

Studiengang : Softwaretechnik
Prüfer : Prof. Dr. B. Mitschang
Betreuer : Uwe Heinkel
begonnen am : 07.01.2004
beendet am : 06.07.2004
CR-Klassifikation : D.2.8, D.4.8, H.3.3

Fachstudie Nr. 32

Bewertung von XSLT - Prozessoren

Manuel Früh Philipp Häuser Michael Marks

Institut für Parallele und
Verteilte Systeme (IPVS)
Abteilung Anwendersoftware (AS)
Universität Stuttgart
70550 Stuttgart

Inhaltsverzeichnis

1	Einleitung	6
1.1	Motivation	6
1.2	Struktur des Dokumentes	6
1.3	Rahmenbedingungen und Ablauf der Fachstudie	7
2	Einführung in das Themengebiet	8
2.1	SFB 467 - Wandlungsfähige Unternehmensstrukturen	8
2.2	XML	8
2.3	SAX	9
2.4	DOM	9
2.5	XSL	10
2.5.1	XPath	10
2.5.2	XSLT	11
2.5.3	EXSLT	12
2.5.4	XSL-FO	12
2.5.5	Die Transformation von XML-Dokumenten	12
2.6	XSLTMark	14
3	Liste verfügbarer XSLT Prozessoren	16
3.1	jd.xslt	16
3.2	libxslt	17
3.3	MSXML	17
3.4	Oracle XDK	17
3.5	Sablotron	18
3.6	Saxon	18
3.7	Transformiix	18
3.8	Xalan	19
3.9	XSLTC	19
3.10	XSL:P	19
3.11	XT	20

4	Kriterienkatalog	21
4.1	Gewichtung	21
4.2	Hauptkriterien	21
4.3	Dokumentation	22
4.4	Integrierbarkeit (Implementierungssprache)	22
4.5	Performanz	22
4.6	Testfälle	23
4.7	Normalisierung	23
5	Beschreibung der Testfälle	25
5.1	AggregateHTML und AggregateXML	25
5.2	AttrToElem	27
5.3	CreateElemSubset	28
5.4	FilterElement	29
5.5	IncludeDocs	29
5.6	RenameElement	31
5.7	ReplaceElementValue	33
6	Die Testumgebung	34
7	Evaluationsergebnisse	34
8	Auswertung der Ergebnisse	36
8.1	Übersicht der Durchsätze	36
8.2	Auswertung der Performanz bei großen Testdateien	37
8.2.1	Zusammenfassung der Ergebnisse bei großen Testdateien . . .	38
8.3	Graphischer Vergleich der gemessenen Performanz bei großen Dateien	38
8.4	Auswertung der Performanz bei kleinen Testdateien	41
8.4.1	Zusammenfassung der Ergebnisse bei kleinen Testdateien . .	41
8.5	Graphischer Vergleich der gemessenen Performanz bei kleinen Dateien	42
8.6	Markante Unterschiede der gemessenen Performanz bei großen und kleinen Dateien	42

8.7	Zusammenfassung der Performanz-Messungen für große und kleine Testdateien	45
8.8	Auswertung der Integrierbarkeit	46
8.9	Auswertung der Dokumentation	47
8.10	Gesamtauswertung	48
9	Zusammenfassung	49
9.1	Interpretation der Gesamtauswertung	49
9.2	Empfehlung auf Basis des Ergebnisses der Evaluation	49
A	Aufgabenstellung	53
A.1	Hintergrund	53
A.2	Aufgabe	53
A.3	Teilaufgaben	53
B	Messwerttabellen	54
B.1	jdxslt	54
B.2	msxslt	54
B.3	oracle-xdk	54
B.4	sablotron	55
B.5	saxon	55
B.6	xalan-c	55
B.7	xalan-j	56
B.8	xsltc	56
B.9	xt	56
C	Auswertungstabellen	57
C.1	jdxslt (große Dateien)	57
C.2	msxml (große Dateien)	57
C.3	oracle-xdk (große Dateien)	57
C.4	sablotron (große Dateien)	58
C.5	saxon (große Dateien)	58
C.6	xalan-c (große Dateien)	58

