Seminar der Arbeitsgruppe
Datenbanken und Informationssysteme
im Wintersemester 02/03

Thema:

XML und Datenbanken

XML – basiertes Publizieren und Visualisieren

Von Steffen Apfel
Mail: **s_apfel@informatik.uni-kl.de**

Inhaltsverzeichnis

1. Einführung in das Themengebiet

XML (Akronym für **eXtensible Markup Language**) ist eine neue, leistungsfähige und vielseitige Markup-Sprache, die seit 1996 entwickelt wurde, um strukturierte Dokumente und Daten zu beschreiben. Seit dem 10.Februar 1998 ist XML eine Recommendation des **W3C** (**World Wide Web Consortium**). [W3C02]
Eines der wichtigsten Grundprinzipe von XML ist die strukturorientierte Datenbeschreibung. Die Daten werden innerhalb eines XML-Dokuments von der Darstellung (Layout) getrennt. Die Informationen fürs Layout stehen zum Teil schon in den XML-Metadaten oder können nachträglich festgelegt werden.
Dadurch erfreut sich XML immer größer werdender Beliebtheit, da es hervorragend zum Datenaustausch im Web geeignet ist und dies für viele Firmen und Privatpersonen mittlerweile essentiell geworden ist.
Die Alternativen zu XML, SGML und HTML, sind leider nur beschränkt zur Datenbeschreibung und Datenaustausch nutzbar. XML ist eine von **SGML** (**Standard Generalized Markup Language**) abgeleitete Untersprache. SGML bietet zwar eine erweiterbare Struktur, ist aber viel zu komplex und umständlich für den Datenaustausch im Netz. **HTML** (**Hypertext Markup Language**) wurde ebenfalls von der Metasprache SGML abgeleitet und unterstützt nur vorgefertigte Konstrukte und ist somit zum Datenaustausch nicht geeignet, da es nicht erweiterbar ist. HTML dient eher der Beschreibung der Präsentation als der Struktur. [DES03]

XML bietet einen Kompromiss zwischen Einfachheit und Flexibilität und ist eine leicht verständliche Metasprache, die nach belieben erweitert werden kann. Es liegt also kein festgelegter Satz von Markup-Befehlen vor. Nach dem Datenaustausch im Netz kann man das plattformunabhängige XML in alle gängigen Ausgabe-Formate transformieren. Man muss also seine Daten nicht in mehreren Formaten speichern, da man aus XML sie konstruieren kann. Man erreicht eine Trennung der Daten und der Präsentation. Die Daten werden in XML und die Präsentationsinformationen in einem XML-Transformation-Dokument gespeichert. [HARE02]
XML-Dokumente müssen im Gegensatz zu weniger restriktiven HTML-Dokumenten immer **wohlgeformt** sein, d.h. dass sie sich exakt an die syntaktischen Regeln des Standards halten. Zusätzlich sollten XML-Dokumente **gültig** sein und den Regeln einer **DTD** oder **XML-Schema** folgen. **D**okumenten-**T**yp-**D**efinitionen und XML-Schema sind formale Spezifikationen aller in einem Dokumenttyp erlaubten Strukturen. Man spricht auch von einer *formalen Grammatik*, in der zulässige Elementtypen, Attribute und deren Verschachtelung definiert sind. DTD und XML-Schema stellen somit sicher, dass verarbeitende Anwendungen XML-Dokumente entsprechend der zu Grunde liegenden Grammatik interpretieren und verstehen können. [FLY02]

In diesem Seminar soll auf die Verwendung von XML beim Publizieren und Visualisieren von Dokumenten eingegangen werden. Zuerst werden im 2. Kapitel einige XML-Standards und Techniken vorgestellt.

XSL (**eXtensible Stylesheet Language**) (Kapitel 2.1) ist eine Sprache um die Struktur von XML-Dokumenten zu transformieren. Es wird dabei näher auf die beiden Teilsprachen **XSL/T** (**eXtensible Stylesheet Language Transformation**) und **XSL/FO** (**eXtensible Stylesheet Language Formatted Objects**)eingegangen.

Im Kapitel 2.2 wird **SVG** (**Scaleable Vector Graphic**) vorgestellt. SVG wird zur Beschreibung von Vektorgraphiken benutzt.

Anschließend wird im Kapitel 2.3 **Docbook** näher betrachtet. Docbook ist eine DTD für überwiegend technische Dokumentationen. DocBook wird verwendet um Bücher und Artikel einheitlich in einem gemeinsamen XML-Format zu repräsentieren.

Im 3. Kapitel wird die Idee des **Single-Source-Publishings** vorgestellt. Das Prinzip des Single-Source-Publishings besteht darin, dass die zugrunde liegenden Daten Format unabhängig in einer gemeinsamen Quelle verwaltet werden.

Im nächsten Kapitel wird **Cocoon** von Apache als ein web-basiertes Framework zum Single-Source-Publishing vorgestellt. Dieses Open Source Framework benutzt die vorgestellten Techniken um die Idee des Single-Source-Publishing „Store once, use everywhere“ zu verwirklichen.

Im 5. und 6. Kapitel folgen noch eine Zusammenfassung und ein Ausblick auf die weitere Entwicklung von XML und die zugrunde liegende Literaturressourcen.

2 Techniken und Standards

In diesem Kapitel werden diverse XML-Techniken und Standards vorgestellt mit deren Hilfe das Publizieren von Dokumenten vereinfacht wird. Mit sämtlichen vorgestellten XML-Techniken kann der Benutzer wohlgeformte XML-Dokumente mit eigenem Namensraum (Namespace) erstellen. Die einzelnen Techniken sind in sich nicht abgeschlossen und können erweitert und somit den Bedürfnissen des Anwenders angepasst werden. Das Ziel dieser Techniken ist es, dass man beliebige Daten und Dokumente plattformunabhängig speichern, übertragen und kombinieren kann. Es erfolgt jeweils eine Trennung der Daten und des Layouts, damit letzteres nach belieben verändert werden kann. Somit muss der Anwender seine Daten nicht in verschieden Formaten vorliegen haben, sondern kann das Format und Layout je nach Gebrauch „just in time“ erzeugen.

2.1 XSL

EXtensible Stylesheet Language (XSL) ist seit dem 15.Oktober 2001 in der Version 1.0 eine Recommendation des W3C. Während ein XML-Dokument eher datenorientiert ist, versucht man nun mit Hilfe von Stylesheets diese Daten für den Browser, Printmedien oder PDA´s grafisch aufzubereiten. das Stylesheet legt dabei Regeln fest, wie spezielle Elemente eines XML-Dokumentes in ein anderes XML-Dokument transformiert werden sollen. Abbildung 2.1.1 zeigt, dass man mit Hilfe der Extensible Stylesheet Language XML-Dokumente in ein anderes Ausgabeformat transformieren kann. [XSL02]

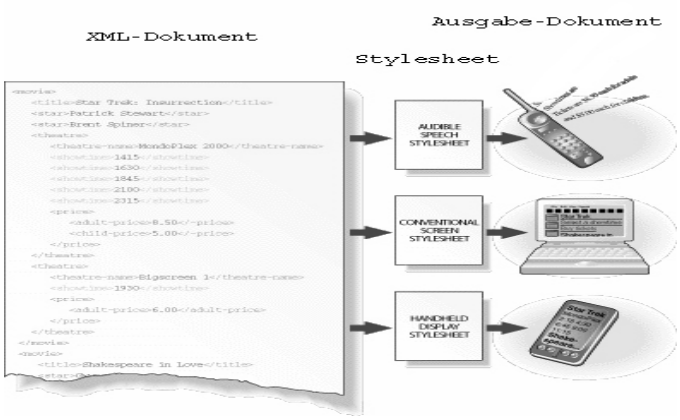

Abb. 2.1.1: Umwandeln von XML durch Stylesheets in das Ausgabeformat.

