

Sonnige Aussichten

Einführung in XML und XSLT, Teil 2: Ein starkes Team?

Die leistungsfähige Verarbeitung von XML-Dokumenten mit XSLT-Stylesheets wird oft als Grund genannt, warum sich XML im Gegensatz zu seinem Vorgänger SGML so schnell etablieren konnte. In diesem Artikel untersuchen wir an einem Anwendungsbeispiel, welche Möglichkeiten XSLT-Stylesheets eröffnen. Im Anschluss daran beantworten wir die zentrale Frage unseres Workshops „XML und XSLT – Ein starkes Team?“.

von Arne Fitschen und
Wolfgang Lezius

In der vergangenen Ausgabe haben wir uns mit XML als Dokumentenaustauschformat befasst. Beim Einsatz im Anwendungsbeispiel „Kodierung von Wetterdaten“ haben wir zunächst die XML-Syntax kennen gelernt und anschließend gesehen, wie Schemata bzw. DTDs die Definition von Dokumentklassen und damit die Validierung von Dokumenten ermöglichen. Das volle Potenzial von XML zeigt sich aber erst in der Weiterverarbeitung. Eine zentrale Rolle spielen dabei die so genannten XSLT-Stylesheets. Wir werden in diesem Artikel am Beispiel der Wetterdatensammlung untersuchen, welche Möglichkeiten XSLT-Stylesheets bei der Weiterverarbeitung von XML-Dokumenten bieten.

> Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

XSL

Zur Verarbeitung von XML-Dokumenten hat das W3C die Extensible Stylesheet Language (XSL) entwickelt ([1]). Sie besteht aus den beiden Komponenten XSLT zur *Transformation* von Dokumenten und XSLF zur *Formatierung* von Dokumenten. Das Einsatzgebiet von XSLT liegt in der Konvertierung von XML-Dokumenten in Formate wie HTML oder XML. Die Idee von XSLF besteht darin, eine standardisierte Layoutsprache für Textdokumente zu schaffen. Die so entstandenen XSLF-Dokumente können dann mittels geeigneter Tools in gängige Druckformate wie PDF überführt werden. Während sich XSLT schnell etablieren konnte, hat sich XSLF u.a. aufgrund seiner hohen Komplexität bislang nicht durchsetzen können. Eine Einführung in XSLF findet sich in [2].

Das XSLT-Verarbeitungsmodell

Bei der Verarbeitung von XML-Dokumenten durch XSLT-Stylesheets kommen drei Komponenten zusammen (vgl. Abb.

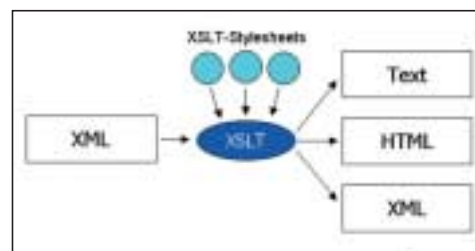


Abb. 1: Das XSLT-Verarbeitungsmodell

1): Die XML-Datei bildet die Datengrundlage, die in ein anderes Format überführt und dabei ggf. umstrukturiert werden soll. Das XSLT-Stylesheet beschreibt durch Transformationsregeln, wie die Bestandteile der XML-Datei (Elemente, Attribute usw.) in das Zielformat überführt werden. Ein XSLT-Stylesheet ist selbst wieder eine XML-Datei – damit stehen dieselben Tools zum Bearbeiten von Stylesheets und XML-Dokumenten zur Verfügung. Für den eigentlichen Transformationsprozess wird ein so genannter Stylesheet-Processor benötigt. Er wendet die Regeln des Stylesheets auf die XML-Datei an und legt die Ausgabe in der

Zieldatei ab. Es gibt mittlerweile eine ganze Reihe frei verfügbarer Stylesheet-Prozessoren. Wir werden in diesem Projekt den Prozessor Xalan der Apache Gruppe verwenden (vgl. Kasten: „Der Stylesheet-Prozessor Xalan“). Als Zielformate kommen beliebige textbasierte Formate wie XML, HTML, WML oder auch Textverarbeitungsformate wie RTF oder LaTeX in Frage.