C.7 xalan-j (große Dateien)	59
C.8 xsltc (große Dateien)	59
C.9 xt (große Dateien)	59
C.10 jdxslt (kleine Dateien)	60
C.11 msxml (kleine Dateien)	60
C.12 oracle-xdk (kleine Dateien)	60
C.13 sablotron (kleine Dateien)	61
C.14 saxon (kleine Dateien)	61
C.15 xalan-c (kleine Dateien)	61
C.16 xalan-j (kleine Dateien)	62
C.17 xsltc (kleine Dateien)	62
C.18 xt (kleine Dateien)	62

D Begriffslexikon **63**

Abbildungsverzeichnis

1	XSLT Überblick	13
2	Stärken und Schwächen (große Testdateien)	36
3	Stärken und Schwächen (kleine Testdateien)	36
4	Prozentuale (normalisierte) Durchsätze	37
5	Performanzvergleich AggregateHTML	39
6	Performanzvergleich AggregateXML	39
7	Performanzvergleich AttrToElem	39
8	Performanzvergleich FilterElement	40
9	Performanzvergleich IncludeDocs	40
10	Performanzvergleich RenameElement	40
11	Performanzvergleich RenameElemValue	41
12	Performanzvergleich AggregateHTML-small	42
13	Performanzvergleich AggregateXML-small	42
14	Performanzvergleich AttrToElem-small	43
15	Performanzvergleich FilterElement-small	43
16	Performanzvergleich IncludeDocs-small	43
17	Performanzvergleich RenameElement-small	44
18	Performanzvergleich RenameElemValue-small	44
19	Gewichtete Performanzen (große und kleine Dateien zusammen)	46

1 Einleitung

1.1 Motivation

Das Ziel dieser Fachstudie ist es, eine Liste der momentan am Softwaremarkt existierenden XSLT-Prozessoren zu erstellen, die für den Einsatz innerhalb des Projektes „Wandlungsfähige Unternehmensstrukturen“ in Frage kommen. Dabei sollen die Eigenschaften und Unterschiede einer Auswahl von Prozessoren untereinander herausgestellt werden.

Aus der Liste der untersuchten Prozessoren soll mittels eines Kriterienkataloges derjenige XSLT-Prozessor bestimmt werden, der den Anforderungen des oben genannten Projektes am besten entspricht.

1.2 Struktur des Dokumentes

Dieses Dokument ist in die folgende Themenbereiche aufgeteilt:

- **Einführung in das Themengebiet:** Dieses Kapitel führt in das Thema der XML-Transformation ein. Es beschreibt die Grundlagen von XML, SAX, DOM sowie XSL und erläutert darüber hinaus das Zusammenspiel zwischen den verschiedenen Standards. Außerdem wird das Rahmenwerk XSLTMark vorgestellt, mit dem die Prozessoren auf ihre Performanz untersucht wurden.
- **Liste verfügbarer XSLT-Prozessoren:** Dieses Kapitel enthält eine Liste der verfügbaren Prozessoren. Es beschreibt weiterhin die Eigenschaften und Besonderheiten jedes Prozessors.
- **Kriterienkatalog:** Die Beurteilung der Prozessoren erfolgte anhand eines Kriterienkataloges. Das Kapitel beschreibt den aufgestellten Kriterienkatalog und die Vorgehensweise bei der Ergebnisauswertung.
- **Beschreibung der Testfälle:** Für die Bewertung der Effizienz der Prozessoren wird die Performanz bei verschiedenen XSL-Transformationen untersucht. Die verschiedenen Transformationsarten wurden in Klassen zusammengefasst und jede Klasse durch einen geeigneten Testfall abgedeckt. Die Beschreibung der Testfälle ist Inhalt dieses Kapitels.
- **Evaluationsergebnisse und Auswertung:** Dieser Teil der Fachstudie stellt die Ergebnisse des Performanztestes sowie die anschließende Auswertung der absoluten Ergebnisse dar, um sie innerhalb des Kriterienkataloges verwenden zu können. Anhand des Kriterienkataloges werden die einzelnen Prozessoren bewertet und die erreichten Punktzahlen einander gegenübergestellt.
- **Zusammenfassung und Empfehlung:** Die Ergebnisse werden zusammengefasst und der am Besten geeignete Prozessor für die Verwendung innerhalb des

