Einsatzgebiete von XSLT

Manfred Knobloch, Institut für Wissensmedien (IWM), manfred.knobloch@web.de

Einleitung

XSLT ist eine Sprache, die es erlaubt XML-Dokumente in andere XML-Dokumente umzuwandeln. Das klingt zunächst nicht besonders spannend. Wozu wurde eine Sprache entwickelt, die nichts weiter kann als Daten innerhalb eines Formats zu manipulieren?

Das Missverständnis, das in dieser Frage zum Ausdruck kommt, liegt darin begründet, dass XML als Datenformat betrachtet wird. Das ist nicht falsch. Dennoch lässt der Begriff Datenformat etwas mitschwingen, was bei den meisten herkömmlichen Datenformaten zutrifft, dass sie nämlich einen festgelegten Aufbau besitzen. Genau das trifft aber auf XML gerade nicht zu. Extensible Markup bedeutet bei XML nicht, dass eine vorgegebene Menge von Schlüsselwörtern um eigene erweiterbar ist, sondern dass jede Anwendung von XML einen eigenen Satz von Schlüsselwörtern verwenden kann. In diesem Sinne ist XML kein Datenformat, sondern wie inzwischen auch in der Literatur überall zu finden, ein Metaformat. Eben ein Ding, mit dem Formate gemacht werden können.

Wenn aber jede Anwendung von XML quasi einem eigenen Format entspricht, dann bedeutet eine Transformation von XML nach XML mehr als das Umrühren von »angedicktem Datenbrei«. XSLT ist in diesem Rahmen ein Formatkonverter, der immer dann angewendet werden kann, wenn die Eingabedaten in XML notiert sind. Produktiv wird XSLT, wenn auch das oder die Zielformate XML-basiert sind, wie das beispielsweise bei XHTML und anderen Formaten der Fall ist.

Bevor ich darauf eingehe, was dies konkret bedeutet, möchte ich kurz und sehr oberflächlich die Entwicklung von XLST und die Einbettung in die XML-Spezifikationsfamilie darstellen.

Entwicklung von XSLT

XML wurde konzipiert, um Daten zu strukturieren. Ganz bewusst wurde die Frage der Darstellung im Sinne von Visualisierung und Formatierung in der XML-Spezifikation nicht bedacht. XML beansprucht dennoch – im Gegensatz zu binären For-

maten – ein von Menschen lesbares Format zu sein. Jeder, der einmal längere oder komplexere XML-Dateien gelesen hat, weiß, dass dies zwar möglich ist, mit herkömmlichem Verständis von Lesbarkeit oder Lektüre jedoch nichts zu tun hat.

Als einfaches Beispiel möchte ich eine Datei vorstellen, die eine Auflistung der Stärken und Schwächen von Entscheidungsvarianten enthält. Diese Datei dient als Grundlage für die meisten der nachfolgend vorgestellten Beispiele.

Als Informatiker hat man ja bekanntlich eine große Auswahl an attraktiven Jobs. Um die Qual der Wahl zu versachlichen wurden die Stärken, Schwächen, Chancen und Risiken einer Anstellung an einer Hochschule der einer Arbeit in der freien Wirtschaft in folgender Datei gegenübergestellt:

```
<swots id="1">
   <desc>Hochschule oder Privatwirtschaft?</desc>
   <swot title="Arbeiten in der Privatwirtschaft"
                id="c-work">
      <strengths>
         <item count="1">
            Arbeitgeber ist gute Referenz
         </item>
         .....
      </strengths>
      <weaknesses>
         <item count="2">
            Hohes Stresspotential
         </item>
         .....
      </weaknesses>
      <opportunities>....</opportunities>
      <threads>....</threads>
   </swot>
   <swot title="Arbeiten an ..." id="s-work">
      ....
   </swot>
</swots>
```

Dateiauszug 1 *"woarbeiten.xml"*

Auch wenn die Auslassungspunkte mit Inhalt gefüllt sind und wir die vollständige Datei vor Augen hätten, eignet sich diese Instanz nicht als Urlaubslektüre. Erst die Überführung von XML-Rohdaten in Endbenutzerformate erzeugt das, was wir Menschen als Lesbarkeit empfinden. Der Einsatz einer Style-Language, mit der das Anschaulichkeitsdefizit ausgeglichen werden kann, liegt auf der Hand. Obwohl es solche Formatsprachen bereits gab, beispielsweise (DSSSL oder Omnimark) wurde vom WWW-Konsortium eine eigene Arbeitsgruppe zur Entwicklung einer eXtensible Style Language (XSL) ins Leben gerufen. Bei der Entwicklung von XSL stellte sich heraus, dass bei der Umwandlung von XML-Daten in ein Zielformat drei Aufgabengebiete bearbeitet werden müssen.

XPATH

Da XML-Dokumente hierarchisch strukturiert sind, ist es wünschenswert die einzelnen Elemente, aus denen diese Struktur aufgebaut ist, wahlfrei ansprechen zu können. Eine typische Fragestellung ist es beispielsweise den zweiten Eintrag in der Liste der Schwächen einer Anstellung in der Privatwirtschaft auzuwählen. Diese Aufgabe löst die Spezifikation von Xpath durch eine Beschreibung der Elemente und Attribute, die den Pfadangaben im Dateisystem ähnelt. Beispielsweise so:

```
/swots/swot[1]/weaknesses/item[2]
```

Die Xpath-Notation wird nicht nur im Umfeld von XSL, sondern auch noch im Rahmen von Xlink/Xpointer verwendet. Ebenso wird Xpath als Abfragesprache in XML-Datenbankprodukten verwendet.

XSLT

Eine zentrale Aufgabenstelleung bei der Erzeugung eines Endbenutzerdokuments ist die Überführung der Struktur des Ausgangs- oder Quelldokuments in Syntax und Semantik des Zielformats. Diese Überführung ist Aufgabe der eXtensible Style Language Transformation (XSLT).