XSL stehen zur Verarbeitung drei Untersprachen zur Verfügung:

- Mit **XPath** (**XML Path Language**) werden bestimmte Teile des XML-Dokuments adressiert und angesprochen. Die „Pfadbeschreibungssprache" wird zum Matching verwendet. Beim Matching wird getestet, ob ein XML-Knoten auf ein Stylesheet-Muster passt und bei Übereinstimmung wird der Body des passenden Stylesheet-Musters abgearbeitet.

- **XSL/T** (**XML Style Language Transformation**) beschreibt, wie man die Baumdarstellung eines XML-Dokuments in eine andere XML-Baumdarstellung verwandelt. XSL/T (Kapitel 2.1) gibt die Regeln zur **Transformation** an.

- Mit **XSL/FO** (**XSL Formatted Objects**) werden die XML-Dokumente **formatiert**. Durch XSL/FO (Kapitel 2.2) kann man einem XML-Dokument neben dem Inhalt noch Layout-Informationen bei fügen.

Die Verarbeitung eines XML-Dokumentes durch ein Stylesheet wird in Abbildung 2.1.2 gezeigt.

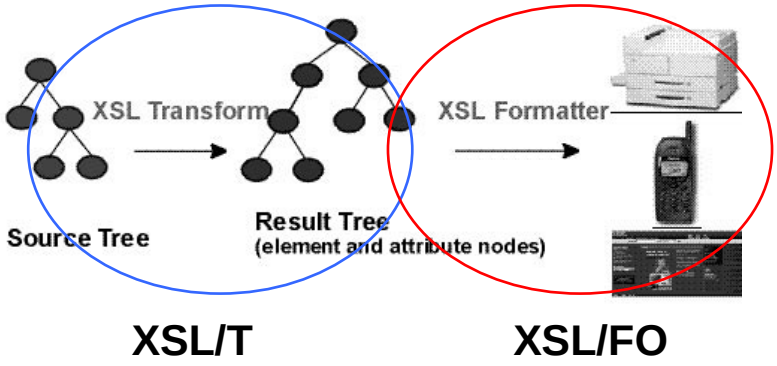

Abb. 2.1.2 zeigt die Transformation und Formatierung eines XML-Dokumentes. [XSL02]

Der **XSL Stylesheet Processor** erzeugt mit Hilfe eines Stylesheets ein XML-Dokument mit Layoutinformationen. Der Prozessor arbeitet dabei in zwei Teilschritten.
Zuerst findet eine **Transformation** des Source Tree (Ausgangsdokument in XML) in einen Result Tree (ebenfalls ein XML-Dokument) statt. Der Prozessor verwendet ein Stylesheet, das die Regeln zur Transformation beinhaltet. Der entstandene Result Tree wird auch „*Element und Attribut Baum*“ genannt. Die ursprünglichen XML-Elemente wurden in diesem Schritt in XSL/FO-Elemente transformiert und gegebenenfalls die Daten umstrukturiert. Man kann z.B. einzelne Daten von Börsenkursen in einer Tabelle anordnen zur besseren Ansicht später.
Danach wird der Result Tree mit einem **XSL Formatter** in das Ausgangsdokument durch Formationssemantik formatiert**.** Ein XSL Formatter kann dabei u.a. eine Rendering-Maschine in einem Browser sein. Die Layoutinformationen des Stylesheets werden nun verarbeitet um ein Formatted Objekt zu erzeugen. Dieses neu erzeugte XML-Dokument kann nun in einem Browser, der XSL/FO-Dokumente unterstützt, angezeigt werden. Außerdem kann man XSL/FO-Dokumente durch Reduktion der Daten und Layoutinformationen in einem Handy oder PDA anzeigen. Letztlich kann man XSL/FO in ein Druckformat wie z.B. PDF, RTF und PS umwandeln. [XSL03]

In den beiden folgenden Unterkapiteln wird nochmals genau auf die Funktionsweise von XSL/T und XSL/FO eingegangen.

2.1.1 XSL/T

XSL/T (**EXtensible Stylesheet Language Transformation**) ist in der Version 1.0 seit dem 16.11.1999 eine Recommendation des W3C. [XSLT02]
XSL/T ist ein wohlgeformtes XML-Dokument mit eigenem Namensraum um XML-Dokumente in andere XML-Dokumente zu transformieren. **XPath** wird von XSL/T benutzt um bestimmte Dokumententeile eines XML-Dokuments anzusprechen. (Auf die Funktionsweise von XPath wird an dieser Stelle nicht eingegangen. Der interessierte Leser findet weitere Information unter: http://www.w3.org/TR/xpath .)

Da XSL/T generisch wohlgeformte XML-Dokumente in andere XML-Dokumente transformiert, sind die entstehenden Dokumente immer wohlgeformt. Dies wird beim Aufbau eines Stylesheets weiter unten verdeutlicht. Möchte man also aus XML ein HTML-Dokument erstellen, entsteht automatisch immer ein *XHTML-Dokument* [1]. Im 3.Kapitel wird anschließend ein Stylesheet zur Transformation in ein Formatted Objekt vorgestellt. Mit Hilfe von XSL/T kann also die Struktur eines XML-Dokuments beliebig umgeordnet werden und so in verschiedenen Applikationen verwendet werden. Im Folgenden wird nun auf den Aufbau eines Stylesheets und die Arbeitsweise näher eingegangen um aus einem XML-Dokument ein XHTML-Dokument zu erzeugen. [SUN02]

Aufbau eines Stylesheets :

Im Folgenden soll der allgemeine Aufbau eines XSL-Stylesheets vorgestellt werden. Nach der XML-Deklaration (Zeile 1) folgt das so genannte Dokumentenelement mit der XSLT-Namensraumdeklaration (Zeile 2). Dabei ist auch die Angabe der XSLT-Version erforderlich (Zeile2). Es folgen Top-Level-Elemente mit den eigentlichen Template-Regeln (template rules) (Zeile 4). Die Template-Regeln wiederum sind durch XSLT-Instruktionen und Ausgabebaumvorlage (result tree template) gekennzeichnet. Ein einfaches Beispiel:

```
1: <?xml version="1.0" ?>
2: <xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
3:
4:   <xsl:template match="fussball">
5:         <html>
6:         <xsl:apply-templates />
7:         </html>
8:   </xsl:template>
9: </xsl:stylesheet>
```

Ein XSLT-Stylesheet besteht aus einer Serie von Templates. Ein Template besteht aus einem Muster (Pattern), das XML-Elemente im XML-Quelldokument selektiert und einem Körper, der spezifiziert, was für eine Ausgabe generiert werden soll, wenn dieses Muster im Quelldokument angetroffen wird. Dabei parst der XSL-Prozessor die gesamte Struktur eines XML-Dokuments, vergleicht sie mit den Mustern der Template-Regeln, ersetzt Inhalte und gibt anschließend das neu erzeugte Dokument

[1] XHTML ist ein wohlgeformtes XML-Dokument. eXtensible HyperText Markup Language ist eine Weiterentwicklung von HTML, die komplett auf XML basiert

aus. Der XSLT-Prozess beginnt im Wurzelknoten des Quelldokuments und durchläuft den Baum in Präfix-Reihenfolge und sucht dabei nach entsprechenden übereinstimmenden Produktionsregeln im Stylesheet. Danach wird der Körper abgearbeitet, wobei wieder andere Templates aufgerufen werden können.
Template-Regeln werden mit dem Element *xsl:template* definiert. Sie legen fest, welche Knotentypen des Ausgangsdokuments den Transformationsvorgang durchlaufen. Es folgt die Definition des Musters (Pattern) über "match". Match gibt an, auf welche Elemente eine Template-Regel angewendet werden soll. Im Beispiel oben erfolgt über die XSLT-Instruktion *xsl:apply-templates* die Auswahl von weiteren Elementen aus dem Quelldokument. Diese Auswahl könnte auch über *xsl:for-each* erfolgen.

