Funktionen zu vermeiden, wird vom Stylesheet-Autor verlangt, die Funktionen mit einem eigenen Namespace zu versehen.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:str="http://example.com/namespace"
  version="2.0" exclude-result-prefixes="#all">
  <!-- -->
  <xsl:function name="str:reverse-xslt" as="xs:string">
    <xsl:param name="sentence" as="xs:string" />
    <xsl:choose>
      <xsl:when test="contains($sentence, ' ')">
        <xsl:sequence
          select="concat( str:reverse-xslt(substring-after($sentence, ' ') ),
                ' ',
                substring-before($sentence, ' ') )" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:sequence select="$sentence" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>
  <!-- -->
<xsl:stylesheet>
```

Beispiel 17: Stylesheet Function: Definition mit XSLT 2.0

```
<xsl:template match="/">
  <output>
    <xsl:value-of select="str:reverse-xslt('DOG BITES MAN')" />
  </output>
</xsl:template>
```

Beispiel 18: Stylesheet Function: Aufruf der Funktion

Dieses Beispiel stammt direkt aus der Spezifikation von XSLT 2.0. Und es gibt sogar noch eine weitere Lösungsmöglichkeit mit der `if ... then ... else`-Anweisung aus XPath 2.0. Dazu mehr im Abschnitt über XPath 2.0.

STRINGVERARBEITUNG MIT REGULÄREN AUSDRÜCKEN

DAS XSLT 2.0 ELEMENT <XSL:ANALYZE-STRING>

Neu in XSLT 2.0 ist das Element `<xsl:analyze-string>`. Dieses Element verarbeitet einen String, selektiert durch den XPath im Attribut `select`, entsprechend eines regulären Ausdrucks, der im Attribut `regex` festgelegt wird.

Zusätzlich besitzt `<xsl:analyze-string>` zwei Kindelemente: `<xsl:non-matching-substring>` und `<xsl:matching-substring>`, die für die Verarbeitung der Teilstrings zuständig sind, die dem regulären Ausdruck entsprechen bzw. eben nicht entsprechen. Beide Kindelemente sind optional, sinnvollerweise sollte natürlich mindestens eines von beiden vorhanden sein.

Ein typisches Beispiel für den Einsatz regulärer Ausdrücke ist die Aufteilung eines Textdokumentes in einzelne Absätze von HTML (`<p></p>`) oder Docbook (`<para></para>`):

Die hier verwendete Funktion `unparsed-text()` ist ebenfalls neu und bildet das Gegenstück zur Funktion `document()`, um jetzt auch auf nicht strukturierte, also auch auf Nicht-XML-Dateien zugreifen zu können.

```
Lorem ipsum dolor sit amet,...
Duis autem vel eum iriure dolor ...
Ut wisi enim ad minim veniam,...
```

Beispiel 19: RegEx: Textdokument lorem.txt

```
<xsl:template match="/">
  <xsl:variable name="text" select="unparsed-text('lorem.txt','UTF-8')" />
  <lorem>
    <xsl:analyze-string select="$text" regex="\r\n">
      <xsl:non-matching-substring>
        <para>
          <xsl:value-of select="." />
        </para>
      </xsl:non-matching-substring>
      <xsl:matching-substring>
        <!-- Nicht von Interesse -->
      </xsl:matching-substring>
    </xsl:analyze-string>
  </lorem>
</xsl:template>
```

Beispiel 20: RegEx: Stylesheet zur Umwandlung von lorem.txt in XML

```
<lorem>
   <para>Lorem ipsum dolor sit amet,...</para>
   <para>Duis autem vel eum iriure dolor ...</para>
   <para>Ut wisi enim ad minim veniam,...</para>
</lorem>
```

Beispiel 21: RegEx: Ergebnis - lorem.txt als XML-Dokument

Reguläre Ausdrücke bieten zudem die Möglichkeit, Teilausdrücke mittels Klammern `()` zu Gruppen zusammen zu fassen. In XSLT 2.0 kann auf diese Gruppen mit der Funktion `regex-group(n)` zugegriffen werden.

Als Beispiel bietet sich hier die Umwandlung von Datumsangaben aus der in Deutschland gebräuchlichen Form in die von W3C XML Schema genutzte und als Datentyp zur Verfügung stehende Form nach ISO/DIN:

```
<termin>12.3.2007</termin>
<termin>1.3.2007</termin>
<termin>1.11.2007</termin>
```

Beispiel 22: RegEx: Umzuformatierende Datumsangaben