Was diese Transformation jeweils produzieren soll, wird in einem XSLT-Stylesheet beschrieben. Ein XSLT-Stylesheet ist eine XML-Datei, die XSLT-Schlüsselwörter und Anweisungen enthält. Diese Anweisungen werden von einem XSLT-Prozessor interpretiert und ausgeführt. Im Unterschied zu den Cascading Stylesheets bietet eine XSLT-Transformation jedoch mehr als eine Anreicherung des Quelldokuments mit Formatvorlagen. XSLT nutzt Xpath, um im Verlauf der Umwandlung des Quelldokuments auf die Elemente und Teilbäume lesend zuzugreifen. XSLT manipuliert niemals eine Quelldatei, sondern liest diese lediglich und gibt das Umwandlungsergebnis auf eine Zieldatei aus.

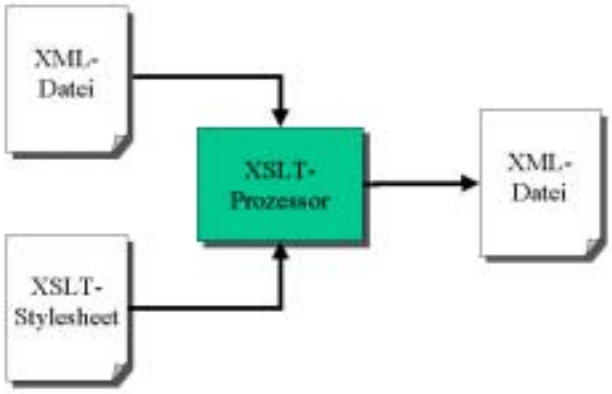

Abb. 1 *Komponenten einer Transformation*

XSL Formatting Objects

Das dritte Aufgabengebiet betrifft das Zielformat selbst. Dieses Format soll einerseits eine exakte Positionierung und Paginierung sowie weit reichende Gestaltungsmöglichkeiten für Schriften etc. bieten. Andererseits soll es aber dennoch ein möglichst neutrales Format sein. Im Unterschied beispielsweise zu PostScript soll es auch nicht auf das Medium Druck ausgerichtet sein. Es soll neutral sein in dem Sinne, dass es lediglich beschreibt, wo welches Element in welcher Farbe, Schriftart etc. erscheinen soll. Diese Beschreibung wird aber nicht auf ein reales Medium produziert. Dies soll Aufgabe von nachgeordneten Formatierern sein, die dieses neutrale Format dann schlussendlich in PDF, PostScript, RTF oder beispielsweise in Datenströme für Belichter umwandelt.

Die Elemente dieses Formats werden Formatting Objects genannt. Die Bezeichnungen XSL, XSL:FO oder einfach nur FO sind synonyme Bezeichnung dieses neutralen Formats. Die Spezifikation dieses Formats ist noch immer keine Empfehlung des W3C. Das Working-Draft-Dokument mit einem Umfang von ca. 400 Seiten hat eine bewegte Versionsgeschichte hinter sich. Für Implementierungen der Formatting Objects ist das ein großer Hemmschuh. Es zeichnet sich m.E. nicht ab, dass dieses Format sich rasch durchsetzen wird.

Zusammenfassung

Setzen wir die Teile zusammen, ergibt sich folgendes Bild: XSLT benutzt Xpath um wahlfrei und wiederholt auf Elemente und Teilbäume der Eingabedatei zuzugreifen. Durch Konstrukte wie Schleifen und Verzweigungen ist XSLT in der Lage, die gelesenen Daten vor der Ausgabe in das Zieldokument zu reorganisieren und zu manipulieren. Ausgegeben wird ein XML-basiertes Datenformat, das Gestaltungsangaben in Form der Formatting Objects enthält. Dieses Dokument wird von Formatierern in ein mediengebundenes Format überführt, um gelesen, gedruckt oder abgespielt zu werden.

Dies ist in groben Zügen die Idee von XSL als Formatierungssprache für XML. In der Praxis fehlt heute jedoch das letzte Glied im Produktionsablauf, da XSL:FO noch keine Empfehlung ist und Formatierer fehlen.

Um so mehr stellt sich die Frage: Was können wir mit XSLT heute machen? Zur Beantwortung dieser Frage werden nachfolgend einige XSLT-Transformationen auf das eingangs vorgestellte Dokument angewendet:

- Erzeugung lesbarer Texte in Form von HTML-Dateien, die durch Links miteinander verbunden sind
- Erzeugung eines grafischen Elements als Visualisierung der Stärken und Schwächen eines Entscheidungspfades
- Erzeugen einer Kombination von Text und Grafik in HTML-Dateien
- Erzeugen von interaktiven Grafiken
- Erzeugen eines druckbaren PDF-Dokuments
- Erzeugen von Text

Transformationen mit XSLT

Transformationsziel HTML

Die bekannteste XML-Anwendung ist HTML. Die Darstellung der Einsatzgebiete von XSLT beginnt deshalb auch mit der Umwandlung unserer XML-Quelldaten in HTML-Dateien.

XSLT ermöglicht es auf sehr flexible und generische Art zu beschreiben, was mit den Auszeichnungselementen des Quelldokuments geschehen soll. Dies geschieht durch Schablonen (`templates`), die bei Auftreten von definierten Mustern im Quelldokument aktiviert werden. Die nachfolgende Schablone wird aktiviert, wenn beim Verarbeiten des Quelldokuments eines der Elemente `<strengths>`, `<weaknesses>` etc. angetroffen wird.

```
<xsl:template match="strengths | weaknesses | threads | opportunities">
  <tr>
    <td valign="top" class="swot-title">
      <b>
        <xsl:value-of select="name()"/>
      </b>
    </td>
    <td class="swot-item">
      <ul>
        <xsl:apply-templates select="item"/>
      </ul>
    </td>
  </tr>
</xsl:template>
```

Dateiauszug 2 *"swot2html.xslt"*

Im Schablonenrumpf werden die gewünschten HTML-Tags notiert und durch den XSLT-Prozessor ausgegeben. In der dargestellten Schablone ist das eine Tabellenzeile, eine Zelle mit dem Namen des Tags als fett gedrucktem Inhalt und eine weitere Zelle, deren Inhalt innerhalb einer HTML-Liste erscheinen soll. Die Listenelemente sollen jedoch durch eine andere Schablone geliefert werden. Diese Delegation der Detailverarbeitung geschieht durch die Anweisung:

```
<xsl:apply-templates select="item"/>
```

Für die Erzeugung der Listenelemente und die Füllung mit Inhalt ist folgende Schablone zuständig:

```
<xsl:template match="item">
  <li>
    <xsl:value-of select="."/>
  </li>
</xsl:template>
```

Dateiauszug 3 *"swot2html.xslt"*

Ein Mustare kann auch die Hierarchieposition eines Elements beschreiben. Das Muster `"strengths/item"` lässt sich von `"weaknesses/item"` unterscheiden und bei Bedarf unterschiedlich ausgeben.