Listing 1: Ein erstes Stylesheet

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="wetter">
  <HTML>
  <BODY BGCOLOR="white">
  <H3>Wetterdaten der Stadt Frankfurt</H3>
  <TABLE BORDER="1">
  <TR><TH>Tag</TH><TH>morgens</TH><TH>mittags</TH><TH>abends</TH><TH>nachts</TH></TR>
  <xsl:apply-templates select="tag"/>
  </TABLE>
  </BODY>
  </HTML>
</xsl:template>
<xsl:template match="tag">
  <TR>
  <TD><xsl:value-of select="@wochentag"/>.
    <xsl:value-of select="@tag"/>.
    <xsl:value-of select="@monat"/>.
    <xsl:value-of select="@jahr"/></TD>
  <xsl:apply-templates select="messung[@stadt=
    'Frankfurt']"/>
  </TR>
</xsl:template>
<xsl:template match="messung">
  <TD><xsl:value-of select="temperatur/
    @morgens"/></TD>
  <TD><xsl:value-of select="
    temperatur/@mittags"/></TD>
  <TD><xsl:value-of select="
    temperatur/@abends"/></TD>
  <TD><xsl:value-of select="
    temperatur/@nachts"/></TD>
</xsl:template>
</xsl:stylesheet>
```

XSLT-Stylesheets

Zum Einstieg wollen wir ein Stylesheet entwickeln, das alle erfassten Wetterdaten der Stadt Frankfurt in Form einer HTML-Tabelle ausgibt. Die Struktur der XML-Wetterdatensammlung ist dem folgenden Beispieldokument zu entnehmen (eine umfangreiche Wetterdatensammlung befindet sich auf der Heft-CD).