Teilprojektes A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ des Sonderforschungsbereiches 467 „Wandlungsfähige Unternehmensstrukturen“ empfohlen.

1.3 Rahmenbedingungen und Ablauf der Fachstudie

Die Fachstudie entstand im Auftrag und in Zusammenarbeit mit der Abteilung „Anwendersysteme“ der Fakultät Informatik an der Universität Stuttgart. Die beteiligten Personen waren:

- Bearbeiter: Manuel Früh, Philipp Häuser, Michael Marks
- Universität Stuttgart: Prof. Dr. B. Mitschang, Dipl. Inf. Uwe Heinkel

Zu Beginn der Studie erfolgte eine Einarbeitung in die Problemstellung durch das Lesen zahlreicher Literatur- und Onlinequellen zu den Themen XML, SAX, DOM und XSLT. Des weiteren erhielten wir Einblick in das Teilprojekt A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ des Sonderforschungsbereiches 467 „Wandlungsfähige Unternehmensstrukturen“, in dem der XSLT-Prozessor eingesetzt werden soll.

Nach der Einarbeitungsphase stellten wir eine Liste verfügbarer XSLT-Prozessoren auf. Im folgenden Zeitraum suchten wir Kriterien, die vorwiegend darauf abzielten, die Prozessoren in Bezug auf den Aufwand und die Schwierigkeit der Installation, des Dokumentationsumfangs, der Performanz sowie der Einfachheit der Prozessorintegration in das vorhandene Rahmenwerk zu bewerten. Die ausgewählten Bewertungskriterien wurden gemeinsam mit Herrn Heinkel besprochen und gleichzeitig Wege gesucht, wie ein Testen der Performanz durchgeführt werden könnte. Aus den gefunden Kriterien wurde anschließend der Kriterienkatalog erstellt, der die Grundlage für die Bewertung der Prozessoren bildete.

Zum Testen der Performanz der einzelnen Prozessoren verwendeten wir das frei erhältliche Benchmark-Rahmenwerk XLSTMark, für welches wir eigene angepasste Testfälle erstellten.

Die Ergebnisse der Fachstudie und die abgeleitete Empfehlung wurden in einer Abschlußpräsentation vorgetragen und sind in diesem Bericht dokumentiert.

2 Einführung in das Themengebiet

2.1 SFB 467 - Wandlungsfähige Unternehmensstrukturen

Turbulente Unternehmensumfelder sorgen heutzutage dafür, dass Wandlungsfähigkeit für produzierende Unternehmen zunehmend zum Erfolgsfaktor wird. Der Sonderforschungsbereich 467 „Wandlungsfähige Unternehmensstrukturen für variantenreiche Serienproduktion“ befaßt sich mit der Erarbeitung von Modellen, Methoden und Verfahren zur Erhöhung der Wandlungsfähigkeit in produzierenden Unternehmen.

Das Teilprojekt A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ untersucht in Zusammenarbeit mit anderen Teilprojekten konkrete Informationstechnologien, mit denen diese Modelle als Systeme realisiert werden können. Zu den untersuchten Technologien zählen vor allem Middleware (Object Request Broker, Message Queues, XML-basierte Verfahren) und Datenbanktechnologien. Das bedeutet, die Interaktion zwischen Programmen und ihre jeweilige Datenhaltung bilden die Hauptuntersuchungsfelder dieses Teilprojektes.