```
<termin>
  <xsl:analyze-string select="." regex="([0-9]+)\.([0-9]+)\.([0-9]+)">
    <xsl:matching-substring>
      <xsl:value-of select="format-number(xs:integer(regex-group(3)),'0000')"/>
      <xsl:text>-</xsl:text>
      <xsl:value-of select="format-number(xs:integer(regex-group(2)),'00')"/>
      <xsl:text>-</xsl:text>
      <xsl:value-of select="format-number(xs:integer(regex-group(1)),'00')"/>
    </xsl:matching-substring>
    <xsl:non-matching-substring/>
  </xsl:analyze-string>
</termin>
```

Beispiel 23: RegEx: Umwandlung zu Datumsformat nach W3C XML Schema

```
<termin>2007-12-03</termin>
<termin>2007-01-03</termin>
<termin>2007-01-11</termin>
```

Beispiel 24: RegEx: Ergebnis

DIE XPATH 2.0 FUNKTIONEN FÜR REGULÄRE AUSDRÜCKE

Seitens XPath 2.0 gibt es drei neue Funktionen, um mit regulären Ausdrücken sehr einfach komplexe Textanalysen durchzuführen.

Die drei Funktionen sind:

- `tokenize()`
- `matches()`
- `replace()`

Der Mehrwert der neuen Funktionen kann sehr schön am Beispiel der Funktion `tokenize()` veranschaulicht werden. In den folgenden Beispielen sollen die Wörter eines Strings gezählt werden, zuerst als 1.0, anschließend als 2.0 Lösung:

In XSLT 1.0 wird ein Named Template benötigt, mit dem String als initialen Parameter. Dieses Template ruft sich anschließend mit dem um ein Wort gekürzten String solange rekursiv auf, bis der String leer ist. Gleichzeitig wird pro Durchlauf eine Zähler-Variable um 1 erhöht. Stoppt die Rekursion, wird der letzte Wert der Variablen ausgegeben:

```
<xsl:variable name="keywords">XSLT XPath XQuery XLink XPointer</xsl:variable>
<!-- -->
<xsl:template match="/">
  <result>
    <xsl:call-template name="count">
      <xsl:with-param name="text"
            select="concat(normalize-space($keywords),' ')"/>
    </xsl:call-template>
  </result>
</xsl:template>
<!-- -->
<xsl:template name="count">
  <xsl:param name="text"/>
  <xsl:param name="zahl" select="1"/>
  <xsl:variable name="rest" select="substring-after($text,' ')"/>
  <xsl:choose>
    <xsl:when test="$rest">
      <xsl:call-template name="count">
        <xsl:with-param name="text" select="$rest"/>
        <xsl:with-param name="zahl" select="$zahl + 1"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$zahl"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Beispiel 25: Wörter zählen: XSLT 1.0

Dank XSLT 2.0 kann das gesamte rekursive Named Template entfallen und die Funktionalität wird zu einem Einzeiler: Die Funktion `tokenize()` erwartet zwei Parameter, den zu verarbeitenden String und den regulären Ausdruck, nach dem der String zu zerlegen ist.

```
<xsl:value-of select="count(tokenize($keywords,' '))"/>
```

Beispiel 26: Wörter zählen: XSLT 2.0

NEUERUNGEN IN XPATH 2.0

FUNKTIONEN FÜR DATUM, ZEIT, DAUER UND DEREN FORMATIERUNG

Für die Verarbeitung von Datum, Zeit bzw. Zeiträumen war in XSLT/XPath 1.0 keinerlei Funktionalität vorgesehen. Es bestand nur die Möglichkeit, das aktuelle Datum beim Aufruf des Stylsheets als Parameter zu übergeben, oder während der Transformation über eine Erweiterung auszugeben. Gleiches galt für die Formatierung und Berechnung von Datum und Zeit, beides war nur mittels eigenen Erweiterungen, oder denen von EXSLT.org möglich.

Zum neuen Datenmodell von XSLT/XPath 2.0 gehören auch die Datentypen von W3C XML Schema, u.a. Typen für Datum, Uhrzeit und Zeiträume. Für deren Verarbeitung gibt es die passenden Funktionen in XPath (Manipulation, Konstruktoren) und XSLT (Formatierung).

Die Funktionen für die aktuellen Angaben lauten:

- `current-date()`
- `current-time()`
- `current-dateTime()`

Sollten z.B. Tag, Monat oder Jahr aus Datumsangaben extrahiert werden, mussten in XSLT 1.0 aufwendige `substring()`-Operationen durchgeführt werden. In XSLT/XPath 2.0 stehen auch dafür passende Funktionen zur Verfügung, u.a.:

- `day-from-date()`
- `hours-from-time()`
- `minutes-from-dateTime()`

Die Funktionen zum Formatieren dieser Daten liefert wiederum XSLT mit den Funktionen `format-date()` und `format-dateTime()`.

