Ausarbeitung für das
Hauptseminar eLearning-Systeme im SS2003
TU München

XSLT Prozessoren

von
Daniel Brügge, bruegged@in.tum.de

Betreuer:
Jan-Thomas Czornack

Inhaltsverzeichnis

1 Einleitung

XSLT Prozessoren dienen zur Formatierung von XML Dokumenten mit Hilfe von XSL Stylesheets. Sie überführen (transformieren) das XML Dokument in ein anderes textbasiertes Dokument wie beispielsweise HTML, Plaintext, RTF oder wiederum XML.

Die folgende Ausarbeitung soll einen Einblick in die Funktionsweise von XSLT Prozessoren im allgemeinen geben und im Anschluss einige bekannte Vertreter näher vorstellen. Aufgrund der detaillierten Angaben die in [Kay-01] gegeben werden, wird SAXON besonders ausführlich erwähnt, aber auch Xalan und MSML sind extra Unterkapitel gewidmet.

Zuallererst wird aber eine kurze Einführung in XSLT gegeben, der Sprache mit der XSL Stylesheets formuliert werden, die für die Arbeit von XSLT Prozessoren unabdingbar sind.

2 XSLT

XSLT steht für *XSL Transformations* und bildet zusammen mit XSL-FO (*XSL Formatting Objects*) und XPath (*XML Path Language*) die *Extensible Stylesheet Language* (XSL). XSLT existiert zurzeit in der Version 1.0 und wurde im November 1999 vom *World Wide Web Consortium*[1] (W3C) als Empfehlung herausgegeben. Eine überarbeitete Version von XSLT, XSLT 2.0, befindet sich mittlerweile schon in Arbeit, hat aber noch nicht den Status einer W3C-Empfehlung erreicht, sondern ist immer noch in der Entwurfsphase.

XSLT ist die Sprache in der XSL-Stylesheets formuliert werden. Diese werden in Verbindung mit XML-Dokumenten verwendet und dienen dazu, diese XML-Dokumente zu formatieren und für die unterschiedlichsten Ausgabemedien vorzubereiten. Dieses kann zum Beispiel ein einfacher Internetbrowser sein, ein PDA, ein Audiogerät und vieles weiteres mehr. XML-Dokumente beinhalten nur Daten und im Gegensatz zu HTML keine Anweisungen, wie diese Daten später dargestellt werden sollen. Aus diesem Grund benötigt ein XML-Dokument immer eine XSL-Stylesheet, welches die Formatierung übernimmt. Bei dieser Formatierung kommt nun auch XSLT als Teil von XSL ins Spiel.

Ein XML-Dokument bildet einen sogenannten Dokumentenbaum, da die Elemente innerhalb des Dokumentes hierarchisch angeordnet sind. XSLT kann nun aus diesem Eingabebaum mittels Transformationen einen neuen Ausgabebaum erzeugen. Dieser Ausgabebaum kann nun entweder wieder ein XML-Dokument sein, aber auch beispielsweise ein reines Textdokument oder auch ein HTML-Dokument usw.

Die Transformation, die XSLT vornimmt, erfolgt durch sogenannte „Template-Regeln“. Eine Template-Regel besteht aus einem Muster und einem Template. Das Muster gibt an, auf welche Knoten in dem XML-Eingabebaum das jeweilige Template angewendet werden soll und das Template wird entsprechend in den Ausgabebaum eingefügt, falls eine Übereinstimmung durch das Muster getroffen wurde. Solch ein Template kann entweder normale Literale enthalten oder auch spezielle XSLT Befehle, welche dann wieder auf Unterknoten, des aktuell durch das Muster identifizierten Knotens, angewendet werden können.

Die Erkennung der Knoten durch die Muster erfolgt mit Hilfe von XPath, einem der drei Bestandteile von XSL. XPath kann Knoten in einem XML-Dokument anhand ihrer Position im Dokumentenbaum, anhand des Namens, des Inhalts, usw. identifizieren. Eine komplette Einführung in XPath würde den Rahmen dieser Ausarbeitung sprengen, aber man kann noch kurz erwähnen, dass XPath zwischen sieben unterschiedlichen Knotentypen in einem XML-Dokument unterscheiden kann: Wurzelknoten, Elementknoten, Attributknoten, Textknoten, Kommentarknoten, Verarbeitungsanweisungsknoten und Namensraumknoten. Weitere Details kann man [Clark-99] entnehmen.

Da es sich bei einem XSL-Stylesheet auch „nur“ um ein XML-Dokument handelt und dieses nicht eigenständig ein Eingabedokument formatieren kann, braucht man dazu noch einen Prozessor. Mit diesen sogenannten XSLT-Prozessoren wollen wir uns in den nächsten Kapiteln ausgiebig beschäftigen und einige wichtige Prozessoren der heutigen Zeit vorstellen.

[1] http://www.w3.org/

3 XSLT Prozessoren

Wie schon in der Einführung zu XSLT (siehe Kapitel 2) erwähnt wurde, benötigt man zur Formatierung eines XML-Dokumentes mit Hilfe eines XSL-Stylesheets einen sogenannten XSLT-Prozessor, der die Template-Regeln innerhalb des Stylesheets auf das Eingabedokument anwendet und daraus ein neues Ausgabedokument erzeugt.

Der XSLT Prozessor benötigt, um die Transformationen korrekt durchführen zu können, zwei Eingabedokumente: das XML Quelldokument und das Stylesheet Dokument. Der Prozessor erhält zwar als Eingabe beide Dokumente beispielsweise als Datei, aber er arbeitet bei seinen Transformationen nicht direkt auf diesen Originaldokumenten, sondern er führt alle Tätigkeiten auf Baumrepräsentationen aus, welche sich im Arbeitsspeicher befinden. Diese Baumrepräsentationen werden in einem ersten Schritt erzeugt, bevor der XSLT Prozessor seine eigentliche Arbeit aufnimmt.