2.2 XML

XML (eXtensible Markup Language) ist eine Sprache, die vor allem für die Beschreibung logischer Strukturen entwickelt wurde. Ähnlich wie HTML bedient sich XML benannter Elemente. Der XML-Standard definiert allerdings anders als bei HTML keine festen Elementnamen. Es liegt in der Hand des Autors, sich passende Elementnamen für die Darstellung der strukturierten Informationen zu überlegen. Wie ein korrektes XML-Dokument aussieht und welche Markierungen verwendet werden dürfen, bestimmen sogenannte Dokumenttyp-Definitionen (DTD) oder Schema Definitionen (XML-Schemata).

Auf Basis von XML können durch Definition verschiedener Dokumenttypen Standards für den Austausch strukturierter Informationen in den verschiedensten Bereichen entstehen. Webservices verwenden beispielsweise XML-Dokumente, um entfernte Methodenaufrufe durchzuführen. Das Resource Description Framework (RDF) benutzt XML für die Beschreibung von Ressourcen im Internet. Dazu könnte beispielsweise ein kompletter Bücherkatalog eines Online-Buchhändlers zählen. Agentensysteme könnten die nach dem RDF-Standard strukturierten Informationen automatisch auswerten und so dem Internetbenutzer eine einfache inhalts- oder preisbasierte Suche nach bestimmten Büchern anbieten.

Für die Auswertung der im XML-Format vorliegenden Informationen existieren zwei verschiedene Modelle - SAX und DOM. Beide Modelle werden in den anschließenden Kapiteln erklärt. Um den Austausch zwischen verschiedenen XML-Standards zu gewährleisten, ist es oft notwendig, die Struktur beziehungsweise die Darstellung der gespeicherten Informationen anzupassen. Das ist beispielsweise der Fall, wenn Daten in XML-Dokumenten vorliegen und auf Internetseiten visualisiert werden sollen. Für derartige Strukturänderungen wurde die Extensible Stylesheet Language (XSL) entwickelt. XSL ist eine auf XML-basierende Sprache, die die Definition von Trans-

formationsregeln für Elemente in XML-Dokumenten ermöglicht.

2.3 SAX

SAX bedeutet „Simple API for XML“. SAX ermöglicht eine serielle Verarbeitung der Elemente von XML-Dateien. Beim Lesen unterschiedlicher Elemente werden auch unterschiedliche Ereignisse ausgelöst. Für jeden Ereignistyp kann eine eigene Verarbeitung erfolgen. So können Informationen in Knoten, die zu verschiedenen Elementtypen gehören, auch unterschiedlich verarbeitet werden.

Durch die serielle Verarbeitung der Elementknoten kann SAX keine kontextabhängige Bearbeitung der Knoteninhalte ermöglichen. Beim Lesen eines Kindknotens ist der zuvor gelesene Elternknoten nicht mehr bekannt, da SAX nur den aktuell gelesenen Knoten sieht und anders als DOM keine Baumstruktur im Speicher aufbaut. SAX bietet eine schnelle Verarbeitung von XML-Dokumenten, wenn die Dokumente nur seriell gelesen werden sollen und der Kontext der gelesenen Elemente für deren Verarbeitung keine Rolle spielt.

Die SAX-API sieht keinerlei Möglichkeit vor, aus den gelesenen Informationen im Speicher ein Modell aufzubauen und dieses vorzuhalten, um darauf arbeiten zu können. Für in XML definierte Strukturen, die nach dem Einlesen verändert werden sollen, wäre dies aber wünschenswert. Das Document Object Model bietet an dieser Stelle Abhilfe.

2.4 DOM

Das Document Object Model, kurz DOM, geht bei der Bereitstellung der aus XML-Dokumenten gelesenen Informationen einen anderen Weg als die SAX-API. Informationen sind in XML-Dateien hierarchisch strukturiert. Das bedeutet, ein Element hat immer genau ein Eltern-Element, kann aber mehrere Kind-Elemente besitzen. Auf diese Weise ergibt sich eine Baumstruktur.

