Fragestunde: Neuerungen in XQuery und XSLT

Mit XQuery und XSL-T/XPath existieren zwei Ansätze zur Verarbeitung von XML Daten.

Felix Sasaki & Jan-Torsten Milde

Einleitung

Der Artikel behandelt die W3C-Standards XQuery und XSLT, die zur Abfrage bzw. Transformation von XML-Dokumenten konzipiert wurden. Diese beiden Aufgaben entstammen unterschiedlichen Anwendungsszenarien: der datenbankorientierten Nutzung von XML versus der Verarbeitung von XML-basierten Textdokumenten im elektronischen Publikationsprozess. Die Unterschiedlichkeit der Szenarien führt dazu, dass viele Entwickler und Nutzer der jeweiligen 'Communities' nur einen der beiden Standards für notwendig halten. Der Artikel hingegen will die Gemeinsamkeiten von XQuery und XSLT herausstellen.

Der vorliegende Artikel strebt keine komplette Beschreibung der Merkmale von XSLT und XQuery an. Der Leser sei hierfür an [1,2] verwiesen.

Die gemeinsame Grundlage: das XPath 2.0 Datenmodell

Die Basis zum Verständnis von XQuery und XSLT stellt das zugrundeliegende XPath 2.0 Datenmodell dar. Das Datenmodell repräsentiert XML Daten als eine *Sequenz von Knoten und / oder atomaren Werten.* Wichtig ist zu beachten, was vom Datenmodell *nicht* repräsentiert wird. So wird die Zeichenkodierung erst bei der Serialisierung der Daten verwendet. Weiterhin werden die Grenzen eine *CDATA* Section, Referenzen auf *Entitäten*, die *DOCTYPE* Deklaration und der interne *DTD Subset* nicht repräsentiert. Hierbei hängt es von der einbettenden Sprache ab (also z.B. XQuery oder XSLT), welche der Informationen erhalten bleiben und welche verloren gehen.

Die auf dem Datenmodell aufsetzende Verarbeitungsweise von XQuery und XSLT sind einander sehr ähnlich. Beide Sprachen erwarten als Input 0 bis n XML Dokumente. Der Output sind 0 bis n XML Dokumente. Die eigentliche Ausgabe bzw. „Serialisierung“ der Daten ist dabei ein vom Verarbeitungsprozess getrennter Schritt, der in einer separaten Spezifikation des W3Cs beschrieben wird[4,5]. Diese Spezifikation gibt eine beträchtliche Anzahl von Konfigurationsparametern vor. Allerdings verlangen XSLT und XQuery nur eine minimale Unterstützung der beschriebenen Serialisierungsmöglichkeiten.

Hier kommt natürlich die Frage auf, was der Unterschied zwischen beiden Sprachen ist. Wir werden sie anhand des folgenden Beispiels beantworten.

```
<meinDok>
 <meinEl meinAttr="meinWert1"/>
 <meinEl meinAttr="meinWert2"/>
</meinDok>
```

[Abb. Eingabedokument]

```
<deinDok>
 <deinEl deinAttr="meinWert1"/>
 <deinEl deinAttr="meinWert2"/>
</deinDok>
```

[Abb. Ausgabedokument]

Eingabe in die Verarbeitung ist das Dokument „meinDok.xml“. Es besteht aus einem Element <meinDok> welches zwei Elemente <meinEl> enthält. Diese besitzen jeweils ein Attribut @meinAttr. Die Aufgabe besteht nun darin, aus diesem Dokument ein Dokument "deinDok.xml" zu

erzeugen. Es ist strukturell isomorph zu "meinDok.xml" und unterscheidet sich nur hinsichtlich der Element- und Attributnamen.

Das Verarbeitungsmodell von XSLT: „Push“ Verarbeitung

Zunächst zur Lösung der Aufgabe mit XSLT. Wichtigster Bestandteil der XSLT Verarbeitung sind Templates, die durch <xsl:template> Elemente dargestellt sind. Zur einfacheren Identifikation der Templates im Text haben wir @xml:id Attribute eingeführt, auf die wir im Folgenden Bezug nehmen.

Eingabe in ein Temlate ist eine Sequenz von Knoten und / oder atomaren Werten, ein Teil des so genannten „*dynamische Kontexts*“. Der dynamische Kontext ist die Information, die bei der Verarbeitung einer XSLT-Anweisung zur Verfügung steht. Zum dynamischen Kontext kommt der „*statische Kontext*“, der unter anderem die im Stylesheet deklarierten Namensräume umfasst.