Diese Generierung der Baumrepräsentationen wird von einem XML Parser übernommen, wobei man hier aber noch zwei unterschiedliche Vorgehensweisen unterscheiden muss: es kann entweder das *Document Object Model* (DOM) als Repräsentation verwendet werden oder man benutzt eine eigene Datenstruktur, die den Baum repräsentieren soll. Beim ersten Fall (DOM) erzeugt der XML Parser selbst den Baum im Speicher und dieser wird vom XSLT Prozessor verwendet. Es ist also nur ein Schritt notwendig bevor der Prozessor mit seiner eigentlichen Arbeit beginnen kann. Der zweite Fall – eine eigene Datenstruktur – erfordert zwei Arbeitsschritte. Als XML Parser benutzt man beispielsweise hier einen SAX Parser und die Ereignisse, die dieser Parser auslöst, dienen als Eingabe für eine zweite Komponente, die daraus die Baumrepräsentation aufbaut. Der XSLT Prozessor SAXON ist, wie später noch gezeigt wird, ein Vertreter der letzteren „Gattung“. (Bemerkung: Da das XML Dokument, sowie das Stylesheet Dokument beides XML Dokumente sind, werden für beide Dokumente die gleichen Arbeitsabläufe beim Einlesen verwendet.)

Als Ausgabe erzeugt der XSLT Prozessor wieder eine Baumrepräsentation – diesmal vom Ergebnisbaum (result tree). Dieser Baum wird dann später serialisiert und kann dann als ein beliebiges Textdokument (XML, HTML, Plaintext, RTF, XSL-FO Designstruktur, etc.) ausgegeben werden.

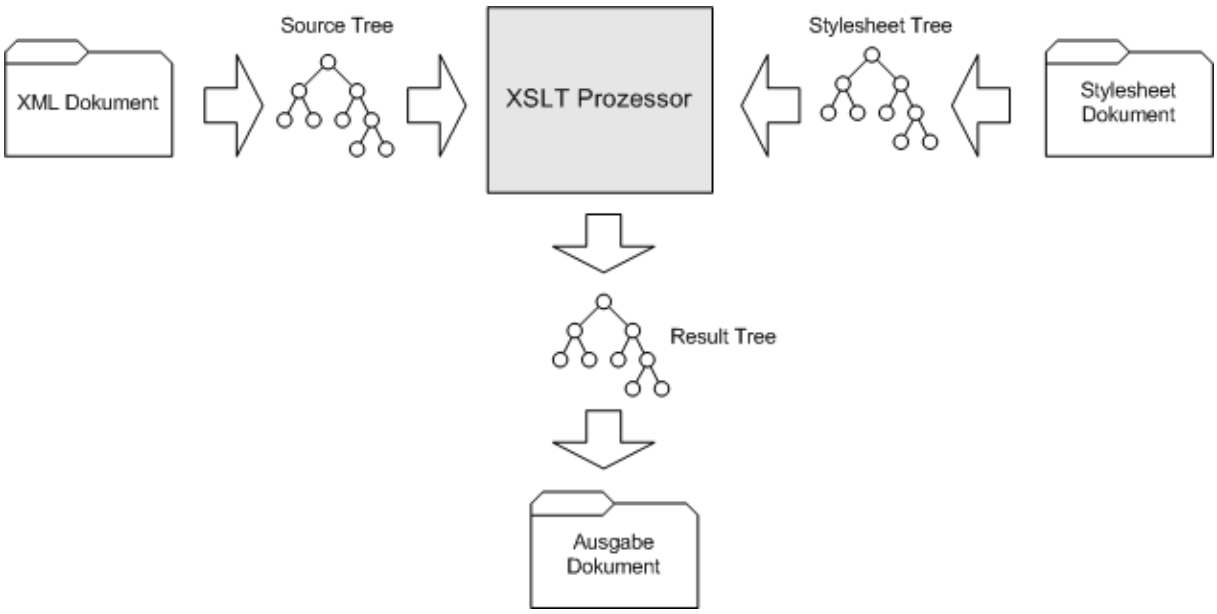

Abbildung 1: XML Verarbeitung mittels eines XSLT Prozessors

3.1 Transformationsablauf

In der Einführung wurde gezeigt, welche Eingaben ein XSLT Prozessor benötigt und wie diese generiert werden. Dieses Kapitel soll nun zeigen, was der Prozessor mit diesen Eingabebäumen anfängt und wie die eigentlichen Transformationen durchgeführt werden. Die meisten Informationen wurden größtenteils aus [Microsoft-03b] entnommen und beziehen sich daher auf dem XSLT Prozessor MSXML, aber die darin beschriebenen Abläufe treffen auf den Großteil der XSLT Prozessoren zu.

Dem XSLT Prozessor stehen dank dem XML Parser nun zwei Bäume im Speicher zur Verfügung: einmal der Stylesheet-Baum und einmal der XML Quellbaum. Die Knoten im Stylesheet-Baum enthalten die Template-Regeln, also die Formatierungsanweisungen, die später benötigt werden. Der XML Quellbaum enthält als Knoten ganz normal die Elemente und Attribute usw., die im XML Dokument angegeben wurden.

Der Prozessor beginnt nun seine Arbeit an dem Wurzelknoten des XML Quellbaumes. Dazu fügt er im Ergebnisbaum, der bei der Abarbeitung generiert wird, einen Platzhalterknoten (Einfügezeiger) ein, an dessen Stelle später die, aus der gewählten Template-Regel, erzeugten Inhalte eingefügt werden [Brügge-03]. Das Auswählen einer passenden Template-Regel aus dem Stylesheet-Baum für jeden gerade zu bearbeitenden Knoten, kann noch einige Konflikte nach sich ziehen, die vom Prozessor während der Transformation eigenständig aufgelöst werden müssen.