Bei der Verarbeitung von XML-Dokumenten liest ein DOM-Parser das Dokument zuerst vollständig ein und konstruiert daraus im Speicher einen virtuellen Baum. Auf diesem Informationsbaum kann anschließend gearbeitet werden. Es ist möglich, Elementinhalte und -attribute zu verändern, Elemente in den Baum einzuhängen, sie innerhalb des Baumes zu verschieben oder ganz aus diesem zu löschen. Das Ergebnis kann anschließend wiederum als XML-Dokument gespeichert werden.

DOM ist die richtige Wahl für die Bearbeitung von XML-Dokumenten, wenn es darum geht, die gelesenen Informationen anschließend interaktiv zu bearbeiten oder in andere Strukturen zu transformieren.

2.5 XSL

XSL definiert einen Standard, mit dessen Hilfe Elemente innerhalb von XML- Dokumenten referenziert und manipuliert werden können, um sie in eine andere Struktur zu bringen. Mit Hilfe von XSL können XML-Dateien in HTML, anders strukturierte XML-Dokumente, PDF-Dokumente oder beliebige andere Formate umgewandelt werden. Der XSL-Standard besteht aus drei wichtige Unterkomponenten - XSL-FO, XPath und XSLT.

2.5.1 XPath

XSLT bietet die Möglichkeit XML-Dokumente in andere Formate zu transformieren. Dabei müssen die zu transformierenden Elemente zuerst einmal benannt werden, damit überhaupt bekannt ist, welche Elemente verändert werden sollen. XPath dient der Adressierung dieser Elemente. Ein Beispiel soll verdeutlichen, wie Elemente mittels XPath-Ausdrücken benannt werden. Angenommen ein XML-Dokument enthält die folgenden Elemente:

```
<AAA>
  <BBB/>
  <CCC/>      ( 1 )
  <BBB/>
  <BBB/>
  <DDD>
    <BBB/> ( 2 )
  </DDD>
  <CCC/>      ( 3 )
</AAA>
```

Der XPath-Ausdruck „/AAA/CCC“ würde alle Elemente CCC selektieren, die Kinder des Wurzelementes AAA sind, also (1) und (3). Der Ausdruck „/AAA/DDD/BBB“ würde alle Elemente BBB finden, die Kinder von DDD-Elementen sind, die wiederum Kinder des Wurzelementes AAA sind. Das entspricht genau dem Element (2). Ein XSL-Stylesheet, welches das Element (2) suchen würde, um anschließend Transformationen daran vorzunehmen, könnte folgendermaßen aussehen:

```
<xsl:template match="/">
<xsl:value-of select="/AAA/DDD/BBB"/>
...
  bearbeite die gefundenen Elemente
...
</xsl:template>
```

Neben der Möglichkeit, Elemente anhand ihres absoluten Pfades innerhalb des Dokumentes zu beschreiben, können die gesuchten Elemente auch flexibler beschrieben werden. Eine Auswahl weiterer XPath-Ausdrücke bietet die nachfolgende Tabelle.

XPath - Ausdruck	selektierte Elemente
//*	alle Elemente des Dokumentes
//BBB	alle Elemente BBB des Dokumentes
//DDD/BBB	alle Elemente BBB, die Kinder eines Elementes DDD sind
//BBB[@*]	alle Elemente BBB, die irgendwelche Attribute besitzen
/*/*/*/BBB	alle Elemente BBB auf der 4 Hierarchieebene
//*[count(*)=2]	alle Elemente des Dokumentes mit genau 2 Kindern
/AAA/BBB[1]	das erste Element BBB, das ein Kind der Wurzel AAA ist

2.5.2 XSLT

XSLT ist eine auf XML-basierende Sprache, die der Beschreibung von Transformationen dient. Mit XSLT können XML-Dokumente zum Beispiel in HTML-Dokumente, anders strukturierte XML-Dokumente oder beliebige andere Datei-Formate transformiert werden. Ein eigenständiges XSL-Dokument definiert Regeln, die festlegen, mit welchen Elementen des Quelldokumentes welche Transformationen durchgeführt werden sollen. Ein Beispiel soll demonstrieren, wie aus einem einfachen XML-Dokument ein HTML-Dokument entsteht. Aus dem linken Quelldokument soll das rechts dargestellte Html-Dokument erzeugt werden.

