3.0-Updates von XSLT und XPath auf einen Blick

tekom-Jahrestagung 2012 – Wiesbaden, 23. Oktober

Dr. Thomas Meinike

Hochschule Merseburg | FB Informatik und Kommunikationssysteme
http://www.iks.hs-merseburg.de/~meinike/
thomas.meinike@hs-merseburg.de

XSLT …

- Extensible Stylesheet Language Transformations
- Funktionale Programmiersprache zur Umwandlung von XML-Strukturen in Ausgabeformate auf der Basis von XML, (X)HTML, Text
- XSLT-Dokument = mittels XML-Syntax ausgezeichnetes Template-basiertes Regelwerk (Verarbeitung mit XSLT-Prozessoren: Saxon, AltovaXML, …)
- Spezifiziert vom W3C
  1999: 1.0 | 2007: 2.0 (siehe tekom-Vortrag) | 2012/13: **3.0**
- Aktueller 3.0-Arbeitsentwurf vom 12. Juli 2012
  http://www.w3.org/TR/xslt-30/

XPath …

- XML Path Language zur Lokalisierung von Knoten und Inhalten innerhalb von XML-Strukturen
- Typische Anwendung als Abfragesyntax für XSLT und XQuery, aber auch mit JavaScript, PHP usw. einsetzbar
- Seit 2.0 >100 spezielle XPath-Funktionen für Datenmanipulationen
- Spezifiziert vom W3C
  1999: 1.0 | 2007: 2.0 (siehe tekom-Vortrag) | 2012/13: **3.0**
- Aktueller 3.0-Arbeitsentwurf vom 13. Dezember 2011
  http://www.w3.org/TR/xpath-30/

XSLT / XPath 3.0 – Was läuft?

➔ Da noch nicht verabschiedet, lassen sich neue Techniken nur ansatzweise oder noch nicht praktisch ausprobieren

➔ Aktuell sind nur Saxon-Versionen 9.3 / **9.4** (ab PE / EE) mit Teilimplementierungen ausgestattet:

Feature	HE	PE	EE	EE-T	EE-Q	EE-V
XPath 2.0 (Basic)	✓	✓	✓	✓	✓	✓
XPath 2.0 (Schema-Aware)			✓	✓	✓	
XPath 3.0*		✓	✓	✓	✓	✓
XQuery 1.0 (Basic)	✓	✓	✓	✓	✓	✓
XQuery 1.0 (Schema-Aware)			✓		✓	
XQuery 3.0*		✓	✓		✓	

Tests mit Saxon-EE 9.4.0.4 im <oXygen/> XML Editor 14.1
http://www.oxygenxml.com/

„Saxon Product/Feature Matrix“ (Ausschnitt)

http://www.saxonica.com/feature-matrix.html

XSLT 3.0 – Grundgerüst

➔ Neue Versionsnummer und Namensräume:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:err="http://www.w3.org/2005/xqt-errors"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:map="http://www.w3.org/2005/xpath-functions/map"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="err fn map math xs">
  <xsl:template match="/">
    <!-- ... -->
  </xsl:template>
</xsl:stylesheet>
```

Namensräume nach Bedarf, neu eingeführt wurden map und math

XSLT 3.0 – Streaming & Co.

- Soll Verarbeitung von sehr großen XML-Dokumenten verbessern

- Sequenzieller Zugriff auf Teile des jeweiligen Dokuments über neue Elemente:

  xsl:stream

    xsl:iterate (auch ohne xsl:stream nutzbar)

      xsl:next-iteration

      xsl:break

      xsl:on-completion (bei vollständiger Ausgabe)

- xsl:merge zum Zusammenführen mehrerer Eingabedokumente bzw. -Streams (z. B. Logfiles) und xsl:fork für kombinierte Ausgaben

✔Saxon 9.4
(ohne xsl:stream / xsl:fork)

XSLT 3.0 – Streaming & Co.

➔ Zerlegung einer umfangreichen Struktur eingabe.xml in kleinere Einheiten ausgabe_1.xml bis ausgabe_n.xml:

```
<xsl:stream href="eingabe.xml">
  <xsl:iterate select="//element">
    <xsl:result-document href="ausgabe_{fn:position()}.xml">
      <xsl:copy-of select="."/>
    </xsl:result-document>
    <xsl:choose>
      <xsl:when test="fn:position() > $n"><xsl:break/></xsl:when>
      <xsl:otherwise><xsl:next-iteration/></xsl:otherwise>
    </xsl:choose>
    <xsl:on-completion><xsl:message>Fertig!</xsl:message></xsl:on-completion>
  </xsl:iterate>
</xsl:stream>
```

✔Saxon 9.4
(ohne xsl:stream)

XSLT 3.0 – Streaming & Co.

➔ Beispiel zu xsl:merge:

```
<xsl:merge>
  <xsl:merge-source select="fn:collection('liste.xml')">
    <xsl:merge-input select="root/test">
      <xsl:merge-key select="@nr" order="ascending"/>
    </xsl:merge-input>
  </xsl:merge-source>

  <xsl:merge-action>
    <eintrag nr="{@nr}">
      <xsl:value-of select="fn:concat(fn:current-group()[1], '|', fn:current-group()[2])"/>
    </eintrag>
  </xsl:merge-action>
</xsl:merge>
```

```
<root>
  <test nr="1">A</test>
  <!-- ... -->
</root>

<root>
  <test nr="1">G</test>
  <!-- ... -->
</root>
```

```
<ausgabe>
  <eintrag nr="1">A|G</eintrag>
  <!-- ... -->
</ausgabe>
```

✔Saxon 9.4

XSLT 3.0 – Dynamische XPath-Abfragen

➔ xsl:evaluate wertet mit Zeichenketten gebildete XPath-Ausdrücke aus (analog zu saxon:evaluate oder dyn:evaluate unter EXSLT):

```
<!-- XML -->
<NASA>
  <MARS>Curiosity has landed!</MARS>
</NASA>
<!-- XSLT -->
<xsl:variable name="str1" select="'NASA'"/> <xsl:variable name="str2" select="'MARS'"/>
<xsl:value-of>
  <xsl:evaluate xpath="fn:concat($str1, '/', $str2)"/>
</xsl:value-of>
```

Ausgabe:
Curiosity has landed!

➔ Besonders praktikabel bei variablen Abfragen über externe Parameter via xsl:param

✔Saxon 9.4

XSLT 3.0 – Fehlerbehandlung

➔ Neue Elemente xsl:try, xsl:catch | xsl:fallback

```
<xsl:variable name="a" select="1"/> <xsl:variable name="b" select="0"/>
<xsl:try>
  <p><xsl:value-of select="$a div $b"/></p>
  <xsl:catch errors="err:FOAR0001">
    <p>Division durch Null!</p>
  </xsl:catch>
  <!-- ggf. weitere Elemente xsl:catch | xsl:fallback für Kompatibilität XSLT < 3.0 -->
</xsl:try>
```

Alternativ select-Attribut für xsl:try / xsl:catch möglich

➔ Namensraum / Fehlerdetails:

```
xmlns:err="http://www.w3.org/2005/xqt-errors"
```

✔Saxon 9.4

B Error Summary

The error text provided with these errors is non-normative.

err:FOER0000, Unidentified error

Unidentified error.

err:FOAR0001, division by zero

This error is raised whenever an attempt is made to divide by zero.

err:FOAR0002, numeric operation overflow/underflow

This error is raised whenever numeric operations result in an overflow or underflow.

{Demos …}

XSLT 3.0 – Akkumulatoren

➔ Einfachere Ermittlung von Werten bezogen auf vorhergehende Schritte, bisher über rekursive Templateaufrufe gelöst

➔ Beispiel aus der Spezifikation zur Nummerierung von Abbildungen:

```
<xsl:accumulator name="f:figNr" as="xs:integer" initial-value="0" streamable="yes">
  <xsl:accumulator-rule match="chapter" new-value="0"/>
  <xsl:accumulator-rule match="figure" new-value="$value + 1"/>
</xsl:accumulator>
...
<xsl:template match="figure">
  <xsl:apply-templates/>
  <p>Figure <xsl:value-of select="f:figNr()"/></p>
</xsl:template>
```

Siehe: http://www.w3.org/TR/xslt-30/#dt-accumulator-function

✘ Saxon 9.4

XSLT 3.0 – Pakete

➔ Bessere Verwaltung umfangreicher Transformationen, als Ergänzung zur Einbindung von Ressourcen über xsl:include / xsl:import