Arbeitsweise eines Stylesheets:

Ein einfaches XML-Dokument mit Fußballergebnissen wird nun mit einem Stylesheet transformiert:

```
01<results group="A">
02 <game>
03   <date>10-Jun-1998</date>
04   <team score="2">Brazil</team>
05   <team score="1">Scotland</team>
06 </ game >
07 < game >
08   <date>10-Jun-1998</date>
09   <team score="2">Morocco</team>
10   <team score="2">Norway</team>
11 </ game >
12 < game >
13   <date>16-Jun-1998</date>
14   <team score="1">Scotland</team>
15   <team score="1">Norway</team>
16 </ game >
17 < game >
18   <date>16-Jun-1998</date>
19   <team score="3">Brazil</team>
20   <team score="0">Morocco</team>
21 </ game >
22 < game >
23   <date>23-Jun-1998</date>
24   <team score="1">Brazil</team>
25   <team score="2">Norway</team>
26 </ game >
27 < game >
28   <date>23-Jun-1998</date>
29   <team score="0">Scotland</team>
30   <team score="3">Morocco</team>
31 </ game >
32</results>
```

Das nachfolgende XSL/T-Dokument erzeugt aus dem XML-Dokument eine XHTML-Ausgabe:

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
```

```
<xsl:template match="results">
  <html>
  <head>
    <title>
      Results of Group  <xsl:value-of select="@group" />
    </title>
  </head>
  <body>
  <h1>
    Results of Group  <xsl:value-of select="@group" />
  </h1>

  <xsl:apply-templates />
  </body>
  </html>
</xsl:template>
```

```
<xsl:template match=" game ">
  <h2>
  <xsl:value-of select="team[1]" />   versus  <xsl:value-of select="team[2]" />
  </h2>
  <p>
    Played on  <xsl:value-of select="date" />
  </p>
  <p>
    Result:
  <xsl:value-of select="team[1]" />
  <xsl:value-of select="team[1]/@score" />
    ,
  <xsl:value-of select="team[2]" />
  <xsl:value-of select="team[2]/@score" />
  </p>
</xsl:template>
```

```
</xsl:transform>
```

Man parst nun das XML-Dokument von der Wurzel beginnend und bei jedem Auftreten von einem *<result> -Tag* im XML-Dokument wird die Ersetzungsregel des Styleshheets des XSL/T-Dokumentes angewendet (1.Kasten). Entsprechend wird die 2. Ersetzungsregel des XSL/T-Dokumentes beim Auftreten eines *<game> -Tag* ausgeführt (2. Kasten).

Das XML-Dokument wird nun schrittweise durchlaufen und durch die Transformation entsteht schließlich folgendes XHTML-Dokument:

Abb.2.1.3 zeigt das XHTML-Dokument nach der Transformation durch ein Stylesheet
Quelle dieses Beispiels: [KAY01]

2.1.2 XSL/FO

XSL/FO (**eXtensible Stylesheet Language Formatted Object**) ist eine XML basierte Sprache, um einem XML-Dokument ein Layout zu geben. Seit dem 15.Oktober.2001 ist XSL/FO als W3C-Recommendation zu XSL hinzugenommen worden. [SEY02]
Das XSL/FO-Dokument organisiert den ursprünglichen Inhalt, das neu hinzugefügte Layout und stellt wiederum ein wohlgeformtes XML-Dokument dar. XSL/FO legt dabei das Layout der Präsentation des XML-Dokuments bis ins kleinste Detail fest. Alle Elemente werden entsprechend ihrer späteren Position auf einer Seite angeordnet. XSL/FO-Dokumente kann man sich im Browser, falls dieser XSL/FO unterstützt, anzeigen lassen und „just in time“ automatisch in druckbare Formate umwandeln lassen. Aus diesem Grund zeigen immer mehr Unternehmen Interesse an diesem neuen Standard. Gespeichert werden muss lediglich ein XML-Dokument, welches durch Stylesheets (siehe Kapitel 3) in ein XSL/FO-Dokument zur Ansicht im Web transformiert wird. Bei Bedarf kann man dieses Dokument in Druckformate wie PDF, RTF oder Postscript umwandeln. [REN03]

Abbildung 2.1.4 zeigt neben dem im vorigen Kapitel vorgestellten Transformationsprozess die Einbindung von externen Dateien in ein XSL/FO-Dokument. Der **XSL/FO Formatter** erzeugt aus den einzelnen Dateien und dem XSL/FO-Dokument ein Ausgabedokument.

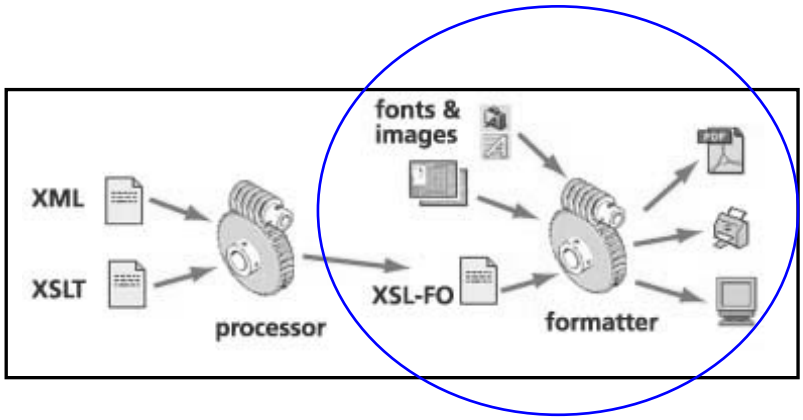

Abb.2.1.4 zeigt die Einbindung von externen Dateien in ein XSL/FO-Dokument [SEY02]

Aufbau eines XSL/FO-Dokumentes:

Das Wurzelelement ist *fo:root*, welches auch den Namensraum festlegt.

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3c.org/1999/XSL/Format"> ...
</fo:root>
```

Innerhalb dieses Tags werden zunächst die möglichen Seitenlayouts beschrieben und in welcher Reihenfolge diese Seiten auftreten können. Das Tag *<layout-master-set>* umschließt die Definitionen für den Aufbau mehrerer Seiten und ihre Abfolge innerhalb des Dokuments (Abb. 2.1.5 blauer Rahmen). Eine Masterseite kann in bis zu fünf **Region-Areas** unterteilt werden. Ein *Body* für den eigentlichen Seiteninhalt, *Before*, *After* für Kopfzeilen, Fußzeilen und *Start, End* für Randbemerkungen. Das Tag *<layout-master-set>* dient somit als Grundgerüst für den Aufbau einzelner Seiten und bestimmt das einheitliche Layout aller Seiten des Dokumentes. Anschaulich kann sich der Leser die Funktionsweise des Tag *<layout-master-set>* analog zu der Funktion des Masterlayouts bei Powerpoint (WindowX Office-Paket) vorstellen.

```
<fo:layout-master-set>
 <fo:simple-page-master master-name="simple"
 page-height="29.7cm" page-width="21cm"
 margin-top="1cm"
 margin-bottom="2cm"
 margin-left="2.5cm"
 margin-right="2.5cm">
 </fo:simple-page-master>
 </fo:layout-master-set>
```