	<HTML>
<?xml version="1.0"?>	<HEAD> </HEAD>
<book>	<BODY>
<title>XSL</title>	<H1>XSL</H1>
<author>John Smith</author>	<H2>John Smith</H2>
</book>	</BODY>
	</HTML>

Das folgende XSL-Stylesheet definiert alle Schritte der nötigen Transformation.

```

<xsl:stylesheet version='1.0'
  xmlns:xsl=
  'http://www.w3.org/1999/XSL/Transform' >
  <xsl:template match="/">                                (1)
    <HTML>
      <HEAD> </HEAD>
      <BODY>
        <H1><xsl:value-of select="//title"/></H1>          (2)
        <H2><xsl:value-of select="//author"/></H2>        (3)
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>

```

Zuerst wird das Wurzelement selektiert (1). In unserem Fall ist dies das Element book. Alle folgenden XPath-Ausdrücke werden relativ zu diesem selektierten Ele-

ment aufgelöst. Der XSL-Befehl `xsl:value-of select="//title"` bei (2) ermittelt den Wert des Elementes `title`. Bei (3) wird auf die gleiche Weise der Wert des Elementes `author` ermittelt.

2.5.3 EXSLT

Die Funktionen des XSL-Standards erfüllen meist nicht die Bedürfnisse aller Nutzer. Erweiterungen des Standard mit speziellen Funktionalitäten werden häufig benötigt. EXSLT bildet eine Menge von Modulen, die den XSL-Standard um zusätzliche Funktionalitäten erweitert. Da diese Erweiterungen nicht Teil des XSL-Standards sind, werden sie von den einzelnen XSLT-Prozessoren auch unterschiedlich stark unterstützt.

2.5.4 XSL-FO

Mit Hilfe von XSL-FO (XSL Formatting Objects) können Entwickler spezifische Layouts und Stile für eine formatierte Ausgabe von XML-daten definieren. Da XSL-FO präzise formatierte, mehrseitige Dokumente erzeugt, ist es für die Generierung von PDF-Dokumenten bestens geeignet. XSL-FO ist ein eigenständiger Teil des XSL-Standards. Dokumente, die XSL-FO Elemente enthalten, können durch XSL-FO Prozessoren verarbeitet werden. Die Verwendung von XSL-FO wurde in der Fachstudie nicht näher betrachtet.

2.5.5 Die Transformation von XML-Dokumenten

Um ein XML-Dokument in ein anderes Format umzuformen, benötigen wir zuerst ein XSL-Dokument, welches die Regeln beschreibt, nach denen die Elemente des Quelldokumentes verändert werden. Beide Dokumente, das XSL-Stylesheet und das zu transformierende XML-Dokument, werden von einem XML-Parser eingelesen (Abb.1 Nr.1 und 2). Der Parser extrahiert aus dem XSL-Stylesheet die Transformationsregeln (Abb.1 Nr.3) und erstellt aus dem zu transformierenden XML-Dokument einen DOM-Baum (Abb.1 Nr.4). Ein XSLT-Prozessor führt anschließend diese Transformationsregeln auf den Knoten des DOM-Baumes aus (Abb.1 Nr.5 und 6). Das Ergebnis der Transformation ist der transformierte Quellbaum (Abb.1 Nr.7) und kann nun beliebig weiterverarbeitet werden. Zum Beispiel könnte eine Ausgabe als Dokument erfolgen. Hierbei sind Ausgaben in beliebigen Formaten, wie zum Beispiel HTML, XML oder PDF, möglich (Abb.1 Nr.8). Des weiteren ist eine Ausgabe als Datenstrom zur Weiterleitung an eine andere Anwendung ebenfalls denkbar.

