Verwendung benutzerdefinierter Funktionen in XSLT-2.0-Stylesheets

Einfach funktional

XSLT 2.0 stellt Entwicklern mit dem neuen Element xsl:function eine lange vermisste Technik zur Verfügung. Es lassen sich eigene Funktionen definieren und somit Bibliotheken mit häufig benötigten Routinen aufbauen. Dieser Beitrag vermittelt die Grundlagen und demonstriert eine praktische Anwendung am Beispiel der Transformation von XML-Daten in vektorbasierte Kreisdiagramme unter Verwendung von Winkelfunktionen.

von Thomas Meinike

Bereits mit XSLT 1.0 können anspruchsvolle Projekte umgesetzt werden. Dabei kommen als funktionale Einheiten häufig benannte Vorlagen unter Verwendung von Elementen wie *xsl:template*, *xsl:call-template*, *xsl:with-param* und *xsl:param* zum Einsatz. Zur Lösung von Standardaufgaben stehen FAQ-Sammlungen wie [1] und Erweiterungsbibliotheken wie EXSLT [2] bereit. Seit Januar 2007 hat die im Rahmen von XSLT üblicherweise eingesetzte Abfragesprache XPath in Version 2.0 mit 84 neuen Funktionen umfangreiche Verbesserungen zu bieten und XSLT 2.0 selbst bringt zehn neue Funktionen mit. Das Element *xsl:function* gehört zu den 14 hinzugekommenen Elementen, die XSLT-Entwicklern mehr Möglichkeiten und Komfort bieten sollen.

Funktionen definieren

Ein oder mehrere *xsl:function*-Elemente lassen sich unterhalb des Wurzelelements *xsl:stylesheet* platzieren. Der jeweilige Funktionsname wird über das *name*-Attribut unter Verwendung eines eigenen Namensraumpräfixes festgelegt (*ns:fktname*). Zusätzlich kann über das Attribut *as* der von der Funktion zurückgegebene Schema-Datentyp notiert werden, z. B. *xs:double* oder *xs:string*. Letztere Angabe sollte nicht fehlen, um im Verlauf der späteren Transformationen vom Prozessor Debug-Informationen über mögliche Typabweichungen zu erhalten. Funktionen verarbeiten üblicherweise ein oder mehrere Argumente, die mittels ebenfalls typisierbaren *xsl:param*-Kindelementen vorzugeben sind. Ihre Abfolge legt die Anordnung beim Funktionsaufruf im *Call*-Operator *(…)* fest. Zur eigentlichen Verarbeitung und Rückgabe werden XSLT-Elemente wie *xsl:value-of* und bei komplexeren Operationen *xsl:variable* zur Zwischenspeicherung von Daten verwendet. Mit der im folgenden Beispiel deklarierten Funktion soll die Addition zweier Dezimalwerte bewerkstelligt werden

```
<xsl:function name="tm:addition" as="xs:decimal">

 <xsl:param name="a" as="xs:decimal"/>
 <xsl:param name="b" as="xs:decimal"/>

 <xsl:value-of select="$a + $b"/>

</xsl:function>
```

Als Namensraum wird in diesem und allen folgenden Beispielen *xmlns:tm="http://www.datenverdrahten.de/tm"* verwendet. Die Beispielfunktion übernimmt als Argumente *a* sowie *b* und liefert die Summe beider Werte zurück. Aufgerufen wird die selbst definierte Funktion wie eine bereits vorhandene XSLT- oder XPath-Funktion über geeignete Elemente wie *xsl:value-of*, *xsl:variable* usw. Das in einem *xsl:template*-Element angeordnete Codefragment

```
...
<td><xsl:value-of select="tm:addition(5,6)"/></td>
...
```

erzeugt als Additionsergebnis *11* innerhalb der HTML-Tabellenzelle, gibt also *<td>11</td>* aus. Damit ist die Arbeitsweise von XSLT-Funktionen bereits ersichtlich. Innerhalb eigener Funktionen lassen sich wiederum eigene oder andere vordefinierte Funktionen aufrufen, etwa *fn:sum()* zur Summenbildung in den nachfolgend beschriebenen Funktionen. Neben atomaren Werten können an Funktionen auch Sequenzen übergeben und gegebenenfalls auch wieder zurückerhalten werden.