Im Template wird die Knotensequenz, die sich aus dem dynamischen Kontext ergibt, abgearbeitet. Die Besonderheit liegt nun darin, dass die Reihenfolge der Templates nicht fest liegt. Sie hängt davon ab, welche Knotensequenz im dynamischen Kontext gerade zur Verfügung steht. Nehmen wir an, dass das folgende Stylesheet „deinDok.xml“ erzeugen soll:

```
<xsl:stylesheet ...> [Die Namensraumdeklarationen sind der übersichtlichkeit halber
ausgespart.]
 <xsl:template match="/" xml:id="t1">
  <xsl:apply-templates/>
 </xsl:template>

 <xsl:template match="meinEl" xml:id="t2">
  <deinEl deinAttr="{@meinAttr}">
 </xsl:template>

 <xsl:template match="meinDok" xml:id="t3">
 <deinDok>
  <xsl:apply-templates select=".//*" />
 </deinDok>
 </xsl:template>

</xsl:stylesheet>
```

[Abb. XSLT Stylesheet zur Erzeugung von "deinDok.xml"]

Der erste Knoten, der bei der Initialisierung der Stylesheetverarbeitung zur Verfügung steht, ist der Dokumentknoten von „meinDok.xml“. Die Regel @match="/" trifft auf diesen Knoten. Das bedeutet, dass das Template "t1" als so genanntes *initiales* Template fungiert: es wird als erstes abgearbeitet. Im Template ist nur eine Anweisung in Form des Elements <xsl:apply-templates/> enthalten. Sie bewirkt, dass der dynamische Kontext verändert wird: mit einer weiteren Knotensequenz werden wieder neue Templates aufgerufen. Die neue Knotensequenz sind hier die Kindknoten des Dokumentknoten. Es gibt dabei natürlich nur einen Kindknoten, das Wurzelelement. Für diesen Knoten wird das Template "t3" mit der Regel @match="meinDok" abgearbeitet.

In diesem Template wird der erste Bestandteil von "deinDok.xml" erzeugt: das Element <deinDok>. Als Inhalt des Elements steht die Anweisung <xsl:apply-templates select=".//*"/>. Der Unterschied zum vorhergehenden Aufruf von <xsl:apply-templates/> liegt im Attribut @select. Es beschreibt, welche Knotensequenz für die Bildung eines neuen dynamischer Kontext genutzt wird und für weitere Templates bereit steht. Im vorliegenden Fall ist es die Sequenz aller Elemente unterhalb es aktuellen Knotens, also unterhalb von <meinDok>. Da dies in "meinDok.xml" die beiden <meinEl> Elemente Kindelemente von <meinDok> sind, hätte man auch verkürzt <xsl:apply-templates/> schreiben können. Bei einem Eingabedokument, welches weitere, tiefer verschachtelte

<myEl> Elemente enthält, würde jedoch nur die Variante von <xsl:apply-tempates select=".//*"/> alle diese Elemente erfassen.

Für die <meinEl> Elemente wird das Template „t2“ mit der Regel match="myEl" abgearbeitet. Das Template wird zwei Mal durchlaufen, für jedes <meinEl> Element einmal. Zur Ausgabe wird jeweils ein <deinEl> Element hinzugefügt. Für jedes <deinEl> Element wird ein Attribut @deinAttr erzeugt. Seinen Wert erhält es vom Attribut @meinAttr aus dem Eingabedokument.

Diese Form der Verarbeitung wurde von Michael Kay, dem Autor der XSLT 2.0 Spezifikation und des XSLT / XQuery Prozessors Saxon [1], als „*Push Verarbeitung*“ bezeichnet: einem XSLT Stylesheet wird als Teil des dynamische Kontexts eine Sequenz von Knoten „*hingeworfen*“ nach dem Prinzip „*mal sehen, welches Template eine passende Regel hat*“.

Die Templateverarbeitung damit hängt von der jeweiligen Eingabesequenz ab, sie ist also *datengetrieben*. Wenn es im Eingabedokument „meinDok.xml“ z.B. keine Kindknoten von <meinDok> gibt, würden die Templates "t2" und "t3" nicht abgearbeitet.