```
<xsl:package  xsl:version? = decimal name? = uri package-version? = string>
  <!-- Content: (xsl:use-package*, (xsl:stylesheet | xsl:transform), xsl:expose*) -->
</xsl:package>
```

➔ Referenzierung von Komponenten:

```
<xsl:use-package name? = uri package-version? = token>
  <!-- Content: (xsl:accept | xsl:override)* -->
</xsl:use-package>
```

➔ Weitere Implementierungsdetails:

http://www.w3.org/TR/xslt-30/#packages-and-modules

✘ Saxon 9.4

XSLT 3.0 – Diverses

➔ xsl:copy erhält ein optionales select-Attribut, Vorgabe beim Weglassen entspricht:

✔Saxon 9.4

```
<xsl:copy select="."/>
```

➔ xsl:assert stellt Behauptung auf, der Prozessor soll ggf. keine weiteren Auswertungen vornehmen:

```
<xsl:template match="/">
  ...
  <xsl:assert test="count(//abc) = 1)"/>
  <!-- es soll nur 1 abc-Element geben, Abfrage nach weiteren
       würde unterbleiben bzw. Fehler liefern -->
  ...
</xsl:template>
```

✘Saxon 9.4

XSLT 3.0 – Diverses

➔ Formale Unterscheidung von XSLT- bzw. XPath-Funktionen wird aufgehoben, wahlweise auch mit fn-Präfix aufrufbar:

current()
document()
element-available()
format-date()
format-dateTime()
format-number()
format-time()
function-available()
generate-id()
key()
system-property()
unparsed-entity-uri()
unparsed-text()
[…]

✔Saxon 9.4

XPath 3.0 – Mathematische Funktionen

➔ 14 Funktionen inkl. Konstante π mit math-Präfix:

```
xmlns:math="http://www.w3.org/2005/xpath-functions/math"
```

➔ Beispiel: Sinus von 0.5 (Bogenmaß)

```
<xsl:value-of select="math:sin(0.5)"/>
<!-- 0.479425538604203 -->
```

➔ Beispiel: Potenz 2^5

```
<xsl:value-of select="math:pow(2,5)"/>
<!-- 32 -->
```

➔ Was fehlt?

math:random()

✔Saxon 9.4

Trigonometrische Funktionen und Pi	
math:acos(0.5)	1.0471975511965979
math:asin(0.5)	0.5235987755982989
math:atan(0.5)	0.4636476090008061
math:atan2(1, 0.5)	1.1071487177940904
math:cos(0.5)	0.8775825618903728
math:sin(0.5)	0.479425538604203
math:tan(0.5)	0.5463024898437905
math:pi()	3.141592653589793
Exponential- und Logarithmus-Funktionen	
math:exp(2) [= e^2]	7.38905609893065
math:exp10(4) [= 10^4]	10000
math:log(5) [= ln(5)]	1.6094379124341003
math:log10(100) [= lg(100)]	2
Potenz- und Wurzel-Funktionen	
math:pow(2, 5) [= 2^5]	32
math:sqrt(81) [= √81]	9

XPath 3.0 – Mathematische Funktionen

➔ Anwendung von Winkelfunktionen für SVG-Kreisdiagramm

Datensatz:

```
<?xml version="1.0" encoding="UTF-8"?>
<daten info="Testdaten">
  <satz>
    <wert>50</wert>
    <text>A</text>
    <farbe>#FFF200</farbe>
  </satz>
  <!-- 5 weitere satz-Elemente -->
</daten>
```

Vektorgrafik:

✔Saxon 9.4

XPath 3.0 – Maps

➔ Mit dem neuen Datentyp map {…} lassen sich Key-Value-Paare in Analogie zu (assoziativen) Arrays ablegen und abfragen

```
xmlns:map="http://www.w3.org/2005/xpath-functions/map"
```

➔ Zuweisung und Abfrage:

```
<xsl:variable name="m" select="map {'a' := 'x', 'b' := 'y', 'c' := 'z'}"/>
<xsl:value-of select="$m('a')"/> <!-- x -->