```
<?xml version="1.0" encoding="iso-8859-1"
  standalone="no"?>
<!DOCTYPE wetter SYSTEM „wetter.dtd“>
<wetter>
  <tag wochentag="So" tag="1" monat="4" jahr="2001">
  <messung stadt="Stuttgart">
  <temperatur morgens = „7“ mittags = „16“
    abends = „13“ nachts = „9“ />
  </messung>
  <messung stadt="Frankfurt">
  <temperatur morgens = „8“ mittags = „16“
    abends = „12“ nachts = „8“ />
  </messung>
</tag>
</wetter>
```

Ein XSLT-Stylesheet stellt eine Sammlung von so genannten Templates dar, d.h. Transformationsregeln, die festlegen, wie die Bestandteile des XML-Dokuments in das Zielformat überführt werden sollen. Eine naheliegende Vorgehensweise besteht für den Entwickler darin, beginnend mit dem Wurzelement den Elementbaum des Dokuments durcharbeiten und für jede Elementebene in einem eigenen Template festzulegen, wie der zugehörige Inhalt (Attributwerte, Elementinhalt) im Zielformat zu erscheinen hat. Anhand des Stylesheets in Listing 1, das die gewünschte HTML-Tabelle erzeugt, zeigen wir die Umsetzung dieser Vorgehensweise. Eine umfassende XSLT-Referenz ist in [3] zu finden.

Für das Wurzelement `<wetter>` wird in einem `<xsl:template>`-Element das zugehörige Template definiert. Die Belegung des Attributs `match` legt fest, dass dieses Template für `<wetter>`-Elemente zuständig ist. Dem Template kommt zunächst die Aufgabe zu, das Gerüst des HTML-Dokuments und eine geeignete Überschrift zu erzeugen. Die dazu nötigen HTML-Elemente `<HTML>`, `<BO-`

`DY>` usw. werden in den Rumpf des Templates eingetragen. Um Namenskonflikte zu vermeiden (z. B. bei der Erzeugung eines Elements `<template>` im Falle von XML als Zielformat), wird für XSLT-Anweisungen ein eigener Namensraum `xsl` verwendet. XSLT-Anweisungen werden dann vom Stylesheet-Prozessor gesondert verarbeitet, während alle übrigen Elemente einfach in die Zieldatei kopiert werden. Es ist zu beachten, dass XSLT-Stylesheets als XML-Dokumente auch den Restriktionen von XML unterworfen sind: Öffnende Tags müssen stets geschlossen werden, ungeschlossene Elemente (z. B. die in HTML-Dokumenten oft verwendeten `<P>` und `
`) sind damit unzulässig. Zudem müssen Attributwerte explizit in Anführungsstriche gesetzt werden.

Nach der Überschrift erzeugt das Template eine Tabellenumgebung. Die oberste Zeile trägt die Überschriften für die einzelnen Spalten `morgens`, `mittags` usw. Um den Inhalt der Tabelle kümmert sich dieses Template nicht, schließlich sind die Messungsdaten erst in den unteren Ebenen der Elementhierarchie zu finden. Damit auch diese Daten ausgegeben werden, gibt das Template mit Hilfe der XSLT-Anweisung `<xsl:apply-templates>` die Kontrolle an ein weiteres Template ab, das für die Verarbeitung von `<tag>`-Elementen zuständig ist. Für alle `<tag>`-Kinder des Wurzelements wird dann nacheinander das zuständige Template aufgerufen.

Das Template für `<tag>`-Elemente gibt zunächst das Datum einer Messung aus. Dazu ist der Zugriff auf die vier Attribute `wochentag`, `tag`, `monat`, `jahr` erforderlich. Um den Wert eines Ausdrucks in die Zieldatei zu kopieren, ist die XSLT-Anweisung `<xsl:value-of>` vorgesehen. Sie fragt den Wert des Attributs `select` ab, wertet den dort abgelegten Ausdruck aus (hier sind beliebige XPath-Ausdrücke zulässig, für eine Referenz vgl. [3]) und kopiert den ermittelten Wert in die Zieldatei. Attribute werden in XPath-Ausdrücken mit dem Klammeraffen gekennzeichnet, um Konflikte mit Elementnamen aufzulösen (im Beispiel: Attribut `@tag` im Element `tag`).

Nachdem auf diese Weise das Datum eines `<tag>`-Elements in eine Tabellenspalte geschrieben wurde, gibt das Template zur Ausgabe der Temperatur-Werte die Kontrolle an das Template für `<messung>`-Elemente ab. Die Besonderheit besteht an dieser Stelle darin, dass nicht alle `<messung>`-Kinder des aktuellen `<tag>`-Elements verarbeitet werden dürfen, da wir lediglich die Messungen der Stadt Frankfurt ausgeben wollen. Dies wird durch eine nähere Spezifikation der weiterzuverarbeitenden Elemente erreicht: In eckigen Klammern wird an `<messung>`-Elemente die Bedingung gestellt, dass das `stadt`-Attribut mit dem Wert `Frankfurt` belegt ist: `[@stadt='Frankfurt']`. Die Zeichenkette steht hier in einfachen Anführungsstrichen, da in XML doppelte Anführungsstriche in Attributwerten (hier Attribut `select`) nicht erlaubt sind.