Die Reihenfolge, in der die Schablonen im Stylesheet notiert werden, spielt keine Rolle. Gibt es für Auszeichnungen in der Quelldatei keine expliziten Schablonen im XSLT-Stylesheet, tritt ein Automatismus in Kraft, der lediglich die Inhalte der XML-Elemente und der Attribute in das Ergebnisdokument kopiert. Dieser Mechanismus von Musterverarbeitung und Delegation kann dokumentgetrieben rekursiv durch das gesamte Eingabedokument führen oder auch durch XSLT-Anweisungen gesteuert iterativ oder wahlfrei erfolgen.

Durch die Mischung dieser Möglichkeiten in einem Stylesheet entsteht die Flexibilität von XSLT. Im Unterschied zu einem Anwendungsprogramm, welches das Eingabedokument direkt durch eincodierte Logik verarbeitet, ist ein XSLT-Prozessor ein generisches Transformationsprogramm, welches sich eben nicht direkt an den Tags des Eingangsdokuments orientiert, sondern an den Mustern, die im Stylesheet zu diesem Dokument beschrieben sind. Ändert sich die Struktur der Eingabedatei, kann dem oft durch »kleinräumige« Änderungen am Stylesheet Rechnung getragen werden. Käme in der vorgestellten Quelldatei neben den Stärken und Schwächen noch eine Rubrik »Sonstiges« dazu, würde dies lediglich die Einfügung dieses Begriffs in das entsprechende Template im XSLT-Stylesheet nach sich ziehen.

Ein XSLT-Prozessor erhält als Eingabe ein XSLT-Stylesheet und ein XML-Quelldokument. Als Ausgabe wird eine einzelne Datei erzeugt. Da dies in vielen Situationen unbefriedigend ist, nutzen die meistverbreiteten XSLT-Prozessoren den definierten Erweiterungsmechanismus von XSLT, um die Ausgabe mehrerer Dateien aus einem Transformationslauf zu ermöglichen.

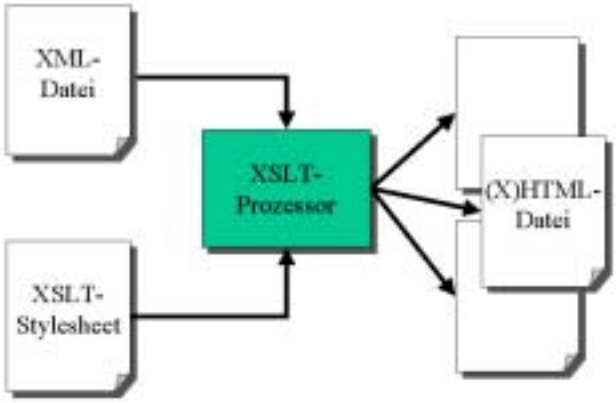

Abb. 2 *Produktion mehrerer Ausgabedateien*

Mit dem Stylesheet `swot2html.xslt` produzieren wir aus einer Eingabedatei mehrere HTML-Ergebnisdateien, die sich durch Links gegenseitig aufrufen lassen.