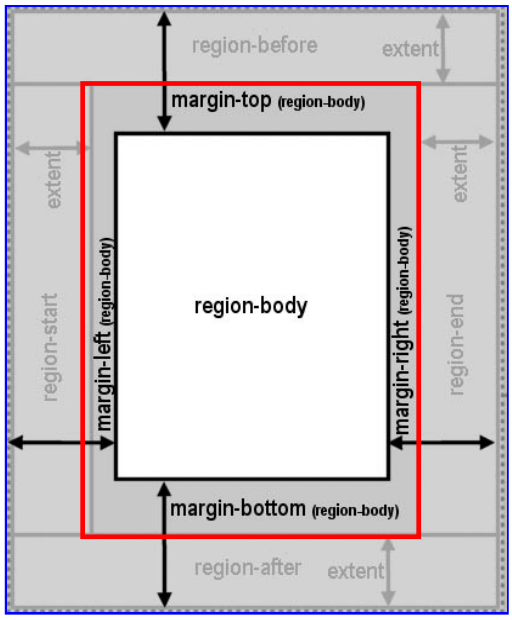

Abb. 2.1.5 zeigt den Aufbau einer XSL/FO-Seite

Danach wird der Seiteninhalt einzelner Seiten festgelegt. Eine einzelne Seite wird mit dem Tag *<simple-page-master>* definiert (Abb. 2.1.5 roter Rahmen). Das Formatierungsmodell von XSL/FO basiert auf Bereichen (**Block-Areas**), die Texte, Grafiken oder weitere Formatierungsobjekte enthalten können. Eine externe Grafik kann leicht durch folgenden Befehl angeordnet werden.

```
<fo:block>
<fo:external-graphic src="bild.jpg"/>
</fo:block>
```

Ein komplettes XSL/FO-Dokument wird im 3.Kapitel vorgestellt und weiter vertieft. Es sei lediglich noch erwähnt, dass man bei XSL/FO an die zunehmende Globalisierung und an die Vielzahl von unterstützenden Landessprachen gedacht hat. Da man in anderen Kulturen nicht wie in Europa von links nach rechts und von oben nach unten schreibt, so kann man diesem Aspekt entgegenwirken, indem man Top, Bottom, Left und Right entsprechend anpasst. XSL/FO-Dokumente kann man somit leicht in verschiedene Landessprachen übersetzen, ohne sich Gedanken über das Layout machen zu müssen. [BDG01]

2.2 SVG (Scaleable Vector Graphic)

SVG (**Scalable Vector Graphics**) Version 1.1 ist seit dem 14.Januar 2003 eine W3C-Recommendation. [SVG03] Die auf XML basierende Vektorgrafiksprache SVG soll die Verwendung von Vektorgrafiken im Internet erleichtern. Das W3C definiert SVG als Sprache, die zweidimensionale Grafiken in XML beschreibt. SVG unterstützt dabei drei Arten von grafischen Objekten: Vektorgrafiken (beispielsweise Linien, Kurven, Polygone, Pfade), Pixelbilder und Text. In SVG erstellte Grafiken können zudem interaktive und dynamische Eigenschaften besitzen. SVG kann separat in einem Texteditor entwickelt werden oder innerhalb eines Grafikprogramms wie z.B. Photoshop von Adope. SVG weißt im Gegensatz zu anderen Grafikformaten viele Vorteile auf:

- Erzeugte Dateien sind kleiner als entsprechende GIF-, JPG- oder PNG-Dateien und der Programm-Code lässt sich zusätzlich komprimieren.
- Dank der Skalierbarkeit von Vektorgrafiken passt sich die Anzeige der Grafiken der jeweiligen Auflösung des Ausgabegeräts an. SVG-Grafiken können also auch mit Handhelds ohne Qualitätsverlust verwendet werden.
- SVG-Inhalte können über XSL formatiert werden.
- SVG unterstützt Animationen und Interaktionen (z. B.: mouseon, onClick)
- Dank Xlink kann jeder Bereich einer SVG-Grafik verlinkt werden.
- SVG bietet zahlreiche Filtereffekte.
- SVG ermöglicht das Schwenken und Zoomen in Grafiken ohne Qualitätsverlust.
- SVG-Dokumente sind, da sie aus Tags und Metadaten bestehen, von Suchmaschinen indexierbar. [CON03]

Allgemeiner Aufbau eines SVG-Dokumentes:

```
1  <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
3     "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
4  <svg xmlns="http://www.w3.org/2000/svg"
5       xmlns:xlink="http://www.w3.org/1999/xlink">
6     <-- SVG content goes here -->
7  </svg>
```

1: XML-Deklaration
2-3: Angabe der benutzten DTD
4: SVG-Rootelement wird geöffnet
5: XLink Namensraum wird deklariert um externe Bilder und Objekte, die eingebunden werden zu referenzieren
6: benutzerdefinierte Anweisungen
7: das Rootelement SVG wird geschlossen

Aus dem reichhaltigen Schatz von SVG-Befehlen werden hier einige angeführt:

Quelle: [BDG01]

Rechteck

```
<rect x="5" y="5" rx="5" ry="5" width="40" height="40" fill="red"/>
```

Optional rx, ry für gerundete Rechtecke.

Text auf einem Pfad

```
<defs>
   <path id="textPath" d="M10 50 C10 0 90 0 90 50"/>
</defs>
<use xlink:href="#textPath" stroke="blue" fill="none"/>
<text fill="red">
   <textPath xlink:href="#textPath">Text on a Path</textPath>
</text>
```

Ellipse

```
<ellipse cx="25" cy="25" rx="20" ry="10" fill="red"/>
```

Definiert werden Zentrum sowie der x/y-Achsenradius der Ellipse.

Befehl fuer eine Animation:

```
<rect y="45" width="10" height="10" fill="red">
   <animate attributeName="x" from="0" to="90" dur="10s"
         repeatCount="indefinite"/>
</rect>
```

Filtereffekte:

Durch Filtereffekte wird das Originalbild modifiziert und das neu entstandene Bild wird angezeigt. Das Originalbild bleibt dabei unverändert. Auf dem Ausgabegerät oder Drucker wird lediglich durch Rendern das neu erzeugte Bild angezeigt. Man kann in seinem SVG-Dokument somit leicht die Originalbilder austauschen bei Veränderungen. Die Filtereffekte werden dann auf die neuen Bilder angewendet und auf dem Ausgabegerät neu angezeigt.

Abb.2.2.1 zeigt die Auswirkung eines Filtereffekts [FIL03]

Zoomen

Da SVG eine Vektorgrafik ist, kann man einzelne Bereiche eines Bildes beliebig vergrößern. Durch das Zoomen wird im Gegensatz zu Pixelbildern durch die Vektorgrafik die Qualität des Bildes nicht gemindert.

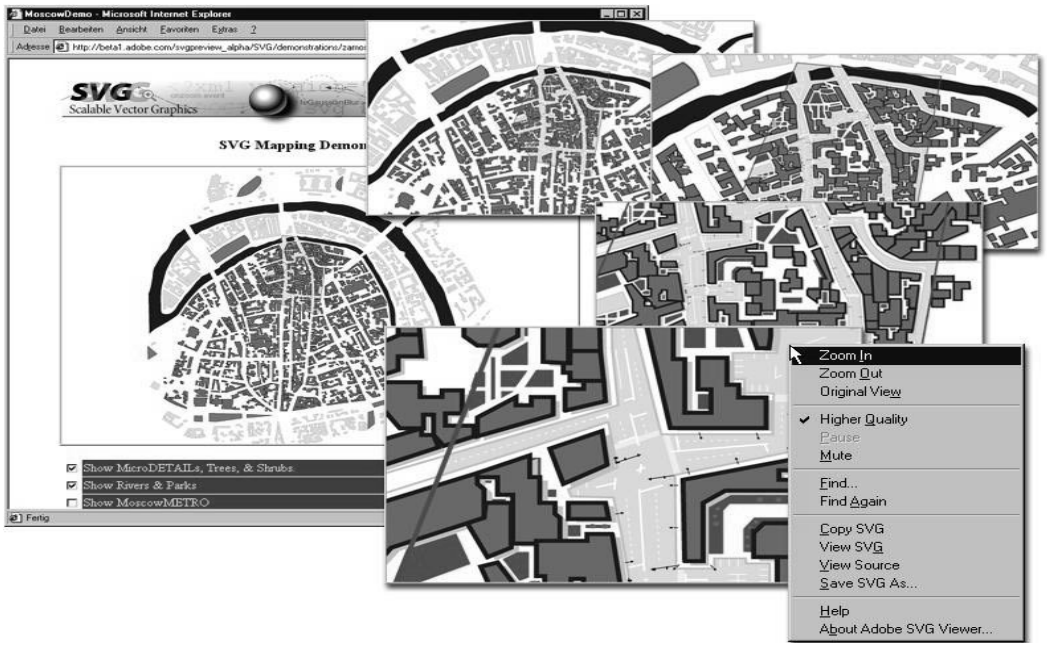

Abb.2.2.2 zeigt die Auswirkung des Zoomens [SMK03]