Variable Parameter

Es existiert keine direkte Möglichkeit, variable Parameter zu erlauben. Jede *xsl:param*-Zuweisung innerhalb von *xsl:function* ist bindend. Wenn zwei Argumente erwartet werden, wirft der XSLT-Prozessor bei der Vorgabe nur eines Parameters einen Fehler. Dennoch ist die Variation von Parametern mit einem kleinen Trick möglich: Die Funktion wird mehrfach mit identischem Namen, aber unterschiedlichen Parametersätzen, definiert. So lässt sich die Beispielfunktion *tm:addition()* um ein Argument erweitern und kann nun wahlweise zwei oder drei Werte addieren:

```
<xsl:function name="tm:addition" as="xs:decimal">

 <xsl:param name="a" as="xs:decimal"/>
 <xsl:param name="b" as="xs:decimal"/>
 <xsl:param name="c" as="xs:decimal"/>

 <xsl:value-of select="$a + $b + $c"/>

</xsl:function>
```

Bei einer größeren Zahl optionaler Parameter steigt der Aufwand entsprechend und somit sollte der konkreten Implemen-

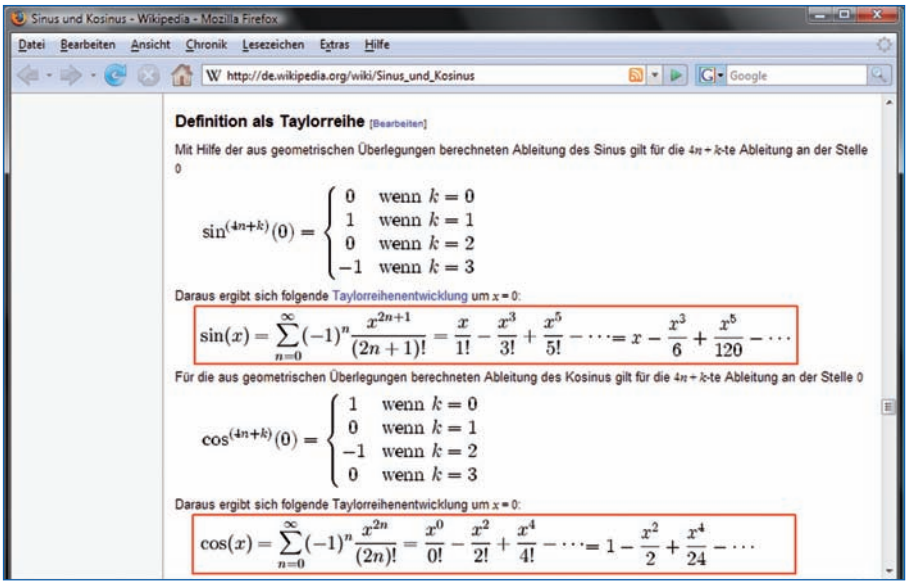

Abb. 1: Formeln zur Reihenentwicklung der Sinus- und Kosinus-Funktionen

tierung von Funktionen eine ausgewogene Planung vorangehen.