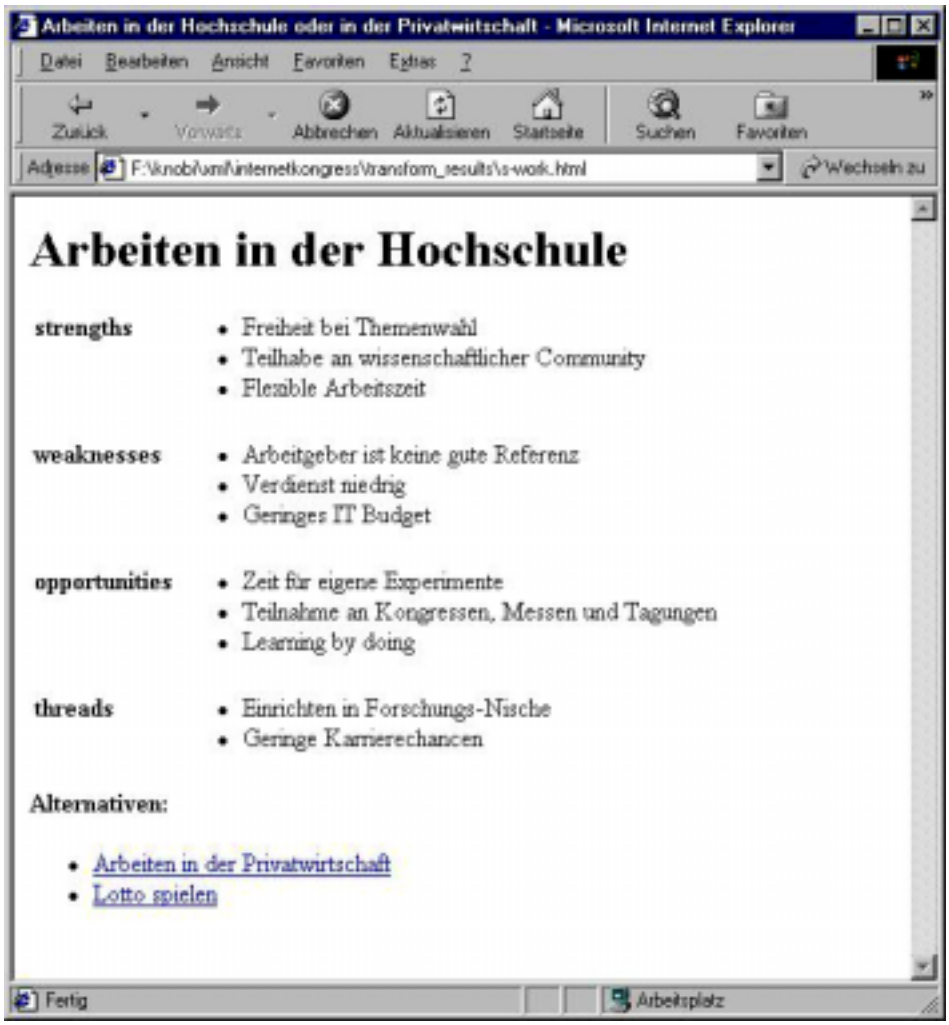

Abb. 3 *HTML-Repräsentation der SWOT-Liste*

Die Ausgabe auf mehrere Dateien wird in der nächsten Version von XSLT (Version 1.1) als Bestandteil der Spezifikation übernommen. Damit entfallen die herstellerabhängigen Erweiterungen.

Für einen sinnvollen Einsatz von XSLT ist diese Funktion bedeutsam. Erst bei einer automatisierten Produktion einer großen Zahl von HTML-Seiten lohnt sich der Aufbau einer Transformationsumgebung. Mit Hilfe von XSLT-Transformationen kann eine gesamte Site, also ein zusammenhängendes Web-Angebot, aus einer zentralen Datenbasis erstellt werden. Hierbei kann auf folgende Weise vorgegangen werden:

- In einem Batch-Durchlauf werden aus den Quelldaten die HTML-Seiten einer Site erzeugt und als Dateien in Verzeichnissen des Web-Servers abgelegt. Tendenziell wird man dieses Vorgehen dazu nutzen, um statische HTML-Seiten zu erzeugen. Diese Vorgehensweise ist unkompliziert, da die Produktion der Seiten auf beliebigen Maschinen mit Standardwerkzeugen erfolgen kann. Mit zusätzlichem Aufwand ist es jedoch auch möglich HTML-Seiten mit dynamischem Inhalt zu erzeugen. Ein Beispiel hierfür wird nachfolgend beim Thema Scalable Vector Graphics geboten.

- Die zweite Möglichkeit besteht darin die Seiten dynamisch zu erzeugen. Hierbei wird serverseitig pro HTTP-Anfrage eine Transformation durchgeführt. Das Transformationsergebnis wird nicht als Datei abgelegt, sondern als HTTP-Datenstrom an den Browser zurückgegeben. Dies ist prinzipiell durch einen CGI-Prozess realisierbar. Häufig werden jedoch Servlet-basierte Techniken genutzt, da die XSLT-Prozessoren zuerst als Java-Klassen verfügbar waren. Der bekannteste Vertreter dieser Variante ist das Web-Publishing Framework Apache Cocoon (xml.apache.org/cocoon).
- Der ursprünglichen Idee von XSL am nächsten käme eine dritte Variante, bei der die Transformation im Browser durchgeführt wird. Dies ist bislang jedoch lediglich im Microsoft Internet Explorer 5.x und der Verwendung der MSXSL3.DLL standardkonform möglich. Weshalb dies nur bei reinen Intranet-Anwendungen in Erwägung gezogen werden kann, wo es unter Umständen möglich ist, den Benutzern einen bestimmten Browser vorzuschreiben.

Die Stylesheets für alle drei genannten Möglichkteiten sind nahezu identisch. Lediglich bei der dynamischen Generierung müssen die erzeugten HTML-Links entsprechend angepasst werden, da sie in diesem Fall einen URL mit Parameter erzeugen müssen.

Der Einsatz von XSLT bietet sich an, wenn eine unformatierte, also möglichst medienneutrale XML-Datenbasis existiert, die häufigen Änderungen unterworfen ist. Da selbst Änderungen oder Erweiterungen in der Struktur der Quelldaten in einem XSLT-Publikationsprozess rasch nachgezogen werden können, liegt die Idee den Transformationsmechanismus als Visualisierungsschicht für Content-Management-Systeme einzusetzen nahe. Es soll jedoch nicht verschwiegen werden, dass folgende Nebeneffekte den Einsatz von XSLT erschweren:

- XSLT-Transformationen brauchen schnelle Prozessoren, sehr viel Speicher und haben dennoch relativ lange Laufzeiten, was im Online-Bereich nicht akzeptiert wird.
- Die Technik ist noch relativ jung und es existiert noch wenig Know-how, auf das Firmen zurückgreifen können.
- Der Einsatz von XSLT erzwingt den regelhaften hierarchischen Aufbau eines Web-Angebots, da sonst die Navigation nicht mitgeneriert werden kann.
- Pixelgenaue Positionierung von HTML-Elementen, wie sie in handgemachten HTML-Seiten häufig zu finden ist, ist nicht möglich.
- Eine Modellierung der Quelldokumente ist unverzichtbar. Dokumente müssen immer auf definierten Strukturen basieren, auch wenn diese Dokumente selbst variantenreich und flexibel sein können.

Gibt es noch weitere Potentiale von XSLT, die diese Nachteile kompensieren? Wenden wir uns einem weiteren XML-basierten Format zu, um zu sehen, ob sich noch mehr mit XSLT anstellen lässt.

Transformationsziel Scalable Vector Graphics

Scalable Vector Graphics (SVG) ist ein Vector-Grafikformat, das als XML-Anwendung definiert ist. Für SVG gilt damit auch, was für HTML gilt: Prinzipiell genügt als Arbeitsschritt eine Transformation, um das Endformat zu produzieren. Für SVG existieren bereits Plug-Ins für die wichtigsten Browser und darüber hinaus auch einige Werkzeuge zur Konvertierung und Bearbeitung dieses Formats. Obwohl auch SVG noch keine Recommendation des WWW Konsortiums ist, wächst die Akzeptanz dieses Formats rasch.