-----------------------------------------------------------------------------------------------------------

<xsl:variable name="wochentag"
 select="map {0 := 'So', 1 := 'Mo', 2 := 'Di', 3 := 'Mi', 4 := 'Do', 5 := 'Fr', 6 := 'Sa'}"/>
<xsl:value-of select="$wochentag(0)"/> <!-- So -->
```

✔Saxon 9.4

XPath 3.0 – Maps

➔ Beispiel zur Nutzung von Farbwerten:

```
<xsl:variable name="farben"
 select="map {'gelb' := '#FF0', 'rot' := '#F00', 'gruen' := '#0F0', 'blau' := '#00F'}"/>
<p style="color: {$farben('rot')}">Hallo Maps!</p> <!-- ... color: #F00 ... -->
```

➔ Map-Funktionen (contains, get, size, keys, remove + new, entry):

```
<xsl:value-of select="map:contains($m, 'a')"/> <!-- true -->
<xsl:value-of select="map:contains($m, 'd')"/> <!-- false -->
<xsl:value-of select="map:get($wochentag,6)"/> <!-- Sa -->
<xsl:value-of select="map:size($wochentag)"/> <!-- 7 -->
<xsl:value-of select="map:keys($m)"/> <!-- b c a (warum nicht a b c ?) -->
<xsl:value-of select="map:keys($wochentag)"/> <!-- 0 1 2 3 4 5 6 -->
<xsl:variable name="m_neu" select="map:remove($m, 'c')"/> <!-- erzeugt neue map ohne c -->
<xsl:value-of select="map:size($m_neu)"/> <!-- 2 -->
```

✔Saxon 9.4

{Demos …}

XPath 3.0 – Higher-Order Functions

➔ **HOF** sind Funktionen, welche andere Funktionen als Argumente übernehmen und / oder Funktionen zurückgeben:

fn:map(), fn:map-pairs(), fn:filter(), fn:fold-left(), fn:fold-right()

➔ fn:map wendet eine Funktion nacheinander auf eine Sequenz an und führt das Ergebnis zusammen:

```
<xsl:value-of select="fn:map(function($x) { math:sin($x) }, 1 to 10)"/>
<!-- erzeugt Sequenz der Sinus-Werte von 1 bis 10 -->

<xsl:value-of select="fn:map(fn:string-length#1, ('XML', 'XSLT', 'ganz toll'))"/>
<!-- erzeugt Sequenz 3 4 9 | fktname#1 ist ein Funktionsliteral mit 1 Argument -->
```

✔Saxon 9.4

XPath 3.0 – Higher-Order Functions

- fn:map-pairs() wendet eine Funktion nacheinander paarweise auf zwei Sequenzen an:

```
<xsl:value-of select="fn:map-pairs(concat#2, ('a', 'b', 'c'), ('x', 'y', 'z'))"/>
<!-- erzeugt Sequenz ax by cz -->
```

- fn:filter() gibt jene Werte eines Funktionsaufrufes zurück, welche die gesetzte Bedingung erfüllen:

```
<xsl:value-of select="fn:filter(function($x) {$x mod 2 = 1}, 1 to 10)"/>
<!-- erzeugt Sequenz der ungeraden Zahlen 1 3 5 7 9 -->
```

- fn:fold-left() bzw. fn:fold-right() verarbeiten eine Sequenz von links bzw. rechts und wenden die angegebene Funktion schrittweise unter Einbeziehung der Zwischenwerte an

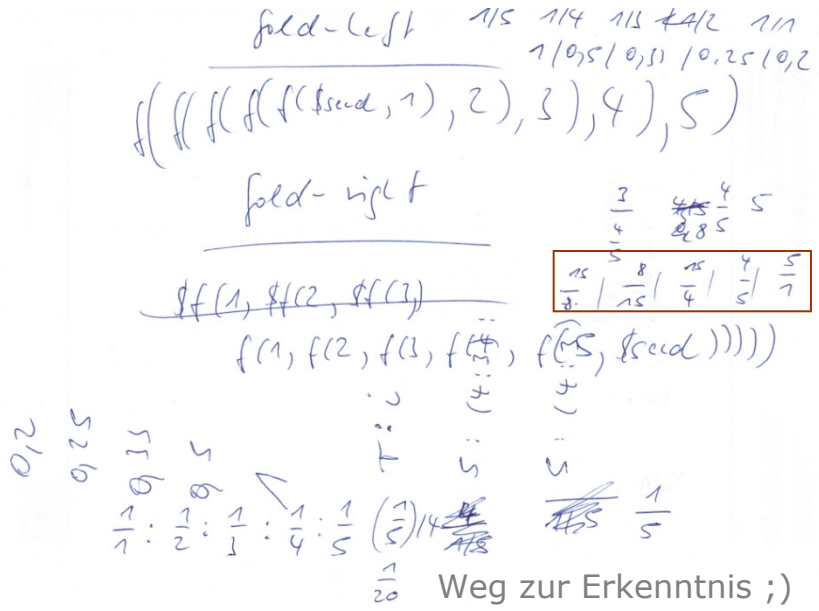
✔Saxon 9.4

XPath 3.0 – Higher-Order Functions

➔ Faltungsexperimente:

```
<xsl:value-of select="fn:fold-left(function($a, $b) { $a div $b }, 1, 1 to 5)"/>
<!-- von links: 1 / 1 / 2 / 3 / 4 / 5 = 0.008333333333333333 -->

<xsl:value-of select="fn:fold-right(function($a, $b) { $a div $b }, 1, 1 to 5)"/>
<!-- von rechts: 5/1 ... 4/5 ... 3/(4/5) = 15/4 ...
     2/(15/4) = 8/15 ... 1/(8/15) = 15/8
     = 1.8750000000000000001 -->
```



✔Saxon 9.4

Weg zur Erkenntnis ;)

XPath 3.0 – Higher-Order Functions

➔ Im Kontext von dynamischen HOF interessant:
fn:function-lookup(), fn:function-name(), fn:function-arity()

Anzahl der Argumente einer Funktion (arity):

```
<xsl:variable name="addiere" select="function($a, $b) { $a + $b }"/>
<xsl:value-of select="fn:function-arity($addiere)"/> <!-- 2 -->
```

➔ Umwandlung einer Funktion mit mehreren Argumenten in eine Funktion mit einem Argument (Currying):

```
<xsl:value-of select="$addiere(23, 7)"/> <!-- 30 -->
<xsl:value-of select="let $f := $addiere(23, ?) return $f(42)"/> <!-- 65 -->
```

Hinweis: Funktion $addiere() wurde als **Inline-Funktion** deklariert, ebenfalls eine neue Technik

✔Saxon 9.4

XPath 3.0 – Sequenzfunktionen

➔ fn:head($seq) alternativ zu $seq[1]

➔ fn:tail($seq) als Kurzform für fn:subsequence($seq, 2)

```
<xsl:variable name="seq" select="('A', 'B', 'C', 'D', 'E', 'F')"/>

<xsl:value-of select="fn:head($seq)"/> <!-- A als Sequenz -->

<xsl:value-of select="fn:tail($seq)"/> <!-- B C D E F als Sequenz -->
```

✔Saxon 9.4

XPath 3.0 – Sequenzfunktionen

➔ fn:innermost($seq as node()*) → node()*

gibt Knoten aus $seq zurück, welche keine Nachfahren besitzen, die ebenfalls Teil dieser Sequenz sind

➔ fn:outermost($seq as node()*) → node()*

gibt Knoten aus $seq zurück, welche keine Vorfahren besitzen, die ebenfalls Teil dieser Sequenz sind

➔ fn:has-children($node as node()?) → xs:boolean (true / false)

Abfrage, ob Knoten einen oder mehrere Kindknoten haben

✘ Saxon 9.4

XPath 3.0 – Umgebungsvariablen

➔ Vom Betriebssystem und installierten Anwendungen nutzen

➔ Welche existieren? fn:available-environment-variables()

```
<ul>
  <xsl:for-each select="fn:available-environment-variables()">
    <xsl:sort select="." data-type="text" order="ascending"/>
    <li><xsl:value-of select="."/></li>
  </xsl:for-each>
</ul>
```

- AHF60_HOME
- CLASSPATH
- COMPUTERNAME
- HOMEDRIVE
- HOMEPATH
- OS
- PATHEXT
- ProgramData
- ProgramFiles
- SystemDrive
- SystemRoot
- TEMP
- TMP
- USERNAME
- ...

➔ Bekannte abfragen: fn:environment-variable('varname')

