Streaming Transformations for XML — STX

Oliver Becker
Humboldt Universität zu Berlin
wiss. Mitarbeiter am Institut für Informatik,
Lehrstuhl für Systemarchitektur
obecker@informatik.hu-berlin.de

Abstract: STX (Streaming Transformations for XML) ist eine Transformationssprache, die als schnelle und speicherschonende Alternative zu XSLT entworfen wurde. STX weist gewisse syntaktische Ähnlichkeiten zu XSLT auf, arbeitet jedoch auf einem Strom von SAX-Events und transformiert diesen unmittelbar in einen entsprechenden Ergebnis-Strom. STX-Transformationen können auf diese Weise unter anderem als SAX-Filter eingesetzt werden.

1 Einleitung

Die Transformation von XML-Dokumenten zählt zu den häufigsten Aufgaben bei der Verarbeitung von XML. Die in einem Vokabular A vorliegenden Daten werden in ein anderes Vokabular B überführt. Typische Beispiele sind neben der Erzeugung von Präsentationsformaten wie XHTML, XSLFO oder SVG insbesondere die Anpassung der Daten an geänderte Dokumenttypen und das Herausfiltern von Teilsichten auf die Daten. Solche Transformationen können zum Beispiel innerhalb einer Middleware-Schicht ablaufen, die für die Verteilung und Auslieferung von XML-Daten verantwortlich ist.

Das W3C hat zu diesem Zweck die Transformationssprache XSLT (XSL Transformations) als Teil der Stilsprache XSL spezifiziert. XSLT ist leicht zu erlernen und erfordert insbesondere für einfache Transformationen keine Programmierkenntnisse. Ihr deklarativer, template-basierter Charakter dürfte wesentlich zum Erfolg von XSLT beigetragen haben.

Zum Navigieren innerhalb des XML-Dokuments benutzt XSLT die Sprache XPath, die wiederum auf der Baumstruktur des Dokuments operiert. Über XPath kann auf alle Knoten dieser Baumstruktur wahlfrei zugegriffen werden. Dazu muss diese Struktur vollständig im Speicher aufgebaut werden. Dies führt bereits bei Dokumenten mit wenigen MByte Größe zu Problemen.[1] Der Aufwand für den Aufbau einer vollständigen Baumrepräsentation erscheint zudem für einfache Transformationen unangemessen.

Als Alternative bietet es sich an, eine Transformationssprache zu entwickeln, die seriell arbeitet und keine Baumrepräsentation benötigt.

[1] Zum Vergleich: die vollständige RDF-Beschreibung des *Open Directory* benötigt etwa 1 GByte, siehe `http://dmoz.org/rdf.html`

2 Serielle Transformation

Auf programmiersprachlicher Ebene hat sich das *Simple API for XML (SAX)* für die serielle Verarbeitung von XML-Daten etabliert. Das XML-Dokument wird durch einen SAX-Parser in einen Strom von Events zerlegt, der durch eine Applikation verarbeitet wird. SAX selbst ist von der Größe des zu verarbeitenden Dokuments unabhängig – die dahinterliegende Applikation bestimmt, ob und wie die gelieferten Daten gespeichert werden. Dies unterscheidet SAX wesentlich vom Document Object Model (DOM) des W3C.

Der Nachteil einer rein SAX-basierten Vorgehensweise besteht darin, dass sämtliche Datenstrukturen sowie Zustandsinformationen während des Parsens durch die Applikation selbst verwaltet werden müssen. Für die Erstellung einer SAX-Anwendung sind Programmierkenntnisse in einer geeigneten Programmiersprache notwendig. Schließlich ist SAX ein low-level-API, das von sich aus keine Unterstützung für die Erzeugung von XML bietet. Es ist ein reines Parser-API.