Als Beispiel für eine Transformation benutzen wir wieder die vorgestellte Ausgangsdatei und erzeugen daraus mit dem Stylesheet `swot2svg.xslt` eine Reihe von SVG-Dateien, in denen die Menge und die Gewichtung der Pro- und Contra-Argumente zu einem tachoartigen Anzeigeelement verarbeitet wird. Dabei werden die Stärken und Chancen als positive, die Schwächen und Risiken als negative Punktwerte zusammenadiert. Die Differenz wird mit fünf multipliziert und damit die Rotation des Zeigers parametrisiert.

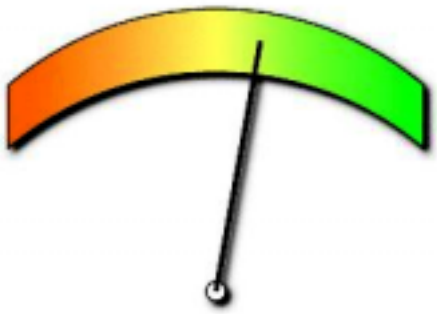

Abb. 4 *Eine errechnete Visualisierung der SWOT-Aufstellung*

War die reine HTML-Variante sehr textlastig, fehlt hier umgekehrt die textliche Erklärung dessen, was hier visualisiert wird. Es ist jedoch nur ein geringer Aufwand unser Stylesheet für die HMTL-Ausgabe so umzuschreiben, dass die SVG-Grafik in die jeweilige HTML-Datei eingebettet wird. Das Transformationsergebnis `s-work_svg.html` produziert durch das zugehörige Stylesheet `swot2html_svg.xslt` finden Sie bei den Beispielen.

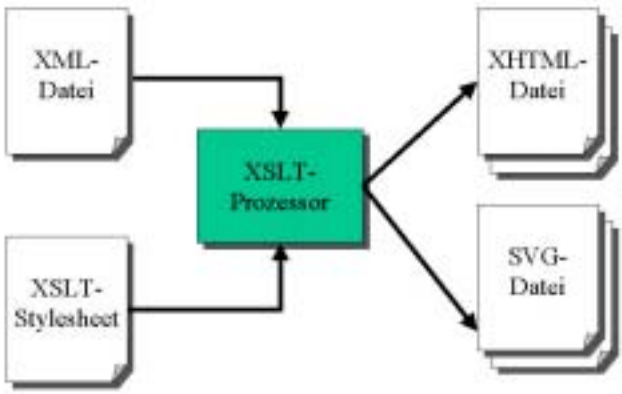

Abb. 5 *Konsistente Produktion von Text und Grafik*

Eine weitere Variante unseres Stylesheets (`swot2html_svg_js.xslt`) erzeugt die HTML-Dateien (`s-work_svg_js.html`), so dass sie lediglich als Navigationsrahmen und Container für interaktive SVG-Grafiken dienen. Die textliche Auflistung der Stärken, Schwächen, Chancen und Risiken beschränkt sich in diesen Grafiken auf lediglich ein sichtbares Thema, also beispielsweise die Stärken. Die anderen Aufzählungen sind unsichtbar. Durch Mausklick auf Schaltflächen, die in die Grafik eingebaut sind, lassen sich wahlweise die anderen Aufzählungen sichtbar schalten.