2.3 DocBook

DocBook Version 5 ist ein Document Type Definition um strukturierte Dokumente in XML zu schreiben. Seit 1991 wurde DocBook in einem gemeinsamen Projekt von HaL Computer Systems und O´Reilly entwickelt und seit 1998 von **OASIS** (Organization for the Advancement of Structured Information Standards) übernommen. [WIE01]
OASIS, eine non-profit-Gesellschaft, ist ein internationales Konsortium, welches Spezifikationen im Zusammenhang mit strukturierter Informationsverarbeitung erstellt. [OAS02]
DocBook ermöglicht es dem Autor seine Bücher und Artikel in einer präsentationsneutralen Form zu speichern. Die Struktur und der Inhalt werden getrennt und mit Hilfe von XSL kann man die DocBook-Dokumente in andere Formate wie z. B. HTML, PDF, PS, XSL/FO, man pages (Unix), u.s.w. transformieren. Viele Autoren benutzen DocBook um ihre Artikel und Bücher einer breiten Masse von Lesern zugänglich zu machen. Eine Reihe von Open-Source-Projekten stellt Tools zur Transformation in gängige Formate bereit. [ORE02]

Die Elemente der DocBook DTD werden in zwei Klassen unterteilt. Zum einen gibt es die **hierarchischen Elemente**. Sie dienen zur Beschreibung einer umfassenden Struktur bzw. Gliederung des Dokumentes in einzelne Teile. Zum anderen gibt es **Info-Pool Elemente**, welche für das Basis-Markup verwendet werden.

Hierarchie Elemente:

- Zusammenstellungen (*set*) - Sie enthalten zwei oder mehr Bücher, welche sich einem Thema widmen, und darum eine Einheit bilden. Das *set* Element bildet die Spitze der Elementhierarchie in DocBook.
- Bücher (*book*) - Das *book* Element ist das geläufigste Top-Level Element unter Set für einzelne Bücher.
- Artikel (*article*) - Dieses Element kann z.B. für Zeitschriftenartikel, White Papers und technische Anmerkungen genutzt werden.
- *Reference Pages* - Sie werden für Unix Man-Pages genutzt.

Diese Elemente können die folgenden Hierarchie Elemente enthalten.

- Widmung (*dedication*)
- Unterteilungen (*part*, *reference*)
- Komponenten, wie Vorwort (*preface*), Kapitel (*chapter*), Anhang (*appendix*), Glossar (*glossary*) und Artikel (*article*)
- Navigationskomponenten, wie Inhaltsverzeichnisse (*toc*), Namenslisten (*lot*) für Listen von Abbildungen, Tabellen und Beispielen, und Indexe (*index*)

Info-Pool Elemente:

Die Info-Pool Elemente sind unterteilt in Block-Elemente und Inline-Elemente.

Block-Elemente:

- Listen (*itemizedlist*, *orderedlist*, ...)
- Ermahnungen (*note*, *tip*, *warning*, ...)

- Zeilen-spezifische Umgebungen (*address*, *programlisting*, ...)
- Beispiele (*example*), Abbildungen (*figure*) und Tabellen (*table*)
- Absätze (*para*, ...)

Inline-Elemente:

- traditionelle (*abbrev*, *footnote*, *quote*, ...)
- Cross Referenz (*anchor*, *citetitle*, *link*, ...)
- Markup (*foreignphrase*, *computeroutput*, ...)
- Mathematik (*subscript*, *superscript*, ...) hierfür besser MathML
- Nutzerinterface (*guibutton*, *guimenuitem*, *shortcut*, ...)
- Programmierung (*classname*, *errorcode*, *returnvalue*, ...)

Beispiel eines DocBook-Dokumentes:

```
<book>
  <bookinfo>
    <title>Mein Buch</title>
    <author>
      <firstname>Steffen</firstname>
      <surname>Apfel</surname>
    </author>
    <copyright>
      <year>2003</year>
      <holder>AGDBIS</holder>
    </copyright>
  </bookinfo>
  <preface>
    <title>Vorwort</title>
                ...
  </preface>
  <chapter>...</chapter>
  <chapter>...</chapter>
  <appendix>...</appendix>
  <index>...</index>
</book>
```

3. Single-Source-Publishing

Single-Source-Publishing-Systeme (**SSP-Systeme**) erfreuen sich bei Unternehmen und auch Privatpersonen, die ihre Daten in mehreren Ausgabeformaten benötigen, einer immer größer werdenden Beliebtheit. Single Source Publishing verfolgt dabei die Idee des „**store once, use everywhere**“. Man speichert seine Daten und Dokumente *unabhängig* von der *späteren Anwendung* in einem XML-Dokument und generiert beim konkreten Anliegen erst sein Ausgabeformat.

Single Source Publishing hilft dabei Zeit und Geld zu sparen. Man speichert Texte, Daten, Unterlagen, SVG-Grafiken u.s.w. in XML-Dokumenten und modifiziert diese Daten an zentraler Stelle. Dadurch reduziert man Fehler, vermeidet doppelte Arbeit und Redundanzen und verringert die Kosten für Übersetzung und Speicherung in mehrere Formate, die „just in time“ bei Gebrauch erstellt werden aus den XML-Dokumenten. [SSP03]
Die XML-Dokumente kann man mit Hilfe von Parsern und Transformationssprachen in beliebige Ausgabeformate umwandeln. Einige Techniken wurden im vorherigen Kapitel kurz vorgestellt. Abbildung 3.1 gibt eine kleine Übersicht über die möglichen Transformationen. Neben der Webausgabe in HTML, der Druckausgabe in PDF, DOC kann man zahlreiche andere Formate erzeugen aus XML.

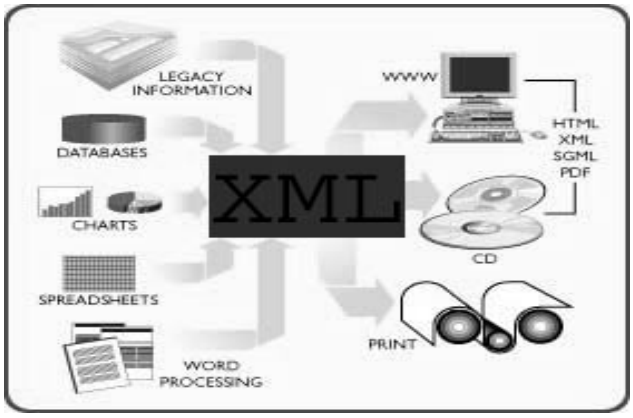

Abb. 3.1: Prinzip des Single-Source-Publishing mit XML [MEK02]

Im Folgenden werden drei SSP-Systeme namentlich vorgestellt. An dieser Stelle soll lediglich ein kleiner Überblick über bestehende Systeme gegeben werden und der interessierte Leser kann mit Hilfe der Literaturangaben seine Recherchen weiterführen. [XML03]
XEP, RenderX wurde in Java implementiert, kann leicht in andere Systeme integriert werden und erstellt aus XML PDF und PS. [XEP03]
FOP, Apache ist ein Open Source Projekt, welches ebenfalls in andere bestehende Systeme integriert werden kann und XML in viele Ausgabeformate transformiert: PDF, PCL, PS, SVG, AWT, MIF und TXT. [FOP03]