Attribut override

Dieses Attribut kann dem Element *xsl:function* explizit zugeordnet werden. Bei Abwesenheit ist der Wert implizit auf *yes* gesetzt, die Alternative ist *no*. Behandelt wird damit eine mögliche Kollision von eigenen mit bereits durch den XSLT-Prozessor bereit gestellten Funktionen. Bekannt sind beispielsweise die Erweiterungen des Prozessors Saxon [3] aus dem Namensraum *xmlns:saxon="http://saxon.sf.net/"*, etwa *saxon:sort()*. Ein anderer Prozessor wie AltovaXML [4] kann mit dieser Funktion nichts anfangen, aber durchaus mit einer Nachbildung derselben:

Listing 1

```
<!-- Potenzfunktion x^n n für nichtnegative
                                    ganzzahlige n -->
<xsl:function name="tm:pow" as="xs:double">

 <xsl:param name="x" as="xs:double"/>
 <xsl:param name="n" as="xs:integer"/>

 <xsl:variable name="ps" as="item()*" select="for $i in
                                    (1 to $n) return $x"/>
 <xsl:value-of select="tm:seq_prod
                                    ($ps, 1, fn:count($ps))"/>

</xsl:function>
```

```
<xsl:function name="saxon:sort" as="item()*
                                    " override="no">

 <xsl:param name="seq" as="item()*"/>
 <xsl:param name="ord" as="xs:string"/>
 <xsl:param name="typ" as="xs:string"/>

 <xsl:perform-sort select="$seq">
  <xsl:sort select="." order="{$ord}" data-type="{$typ}"/>
 </xsl:perform-sort>

</xsl:function>
```

Kommt hingegen Saxon 9.0 zum Einsatz, stellt sich die Frage, ob man nicht lieber

Listing 2

```
<!-- Hilfsfunktion für Produkt von Sequenzen mit
                                    atomaren Werten -->
<xsl:function name="tm:seq_prod" as="xs:double">

 <xsl:param name="seq" as="item()*"/>
 <xsl:param name="p" as="xs:double"/>
 <xsl:param name="i" as="xs:integer"/>

 <xsl:choose>
  <xsl:when test="$i gt 0">
   <xsl:variable name="p_i" as="xs:double" select=
                                    "$p * $seq[$i]"/>
   <xsl:value-of select="tm:seq_prod
                                    ($seq, $p_i, $i - 1)"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:value-of select="$p"/>
  </xsl:otherwise>
 </xsl:choose>

</xsl:function>
```

der Originalfunktion von Michael Kay als der genannten Nachbildung vertraut. Genau das lässt sich mit dem *override*-Attributwert *no* ausdrücken. Originalfunktion und Nachbildung liefern beim Aufruf von *saxon:sort((8,5,7,3,4,2,9,1,6),'descending','number')* übereinstimmend die absteigend sortierte Sequenz *(9,8,7,6,5,4,3,2,1)* zurück.

Funktionsbibliotheken

Mehrere Funktionen können in externe Dateien ausgelagert und bei Bedarf über die Elemente *xsl:include* oder *xsl:import* in Transformationsvorlagen eingebunden werden. Dadurch entstehen portable Bibliotheken. Vom Autor wurden im Rahmen des Moduls *tm_mathlib.xsl* einige mathematische Funktionen und benötigte Hilfsfunktionen umgesetzt, die sich nicht im regulären Angebot von XSLT bzw. XPath befinden. Dazu gehören die Winkelfunktionen Sinus und Kosinus. Gute Näherungswerte beider Funktionen sind über Reihenentwicklungen [5] zugänglich, die sich mit wenigen Zeilen Code umsetzen lassen. In den einzelnen Summanden beider Reihen kommt die Fakultät vor, als separate Funktion formuliert.

```
<!-- Fakultät -->
<xsl:function name="tm:fact" as="xs:double">

 <xsl:param name="n" as="xs:integer"/>
 <xsl:value-of select="if($n eq 0 or $n eq 1) then 1 else $n *
                                    tm:fact($n - 1)"/>

</xsl:function>
```