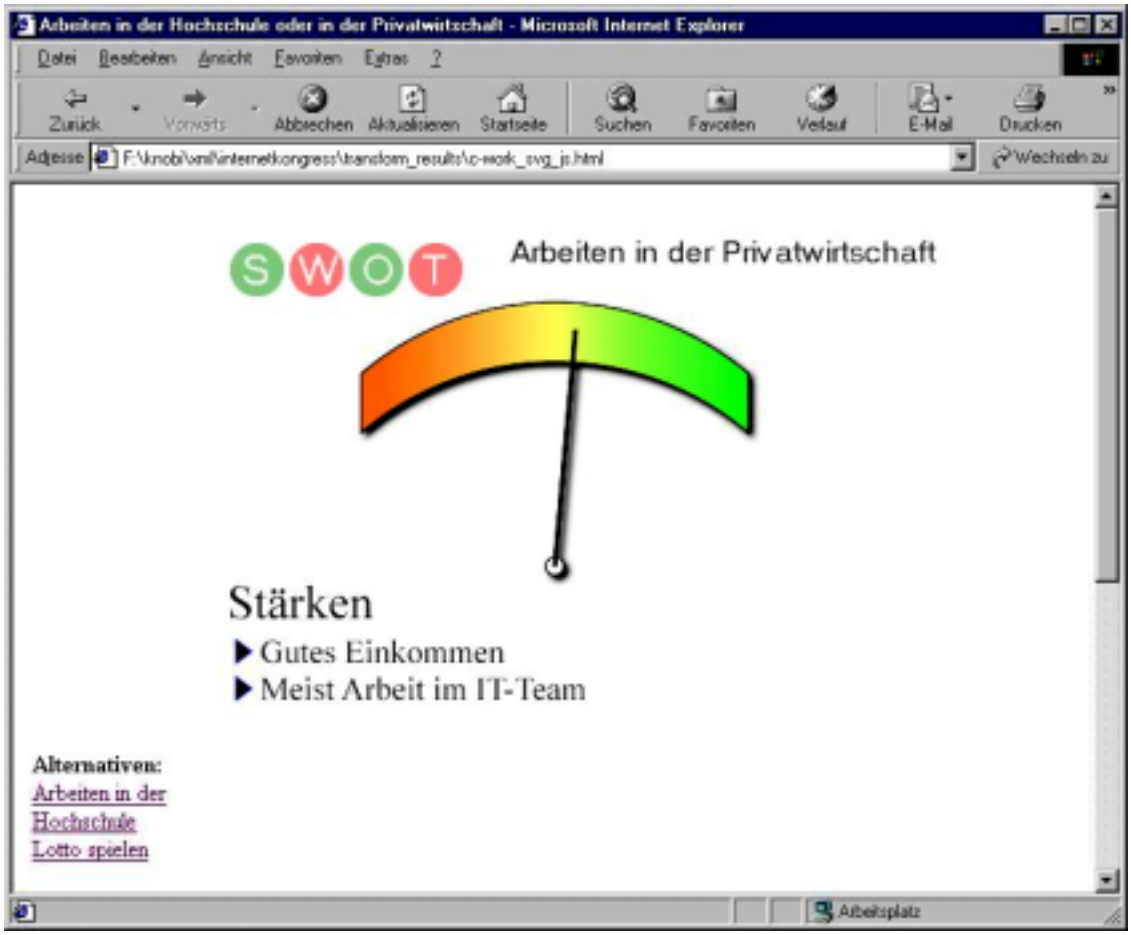

Abb. 6 *HTML-Datei mit interaktiver SVG-Grafik*

Solche interaktive SVG-Grafiken verarbeiten Benutzereingaben mit Hilfe von JavaScript. Auch dies lässt sich durch die XSLT-Stylesheets mit ausgeben. Aus einer Quelle wird somit JavaScript-Programmcode eingebettet in SVG, dies wiederum eingebettet in HTML generiert.

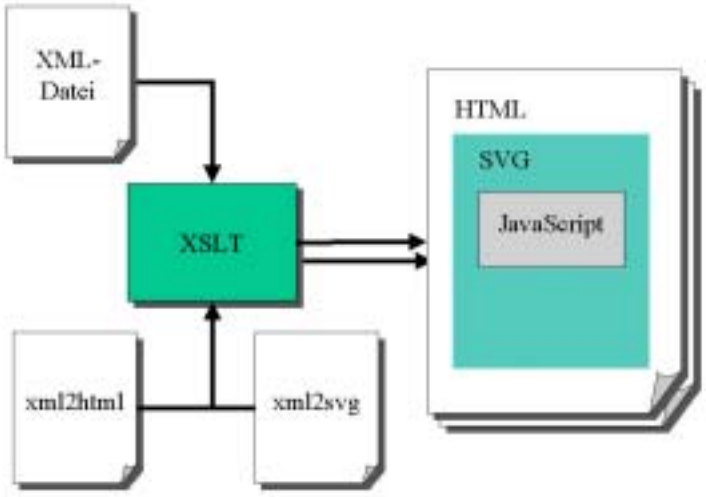

Abb. 7 *Produktion unterschiedlicher Codes aus mehreren Transformationen*

Sehr viele Navigationselemente oder Beschriftungen auf Web-Seiten sind heutzutage Grafiken. Mehrheitlich werden diese manuell erzeugt. Wird beispielsweise eine Web-Site um ein thematisches Angebot erweitert, muss ohne ein Web-CMS zu verwenden die Navigation angepasst werden. Sind die Navigationselemente durch grafisch aufgewertete Texte realisiert, müssen die neuen Grafiken für die neuen Navigationspunkte erzeugt werden. Sollte sich SVG durchsetzen, liegt hier ein hohes Potential für die Einsparung manueller Tätigkeiten. Wie dargestellt, kann XSLT bei einer Automation der Erzeugung von Text und Grafik eine Hilfe sein.

Transformationsziel XSLT

Ein weiteres wichtiges XML-basiertes Zielformat muss noch erwähnt werden. Dabei handelt es sich um XSLT selbst. XSLT-Stylesheets sind XML-Dokumente, die sowohl als Eingabedatei als auch als Ausgabedatei in Frage kommen.

Die Produktion von Stylesheets durch XSLT wird in dynamischen Publikationsumgebungen eingesetzt, um abhängig von Request-Parametern verschiedene Verarbeitungspfade zu erzeugen. Die Verarbeitung eines Requests geht dann in mehreren Durchgängen vor sich. Die erste Transformation erzeugt abhängig von Eingabeparametern und Eingabedateien ein Stylesheet, das in einem zweiten oder n-ten Durchgang verwendet wird, um das Ausgabeformat zu erzeugen.

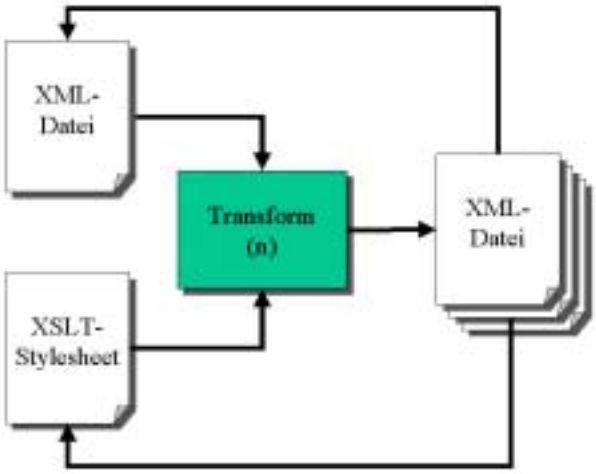

Abb. 8 *Wiederholte Transformationen*

Gleiches gilt auch für die XML-Eingabedatei, die in mehreren Durchgängen verarbeitet werden kann. Jeder Transformationslauf reichert dabei die Ausgangsdatei mit Informationen an, die im nächsten Durchgang benötigt werden. Die Ergebnisdaten dienen dann als Eingabedaten für die Folgetransformation. Auch diese Art des Dokument-Pipelining wird in Publishing-Umgebungen verwendet.

Diese Beispiele sollen als Veranschaulichung für die Erzeugung von XML-basierten Formaten genügen. Die gleiche Vorgehensweise lässt sich auch nutzen, um WML für mobile Geräte zu erzeugen oder andere XML-Dialekte, wie MathML, CML und was noch in Zukunft an Markup-Sprachen auf XML-Basis kommen mag.