```
<xsl:value-of select="fn:environment-variable('TEMP')"/>
<!-- C:\Users\User\AppData\Local\Temp -->
```

✔Saxon 9.4

XPath 3.0 – Diverses

➔ String-Verknüpfungsoperator || als Alternative zu fn:concat()

```
<xsl:value-of select="'a' || 'b'"/> <!-- ab -->
```

➔ Simple-Mapping-Operator für Sequenzen !

```
<xsl:value-of select="(1 to 5) ! ('a', 'b')"/> <!-- a b a b a b a b a b (als Sequenz) -->
<xsl:value-of select="fn:string-join((1 to 10) ! '+')"/> <!-- ++++++++++ -->
```

➔ fn:format-integer() ergänzt fn:format-number()

```
<xsl:value-of select="fn:format-integer(2342, '00000000')"/> <!-- 00002342 -->
<xsl:value-of select="fn:format-integer(2342, 'w')"/> <!-- zweitausend dreihundertzweiundvierzig -->
<xsl:value-of select="fn:format-integer(2342, 'I')"/> <!-- MMCCCXLII -->
```

✔Saxon 9.4

XPath 3.0 – Diverses

➔ fn:unparsed-text-lines() ergänzt fn:unparsed-text() aus XPath 2.0, automatische Behandlung von Zeilenumbrüchen (\n … \r … \r\n):

```
<xsl:for-each select="fn:unparsed-text-lines('test.csv')"> <!-- wert1;wert2;wert3 -->
  <xsl:variable name="zeile" select="fn:tokenize(., ';')"/>
  <eintrag attr1="{$zeile[1]} attr2="{$zeile[2]} attr3="{$zeile[3]}"/>
</xsl:for-each>
<!-- <eintrag attr1="wert1" attr2="wert2" attr3="wert3"/> pro Zeile  -->
```

➔ fn:analyze-string($String, $RegEx)
nützlich unter XQuery 3.0, gibt XML-Struktur zurück:

```
<analyze-string-result><match>...</match><non-match>...</non-match></analyze-string-result>
```

➔ fn:path($node?) → Pfadausdruck vom Knoten relativ zur Wurzel:

`fn:path(x)` für /root/x: `/"":root[1]/"":x[1]` (in "" stehen ggf. Namensräume)

✔Saxon 9.4

XPath 3.0 – Diverses

➔ Verarbeitung von XML sowie JSON (JavaScript Object Notation), sinnvoll vor allem für XQuery 3.0, unter XSLT ggf. via xsl:param

XML: fn:parse-xml(...) | fn:parse-xml-fragment(...) | fn:serialize(...)

```
<!-- $xmlstr = '<a><b>123</b></a>' als Parameter übergeben -->
<xsl:variable name="doc" select="fn:parse-xml($xmlstr)"/>
<xsl:value-of select="$doc/a/b"/> <!-- 123 -->
```

JSON: fn:parse-json(...) | fn:serialize-json(...)

```
<!-- $jsonstr = '{ "x" : 1, "y" : 2, "z" : 3}' als Parameter übergeben -->
<xsl:variable name="obj" select="let $obj := fn:parse-json($jsonstr) return $obj('y')"/>
<xsl:value-of select="$obj"/> <!-- 2 -->
```

✔Saxon 9.4

{Demos …}

Fazit und Ausblick

- Die 3.0-Versionen bringen mehr Evolution als Revolution
- Streaming ist für die Verarbeitung großer Datenmengen ein Gewinn
- Mathematische Funktionen und der map-Datentyp schließen Lücken
- Higher-Order Functions vertiefen den funktionalen Ansatz und ermöglichen kompaktere Formulierung von Algorithmen
- XQuery 3.0 profitiert ebenfalls von neuer Syntax, siehe XML-Datenbanken wie eXist und BaseX
- Candidate Recommendations und weitere Prozessor-Anpassungen lt. M. Kay in Vorbereitung …