Konflikte bei der Transformation

Mehrere Template-Regeln passen auf einen Knoten:

Ein möglicher Konflikt bei der Transformation durch einen XSLT Prozessor ist es, wenn auf einen Knoten im XML Quellbaum mehrere Template-Regeln passen. Hier muss der Prozessor entscheiden, in welcher Reihenfolge diese Regeln ausgeführt werden. Meistens werden dazu Prioritäten an die einzelnen Regeln vergeben, welche dann die Reihenfolge eindeutig festlegen.

MSXML beispielsweise definiert es allgemein so, dass der Template-Regel, die das speziellste Matching aufweist, die höchste Priorität zugewiesen wird. Eine Konfliktauflösung erfolgt dadurch folgendermaßen:

XSLT Stylesheets können weitere XSLT Stylesheets importieren (`xsl:import`). Die Template-Regeln, die sich in den importierten Stylesheets befinden, genießen eine höhere Priorität als normale Regeln. Falls mehrere Import-Anweisungen ausgeführt werden, haben die Stylesheets bzw. Template-Regeln, die zuletzt importiert werden, die höchste Priorität. Ist nach diesem Schritt immer noch ein Konflikt vorhanden, d.h. es gibt immer noch Template-Regeln, bei denen die Ausführungsreihenfolge unklar ist, dann wird noch eine zusätzliche Prioritätenvergabe anhand der Muster der Template-Regeln durchgeführt. Beispielsweise erhält eine Regel, die ein Wildcard-Matching (*) beinhaltet, eine niedrigere Priorität als eine Regel, die auf einen Element- oder Attributnamen „matched“. Sollte nach diesem Schritt immer noch ein Konflikt existieren, dann wird die letzte vorkommende Regel als erstes ausgeführt.

Bemerkung: Regeln, die eine Oder-Verknüpfung (|) in ihrem Matching vornehmen, werden vom Prozessor so betrachtet als wären es einzelne Template-Regeln.

Auf einen Knoten passen keine Template-Regeln:
Falls auf einen, im XML Quellbaum befindlichen, Knoten keine passenden Template-Regeln im Stylesheet-Baum gefunden werden, dann wendet der XSLT Prozessor so genannte Built-In-Rules an. Diese Regeln werden vom Prozessor selbst definiert und regeln das Verhalten, wenn ein solcher Konflikt auftritt.

Eine übliche Built-In Regel ist es beispielsweise den Inhalt von Text- und Attributknoten in den Ergebnisbaum zu schreiben oder Kommentarknoten einfach zu ignorieren, so dass sie später nicht im result tree auftreten.

3.2 Anwendungsbereiche

Für XSLT Prozessoren gibt es vielfältige Anwendungsmöglichkeiten, wobei die am häufigsten genutzte sicherlich die Verwendung in einem Webbrowser ist. Das ist vor allen Dingen der starken Verbreitung des Microsoft Internet Explorers zu verdanken, da dieser einen integrierten XSLT Prozessor besitzt – den MSXML Prozessor. Aber auch andere Browser nutzen eingebaute XSLT Prozessoren zur Transformationen (z.B. Mozilla).

Ein weiterer Anwendungsbereich ist das Nutzen von vorhandenen XSLT Prozessoren in selbst programmierten Programmen. Die meisten bekannten Prozessoren (z.B. SAXON, Xalan, MSXML) stellen Programmbibliotheken zur Verfügung, die vom Entwickler genutzt werden können.

Web- und Applikations-Server wie z.B. Cocoon[2] aus dem Apache Projekt nutzen intern XSLT Prozessoren für die XML/XSLT-Verarbeitung. Cocoon nutzt beispielsweise als Standardprozessor den Xalan XSLT Prozessor, welcher ebenfalls dem Apache Projekt entsprungen ist und in einem extra Kapitel noch ausführlicher erwähnt wird.

Eine weitere Anwendungsmöglichkeit eines XSLT Prozessors ist die Ausführung als stand-alone Anwendung. Dieses ist besonders zu Testzwecken sinnvoll, wenn man z.B. ein XSLT Stylesheet auf korrekte Funktionalität überprüfen will. Die meisten Prozessoren bieten dafür Ausführungsmöglichkeiten auf Kommandozeilenebene an, wie man im folgenden Beispiel sehen kann (Xalan):

```
java org.apache.xalan.xslt.Process
    -IN   data.xml
    -XSL  style.xsl
    -OUT  result.html
```

3.3 Xalan

Xalan ist ein Open Source XSLT-Prozessor der im Rahmen des Apache XML Projektes[3] entstanden ist und vollständig die XSLT 1.0 und XPath 1.0 Spezifikation des World Wide Web Consortiums (W3C) implementiert. Das heißt, dass Xalan zwei der insgesamt drei Bestandteile von XSL umsetzt. Der dritte Bestandteil, *XSL Formatting Objects* wird vom Xalan-Prozessor nicht behandelt und man ist hierbei zusätzlich auf andere Prozessoren angewiesen, wie beispielsweise XML FOP, welcher ebenfalls ein Bestandteil des Apache

[2] http://xml.apache.org/cocoon/index.html
[3] Mehr Informationen unter http://xml.apache.org/.

XML Projektes ist. Xalan existiert in einer Java- (Xalan-J) sowie in einer C++-Version (Xalan-C++) und nutzt als XML Parser Xerces[4], einen Parser aus dem Apache Projekt.