Das Verarbeitungsmodell von XQuery: „Pull“ Verarbeitung

Komplementär zur Push Verarbeitung ist die so genannte „*Pull Verarbeitung*“. Hierbei werden die zu verarbeitenden Knoten vom Programm aus der Eingabe „*gezogen*“. Die Abfolge der Verarbeitung hängt weitestgehend nicht von den Eingabedokumenten, sondern vor allem von der Anordnung der Anweisungen im Programm ab. Diese Verarbeiung wird anhand der folgenden XQuery Abfrage näher erklärt. Sie erzeugt die gleiche Ausgabe „deinDok.xml“ wie das bereits beschriebene XSLT Stylesheet.

```
xquery version "1.0";
<deinDok>
{
let $input := doc("meinDok.xml")
for $elements in $input/meinDok/meinEl
return
<deinEl deinAttr="{$elements/@meinAttr}"/>
}
</deinDok>
```

[Abb. Pull Verarbeitung mit XQuery]

Der erste Verarbeitungsschritt wird vollkommen unabhängig von einem Eingabedokument durchlaufen: In der Ausgabe wird ein Element <deinDok> erzeugt. Der Inhalt des Elements ist eine Folge von XQuery Anweisungen. Sie werden durch geschweifte Klammern abgegrenzt. Die erste Anweisung bindet eine Variable $input an den Dokumentknoten von „meinDok.xml“. Man kann auch sagen: Die „let“ Anweisung „*zieht*“ oder „*pullt*“ einen Knoten - oder eine Knotensequenz - aus einer Eingabe und bindet die Variable an diesen Knoten.

Die folgende „for“ Anweisung greift auf $input zurück und iteriert über den <meinEl> Elementknoten aus $input, die unterhalb des Wurzelknotens <meinDok> liegen. Die Variable $elements wird an diese Knoten gebunden. Der Unterschied zwischen „for“ und „let“ liegt demnach in der Iteration: die „for“ Anweisung arbeitet alle Knoten der Sequenz nacheinander ab.

Die folgende „return“ Anweisung wird deshalb für jeden Knoten in $elements ausgeführt. Dabei werden <deinEl> Elemente erzeugt. Sie erhalten jeweils ein Attribut @deinAttr. Der Wert wird durch eine weitere XQuery Anweisung bestimmt: $elements/@meinAttr zieht die Werte von @meinAttr Attributen aus den Knoten in $elements. Da $elements die Elementknoten <myEl> enthält, sind es die @meinAttr Attribute von den <meinEl> Elementen.

Der zentrale Unterschied der XSLT und XQuery Verarbeitung sind die einer Anweisung zur Verfügung stehenden Informationen, d.h. die Art, wie der dynamische Kontext definiert und genutzt

wird. Bei XSLT können Anweisungen in einem Template auf alle Knoten zurückgreifen, die als dynamischer Kontext in das Template eingegeben wurden. Zur Bildung des dynamischen Kontextes eines Templates dient wie gesagt diese Knotensequenz. Aus dem dynamischen Kontext heraus ist deshalb bei der Anweisung <deinEl deinAttr="{@meinAttr}"> eindeutig, welche @meinAttr Attribute verarbeitet werden: es sind das Attribut des <meinEl> Elemente, welches aktuell im Template verarbeitet wird.

Bei XQuery hingegen wird der dynamische Kontext einer Anweisung unter anderem aus den aktuell gebundenen Variablen heraus bestimmt. In der XQuery Anweisung steht für die Anweisung {$elements/@meinAttr} die Variable $elements zur Verfügung. Die geschweiften Klammern lösen die Evaluierung des Ausdrucks aus, wodurch der Wert des @meinAttr Attributs ausgelesen werden kann. Gemeinsam ist beiden Verarbeitungsweisen „*Push*“ und „*Pull*“, und auch den beiden Sprachen XSLT und XQuery, die Nutzung von XPath 2.0. Mittels XPath werden Knoten zur weiteren Verarbeitung selektiert. Die Selektion erfolgt über *Funktionen*, *Pfadausdrücke* oder *Filterausdrücke*.

Daumenregel: Wann XSLT, wann Xquery?

Push Verarbeitung lässt sich leicht mit XSLT realisieren. Wann diese Verarbeitungsform angebracht ist, hängt von den Daten ab. In hohem Masse strukturierte, textuelle Daten mit vielen *gemischten Inhaltsmodellen* sind ideal für Push Verarbeitung mit XSLT geeignet. Nehmen wir z.B. das folgende <p> Element als Teil der Eingabe an:

```
<para>Meine <emph>grosse</emph> <note>und süsse</note> Blume.</para>
```

Aus dieser Eingabe soll folgende Ausgabe erzeugt werden:

```
<p>Meine <em>grosse</em> (und süsse) Blume.</p>
```

Mit XSLT ist diese Aufgabe einfach zu bewältigen: Jedes Element innerhalb von <p> wird mit einem entsprechendem Template verarbeitet, z.B. <xsl:template match="em">.