Transformationsziel PDF

Für die Ausgabe von Nicht-XML-Daten ist XSLT keine Ideallösung, da planmäßig XML-Dokumente als Ausgabe erzeugt werden. Als Schnittstelle zu nicht XML-basierten Formaten ist für den Publikationsbereich XSL:Formatting Objects vorgesehen. Eine XSLT-Transformation erzeugt also zunächst das FO-Dokument. Dieses wird dann von nachgeschalteten Formatierern in die jeweiligen proprietären Formate umgesetzt wird. Mit einer Transformation in XSL:FO wird also wieder eine weitere XML-Schicht angesteuert. Damit unterscheidet sich eine Transformation in Formatting Objects nicht grundsätzlich von den vorhergehenden Beispielen. Bemerkenswert ist jedoch die nahtlose Integration von grafischen Elementen, wie sie der Working Draft von XSL Formatting Objects vorsieht. Eingebettet in das FO-Dokument kann SVG-Code stehen. Im Entwurf der Formatting Objects ist dafür das Element `<fo:instream-foreign-object>` vorgesehen. Es kapselt die Sprachelemente des Grafik-Codes, wie der nachfolgende Ausschnitt zeigt.

```
<fo:block>
<fo:instream-foreign-object>
   <svg:svg xmlns:svg="http://www.w3.org/2000/svg"
      width="800px" height="250px">
      <svg:desc>Tacho mit Gradient</svg:desc>
         <xsl:call-template name="get-defs"/>
         <svg:g id="tacho-text-group"
                transform="translate(0 30)">
            <xsl:call-template name="do-tacho"/>
         </svg:g>
   </svg:svg>
</fo:instream-foreign-object>
</fo:block>
```

Dateiauszug 4 *"swot2pdf_svg.xslt"*

Die Abgrenzung der Vokabularien gegeneinander findet durch die Verwendung von XML-Namespaces statt. Da wir uns innerhalb eines XSLT-Stylesheets befinden, sind in obigem Ausschnitt drei Namensräume gemischt: der Namensraum von XSLT mit dem Kürzel `xsl:`, der Namensraum der Formatting Objects mit dem Kürzel `fo:` und der von SVG mit Kürzel `svg:`. Der prinzipielle Produktionsablauf sieht wie folgt aus:

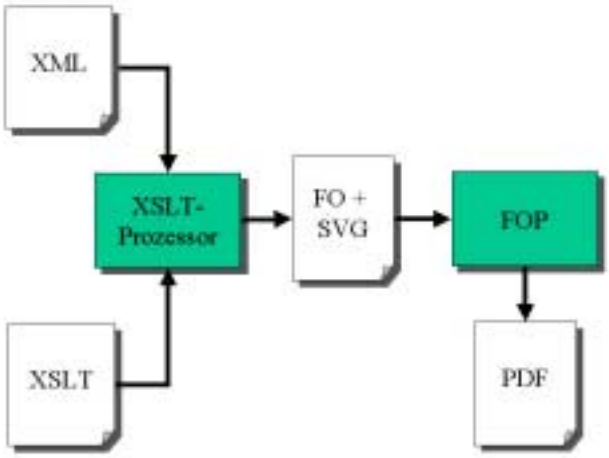

Abb. 9 *Produktion von PDF aus Formatting Objects als Zwischenformat*

Der Vorteil an den Formatting Objects ist, dass dieses Format keine Kenntnis des endgültigen Zielformats verlangt. Dieses Wissen ist im Formatierer gekapselt. Wie bereits erwähnt, gibt es noch keine wirkliche Auswahl von Formatierern, die den breiten Einsatz der Formatting Objects beschleunigen würden. Lediglich für das sicherlich wichtige PDF-Format sind Formatierer vom XML-Apache-Projekt (http://xml.apache.org/fop) und von kommerzieller Seite (http://www.renderx.com) verfügbar. Beide Werkzeuge sind noch nicht im Produktstadium, d.h., sie implementieren den Working Draft noch nicht vollständig. Da die SVG-Implementierung beim Apache-Formatierer FOP weiter fortgeschritten ist, wurde dieses Produkt für unsere Beispieltransformation verwendet. Das Ergebnis, eine PDF-Datei mit integrierten Grafiken, finden Sie bei den Beispielen unter dem Namen `"woarbeiten.pdf"`.

Ausgabe von Text

Neben der schönen neuen XML-Welt existiert aber immer noch die Welt der ASCII-basierten Codes, die beim Umgang mit Programmiersprachen und diversen DV-Werkzeugen benutzt werden müssen. Mit der Aussage, dass XSLT in der Lage ist auch XML-Fragmente zu generieren hat das WWW-Konsortium die Tür zu dieser traditionellen Welt offen gehalten. Ein XML-Fragment ist alles, was ein XML-Dokument wäre, wenn es mit einem korrekten öffnenden und einem schließenden XML-Tag umhüllt wäre. Und das kann vieles sein. Die Ausgabe von Text durch ein XSLT-Stylesheet wird durch Anweisung `<xsl:output method="text" />` möglich.