Der Xalan XSLT Prozessor ist eine Implementierung der *Transformation API for XML* (TrAX), welche Bestandteil der *Java API for XML Processing* (JAXP) ist. Xalan kann seit der Version 2.5 auf zwei unterschiedliche Art und Weisen verwendet werden: erstens als ganz normaler interpretierender XSLT Prozessor (*Xalan Interpretive Processor*) oder zweitens als Prozessor, der mit vorkompilierten Stylesheets arbeitet (*Xalan Compiled Processor*). Das letztere ist neu als Funktionalität zum Prozessor hinzugekommen, nachdem Sun seinen eigenen XSLT Prozessor XSLTC dem Apache Project zur Verfügung gestellt hat.

Beide Arten von Prozessoren arbeiten intern mit einer eigenen Baumrepräsentation – dem *Document Table Model* DTM. DTM wurde extra auf die XSLT und XPath Implementierungen zugeschnitten und ist dadurch effizienter in der Verarbeitung als beispielsweise das allgemeine *Document Object Model* DOM. Ein Hauptunterschied zwischen DOM und DTM liegt darin, dass das DTM Modell ein read-only Model ist, im Gegensatz zu DOM, bei dem man die Möglichkeit hat, Knoten neu zu erzeugen, oder Daten in das Modell zu schreiben

Der Xalan Compiled Processor XSLTC ist eine weitere Besonderheit der neuen Xalan Versionen [Sun-02]. Er bietet die komfortable Möglichkeit, XSL Stylesheets in Java Bytecode zu kompilieren (*Translets*) und diese Translets auf XML Dokumente anzuwenden. Der Vorteil ist klar ersichtlich: es ist nun möglich, nur einmal ein XSL Stylesheet zu verarbeiten und dann mehrfach auf XML Dokumente anzuwenden. Je mehr XML Dokumente mit dem gleichen Stylesheet formatiert werden sollen, desto mehr lohnt sich die Anwendung von XSLTC.

Die Kompilierung eines Stylesheets kann auch auf Kommandozeilenebene erfolgen und man nutzt danach nur noch das fertige Translet in seiner Applikation. Das folgende Beispiel zeigt wie man ein Translet erzeugen kann.

```
java org.apache.xalan.xsltc.cmdline.Compile
   -o    meinTranslet
         meinStylesheet.xsl
```

3.4 SAXON

SAXON ist ein von Michael Kay[5] entwickeltes Paket zum Verarbeiten von XML Dokumenten. Es enthält als wichtigsten Bestandteil einen XSLT Prozessor und zusätzlich noch eine Java-Bibliothek, die es erlaubt, die XSLT Funktionalitäten in Java-Applikationen zu verwenden, und den Vorteil einer höheren Programmiersprache – beispielsweise für Datenbankzugriffe – zu nutzen.

SAXON existiert zurzeit in der Version 7.5, welche die meisten Bestandteile des XSLT 2.0 und XPath 2.0 Arbeitsentwurfes (*working draft*) des W3C schon umsetzt, sich aber noch in einer Entwicklungs- und Experimentierphase befindet. Aus diesem Grund wird auch die stabilere Version 6.5.2 empfohlen, die komplett die Version 1.0 von XSLT und XPath unterstützt und genauso wie die Version 7.5 eine Menge an nützlichen SAXON spezifischen Erweiterungen für XSLT enthält. Zudem werden einige interessante Funktionalitäten aus dem Arbeitsentwurf XSLT 1.1 in SAXON umgesetzt. Darunter fällt beispielsweise die Möglichkeit mittels des Elementes `xsl:document` mehrere Ausgabedokumente zu erzeugen

[4] http://xml.apache.org/xerces2-j/index.html
[5] http://homepage.ntlworld.com/michael.h.kay/

(Bemerkung: SAXON war sogar der erste Prozessor, der diese Funktionalität umgesetzt hat), oder mittels `xml:base` die Basis URI des Dokumentes festzulegen.

Als XML Parser benutzt SAXON intern eine minimal überarbeitete Version des Open Source Parsers Ælfred[6], welche aber jederzeit durch einen anderen SAX kompatiblen Parser ersetzt werden kann.

Wie oben schon erwähnt wurde, enthält SAXON eine Reihe von nützlichen XSLT Erweiterungen – sog. *extension elements* bzw. *extension functions*. Diese Erweiterungen umfassen dabei einige Bibliotheken der EXSLT Initiative[7], aber auch aufwendigere Funktionen, die nur bei SAXON zu finden sind.

3.4.1 Aufbau

Durch [Kay-01] gibt Michael Kay einen guten Einblick in den Aufbau seines SAXON XSLT Prozessors. Der Prozessor ist grob in fünf Hauptbestandteile zu unterteilen: den *Tree Constructor*, den *Tree Navigator*, den *Stylesheet Compiler*, den *Stylesheet Interpreter* und den *Outputter*. Abbildung 2 stellt diese Komponenten und ihren Zusammenhang grafisch dar.

[6] http://saxon.sourceforge.net/aelfred.html

[7] Die EXSLT Initiative – erreichbar unter http://www.exslt.org/ – stellt einige Erweiterungs-Bibliotheken für XSLT zur Verfügung.