Das abschließende `<messung>`-Template gibt die vier Tagestemperaturen in jeweils eigenen Tabellenspalten aus. Hier wird ersichtlich, dass Elementebenen übersprungen werden können. Für `<temperatur>`-Elemente wird kein eigenes Template definiert, stattdessen erfolgt der Zugriff auf die Temperaturattribute über eine Pfadangabe.

Ausgabe

Zu Beginn des Stylesheets legen wir durch die Anweisung `<xsl:output>` HTML als Ausgabeformat fest. Dadurch wird dem Stylesheet-Prozessor mitgeteilt, dass Sonderzeichen automatisch in die korrekte HTML-Schreibweise konvertiert werden müssen. Starten wir nun den Stylesheet-Prozessor, d.h., verknüpfen wir unsere Wetterdatensammlung mit dem Beispiel-Stylesheet (z.B. mit dem Stylesheet-Prozessor Xalan, vgl. Kasten), stößt der Stylesheet-Prozessor den Prozessierungsvorgang für das Wurzelement an. Als Ausgabe der Verarbeitung entsteht ein HTML-Dokument (vgl. Abb. 2).

Parametrisierung

Ein Manko des Beispiel-Stylesheets besteht darin, dass wir die Auswahl der Stadt Frankfurt explizit kodiert haben. Sollen die Temperaturen einer anderen

Stadt angezeigt werden, müssen wir den Code stets entsprechend anpassen. Eine elegante Lösung dieses Problems besteht in der Verwendung sogenannter Top-Level-Parameter. Wie der Name bereits nahe legt, werden Top-Level-Parameter zu Beginn des Stylesheets (auf der obersten Elementebene) durch die Anweisung `<xsl:param>` deklariert und mit einer Default-Belegung versehen. Diese Parameter stehen dann wie globale Variablen in allen Templates zur Verfügung. Zur Unterscheidung von Elementnamen wird in XSLT einem Parameternamen das `$`-Zeichen vorangestellt.

Da Top-Level-Parameter sowohl Zeichenketten als auch ganze Elementbäume enthalten dürfen, muss die Default-Belegung `select="'Frankfurt'"` in unserem Falle durch einfache Anführungsstriche explizit als Zeichenkette gekennzeichnet werden (vgl. Listing 2). Ansonsten würde der Wert als Element `<Frankfurt>` interpretiert. Wir verwenden die Zeichenkette bei der Auswahl der `<messung>`-Elemente im `<tag>`-Template: Hier schränken wir die Ausgabe der Temperaturen auf die im Parameter gespeicherte Stadt ein. Beim Aufruf des Stylesheet-Prozessors können wir nun für den Parameter `stadt` einen Wert übergeben, der die Default-Belegung überschreibt.

SVG

Das parametrisierte Stylesheet illustriert zwar den XSLT-Verarbeitungsprozess, wird aber dem Anwendungsbeispiel nicht

The screenshot shows a Netscape browser window with the title "Wetterdaten der Stadt Frankfurt". The browser's address bar shows "http://www.w3.org/1999/XSL/Transform". The main content is a table with the following data:

| Tag | morgens | mittags | abends | nachts |
|-----------------|---------|---------|--------|--------|
| So. 1. 4. 2001 | 8 | 16 | 12 | 8 |
| Mo. 2. 4. 2001 | 9 | 20 | 19 | 12 |
| Di. 3. 4. 2001 | 7 | 12 | 8 | 4 |
| Mi. 4. 4. 2001 | 7 | 14 | 12 | 5 |
| Do. 5. 4. 2001 | 7 | 12 | 10 | 4 |
| Fr. 6. 4. 2001 | 8 | 13 | 12 | 10 |
| Sa. 7. 4. 2001 | 9 | 12 | 12 | 2 |
| So. 8. 4. 2001 | 4 | 13 | 11 | 7 |
| Mo. 9. 4. 2001 | 5 | 9 | 8 | 7 |
| Di. 10. 4. 2001 | 9 | 10 | 10 | 8 |
| Mi. 11. 4. 2001 | 11 | 11 | 11 | 7 |

Abb. 2: Ergebnis der Stylesheet-Prozessierung

gerecht – Temperaturdaten werden am eindrucksvollsten in Form einer Grafik visualisiert. Zwar lassen sich mit einigen HTML-Tricks recht ordentliche Ergebnisse erzielen (vgl. das Stylesheet zur Erzeugung einer HTML-Balkengrafik auf der Heft-CD), doch so richtig zu begeistern vermögen auch diese Lösungen nicht. Doch wie können wir mit XSLT eine Grafikdatei erzeugen, zumal die gängigen Grafikformate wie GIF oder JPG binäre Formate sind?

Die Antwort auf diese Frage führt uns zu einem neuen, XML-basierten Grafikformat: *Scalable Vector Graphics (SVG)* befindet sich kurz vor der Standardisierung durch das W3C und stellt vermutlich

Listing 2: Beispiel-Stylesheet mit Top-Level-Parameter

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:param name="stadt" select="'Frankfurt'"/>
  ...
  <xsl:template match="tag">
    <TR>
      <TD><xsl:value-of select="@wochentag"/>. <xsl:value-of select="@tag"/>. <xsl:value-of select="@monat"/>.
      <xsl:value-of select="@jahr"/></TD>
      <xsl:apply-templates select="messung[@stadt=$stadt]"/>
    </TR>
  </xsl:template>
  ...
</xsl:stylesheet>
```