AntennaHouse XSLFormatter 2.0 wird nun stellvertretend näher vorgestellt [ATH03]
AntennaHouse bietet eine komfortable grafische Benutzeroberfläche, um XML in XML/FO umzuwandeln. Die XML/FO-Dokumente kann man sich in einem Browser anzeigen lassen oder in PDF umwandeln, um eine Druckausgabe zu ermöglichen. Das folgende Beispiel zeigt, wie ein XML-Dokument mittels eines XSL-Dokument transformiert wird. Ein XML-Dokument dient als Eingabe für dieses Tool:

```
01 <?xml version="1.0" encoding="utf-8" ?>
02<test>
 03<section>
 04  <subsection>
 05    <title>XML Publishing</title>
 06    <para>creating a first table</para>
 07    <tbl>
 08    <row>
 09      <col>
 10      <para>Ausgangsdatei: XML</para>
 11      <para>mit Hilfe von XSL</para>
 12      <para>ensteht ein XSL/FO - File</para>
 13      </col>
 14   </row>
 15  </tbl>
 16   <para />
```

```
17 <para>creating a second table</para>
18 <tbl>
19 <row>
20   <col>
21     <para>Dieses XSL/FO - File kann </para>
22   </col>
23   <col>
24     <para>im Web dargestellt werden</para>
25  </col>
26  <col>
27    <para>oder als PDF zum Drucken</para>
28  </col>
29 </row>
30 </tbl>
31</subsection>
32</section>
33</test>
```

Ein Stylesheet, das XML in ein XSL/Formatted Objekt transformiert:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:psmi="http://www.CraneSoftwrights.com/resources/psmi"
   xmlns="http://www.w3.org/1999/XSL/Format" version="1.0">
```

```
 <xsl:template match="/">
    <root font-family="Times" font-size="20pt">
    <layout-master-set>
            .
            .
    </layout-master-set>
    <page-sequence master-reference="frame" initial-page number="1">
            .
            .
    </page-sequence>
    </root>
 </xsl:template>
```

```
 <xsl:template match="title">
          <block text-align="center" font-size="2em" space-after.optimum="40pt">
                  <xsl:apply-templates />
          </block>
</xsl:template>
```

```
<xsl:template match="para">
        <block space-after.optimum="15pt">
                <xsl:apply-templates />
        </block>
</xsl:template>
```

```
<xsl:template match="tbl" name="do-a-table">
        <table color="blue">
        <table-body>
                <xsl:apply-templates />
        </table-body>
        </table>
</xsl:template>
```

```
<xsl:template match="row">
        <table-row>
                <xsl:apply-templates />
        </table-row>
</xsl:template>
```

```
<xsl:template match="col">
        <table-cell border="solid">
                <xsl:apply-templates />
        /table-cell>
</xsl:template>
```