Die Transformationssprache *Streaming Transformations for XML (STX)* entstand in dem Bestreben, das Beste der beiden Welten XSLT und SAX zu vereinen. STX ist, ähnlich wie XSLT, template-basiert. Ein STX-Prozessor verarbeitet als Eingabe einen SAX-Eventstrom und liefert als Ausgabe ebenfalls SAX-Events. STX kann also als erweiterter SAX-Filtermechanismus verstanden werden, wobei die Filter- bzw. Transformationsregeln in XML-Syntax notiert werden. STX bietet sich damit als leichtgewichtige, schnelle und ressourcenschonende Alternative zu XSLT an. Insbesondere in Serverumgebungen, in denen mehrere Prozesse parallel auf Klientenanfrage Transformationen ausführen, kann dieser Aspekt entscheidende Bedeutung besitzen.

Für die folgende Darstellung wird ein Grundverständnis von XSLT und SAX vorausgesetzt. Spezielle Eigenschaften von STX werden denen von XSLT gegenüber gestellt.

3 Syntax und Verarbeitungsmodell

STX-Transformationsregeln (im Folgenden *STX-Stylesheets* genannt) werden genau wie in XSLT in Form von Templates notiert. Alle Elemente aus dem STX-Namensraum `http://stx.sourceforge.net/2002/ns` werden als STX-Anweisungen interpretiert, alle anderen Elemente innerhalb von Templates sind Teil der zu erstellenden Ausgabe.

Obwohl SAX bei Elementen zwei zueinander gehörende Events (`startElement` und `endElement`) an die dahinterliegende Applikation liefert, erlaubt STX nur die Verarbeitung von Elementen als Ganzes. Auf diese Weise wird sichergestellt, dass das Resultat der Transformation wohlgeformt ist. Bei unabhängiger Verarbeitung der Events wäre eine solche Garantie nicht mehr gegeben.

In Analogie zu XSLT und in Abweichung zu SAX werden Textdaten (aufeinanderfolgende `character`- oder `ignorableWhitespace`-Events) immer als einzelner Textknoten behandelt.

Da in STX gerade keine Baumrepräsentation erzeugt werden soll, existiert auch der aus XPath bekannte Begriff *Knotenmenge* nicht in derselben Form. Einem Template stehen in seinem Kontext weniger Daten zur Verfügung als in XSLT. Dies sind

- der aktuelle Knoten mit allen zugehörigen Eigenschaften (z.B. der Wert bei Text- und Attributknoten, Namen bei Element- und Attributknoten, etc.)
- die Liste der Vorfahrenknoten
- die Position des aktuellen Knotens innerhalb seiner Geschwister[2]

Diese Daten ermöglichen einen match-Algorithmus mit sehr ähnlichen Pattern, wie sie aus XSLT bekannt sind. Darüber hinaus arbeitet STX mit einem Event als Look-Ahead, sodass auf den Textinhalt eines ausschließlich Text enthaltenen Elements einfacher zugegriffen werden kann.

Anstelle des flexiblen `xsl:apply-templates` aus XSLT sieht STX spezielle Anweisungen zum Bearbeiten von Knoten vor: für Kindknoten `stx:process-children`, für Attribute `-attributes`, für denselben Knotens erneut mit einem anderen Template `-self`, für einen Zwischenspeicher `-buffer`, für andere Dokumente `-document`.

Nach dem Lesen eines `startElement`-Events wird das passende Template gesucht und dessen Inhalt abgearbeitet. Durch `stx:process-children` wird diese Verarbeitung unterbrochen, um die Events für die Kindknoten des Elements verarbeiten zu können. Durch das dazugehörige `endElement`-Event wird die Verarbeitung an der unterbrochenen Stelle wieder aufgenommen. Es ist klar, dass die Anweisung `stx:process-children` nur einmal pro Template verarbeitet werden kann.

Dieser Ansatz bringt, verglichen mit XSLT, zunächst folgende Einschränkungen mit sich:

- Alle Knoten des XML-Dokuments müssen in ihrer Originalreihenfolge (in Dokumentordnung) verarbeitet werden.
- Alle Knoten können nur einmal verarbeitet werden. STX beschreibt eine *one-pass*-Transformation.