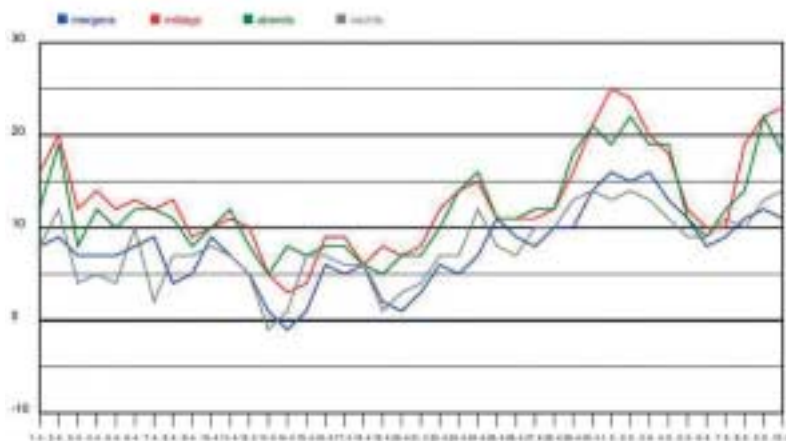


Abb. 3: SVG-Grafik zur Wetterdatensammlung

den Standard der Zukunft zur Beschreibung zwei-dimensionaler Grafiken dar ([5],[6]). Die zentrale Idee besteht darin, eine Grafik aus skalierbaren Primitiven

zusammensetzen, die durch XML-Elemente beschrieben werden. Das folgende XML-Dokument stellt eine vollständige SVG-Grafik dar, die einen Polygonzug enthält:

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
„http://www.w3.org/TR/2000/CR-SVG-20001102/
DTD/svg-20001102.dtd“>