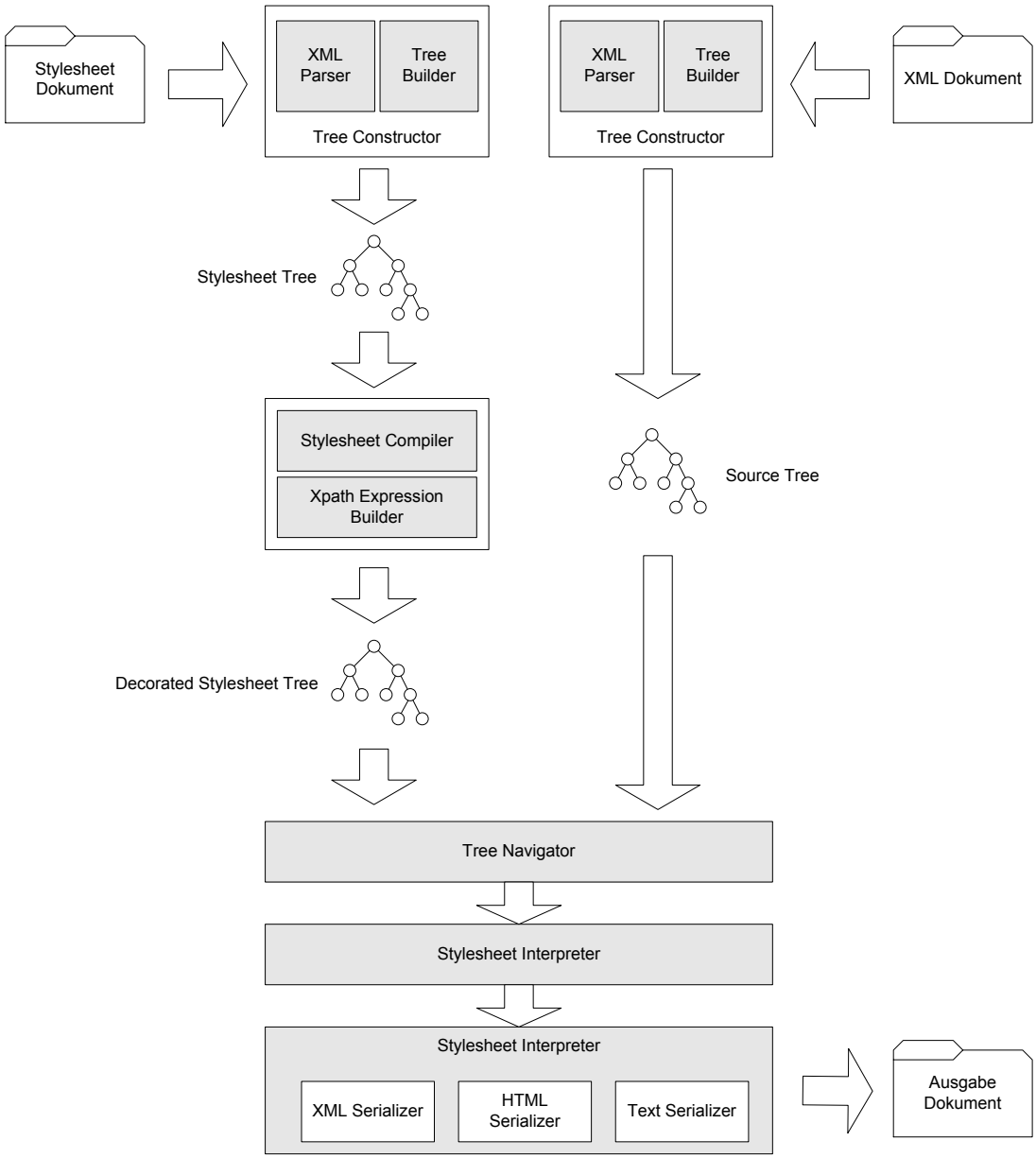

Abbildung 2: SAXON Architektur [Kay-01]

Tree Constructor

Der Tree Constructor unterteilt sich intern in zwei Komponenten – den *XML Parser* und den *Tree Builder*. Der Tree Constructor ist dafür zuständig, aus einem XML Eingabedokument im Arbeitsspeicher eine Baumrepräsentation für die weitere Verwendung im XSLT Prozessor aufzubauen.

In SAXON wird der Tree Constructor bei zwei unterschiedlichen Aufgaben eingesetzt. Als erstes wird der Constructor zum Aufbau einer Baumstruktur für das XML Quelldokument genutzt und als zweites wird ebenfalls das Stylesheet Dokument im Speicher in einer Baumstruktur abgelegt.

Einer der Tree Contructor Komponenten ist ein XML Parser, welcher bei SAXON, wie in der Übersicht schon erwähnt wurde, standardmäßig der Open Source Parser Ælfred ist. Dieser Parser benutzt SAX (Simple API vor XML) zum Parsen eines XML Dokumentes. Das bedeutet, dass er nach dem so genannten „ereignisorientierten Push-Modell“ arbeitet und dabei der Anwendung noch während des Lesens des Quelldokumentes Informationen über das

Dokument zusendet. Er informiert also die Anwendung beispielsweise über ein öffnendes oder schließendes Element, über Zeichendaten usw. Die „Anwendung“ ist im Falle von SAXON die zweite Komponente des Tree Constructor – der Tree Builder. Er baut aus den Ereignissen, die Ælfred sendet, eine Repräsentation des XML Dokumentes im Arbeitsspeicher auf. Da es sich bei Ælfred um einen ganz normalen SAX Parser handelt, also einen, der die SAX API implementiert, kann man in SAXON diesen auch durch jeden anderen SAX konformen Parser (z.B. Xerces, Crimson, etc.) ersetzen.

Tree Navigator

Der Tree Navigator dient dazu, in dem durch den vom Tree Builder erzeugten Baum, auf bestimmte Knoten zugreifen zu können. Die Repräsentation des Baumes, der von SAXON verwendet wird, ist kennzeichnend für diesen XSLT Prozessor und unterscheidet sich von den meisten Baumrepräsentationen, die sonst von anderen Prozessoren verwendet werden.

Die meisten anderen XSLT Prozessoren nutzen das sog. *Document Object Model* (DOM) zur Erstellung eines Baumes im Speicher. Da das DOM Modell ein vom W3C verabschiedeter Standard[8] ist, entsteht der Vorteil, dass diese XSLT Prozessoren mit DOM Bäumen von anderen Anwendungen arbeiten und selbst erzeugte DOM Bäume anderen Anwendungen zur Verfügung stellen können.