Beide Funktionen sind ähnlich aufgebaut und verfügen über zwei obligatorische Attribute: Datum bzw. Datums- und Zeitangabe und einen sog. Picture-String, der eine Schablone für die Formatierung der Angabe enthält.

Beispiele zur Formatierung von Datum und Zeitangaben:

- Funktion: `format-date('2008-02-28',[D].[M].[Y])`

  Ergebnis: `28.2.2008`
- Funktion: `format-date('2008-02-28',[D]. [MNn] [Y])`

  Ergebnis: `28. Februar 2008`

Die Möglichkeiten über die Formatierung mittels Picture-String sind umfangreich, die optionalen Attribute `language`,`calendar` und `country` ermöglichen zusätzlich eine internationalisierte Ausgabe, aber auch die Anpassung an die verschiedensten Kalender-Arten, wie bspw. gregorianische Zeitrechnung, den jüdischen Kalender u.s.w.

WEITERE NEUE XPATH-AUSDRÜCKE UND FUNKTIONEN

Neben den äußerst interessanten Funktionen zur Verarbeitung von Datums- und Zeitangaben liefert XPath 2.0 noch viel mehr neue Funktionen und Operatoren. Das zeigt schon der anfänglich erwähnte Größenvergleich beider Spezifikationen. Davon auch nur einen Bruchteil zu würdigen, würde den Umfang dieses Einführungsartikel bei Weitem sprengen. Deshalb nachfolgend nur noch je ein Beispiel für die `FOR-IN-RETURN`- und die `IF-THEN-ELSE`-Anweisung:

Zuerst einmal eine FOR-IN-RETURN-Anweisung, die innerhalb eines Ausdrucks `<item>`-Elemente verarbeitet, in dem sie deren Anzahl mit dem Preis multipliziert und schließlich die Gesamtsumme über alle Ergebnisse der Multiplikationen bildet:

```
<item id="01">
  <quantity>2</quantity>
  <price>99.95</price>
</item>
<item id="02"> ... </item>
```

Beispiel 27: FOR-Expression: Quelle

```
<xsl:value-of select="
  sum( for $i in //item
    return xs:decimal($i/price) * xs:integer($i/quantity)
  )
"/>
```

Beispiel 28: FOR-Expression: Stylesheet

Das zweite Beispiel ist die bereits bekannte Stylesheet Function `str:reverse`, hier allerdings formuliert mit der XPath-Anweisung `IF-THEN-ELSE` anstatt `<xsl:choose>`:

```
<xsl:function name="str:reverse-xpath" as="xs:string">
  <xsl:param name="sentence" as="xs:string" />
  <xsl:sequence select=" if (contains($sentence, ' '))
    then concat(str:reverse-xpath(substring-after($sentence, ' ')),
         ' ',
         substring-before($sentence, ' ')
       )
   else $sentence" />
</xsl:function>
```

Beispiel 29: IF-THEN-ELSE: Definition der Funktion str:reverse mit XPath

```
<xsl:value-of select="str:reverse-xpath('DOG BITES MAN')" />
```

Beispiel 30: IF-THEN-ELSE: Aufruf der Funktion str:reverse

XML SCHEMA SUPPORT

Seit XSLT 2.0 kann ein Stylesheet mit Typ-Informationen aus W3C XML Schema arbeiten. Dabei wird zwischen zwei XSLT Prozessoren unterschieden:

- Basic XSLT Prozessor
- Schema-Aware XSLT Prozessor

Unterschieden werden die beiden Prozessoren-Typen anhand ihrer Unterstützung von W3C XML Schema. Ein Basic XSLT Prozessor kennt etwa 20 atomare (Textknoten) Datentypen. Ein Schema-Aware XSLT Prozessor kennt alle in W3C XML Schema definierten Typen plus alle Datentypen, die in einem zu importierenden Schema definiert sind. Das können außer weiteren atomaren Typen auch komplexe Typen (Elemente mit Attribute(n) oder/und Nachfahren) sein.

Schema-Aware XSLT Prozessoren sind daher nicht nur in der Lage, einzelne Textknoten auf ihren Datentyp zu prüfen (gültiges Datum, Integer etc.), vielmehr können sie schon während der Tranformation erstellte Sequenzen auf ihre Gültigkeit in Bezug auf ein konkretes XML Schema validieren und entsprechend darauf reagieren.