In XQuery ist gemischter Inhalt schwer zu verarbeiten. Die Daumenregel lautet, XQuery für Pull Verarbeitung von Daten mit *wenig gemischten Elementinhalten* zu nutzen. Die Stärke von XQuery liegt in der kombinierten Verarbeitung mehrerer Knotensequenzen aus möglicherweise mehreren Eingabedokumenten. Der Mechanismus für diese Verarbeitung sind FLWOR Ausdrücke. Die Bezeichnung ergibt sich aus den Anfangsbuchstaben der Teilausdrücke: (f)or bindet Knotensequenzen an Variablen und iteriert über diese; (l)et bindet Knotensequenzen ohne Iteration; (w)here beschreibt eine Filterbedingung; (o)rder-by definiert ein Ordnungskriterium; (r)eturn gibt Sequenzen zurück. Diese Operationen sind zwar auch mit XSLT realisierbar. Die XML Repräsentation von XSLT führt dann allerdings schnell zu unübersichtlichem Programmcode.

Die folgende Liste fasst die Unterschiede zwischen XSLT und XQuery Verarbeitung zusammen.

- Grundprinzip
  - XSLT: Templates "warten" auf matchende Knotensequenzen, die sich aus dem dynamischen Kontext ergeben
  - XQuery: Anweisungen "ziehen" Informationen aus den Eingabesequenzen
- Abfolge der Verarbeitung
  - XSLT: Datengetrieben, d.h. abhängig vom Eingabedokument
  - XQuery: Weitgehend unabhängig vom Eingabedokument, nur die Reihenfolge der Anweisungen zählt
- Dynamischer Kontext:

  - XSLT: Im wesentlichen eine Knotensequenz, die im aktuellen Template verarbeitet wird bzw. sich aus der Verarbeitung von XPath Ausdrücken ergibt. Kann durch <xsl:apply-templates> verändert werden
  - XQuery: Im wesentlichen Knotensequenzen, die sich aus aktuell gebundenen Variablen und der Auswertung von XPath Ausdrücken ergeben
- XPath 2.0 Nutzung
  - XSLT: Nutzung aller Pfad- und Filterausdrücke, Funktionen und Operatoren. Zur Template Auswahl / Veränderung des dynamischen Kontexts: Nutzung von XSLT "Pattern".
  - XQuery: Nutzung aller Pfad- und Filterausdrücke, Funktionen und Operatoren

[Kasten: Unterschiede der Verarbeitung mit XSLT und Xquery]

Zum Schluss soll ein kombiniertes Anwendungsbeispiel von XQuery und XSLT stehen. Die in XQuery implementierte Funktion *eg:checkUniqueness* überprüft die Einzigartigkeit von ID Attributen und gibt eine Fehlermeldung wieder. Von XSLT aus lassen sich solche Funktionen aufrufen, wenn dies vom entsprechenden Prozessor (z.B. Saxon) unterstützt wird.

```
xquery version "1.0";
module namespace eg = "http://example.com#";
declare function eg:checkUniqueness
($inputSequence as item()*) as item()*
{
for $node in $inputSequence//element()
for $comparedNodes in $inputSequence//element() except $node
where $comparedNodes/@id = $node/@id
return "Fehler! ID bereits definiert"};
```

Resumee

Die Entscheidung, welche der beiden vorgestellten Sprachen man für eine spezifische Aufgabenstellung verwendet, ist nicht ganz einfach. Als Daumenregel gilt, dass (rekursive) Funktionen, die viele Datenquellen zusammenführen, in XQuery einfacher zu formulieren sind. Querbezüge lassen sich leicht konstruieren, da im *Pull* Ansatz die Daten an Variablen gebunden werden, über denen anschliessend iteriert und gesucht werden kann. Die Verarbeitung von in hohem Masse strukturierten Daten mit gemischtem Inhalt lässt sich mit XSLT einfacher bewerkstelligen. Durch den datengetriebenen Ansatz der *Push* Verarbeitung können allgemeine Templates für unterschiedliche Knoten definiert werden, welche bei Bedarf aktiviert werden. Die kombinierte Nutzung der beiden Ansätze ist möglich, wobei aus XSLT heraus selbstdefinierte XQuery Funktionen aufgerufen werden können.

Links & Literatur

- [1] Micheal Kay, XSLT Reference, 2nd Edition, Hungry Minds Inc.
- [2] Wolfgang Lehner, Harald Schöning, XQuery, dpunkt Verlag, 2004
- [3] XSL-TSpezifikation: http://www.w3.org/TR/xslt
- [4] Serialisierung: http://www.w3.org/TR/xslt-xquery-serialization/
- [5] Serialisierung: http://www.w3.org/TR/xslt-xquery-serialization/#serparam
- [6] Wolfgang Lezius, XPath, Artikel im XML Magazin
- [7] Heinz Wittenbrink, Werner Köhler, XML, SPC Lehrbuch, Berlin