Michael Kay hat sich bewusst gegen die Verwendung des DOM Modells bei der Entwicklung seines XSLT Prozessors entschieden, da die Performanceeinbußen seiner Meinung nach zu hoch waren. Diese Einbußen kommen vor allem aus dem Grund zustande, da sich das DOM Modell in einigen Punkten vom XPath Modell unterscheidet, welches vom XSLT Prozessor aber benötigt wird. Das DOM Modell enthält beispielsweise zusätzliche Informationen (z.B. über Entity Knoten), welche vom XPath Modell nicht benötigt werden. Zudem erlaubt DOM, dass Änderungen an Bäumen auf der Stelle durchgeführt werden können, wobei das XSLT Modell vorsieht, dass Bäume nur sequentiell geschrieben werden. Dieses sequentielle Schreiben wurde dann auch in den SAXON Bäumen umgesetzt. Zusätzlich wurde eine einfachere Synchronisationslogik in dem SAXON Baummodell implementiert, da das DOM Modell viel überflüssigen Code zur Synchronisation von Multithread-Zugriffen beinhaltet, der eigentlich nicht benötigt wird, da nach dem XSLT Modell nur einmal alles geschrieben wird und danach nur noch lesende Zugriffe stattfinden („*write-once, read-many*“ [Kay-01]).

Stylesheet Compiler

Der Stylesheet Compiler erhält als Eingabe die, in einem vorherigen Schritt vom Tree Constructor erzeugte, Repräsentation des Stylesheet-Baumes. Als Ausgabe erzeugt der Compiler ebenfalls einen Baum, nur dass es sich hierbei um einen Entscheidungsbaum (*decision tree*) handelt, welcher später dem Stylesheet Interpreter als Eingabe dienen soll.

Dieser Entscheidungsbaum wurde vom Compiler insoweit vorbereitet, dass in dem Baum schon alle XPath Ausdrücke vollständig validiert und geparst wurden und es wurden ebenfalls alle Querverweise aufgelöst. Dieser überarbeitete Stylesheet-Baum erlaubt es dem Interpreter später, zu jedem Knoten schneller die passende Templateregel zu finden. Ein Vergleich von jedem Knoten mit jeder vorhandenen Templateregel wäre laut Kay ziemlich ineffizient gewesen [Kay-01].

Der Stylesheet-Baum, der vom Compiler erzeugt wird, kann nicht persistent gespeichert werden, sondern verbleibt immer nur im Arbeitsspeicher. Der Grund hierfür ist, dass ein

[8] http://www.w3.org/TR/DOM-Level-2-Core/

erneutes Kompilieren des Original Stylesheet-Baumes erheblich weniger Zeit benötigt, als ein Einlesen von einem Festspeicher.

Stylesheet Interpreter

Der Stylesheet Interpreter bildet den eigentlichen Kern des SAXON Prozessors, denn hier wird die eigentliche Transformation durchgeführt. Als Eingabe nimmt der Interpreter den XML Quellbaum und den vom Compiler vorbereiteten Stylesheet-Entscheidungsbaum entgegen. Begonnen wird die Transformation an der Wurzel des XML Baumes, auf welche die erste gefundene Templateregel angewendet wird. Als Ergebnis liefert der Interpreter einen Ergebnisbaum (*result tree*).

In [Kay-01] geht Michael Kay sehr auf die Implementierungsdetails des Stylesheet-Interpreters ein. In dieser Ausarbeitung, die nur einen kleinen Überblick über verschiedene XSLT-Prozessoren liefern soll, wird nicht weiter darauf eingegangen, da das den Rahmen sprengen würde.

Outputter

Der Outputter ist die letzte Komponente im SAXON Prozessablauf und ist für die Kontrolle des Ausgabeprozesses des result trees zuständig. Im Unterschied zu anderen XSLT Prozessoren wird der Ergebnisbaum nicht im Arbeitsspeicher komplett erzeugt und dann serialisiert. Dadurch, dass die Knoten schon in Dokumentenreihenfolge vorliegen, können sie auch gleich in dieser Reihenfolge serialisiert werden. Dass heißt, dass schon bei der Erzeugung des result trees die Elemente geschrieben werden können. In der Praxis existiert aber nicht nur ein Ergebnisbaum, sondern es existieren mehrere, die auf einer Stack abgelegt werden. Eine Ursache könnte zum Beispiel das Element `xsl:message` sein, welches eine Nachricht in die Ausgabe schreibt (z.B. zu Debugging-Zwecken) und man dadurch den normalen result tree verlassen muss.

Die Ausgabe des Outputter erfolgt mittels eines Serialisierers in eine Datei. Dieser Serialisierer existiert für die Formate XML, HTML und Klartext, genauso wie es in der XSLT Empfehlung vom W3C definiert ist. SAXON erlaubt ebenfalls, den result tree an eigenen Programmcode zur weiteren Verarbeitung weiterzuleiten.

3.5 MSXML

MSXML steht für *Microsoft XML Core Services* und ist ein Software Development Kit (SDK), welches Microsoft für die Verarbeitung von XML Dokumenten zur Verfügung stellt. Früher bestand MSXML nur aus einem reinen XML Parser und war deswegen auch unter der Bezeichnung *Microsoft XML Parser* bekannt. Ab der aktuellen Version 4.0 wurde der lang erwartete Schritt zum SDK durchgeführt, da MSXML mittlerweile nicht mehr nur ein eine reinen XML Parser enthielt, sondern auch eine Reihe anderer XML Werkzeuge – beispielsweise einen XSLT Prozessor.

Im Gegensatz zu früheren Versionen von MSXML hält sich der XSLT Prozessor der aktuellen Version 4.0 an die Standards des W3C, XSLT 1.0 und XPath 1.0. Eine Kompatibilität zum XSLT 2.0 Standard ist für MSXML aber nicht vorgesehen, da sich bei Microsoft in Zukunft alle Anstrengungen im Bereich der XML Verarbeitung auf das .NET XML Framework System.Xml konzentrieren werden [Microsoft-03a].