```
</xsl:stylesheet>
```

Der AntennaHouse XSL Formatter transformiert mit Hilfe des Stylesheets ein XML-Dokument in ein XSL/FO-Dokument, welches in der Abbildung 3.2 angezeigt wird. Die Transformation eines XML-Dokumentes mittels eines Stylesheets wurde im 2. Kapitel näher vorgestellt. Dieses XSL/FO-Dokument wird in Abb.3.2 angezeigt. Man kann nun im Menü „File“ leicht diese zu einem PDF-Dokument umformen.

Durch dieses Beispiel soll die Idee des Single-Source-Publishings nochmals verdeutlicht werden. Man hält seine Daten in einem XML-Dokument und kann nach Gebrauch eine Webdarstellung und PDF als Druckformat erzeugen.

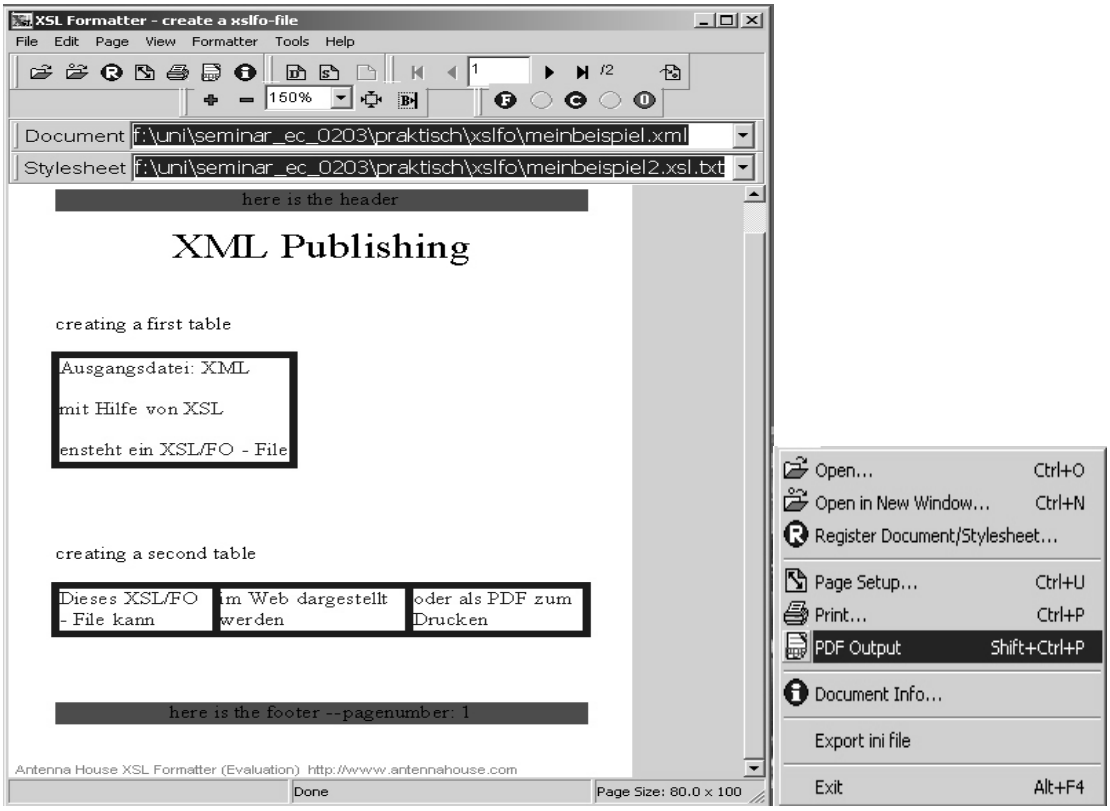

Abb. 3.2: Screenshot des AntennaHouse XSL Formatter

4. Systeme/Frameworks für Web-basierte Systeme

Systeme/Frameworks für web-basierte Systeme stellen eine Plattform für Single-Source-Publishing-Systeme dar. Die Frameworks bieten Schnittstellen um vorhandene SSP-Systeme zu erweitern und an neue Problemstellungen anzupassen oder neue Publishing-Plattformen zu erstellen. Stellvertretend für Frameworks für XML-Publishing und Visualisieren wird **Cocoon** von Apache näher vorgestellt.
Apache, ein Open Source Projekt, wurde im Februar 1999 ins Leben gerufen zur Erstellung von hochwertigen XML-Lösungen, die kommerziellen Produkten in nichts nachstehen. Apache stellt eine Austauschplattform für XML-Projekte dar und eines der bekanntesten Teilprojekte ist Cocoon. Aufgabe dieses Publishing Framework ist die **dynamische Gestaltung** von **Webseiten** und Erstellung von **Präsentationen**. Angeforderte Ressourcen werden je nach Anfrage für die Nutzung aufbereitet. [APA03]
Ein wesentliches Konzept von Cocoon ist der modulare Aufbau mit Standardschnittstellen, so dass einzelne Komponenten ausgetauscht werden können. Cocoon baut auf dem Prinzip vom Single-Source-Publishing auf.

Mit Cocoon soll die Erstellung von Webseiten erleichtert werden. Man möchte eine **Separation of Concerns (SoC)**, d. h. Trennung von *Inhalt*, *Logik* und *Darstellung* bei der Entwicklung von Webseiten erreichen. Abb.4.1 zeigt einen Überblick über diese Aufgabenteilung.

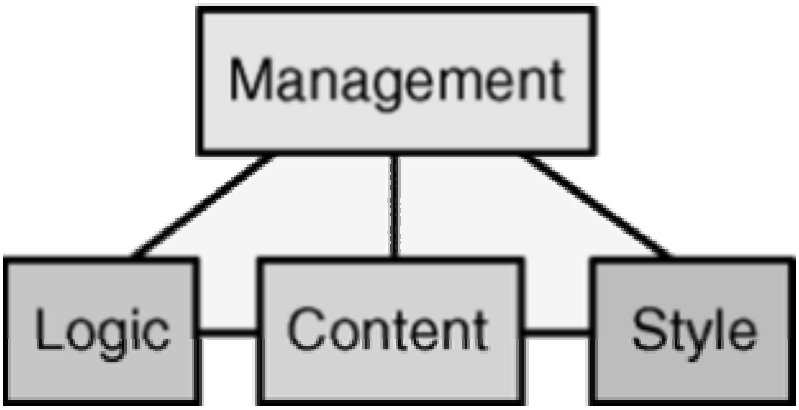

Abb. 4.1 zeigt die separate Entwicklung einer Webseite (SoC) [APA03]

Es soll eine klare Trennung der verschiedenen Arbeitsbereiche in einem Entwicklungsteam von Webseiten entstehen. Das **Management** kümmert sich um den Ablauf, Aufbau aller Seiten und organisiert den gesamten Entwicklungsablauf. Der redaktionelle Teil soll die Inhalte in einem XML-Dokument erstellen. Die Redakteure brauchen sich dabei nur auf den **Content-**Teil zu konzentrieren und lassen alle Designentscheidungen bei der Entwicklung ihrer XML-Dokumente ausser Acht. Eine weitere Arbeitsgruppe organisiert die dynamischen Inhalte einer Seite. Dieser Bereich wird im **Logic-**Teil bearbeitet. Der dynamische Webinhalt wird mittels **XSP (eXtensible Server Pages)**[2] entworfen. Der **Style**-Bereich kümmert sich um die Präsentation und Grafik der Webseiten. Sie entwerfen hauptsächlich Stylesheets um die XML-Dokumente zu transformieren.
Durch die Verwendung von XML-Dokumenten kann man nun auch leicht externe **Datenquellen einbinden**, wie z.B.: Filesysteme, native XML Datenbanken und netzwerkbasierte Quellen. Je nach Anfrage eines Clients kann das gewünschte Ausgabeformat der Webseiten geändert werden. Die XML-Dokumente lassen sich leicht in unterschiedliche **Präsentationsformate überführen** und ausliefern (in HTML, WML, PDF, SVG, RTF).

Verarbeitung eines XML-Dokumentes:

Mittlerweile ist Cocoon2 entwickelt worden. Um allerdings die Verarbeitung eines XML-Dokuments zu verdeutlichen wird kurz auf den Verarbeitungsprozess von Cocoon1 eingegangen. Bei Cocoon1 kann man den Verarbeitungsprozess sehr plastisch und anschaulich darstellen und danach werden die Erneuerungen von Cocoon2 erklärt.

[2] XSP ist eine Weiterentwicklung von Java Server Pages (JSP), die komplett auf XML basieren

Funktionsweise von Cocoon1:

Cocoon1 bietet ein Framework, um in einer Verarbeitungskette verschiedene Verarbeitungsprozessoren einzusetzen, ähnlich einer Pipe in Unix. Über sogenannte **Processing Instructions** (**PI**) wird Cocoon1 mitgeteilt, welcher Prozessor im nächsten Schritt das XML-Dokument, bzw. den erzeugten XML-Baum bearbeiten soll. Sind alle PI´s abgearbeitet wird das Dokument an den Client gesendet. In Abbildung 4.2 wird die Verarbeitungsstruktur von Cocoon1 noch einmal präziser dargestellt.

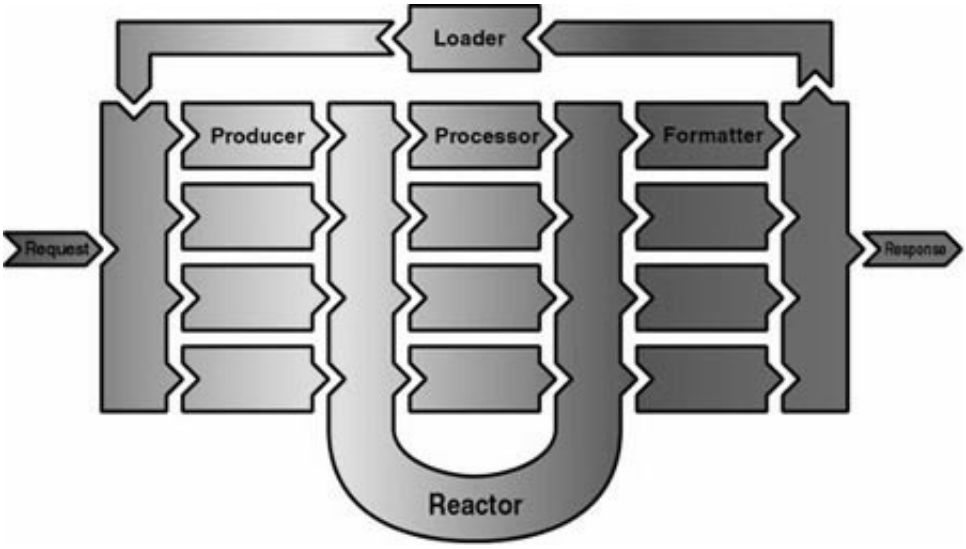

Abb. 4.2 zeigt den Verarbeitungsprozess von Cocoon1 [COC03]

Request:
Der Request enthält alle für die Verarbeitung der Anfrage wichtigen Informationen (welche URI, Typ des Browser, welcher Producer aktiviert werden soll) und wird vom Client an den Server geschickt.

Producer:
Ein Producer bearbeitet den eingegangenen Request und liefert als Ergebnis ein XML-Dokument, dessen Inhalt abhängig vom entsprechenden Producer ist.

Reactor :
Der Reactor bestimmt auf Basis von XML-PI´s, welcher Prozessor das XML-Dokument weiterverarbeiten soll. Nachdem die Verarbeitung abgelaufen ist, ist es Aufgabe des Reaktors, das überarbeitete Dokument an den vorgegebenen Formatter weiterzuleiten.

Formatter:
Ein Formatter formt das XML-Dokument in einen für Browser/Client verarbeitbaren Datenstrom um (MIME Typ setzen, ausführbarer Code). Der richtige Formatter wird mittels eines PI-Tag bestimmt (<?cocoon-format type="yyy"?>).

Response:

Der Response fasst das neu erzeugte Dokument mit weiteren Eigenschaften wie Länge, MIME Typ usw. zusammen und schließt den Verarbeitungsprozess ab.

Loader:
Der Loader übergibt Dokumente mit zusätzlichen PI´s wieder an den Producer und gliedert diese wieder in die Verarbeitungskette ein für einen evtl. zweiten Verarbeitungsschritt (inkrementelle Verarbeitung). Ansonsten werden diese an die Response weitergeleitet.

Processor:
Der für die Verarbeitung benötigte Prozessor wird über die PI´s bestimmt. Das folgende Tag <?cocoon-process type="xslt"?> enthält die Information, welcher Prozessor benutzt werden soll (in diesem Beispiel wird der XSL/T-Prozessor verwendet).

- XSL/T-Prozessor: XSLT-Transformationen werden angewendet.
- XSP-Prozessor: verarbeitet XSP-Seiten.