Durch die Verwendung der neuen Inline-XPath-Technik *if-then-else* ergibt sich ein sehr kompakter Code. Neben der Fakultät fallen ganzzahlige Potenzen in den Zählern der einzelnen Summanden auf. Da XPath für derartige Berechnungen keine adäquate Funktion anbietet, wurde diese in Form von Listing 1 ebenfalls in die Bibliothek aufgenommen. Neben der ebenfalls neuen XPath-Schleifensyntax *for-in-return* kommt eine weitere Funktion zum Ermitteln des Produktes über eine Sequenz zum Einsatz (Listing 2). Die Listings 3 und 4 enthalten den Code der auf den Reihenentwicklungen aufbauenden Funktionen *tm:sin()* und *tm:cos()*. Als globale Variablen werden *$pi*

ANZEIGE

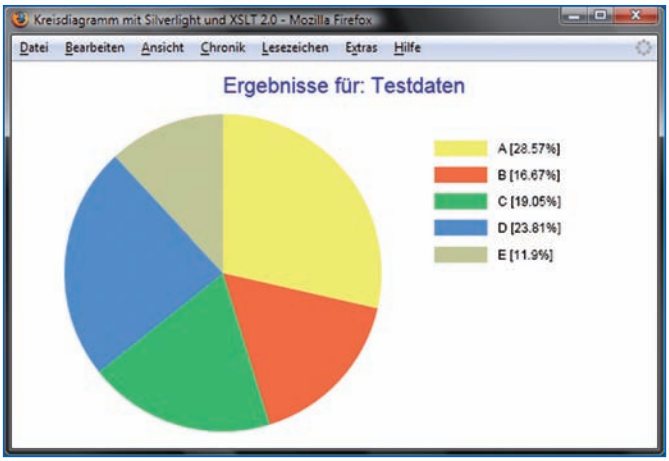

Abb. 2: Vektorgrafische Umsetzung von XML-Daten

(Kreiszahl) und *$max* (Anzahl der Summenglieder für die Reihenentwicklungen) vorgegeben. Mit *$max=15* wird eine Rechengenauigkeit von etwa acht Nachkommastellen erreicht. Zum Runden bietet sich die neue XPath-Funktion *fn:round-half-to-even()* an. Beide Winkelfunktionen erwarten ihre Argumente im Bogenmaß. Wenn Winkel übergeben werden sollen, kann eine weitere Hilfsfunktion *tm:deg2rad()* bemüht werden und auch an die Umkehrung *tm:rad2deg()* wurde gedacht (s. Code der Heft-CD). Beim Aufruf von *tm:sin(1)* wird als Sinus-Funktionswert von *1* das Ergebnis *0.84147098* erhalten.

Listing 3

```
<!-- Sinus-Funktion -->
<xsl:function name="tm:sin" as="xs:double">

 <xsl:param name="arg" as="xs:double"/>
 <xsl:variable name="x" as="xs:double" select=
        "if($arg ge 2 * $pi) then $arg - fn:floor($arg div
            (2 * $pi)) * 2 * $pi else(if($arg le -2 * $pi)
              then $arg + fn:floor($arg div (-2 * $pi))
                    * 2 * $pi else $arg)"/>

 <xsl:variable name="sum_seq" as="item()*" select=
              "for $n in (0 to $max) return fn:number
             (tm:pow(-1, $n) * tm:pow($x, 2 * $n + 1)
                    div tm:fact(2 * $n + 1))"/>
 <xsl:value-of select="fn:round-half-to-even(fn:sum
                    ($sum_seq), 8)"/>

</xsl:function>
```

Listing 4

```
<!-- Kosinus-Funktion -->
<xsl:function name="tm:cos" as="xs:double">

 <xsl:param name="arg" as="xs:double"/>
 <xsl:variable name="x" as="xs:double" select="if
        ($arg ge 2 * $pi) then $arg - fn:floor($arg div
            (2 * $pi)) * 2 * $pi else(if($arg le -2 * $pi)
              then $arg + fn:floor($arg div (-2 * $pi))
                    * 2 * $pi else $arg)"/>

 <xsl:variable name="sum_seq" as="item()*" select=
        "for $n in (0 to $max) return fn:number(tm:pow
            (-1, $n) * tm:pow($x, 2 * $n) div tm:fact
                    (2 * $n))"/>
 <xsl:value-of select="fn:round-half-to-even
                    (fn:sum($sum_seq), 8)"/>

</xsl:function>
```