3.5.1 Ablauf

Leider hält sich Microsoft sehr mit der Beschreibung der Architektur des XSLT Prozessors zurück und bietet nicht so viele interne Informationen wie beispielsweise SAXON. Den Ablauf einer XSL Transformationen mit dem MSXML XSLT Prozessor kann man sich aber trotzdem anschaulich vorstellen.

Als Eingabe erhält der Prozessor eine DOM Repräsentation (Document Object Model) des Stylesheet Dokumentes sowie des XML Quelldokumentes. Der DOM Baum wird dabei durch einige aufeinander folgende Verarbeitungsschritte aufgebaut. Zuerst liest der MSXML Parser das XML Dokument ein und übergibt es danach dem MS DOM Builder. Beim Einlesen des Dokumentes wird zusätzlich überprüft, ob das Dokument wohlgeformt und gültig ist. Der DOM Builder entfernt danach unnötige Leerzeichen (*whitespaces*) aus dem Dokument und es wird im Speicher ein Baum aufgebaut, welcher der Anwendung danach zur Verfügung steht. Abbildung 3 stellt den Ablauf zum Aufbau eines DOM Baumes grafisch dar.

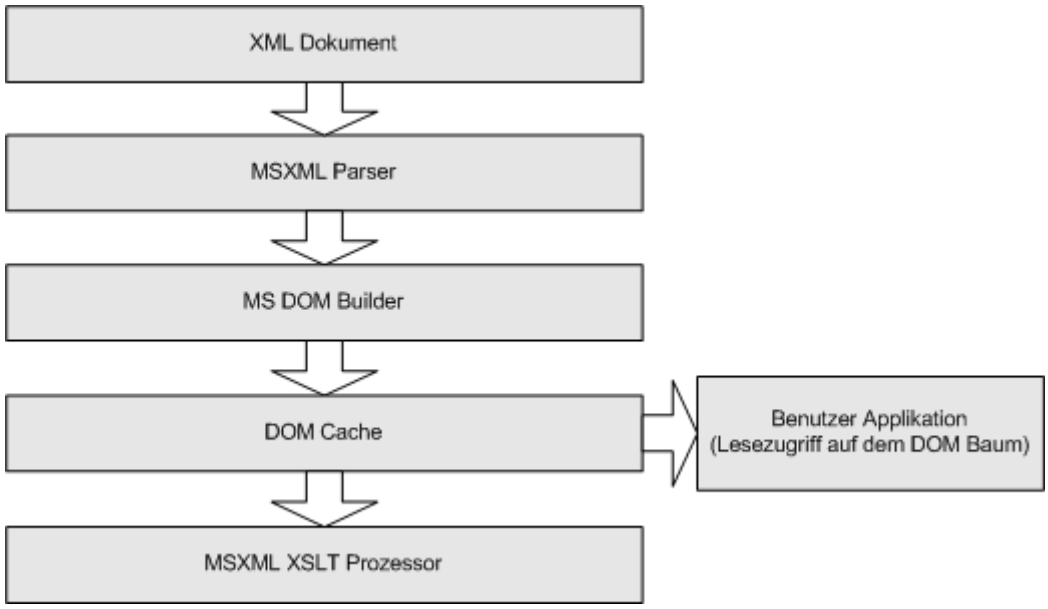

Abbildung 3: MSXML DOM Baum Erzeugung

Nachdem dem XSLT Prozessor nun beide Dokumente, XSLT und XML, als DOM Baum zur Verfügung stehen, kann die eigentliche Transformation erfolgen, die als Ergebnis den Ergebnisbaum (*result tree*) hat. Bei dieser Transformation sucht der Prozessor in dem XML DOM Baum nach Knoten, die zu den Mustern passen, welche in den Template Regeln im XSLT DOM Baum definiert wurden. Danach wird aus den entstehenden Transformationen der Ergebnisbaum konstruiert, der beim MSXML XSLT Prozessor auch in verschiedensten Formaten, wie beispielsweise XML, HTML oder RTF (*rich text Format*), vorliegen kann.

3.6 Weitere Prozessoren

Neben den oben besprochenen drei bekannten XSLT Prozessoren SAXON, Xalan und MSXML gibt es natürlich noch eine Reihe weiterer Prozessoren, die in dieser Ausarbeitung aber nicht alle besprochen werden können. Die folgende Liste gibt aber noch einen kleinen Überblick über einige dieser Prozessoren:

3.6.1 XT

Ein von James Clark entwickelter auf Java basierter Open Source Prozessor. Er ist vor allen Dingen durch seine hohe Verarbeitungsgeschwindigkeit bekannt, unterstützt aber den XSLT 1.0 Standard nicht vollständig.
Mehr Informationen: http://www.blnz.com/xt/index.html

3.6.2 XSLTC

XSLTC wurde von Sun entwickelt, aber dann dem Apache Projekt zur Verfügung gestellt, wo er beim Xalan Prozessor weiterverwendet wurde.
Mehr Informationen: http://xml.apache.org/xalan-j/xsltc_usage.html

3.6.3 XSL:P

Einer der ersten XSLT Prozessoren, welcher heute aber nicht mehr weiterentwickelt wird. Teile des in Java implementierten Prozessors wurden nach C++ portiert und in TransforMiiX verwendet.
Mehr Informationen: http://xslp.kvisco.com/

3.6.4 Sablotron

Sablotron wird von der Firma Ginger Alliance unter einer Open Source Lizenz entwickelt und ist ein XML Toolkit, welches u.a. einen XSLT Prozessor enthält. Sablotron basiert auf C++.
Mehr Informationen: http://www.gingerall.com/charlie/ga/xml/p_sab.xml