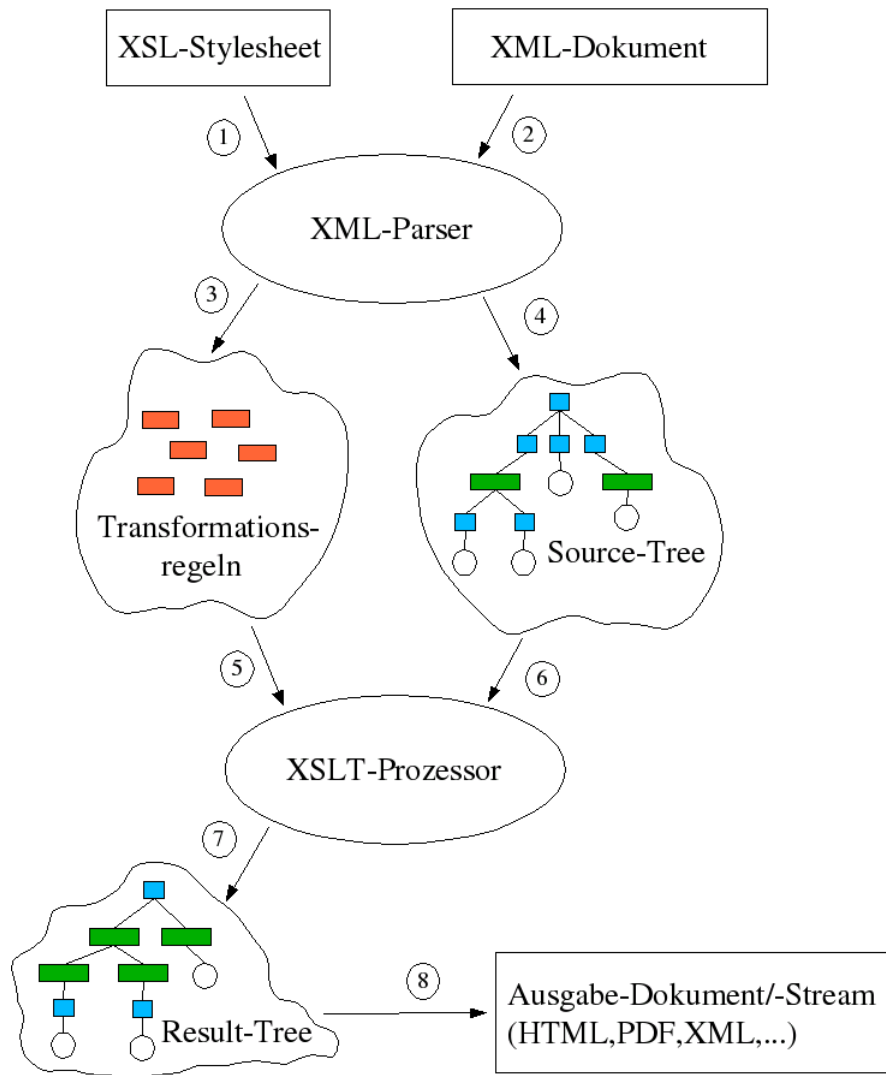


Abbildung 1: XSLT Überblick

2.6 XSLTMark

Für das Messen der Performanz der XSLT-Prozessoren benötigten wir ein Benchmark-Werkzeug, welches uns die Möglichkeit gab, Prozessoren verschiedener Implementierungssprachen zu testen.