Kreisdiagramme mit XSLT

Die beschriebenen Winkelfunktionen sind nicht nur als Beispiele zur Veranschaulichung von *xsl:function* entstanden. Primär wurden sie für die Ausgabe von Tortendiagrammen in den Vektorgrafikformaten SVG bzw. XAML (Silverlight) entwickelt. Diese erlauben die Verwendung einer identischen Syntax zur Beschreibung von Pfaden, wobei sich lediglich die Schreibweise der Element- und Attributnamen unterscheidet (*<path d="…"/>* bzw. *<Path Data="…"/>*). Somit lassen sich die benötigten Kreissegmente mit moderatem Aufwand für den SVG- und den XAML-Einsatz aufbereiten. Auf der Heft-CD sind entsprechende XSLT-Stylesheets und die erhaltenen Ergebnisse für beide enthalten. Die Grundlage für die Transformationen bildet das in Listing 5 ersichtliche XML-Dokument. Für jedes im XML-Dokument enthaltene *satz*-Element werden in der gewünschten Zielstruktur (SVG/XAML) ein Kreissegment und ein Legendeneintrag erzeugt (Listing 6). Abbildung 2 zeigt das vollständige grafische Ergebnis in der Silverlight-Variante, die sich optisch nicht von der SVG-Darstellung unterscheidet.

Listing 5

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<daten info="Testdaten">
 <satz>
  <wert>120</wert>
  <text>A</text>
  <farbe>#FF6</farbe>
 </satz>
 <!-- weitere satz-Elemente ... -->
</daten>
```

Listing 6

```
<!-- XAML-Ausgabe für den Datensatz aus Listing 5 -->
<Canvas x:Name="Segment1">
 <Path Data="M 200,200 L 200,50 A 150,150 0 0,1
                    346.24,233.38 Z" Fill="#FF6"/>
 <Rectangle Canvas.Left="400" Canvas.Top="75"
                Width="50" Height="15" Fill="#FF6"/>
 <TextBlock Canvas.Left="460" Canvas.Top="76"
                FontFamily="Arial" FontSize="12"
                Foreground="#000" Text="A [28.57%]"/>
</Canvas>
```

Fazit

Mit dem neuen XSLT-Element *xsl:function* können Entwickler ihre Stylesheets deutlich aufwerten. Neben der Kapselung und Modularisierung von Algorithmen ist insbesondere der direkte Aufruf eigener Funktionen innerhalb von XPath-Ausdrücken ein wesentlicher Vorteil gegenüber der aufwändigeren Ansprache von benannten Templates mit Parameterübergabe. Weiteres Material zum Thema ist unter [6] und [7] verfügbar.

Dr. Thomas Meinike *ist seit 1997 an der Hochschule Merseburg (FH) als Lehrkraft tätig. Seine Arbeitsschwerpunkte sind XML-Anwendungen in der Technischen Dokumentation, Onlinehilfen und Webentwicklung. Er verfasst regelmäßig Fachartikel und hält Vorträge zu Themen im XML-Umfeld. E-Mail-Kontakt: thomas.meinike@hs-merseburg.de*

Links & Literatur

[1] XSLT Questions and Answers: www.dpawson.co.uk/xsl/sect2/sect21.html

[2] EXSLT: www.exslt.org

[3] Saxon: saxon.sourceforge.net

[4] AltovaXML: www.altova.com/altovaxml.html

[5] Wikipedia, Sinus und Kosinus: de.wikipedia.org/wiki/Sinus_und_Kosinus

[6] Meinike, T.: www.iks.hs-merseburg.de/~meinike/vortraege.php

[7] Meinike, T.: www.datenverdrahten.de