Tatsächlich kann durch erneute Transformationen eines Zwischenergebnisses die gleiche Mächtigkeit wie in XSLT erreicht werden. Dazu werden die Ergebnis-Events temporär in einem Puffer zwischengespeichert und können dann erneut transformiert werden. Mit Hilfe dieser Technik lässt sich ein STX-Stylesheet implementieren, das eine in TMML[3] notierte *Universal Turing Machine* ausführt. STX ist damit (wie XSLT) turing-vollständig.

Die Verwendung von Puffern kann bei großen Zwischenergebnissen erheblichen Speicher beanspruchen. Allerdings kann ein STX-Programmierer selbst entscheiden, ob und in welchem Umfang er diese Technik überhaupt einsetzt.

Im Folgenden werden beispielhaft zwei weitere STX-spezifische Eigenschaften vorgestellt, die STX von XSLT abheben.

[2] Die Position eines Knotens ist in STX anders definiert als in XSLT. Sie hängt insbesondere vom `match`-Pattern des Templates ab.

[3] Turing Machine Markup Language, siehe `http://www.unidex.com/turing/`

3.1 Variablen sind änderbar

Der funktionale Charakter von XSLT bedingt u.a., dass Variablen nach einer Wertzuweisung nicht mehr geändert werden können. Die Ausführung eines XSLT-Stylesheets ist frei von Nebeneffekten. Einzelne Templates können von einem geeigneten Prozessor ohne weiteres parallel und unabhängig voneinander ausgeführt werden.

Da ein STX-Prozessor eine Folge von SAX-Events verarbeitet, deren Reihenfolge sich eindeutig aus dem XML-Dokument ergibt, ist auch die Reihenfolge der instanziierten Templates durch die Eingabe vorgegeben. Es gibt keine parallelen Zweige während der Verarbeitung. Tatsächlich erfordert das Mitführen eines Zustands während der Transformation veränderbare Variablen.

Variablen werden wie in XSLT mit einer entsprechenden Anweisung `stx:variable` definiert. Zuweisungen werden durch ein eigenes Element `stx:assign` vorgenommen:

`<stx:assign name="`*Variablenname*`" select="`*Ausdruck*`" />`

bzw.

```
<stx:assign name="Variablenname">
    Inhalt
</stx:assign>
```

3.2 Templates können gruppiert werden

In XSLT stehen alle Templates auf globaler Ebene. Die Auswahl des jeweils passendsten Templates geschieht über das Pattern im `match`-Attribut und anschließend über implizite oder explizite Prioritäten. Das `mode`-Attribut erlaubt die Einschränkung des Suchraums, indem nur die Templates des ausgewählten Modus in Betracht gezogen werden. Allerdings fördert XSLT die Erstellung gut strukturierter Stylesheets nicht, da alle zu einem Modus gehörenden Templates beliebig angeordnet werden können.

STX kennt ebenfalls `match`-Pattern und Prioritäten. Anstelle eines `mode`-Attributs für Templates können hier mehrere Templates in einer benannten Gruppe zusammengefasst werden. Eine solche Gruppe bildet einen lokalen Suchraum, sodass nur die enthaltenen Templates beim Matchen berücksichtigt werden. Für jede der `stx:process-...`-Anweisungen kann ein zusätzliches `group`-Attribut angegeben werden, das einen Wechsel in die spezifizierte Gruppe bewirkt.

Eine Gruppe wirkt damit wie ein lokales Stylesheet. Dies hat zweierlei Vorteile:

- Der Stylesheet-Entwickler kann relativ leicht die Menge der Templates überblicken, die für die folgenden Transformationsschritte eine Rolle spielen.
- Der STX-Prozessor muss nicht alle Templates beim Matchen in Betracht ziehen, sondern nur die aus der aktuellen Gruppe. Auf diese Weise kann die Transformation entsprechend schneller ausgeführt werden.