Die Ausgabe unseres Beispieldokuments in ASCII-Text ist nicht besonders reizvoll. Deshalb möchte ich für dieses letzte Beispiel einer XSLT-Transformation eine Aufgabe vorstellen, die nicht in den Bereich der Aufbereitung von Darstellungen fällt, sondern im Gegenteil mit der Sammlung von Informationen zu tun hat. Die Sammlung von Daten durch HTML-Formulare auf Web-Seiten, die von den Benutzern selbst ausgefüllt werden, ist in vielen Bereichen attraktiv, weil die Papierform und damit die Mehrfachbearbeitung von Quelldaten entfällt. Solcherart erhobene Daten landen sinnvollerweise in Datenbanken, die mit dem Web-Auftritt gekoppelt sind. Das Angebot von solchen HTML-Seiten setzt also die Definition von Datenbanktabellen voraus. Der Eintrag der Daten in diese Tabellen wird von den HTML-Seiten angestoßen. Dabei müssen die Namen und Bezeichner der Datenbankdefinition bekannt sein und bei Übertragung der Daten an die Datenbank verwendet werden. Wer diese Arbeiten je von Hand erledigt und die entsprechenden Skripte getippt hat ,weiß, dass schon bei einfachen Datenstrukturen die Übersicht verloren geht und sich beim Codieren jede Menge Schreibfehler einschleichen, Plausibilisierungen von Feldinhalten oder ganze Felder vergessen werden. Es liegt also nahe ein unabhängiges Format zu schaffen, das die Anforderungen an die Tabelle, das Eingabeformular und die Speicherprozedur beschreibt. Dieses Meta-Format formuliert man am einfachsten in XML. Damit entsteht die Möglichkeit, flexible XSLT-Skripte zu erstellen, die den Rest generieren.

```
<info name="tb_kurs_uebersicht" label="Kurse">
   <info-item name="ident" label="ident" type="int4"
          size="4" edible="no" pkey="yes"
          constraint=" default nextval( 'kmmt_seq' )"
          input="IMPLIED"/>
   <info-item name="art" label="Art" type="varchar"
          size="120" edible="yes" input="REQUIRED"/>
   <info-item name="name" label="Kursname" type="varchar"
          size="254" constraint="not null" edible="yes"
          input="REQUIRED"/>
...
</info>
```

Dateiauszug 5 *"kurs_def.xml"*

Dieses XML-Dokument orientiert sich sehr stark an den Informationen und Bezeichnungen, die von Seiten der verwendeten PostgreSQL-Datenbank benötigt werden. Über zusätzliche Attribute wie »edible« wird jedoch noch signalisiert, dass diese Felder in der Edit-Maske sichtbar sein sollen oder dass ein Eintrag in diese Felder verpflichtend ist und dies vor dem Abspeichern geprüft werden muss. Der Produktionsablauf für diese Anwendung sieht folgendermaßen aus:

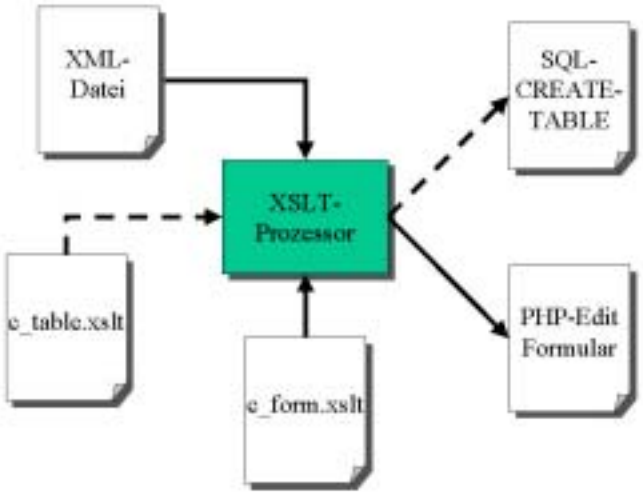

Abb. 10 *Erzeugung textbasierter Formate*

Das Stylesheet `c_form.xsl` generiert einen einfachen create-table-Job für eine PostgreSQL-Datenbank. Passend dazu erzeugt das Stylesheet `c_form.xsl` in einer weiteren Transformation ein PHP-Skript, das eine Dateneingabemaske generiert. Dieses PHP-Skript ruft sich beim Absenden selbst auf und überprüft dabei den Zustand der Eingabefelder. Sind alle Pflichtfelder gefüllt, werden die Daten in der Tabelle abgespeichert; sind sie nicht gefüllt, wird die Eingabemaske mit entsprechenden Hinweisen erneut angezeigt. Die bisher eingegebenen Daten bleiben dabei erhalten. Die vollständigen Beispiele (`kurs_def.xml`, `c_form.xsl` und `c_table.xsl`) finden Sie auf dem Kongresserver oder bei www.xml-web.de.

Resümee

Es ist sicher deutlich geworden, dass bereits mit relativ schlanken XSLT-Skripten mehrere Repräsentationen von Daten aus einer einzigen Quelle erzeugt werden können. Für den Bereich Web-Publishing bietet sich mit dieser Technik die Chance eine bisher nicht gekannte Konsistenz von HTML-Seiten und Multimedia-Formaten zu erreichen. Wenn aus einer Quelle alles Notwendige generiert wird, genügt nach einer Änderung der Ausgangsdaten eine erneuter Durchlauf der Produktion. Der Preis, der dafür bezahlt werden muss, ist der Verzicht auf individuell gestaltete Web-Seiten. Jede Seite aus einer XSLT-Produktion gleicht mit geringen Varianzen den anderen. Diese Normierung und Standardisierung wirkt auch auf die Inhaltsproduktion zurück. Ob eine Überschrift einzeilig oder dreizeilig ist, kann während der Transformation nicht berücksichtigt werden. Die Tatsache, dass ein Automat mit Hilfe von XSLT eine Seite daraus produziert, muss demnach bei der Inhaltserstellung mit berücksichtigt werden. Daraus ergibt sich eine Beschränkung für einen sinnvollen Einsatz von XSLT im Web-Bereich auf große und gut strukturierte Angebote. Die Revolution der persönlichen Homepages findet jedenfalls nicht durch XML-Technologie statt.

Unsere Beispiele waren einfach und passen darin zum Stand der Entwicklung der XML-Welt, die sich einmal auf die Fahnen geschrieben hat einfach zu sein. Mit der Verbreitung der XML-Notation, mit jeder neuen XML-Anwendung wächst jedoch auch die Anzahl der Vokabularien, die unter Umständen in Syntax und Semantik eines anderen Vokabulars übertragen werden müssen. Hier kann XSLT seine Stärke ausspielen, die in der Beschränkung auf XML-Formate liegt. XSLT erweist sich als flexibles Werkzeug für Transformations- und Mapping-Prozesse. Es wird in desem Bereich rascher als im Publishing eine wichtige Rolle spielen.

Ob sich die Nutzung von XSLT durchsetzt, liegt natürlich auch an den AnwenderInnen. Bislang ist das Know-how zu XSLT noch wenig verbreitet. Vielleicht hilft ja dieser Kongress dies zu verändern.