- SQL-Prozessor: wickelt SQL-Datenbankabfragen mittels einfacher Tag-Sprache ab.
- LDAP-Prozessor: ermöglicht den Zugriff auf LDAP-Verzeichnisse. Liefert ein XML-Dokument zurück.

Durch ein PI-Tag kann zusätzlich bei einer Webausgabe angegeben werden, welcher Browser den Request stellte, um die Ausgabe dementsprechend anzupassen (z.B.: <?xml-stylesheet href="hello-lynx.xsl" type="text/xsl" media="**netscape**"?>).

Ein Client schickt also einen Request an den Server mit den Informationen welches Ausgabeformat er zurück erhalten möchte. Cocoon arbeitet den Auftrag ab und sendet dem Client das gewünschte Ausgabeformat. Cocoon stellt somit ein sehr mächtiges Framework zur Verfügung um mit XML-Dokumenten zu arbeiten.

Erneuerungen in Cocoon2:

Das Hauptziel bei der Entwicklung von Cocoon2 war eine Verbesserung der Performance. Es wurde von ein **SAX-Parser** anstelle eines DOM-Parsers verwendet um den Speicherverbrauch zu verringern. Die **Carbage Collection** wird durch SAX-Verarbeitung ebenfalls geringer, weil nicht das komplette XML-Dokument im Speicher gehalten wird. (nähere Informationen im Seminarvortrag „*XML-Verarbeitungsmodelle und Language Bindings*“ dieser Vortragsreihe)
Außerdem wurde nun eine **simultane Verarbeitung** mehrerer Anfragen ermöglicht und das Framework ist **skalierbar** je nach Lastsituation. Die Response wird nun schon während der Bearbeitung gesendet. Man wartet nicht bis ein Auftrag komplett bearbeitet wurde, sondern sendet dem Client schon Zwischenergebnisse, wodurch natürlich die Performance der Verarbeitung gesteigert wird. Allerdings ist diese Neuerung schlecht bei inkrementeller Verarbeitung, wenn die Ergebnisse einer ersten Verarbeitung nochmals transformiert und verändert werden müssen. Bei Cocoon2 gibt es eine Erkennung von wieder verwendbarem Kode. Durch diese **„Hot Spot“-Erkennung** wird eine schnellere Verarbeitung durch Speicherung relevanter Kodefragmente erreicht.

Eine der wichtigsten Erneuerungen ist die **Pipeline Mapping Technique** (flexible Nutzung durch Sitemaps) anstatt dem Reactor-Prinzip (starre Folge von Bearbeitungsschritten). Es stellte sich heraus, dass das Reactor Prinzip, welches durch PI´s die Verarbeitungsschritte genau vorgibt, zu unflexibel ist. An dessen Stelle tritt nun eine flexible Pipelinebearbeitung. Die Verarbeitung erfolgt ähnlich zu dem Reactor-Prinzip. Lediglich eine Sitemap enthält nun Konfigurationsinformationen für Cocoon, welche Verarbeitungsschritte als nächstes dynamisch zu bewältigen sind.

5. Zusammenfassung und Ausblick

XML liefert flexible Mechanismen für den Datenaustausch und die Aufbereitung von Inhalten für die unterschiedlichsten Zielmedien. Realisiert wird dies über die Trennung von Inhalt, Struktur und Erscheinungsbild. Ein klassisches Beispiel: Ein zu druckender Artikel wird in einem Textverarbeitungsprogramm verfasst, während derselbe Artikel nochmals in HTML fürs Web oder Ausgabe in PDA´s neu aufbereitet werden muss. XML wird auf Dauer dieses grundlegende Problem beseitigen. Egal, ob für den Druck, fürs Internet oder für andere Anwendungen - idealerweise greift man lediglich einmal auf in XML aufbereitete Inhalte zurück.
Ein weiterer wichtiger Aspekt bei der Weiterentwicklung und Verbreitung von XML ist die Erweiterbarkeit von XML an bestehende Problemlösungen und zudem auch die leichte Erlernbarkeit. XML ist außerdem von Mensch und Maschine leicht lesbar und muss deshalb nicht zum besseren Verständnis in mehreren Formaten vorliegen.
Dieses Seminar sollte dem Leser einen kleinen Einblick in die vielfältigen Verarbeitungs- und Einsatzmöglichkeiten von XML geben.
Es wurden einige Techniken von XML vorgestellt und danach die Idee des Single-Source-Publishings beschrieben. Das Ziel „Store once, use everywhere“ wird durch XML schon heute in vielen Teilbereichen des Datenaustauschs erreicht, sodass die unterschiedlichsten Applikationen über XML miteinander kommunizieren können. Eine Verwirklichung dieses Single-Source-Ansatzes wurde stellvertretend für webbasierte Frameworks anhand von Cocoon vorgestellt, welches einige der vorgestellten Techniken verwendet. Der interessierte Leser kann selbstständig in den aufgeführten Literaturangaben recherchieren um sich über weitere Handhabungen von XML zu informieren. XML gehört, nach heutigem Stand der Technik, die Zukunft u.a. im Bereich des Daten- und Dokumentenaustausch und der Präsentation von Daten.

6. Literatur

[W3C02] http://www.w3.org,12/2002

[DES03] S. Dessloch: Vorlesung Workflow & Web Services (Kapitel 2: XML Basics for Web Services), Universität Kaiserslautern, WS 2002/03

[HARE02] W. Hausknecht, F. Ressel: http://www.segma.de/vorlesung02/10_xml_folien.pdf, 2002

[FLY02] P. Flynn: http://www.ucc.ie/xml/faq.xml, 01/2002

[XSL02] http://www.w3.org/TR/xsl, 12/2002

[XSL03] http://www.w3.org/TR/xsl/slice1.html#section-N629-Introduction-and-Overview, 01/2003

[XSLT02] http://www.w3.org/TR/xslt, 12/2002

[KAY01] M. Kay: Web Services/XML CD March 2002 : What kind of language is XSLT?, IBM, Feb.2001

[SUN02] J. Sundman : XSL transformation basics, developerWorks Journal, Aug.2002

[SEY02] http://www.seyboldreports.com/TSR/free/0217/techwatch.html, 12/2002

[REN03] http://www.renderx.com/Tests/doc/html/tutorial.html, 01/2003

[BDG01] M. Bireck, J. Duckett, O. G. Gudmundsson: Professional XML, Wrox Press, 05/2001

[SVG03] http://www.w3.org/TR/SVG11/, 01/2003

[FIL03] http://www.w3.org/TR/SVG11/filters.html, 01/2003

[SMK03] http://www.svgmaker.com, 01/2003

[CON03] http://www.w3.org/TR/SVG11/concepts.html, 01/2003

[WIE01] M.Wiedmann: http://www.miwie.org/db-beginners, 11/2001

[OAS02] http://www.oasis-open.org, 12/2002

[ORE02] *http://www.oreilly.com/catalog/docbook/chapter/book/ch01.html*, 12/2002

[SSP03] http://www.schema.de/sitehtml/site-d/singleso.htm, 01/2003

[XML03] http://www.xmlsoftware.com/xslfo.html, 01/2003

[XEP03] http://www.renderx.com, 01/2003

[FOP03] http://xml.apache.org/fop/index.html, 01/2003

[MEK02] http://www.mekon.com/bureau/single-source.htm, 2002

[ATH03] http://www.antennahouse.com, 01/2003

[APA03] http://xml.apache.org/cocoon/index.html, 01/2003

[COC03] http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/8.cocoon/cocoon0.html, 01/2003