3.6.5 TransforMiiX

Open Source C++ XSLT Prozessor, welcher u.a. im Mozilla Projekt[9] als XSLT Prozessor eingesetzt wird.
Mehr Informationen: http://www.mitre.org/mii/techproducts/transform.htm

3.6.6 FastXML

FastXML ist im Rahmen der Master-Abschlussarbeit von Helena Kupková entstanden und ist kompatibel zum MSXML Prozessor. Er soll ca. fünfmal schneller sein, als sein Vorbild.
Mehr Informationen: http://www.geocities.com/fastxml/

3.6.7 Oracle XDK

Oracle XDK ist ein XML Developer Kit und beinhaltet neben vielen anderen Komponenten auch einen XSLT Prozessor.
Mehr Informationen: http://otn.oracle.com/tech/xml/xdk/content.html

3.6.8 iXSLT

Kommerzieller XSLT Prozessor der Firma infoteria.
http://www.infoteria.com/products/product_page.jsp?id=/product/product_1.xml

[9] http://www.mozilla.org/

4 Zusammenfassung

In den vorherigen Kapiteln wurde beschrieben wie ein XSLT Prozessor arbeitet, welche XSLT Prozessoren erhältlich sind und in welchen wichtigen Punkten sich die einzelnen Prozessoren unterscheiden. Ein oft gesehener Unterschied ist sicherlich die interne Repräsentation des XML Quellbaumes und des XSL Stylesheet-Baumes. So wurden die Prozessoren SAXON und Xalan vorgestellt und man hat gesehen, dass beide eine eigene auf die jeweilige Implementierung zugeschnittene Repräsentation verwenden. Dieser „Alleingang“ in Sachen Baumrepräsentation hat den großen Vorteil, dass die Verarbeitungsgeschwindigkeit zunimmt, da man die Struktur auf die eigene Implementierung optimieren kann. Aber es bringt natürlich auch einen Nachteil mit sich und zwar der Verlust von Kompatibilität.

Würden alle Prozessoren ein einheitliches, standardisiertes Modell wie beispielsweise das Document Object Model (DOM) verwenden, wäre eine gute Kompatibilität zwischen den einzelnen Prozessoren gewährleistet. MSXML z.B. hält sich an das DOM Modell.

Man muss aber auch sagen, dass fast alle Prozessoren mit DOM umgehen können und Konvertierungen zwischen dem eigenen Modell und DOM durchführen können.

Als Anwender stellt man sich nun die Frage: Welchen XSLT Prozessor soll ich verwenden? Dieses muss man für jeden Anwendungsfall neu entscheiden und sollte sich vorher einige Fragen stellen:

Welche XSLT und XPath Standards sollen unterstützt werden?
Für viele Anwendungen reicht eine teilweise Unterstützung der W3C Standards aus und so muss man nicht unbedingt auf Prozessoren wie SAXON oder Xalan zurückgreifen, sondern kann eine schnellere Variante wie beispielsweise XT nehmen, die sich nicht komplett an die Standards halten.

Wo möchte ich den Prozessor einsetzen?
Wenn man z.B. den Prozessor nur zum Testen von Stylesheets braucht und nicht programmieren möchte, dann benötigt man einen Prozessor, der auch stand-alone ausgeführt werden kann. Einige Prozessoren unterstützen dieses nicht und fallen dann automatisch weg.

Die Verwendung des Prozessors in einem eigenen Programm setzt voraus, dass der Prozessor mit Bibliotheken ausgestattet ist, die man nutzen kann.

Wie kompatibel soll das Baummodell sein?
Falls man direkt auf der Baumrepräsentation des Prozessors arbeiten möchte, dann bietet es sich an, einen Prozessor zu verwenden, der sich an Standards wie beispielsweise DOM hält. Dadurch hat man den Vorteil, dass man sich nicht extra in ein neues Baummodell einarbeiten muss.

Wie schnell soll der Prozessor arbeiten?
Wenn die Verarbeitungsgeschwindigkeit des Prozessors entscheidend ist, dann sollte man sich mehrfach überlegen, ob man einen Prozessor der normalen Sorte nimmt (SAXON, Xalan, …) oder doch lieber einer extra optimierten Variante wie XT oder FastXML den Vorzug gibt, da die letzteren um einiges schneller sind [Kuznetsov-01]

Abbildungsverzeichnis

Quellenverzeichnis

[Brügge-03]
Brüggemann-Klein, Anne: Prozessmodell von XSLT, Vorlesung Elektronisches Publizieren, TU München, 2003
http://www11.in.tum.de/~brueggem/Vorlesungen/epWS2002/Folien/K.XSLT/XSLT.ppt

[Clark-99]
Clark, J., DeRose, S.: XML Path Language (XPath) Version 1.0, W3C, 1999
http://www.w3.org/TR/xpath.

[Harold-03]
Harold, E., Means, S.: XML In A Nutshell, O'Reilly, 2003

[Kay-01]
Kay, H. Michael: Saxon: Anatomy of an XSLT Processor, developerWorks, 2001
http://www-106.ibm.com/developerworks/xml/library/x-xslt2/

[Kuznetsov-01]
Kuznetsov, E., Dolph, C.: XSLT Processor Benchmarks, XML.com, 2001
http://www.xml.com/pub/a/2001/03/28/xsltmark/

[Microsoft-03a]
Microsoft: Frequently asked questions about XSLT, Microsoft Corporation, 2003
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/xslt_starter_8f3o.asp

[Mircosoft-03b]
Microsoft: The XSLT Processor, Microsoft Corporation, 2003
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/xslt_processornew_3rg3.asp

[Sun-02]
Sun Microsystems: Apache Xalan XSLT Compiler, Sun Microsystems, 2002
http://wwws.sun.com/software/xml/developers/xsltc/xsltc_webpack.html