Darüber hinaus können Gruppen hierarchisch ineinander verschachtelt und anonym (ohne Namen) angelegt werden. Das Dokumentelement `stx:transform` bildet die äußerste Gruppe.

```
<stx:transform ...>
   <stx:template ...> ... </stx:template>
   <stx:group ...>
      <stx:template ...> ... </stx:template>
      <stx:template ...> ... </stx:template>
      <stx:group ...>
         <stx:template ...> ... </stx:template>
         ...
      </stx:group>
      <stx:group ...>
         <stx:template ...> ... </stx:template>
         ...
      </stx:group>
      ...
   </stx:group>
   ...
</stx:transform>
```

In diesem Fall wird automatisch in eine Gruppe gewechselt. Die Templates einer Gruppe können verschiedene Sichtbarkeiten besitzen, die diesen Prozess steuern:

- `private`: das Template ist nur innerhalb der Gruppe, zu der es gehört, sichtbar. Dies ist der voreingestellte Wert, falls das Attribut `visibility` nicht angegeben wurde.
- `public`: das Template ist auch aus seiner Elterngruppe heraus sichtbar.
- `global`: das Template ist global, d.h. stylesheet-weit sichtbar. Globale Templates sind per Definition immer öffentlich (`public`).

Die Auswahl des jeweils nächsten Templates innerhalb einer Gruppe geschieht dann nach folgendem Algorithmus:

1. Es werden alle Templates aus der gleichen Gruppe und alle öffentlichen Templates aus den Kind-Gruppen betrachtet.
2. Wurde dann kein passendes Template gefunden, werden alle globalen Templates betrachtet.

Globale Templates ermöglichen damit das Springen über beliebige Gruppengrenzen hinweg und sollten daher mit Bedacht eingesetzt werden.

Schließlich können Gruppen eigene Variablen besitzen, die das Konzept der globalen Variablen erweitern.

3.3 Weitere Neuerungen

Weitere durch STX eingebrachte Innovationen können hier nur nur stichpunktartig genannt werden, zum Beispiel

- das getrennte Erzeugen von Start- und Endtags; wichtig für Gruppierungsprobleme
- die Template-artige Verarbeitung von Textdaten; erlaubt einfaches Ersetzen von Zeichenfolgen durch Markup
- die Verarbeitung von CDATA-Abschnitten in Ein- und Ausgabe
- die Serialisierung von XML-Markup als Text; sinnvoll, um Teile der XML-Eingabe als Text darzustellen oder per STX-Transformation auszukommentieren

4 Erfahrungen und Ausblick

Die Arbeit an STX begann im Februar 2002 und wird seitdem auf einer eigenen Mailingliste vorangetrieben [1]. Parallel dazu wurden erste prototypische Implementationen in Java [2] und Perl [3] begonnen, um die Realisierbarkeit der Sprache sicherzustellen.

Ein wichtiges Designziel (neben Einfachheit, Mächtigkeit und Effizienz) ist die Nähe zu XPath. Da kein vollständiger Baum im Speicher gehalten wird, kann STX nur eine XPath-Teilmenge beinhalten. Diese wird aber kompatibel zu XPath 2.0 [4] sein. Anwendern mit XSLT-Kenntnissen sollte so das Erlernen von STX leicht fallen.

Erste Tests mit der vom Autor erstellten Java-Implementierung Joost [2] zeigen, dass Dokumente unabhängig von ihrer Größe problemlos transformiert werden können. Da Joost zusätzlich die TrAX-Schnittstelle [5] implementiert, ist eine Einbindung in Java-Applikationen und der Umstieg von XSLT auf STX innerhalb solcher Applikationen ohne Schwierigkeiten möglich.

Literatur

[1] STX-Homepage mit aktueller Spezifikation und Archiv der Mailingliste, siehe `http://stx.sourceforge.net/`

[2] Joost, Java-Implementierung eines STX-Prozessors von Oliver Becker, siehe `http://joost.sourceforge.net/`

[3] XML::STX, Perl-Implementierung eines STX-Prozessors von Ginger Alliance, siehe `http://stx.gingerall.cz/stx/xml-stx/`

[4] XPath 2.0, W3C Working Draft, siehe `http://www.w3.org/TR/xpath20/`

[5] Transformation API for XML, Bestandteil der *Java API for XML Processing – JAXP*, siehe `http://java.sun.com/xml/jaxp/`