Ferner sind sie in der Lage, Transformationen nicht nur anhand von Element-Namen, sondern auch auf Basis derer Typisierung durchzuführen. Wenn also in einem XML Schema mehrere Elemente eines Typs `currency` definiert sind, ermöglicht ein Schema-Aware XSLT Prozessor gleichzeitig alle diese Elemente zu verarbeiten bzw. das selbe Template anzuwenden, ohne dazu den tatsächlichen Element-Namen kennen zu müssen.

```
<xs:complexType name="amountType">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="currency">... </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:element name="betrag" type="amountType"/>
<xs:element name="amount" type="amountType"/>
<xs:element name="absValue" type="amountType"/>
```

Beispiel 31: Schema-Aware: Auszug Schema

```
<root>
  <betrag currency="Euro">1.23</betrag>
  <amount currency="$">10.34</amount>
  <absValue currency="Euro">9.99</absValue>
</root>
```

Beispiel 32: Schema-Aware: XML-Quelle

```
<xsl:template match="element( * , amountType )">
  <xsl:value-of select="@currency, format-number(., '#,##0.00')" />
</xsl:template>
```

Beispiel 33: Schema-Aware: Stylesheet

DIE XSLT/XPATH 2.0 PROZESSOREN

Im Moment gibt es zwei XSLT Prozessoren, die XSLT/XPath 2.0 und XQuery 1.0 implementieren:

- **Saxon**- XSLT and XQuery processing

  http://www.saxonica.com/

  Hinter dem XSLT Prozessor SAXON steht der Autor der XSLT 2.0 Spezifikation Michael Kay. Er bietet seinen XSLT-Prozessor in der Basic-Version kostenlos, und als Schema-Aware-XSLT-Prozessor zum Kauf für die verschiedensten Plattformen an.

- **AltovaXML™**

  http://www.altova.com/de/components_processors.html

  Der Hersteller ist die durch den XMLSpy bekannte Firma Altova. Beide Versionen, Basic- als auch Schema-Aware-XSLT, sind hier kostenlos.

Über die Pläne der Apache Foundation, XSLT/XPath 2.0 im XALAN zu unterstützen, ist bisher nichts bekannt. Microsoft plant für 2008 die Integration in die .Net-Umgebung und empfehlt bis dahin den SAXON von Michael Kay.

FAZIT

XSLT/XPath 2.0 bieten so viele neue Features, dass man an die Arbeit mit der 1.0-Version gar nicht mehr zurück denken möchte. Ganz sicher kann man durch XSLT/XPath 2.0 für das Gros der Anwendungen auf sämtliche bisher benötigte Erweiterungen verzichten. Portabilität ist hier das Stichwort. Gruppieren, Stringverarbeitung, Funktionalität für Datum und Zeit und die neue schema-basierte Typsicherheit sind ein paar Punkte, die für einen Umstieg sprechen. Außerdem: Zwei zuverlässige XSLT-Prozessoren stehen bereits zur Verfügung und auch die anderen Hersteller werden nicht umhin kommen, diesem Entwicklungssprung baldmöglichst zu folgen. Aus dieser Sicht steht dem Erfolg von XSLT/XPath 2.0 in der Praxis nichts mehr im Wege.

REFERENZEN

- [1] XML Verarbeitung
  Mintert, Stefan
  iX Magazin; 8 2005;Seiten: 60
- [2] Muenchian Grouping
  (http://www.biglist.com/lists/xsl-list/archives/200005/msg00270.html)
- XSL Transformations (XSLT) Version 2.0
  (http://www.w3.org/TR/xslt20/)
- XML Path Language (XPath) 2.0
  (http://www.w3.org/TR/xpath20/)
- XQuery 1.0 and XPath 2.0 Functions and Operators
  (http://www.w3.org/TR/xquery-operators/)
- Peek into the Future of XSLT 2.0
  (http://www.devx.com/xml/Article/11147)
- XSLT 2.0 and Beyond
  (http://www.cafeconleche.org/slides/sd2002west/xslt2/XSLT_2.0_and_Beyond.html)
- XSLT 2.0 Programmer's Reference, 3rd Edition
  Kay, Michael
  Wrox, Wiley
  (http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764569090.html)
- XSLT 2.0 and XPath 2.0 Programmer's Reference, 4th Edition
  Kay, Michael
  Wrox, Wiley
  (http://www.wrox.com/WileyCDA/WroxTitle/productCd-0470192747.html)
- XPath 2.0 Programmer's Reference
  Kay, Michael
  Wrox, Wiley
  (http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764569104.html)
- XSLT 2.0 - Das umfassende Handbuch
  Bongers, Frank
  Galileo Computing
  (http://www.galileocomputing.de/1085?GPP=nlch&GalileoSession=58529113A3ZC010xY5Y)