<svg height="200" width="300">
<polyline points="0 0 100 75 300 200"/>
</svg>
```

Zu den Vorteilen eines XML-basierten Grafikformats zählen die nachträgliche Veränderbarkeit in jedem Texteditor sowie die leichte Verarbeitbarkeit sowohl bei der nachträglichen Bearbeitung (z. B. Veränderung der Grafik durch Stylesheets) als auch bei der Erzeugung des Formats. In unserem Wetterbeispiel lässt sich eine SVG-Grafik bequem per XSLT-Stylesheet erzeugen. Listing 3 zeigt ein Stylesheet, das den Verlauf der Mittagstemperaturen als Polygonzug in einer SVG-Grafik darstellt.

Das Stylesheet produziert lediglich einen Grafikbaustein: Das Template für das Wurzelement `<wetter>` legt den Polygonzug und seine Formatierung (Farbe, Linienstärke usw.) fest. Ein Problem besteht darin, dass die Punkte des Polygonzugs in SVG nicht durch eigene Elemente deklariert werden, sondern allesamt im Attribut `points` aufgelistet werden (vgl. auch obiges SVG-Beispiel). Hier kommt die XSLT-Anweisung `<xsl:attri-`

bute> zum Zuge. Sie fügt das Attribut `points` ein, dessen Wert durch den Elementinhalt des `<xsl:attribute>`-Elements bestimmt wird.

Die Auflistung selbst wird im untergeordneten Template für das `<tag>`-Element erzeugt. Für jede Temperatur wird die x- und y-Koordinate ausgerechnet und ausgegeben. Die x-Koordinate ergibt sich aus der Breite der Zeichenfläche 800 und der Nummer der aktuellen Temperatur. Die y-Koordinate wird relativ zur Nullmarke auf der Höhe 300 aus dem Temperaturwert errechnet.

Auf der Heft-CD befindet sich ein Stylesheet, das alle vier Temperaturkurven sowie ein Liniengitter und Beschriftung erzeugt. Die Ausgabegrafik dieses Stylesheets (vgl. Abb. 3) zeigt, dass sich bereits mit einfachen Mitteln ansprechende SVG-Grafiken herstellen lassen. Zum Betrachten von SVG-Dateien wird für die gängigen Browser ein Plug-In von Adobe angeboten ([7]). Hersteller führender Grafiktools haben SVG-Unterstützung für die kommenden Versionen angekündigt. Hilfreich ist ebenfalls das Projekt Batik der XML Apache Gruppe, das Java-Unterstützung zum Anzeigen, Erzeugen von SVG-Grafiken und Konvertieren in andere Formate bietet ([4]).

Diskussion

Die Möglichkeiten, die XSLT eröffnet, sind durchaus beachtenswert. XML-Dokumente lassen sich in beliebige Formate verwandeln und dabei problemlos umstrukturieren (Inhalte gruppieren, sortieren usw.). Weiterführende Möglichkeiten bestehen u.a. darin, in XML kodierte Datensammlungen mit Hilfe von XSLT im WWW zu präsentieren und mittels der Parametrisierung abfragbar zu machen. Ein Projekt zur Verwaltung von Bibliographiedaten wird beispielsweise in [8] beschrieben. Die größten Nachteile von XSLT bestehen in der recht gewöhnungsbedürftigen Syntax und der zum Teil schwierigen (rekursiven) Umsetzung einfacher Problemlösungen. Bedingt durch die rekursive Verarbeitung tun sich hier Entwickler leichter, die im Bereich der funktionalen Programmierung (Prolog, Lisp) groß geworden sind.

Der Stylesheet-Prozessor Xalan

Der bekannteste Stylesheet-Prozessor ist Xalan, ein Projekt der XML Apache Gruppe ([4]). Xalan bindet wiederum den XML-Parser Xerces ein, der ebenfalls von der XML Apache Gruppe entwickelt wird. In der Xalan-Distribution, die mittlerweile als Java- und C++-Bibliothek vorliegt, ist Xerces bereits enthalten. Um Xalan unter Java als Stylesheet-Prozessor zu verwenden, entpacken Sie die Xalan-Distribution und starten folgende Befehlszeile (unter Unix den Doppelpunkt statt Semikolon als Trenner zwischen den beiden Bibliotheken `xalan.jar` und `xerces.jar` verwenden):

```
java -classpath xalan.jar;xerces.jar
org.apache.xalan.xslt.Process -IN wetter.xml -XSL beispiel1.xslt -OUT beispiel1.html
```

Um einen Top-Level-Parameter zu übergeben, erweitern Sie die Befehlszeile um den Parameter `-PARAM`:

```
java -classpath xalan.jar;xerces.jar
org.apache.xalan.xslt.Process -IN wetter.xml -XSL beispiel2.xslt -OUT beispiel2.html -PARAM stadt 'Stuttgart'
```

Da Unix einfache Anführungsstriche als Metazeichen verwendet, müssen sie auf der Shell durch Backslashes maskiert werden (`\`Stuttgart\``).


Ein starkes Team!

Nachdem wir uns ausführlich mit den beiden Partnern XML und XSLT beschäftigt haben, können wir nun die eingangs gestellte Frage „XML und XSLT – Ein starkes Team?“ beantworten. Wir haben gesehen, dass sowohl XML als auch XSLT eine Reihe nützlicher Vorteile bringen. Doch erst ihr Zusammenspiel eröffnet eine breite Palette von Möglichkeiten: XML erlaubt die standardisierte Kodierung von Daten, die mit XSLT in einer standardisierten Weise aufbereitet werden. Somit stehen nicht nur Mittel und Wege zur reinen Datenhaltung, sondern

zugleich zur weiteren Verarbeitung zur Verfügung.

Doch wo liegt der Unterschied zu früheren Datenhaltungsmodellen? Weshalb auf textbasierte Datenhaltung zurückgehen, nachdem sich relationale Datenbanken als Standard etabliert haben? Zum einen lässt sich der Austausch der Daten in XML viel leichter realisieren als bei Datenbanken. Der Zeichensatz ist stets angegeben, Sonderzeichen sind in eindeutiger Weise kodiert (Unicode). Die Beschreibung des Datenmodells durch DTD oder Schema garantiert das problemlose Zusammenfügen verschiedener Datenbe-

stände. Zum anderen steht den Entwicklern mit XSLT-Stylesheets ein Werkzeug zur Verfügung, um Datenbestände in einer standardisierten Art und Weise weiterzuverarbeiten. Stylesheets bieten hier die Chance, an die Stelle der bislang verwendeten Programmiersprachen (z. B. Java oder Perl) zu treten und eine **einheitliche** Verarbeitung von Daten zu gewährleisten.

Leider ist diese Sichtweise naiv und nur für den Fall kleiner Datenbestände uneingeschränkt zutreffend. Im Falle großer Datenbestände ist der Speicherbedarf bei der Stylesheet-Prozessierung enorm und damit schnell unpraktikabel (bei der Verarbeitung einer 1 MB großen XML-Datei belegt Xalan bis zu 100 MB Hauptspeicher). Außerdem entfallen bei der Datenhaltung in Textdateien sämtliche Vorteile der sicheren Verwaltung größerer Datenmengen, die Datenbanken von vornherein bieten. Der derzeit praktizierte Kompromiss sieht eine Symbiose vor: Die Daten werden in einer Datenbank gehalten, aber XML wird als Zwischenformat für alle Weiterverarbeitungsschritte verwendet. Man vereint die Vorzüge beider Modelle und kann das Potenzial des starken Teams XML und XSLT voll nutzen. 

Listing 3: Ein Stylesheet zur Erzeugung von SVG

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"
    standalone="no"
    indent="yes"
    omit-xml-declaration="no"
    encoding="ISO-8859-1"
    doctype-public="-//W3C//DTD SVG 20001102//EN"
    doctype-system="http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd"
  />
  <xsl:param name="stadt" select="'Frankfurt'"/>
  <xsl:template match="wetter">
    <svg width="950" height="550" viewBox="-50 -50 900 500">
      <polyline style="fill:none;stroke:red;stroke-width:2">
        <xsl:attribute name="points">
          <xsl:apply-templates select="tag"/>
        </xsl:attribute>
      </polyline>
    </svg>
  </xsl:template>
  <xsl:template match="tag">
    <xsl:variable name="anzahl" select="last()-1"/>
    <xsl:variable name="abstand" select="800 div $anzahl"/>
    <xsl:variable name="x" select="$abstand*(position()-1)"/>
    <xsl:value-of select="$x"/>
    <xsl:text> </xsl:text>
    <xsl:variable name="wert" select="messung[@stadt=$stadt]/temperatur/@mittags"/>
    <xsl:value-of select="300-((($wert)*10)"/>
    <xsl:text> </xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Links & Literatur

- [1] WWW Consortium W3C - XSL: www.w3.org/Style/XSL/
- [2] Manfred Knobloch, Matthias Kopp: Web-Design mit XML, dpunkt Verlag
- [3] Michael Kay: XSLT Programmer's Reference, Wrox Press
- [4] XML Apache Group - Xalan, Xerces, Batik: <http://xml.apache.org>
- [5] WWW Consortium W3C - SVG: www.w3.org/Graphics/SVG
- [6] Volkmar Großwendt: „Blitzschnelle Bilder - Scalable Vector Graphics: Professionelle Vektorgrafiken fürs Web“, in: *Java Magazin* 4.01
- [7] Adobe SVG-Plug-In: www.adobe.com/svg/viewer/install
- [8] Wolfgang Lezius: „Offline garniert, online serviert - Ein Java Servlet zur Online-Präsentation von Daten mit XML und XSLT“, in: *Java Magazin* 3.01