Das Benchmarkwerkzeug XSLTMark ist ein frei verfügbares Benchmarkprogramm der Firma DataPower¹. XSLTMark ermöglicht das Einbinden von XSLT-Prozessoren über Treiberklassen. Jeder einzubindene Prozessor benötigt einen eigenen Treiber. XSLTMark kommuniziert über diese Treiber mit den Prozessoren, um die einzelnen Schritte der Transformation anzustossen. Dazu zählt das Laden der XML- und XSL-Dokumente, das Starten der Transformation und das Ausschreiben des Ergebnisses. Während der Transformation misst XSLTMark die benötigte Zeit sowie das verarbeitete Datenvolumen und führt auf Wunsch einen Vergleich mit einer Referenzausgabe durch. Dadurch können Performanz und Korrektheit der Transformation bestimmt werden.

Von sich aus bringt XSLTMark bereits Treiber für die bekanntesten XSLT-Prozessoren mit. Dazu gehören XalanJ, XalanC, OracleXDK, MSXML, Saxon, Sablotron, XT und jd.xslt. Sollen weitere Prozessoren in den Benchmark einbezogen werden, so ist es möglich, eigene Treiber für diese Prozessoren zu schreiben und so das Rahmenwerk zu erweitern.

Das Rahmenwerk enthält bereits Testfälle für die häufigsten 40 XSL-Transformationen. Es kann aber durch das Schreiben eigener Testfälle beliebig erweitert werden. Ein Testfall besteht in der Regel aus einem Eingabedokument (XML-Datei), einem XSL-Stylesheet und einer Referenzdatei, gegen die die erzeugte Ausgabe verglichen wird. Die Konfiguration der Testfälle erfolgt über eine Textdatei. Diese Datei enthält pro Testfall Einträge für die folgenden Parameter:

- den Namen des Testfalles,
- den Namen der Eingabedatei (Input.xml),
- den Namen der Stylesheet-Datei (Stylesheet.xml),
- den Namen der Ausgabedatei (Output.out),
- den Namen der Referenzausgabe (Output.ref),
- die Anzahl der Wiederholungen pro Testfall und Prozessor sowie
- die Namen der Prozessoren, die bei diesem Testfall nicht berücksichtigt werden sollen.

Pro Prozessor werden nacheinander alle Testfälle, den eben genannten Parametern entsprechend, durchlaufen. Das Ergebniss wird nach dem Durchlaufen aller Testfälle mit einem Prozessor auf die Standardausgabe geschrieben. Diese Ausgabe wurde von uns

¹www.datapower.com

jeweils in eine eigene Textdatei mit dem Namen des getesteten Prozessors umgeleitet, so dass ein Vergleich der Ergebnisse zu einem späteren Zeitpunkt möglich war. Die einzelnen Ergebnisausgaben sind im Kapitel 7 abgedruckt. Die Initialisierung der Java Laufzeitumgebung hat keinen Einfluss auf die gemessenen Transformationszeiten, da XSLTMark vor dem eigentlichen Benchmarktest einen Initialisierungsdurchlauf mit einem einzelnen Testfall außer der Reihe durchführt. Erst im Anschluß daran wird der Test komplett durchlaufen und die Performanz gemessen. Somit wird die Korrektheit der Messergebnisse gewährleistet, da XSLTMark nur die tatsächlichen Transformationszeiten misst.

3 Liste verfügbarer XSLT Prozessoren

Auf dem Softwaremarkt gibt es derzeit eine ganze Reihe von XSLT Prozessoren. Einige Eigenschaften haben mehrere dieser Prozessoren gemeinsam, jedoch unterscheiden sie sich zuweilen durch die verschiedenen Standards, welche unterstützt werden. Bei genauerer Betrachtung werden die Unterschiede der Prozessoren deutlich und man erkennt schnell die große Vielfalt an verschiedenen Produkten, ein jedes mit eigenen Stärken und Schwächen.

In der vorliegenden Arbeit werden allerdings nicht alle Prozessoren untersucht, sondern nur die Prozessoren, die aktuell in der Praxis Anwendung finden, aktiv weiterentwickelt werden und entweder in C/C++ oder Java implementiert sind. Zu den in dieser Fachstudie getesteten Prozessoren zählen: jd.xslt, MSXML, OracleXDK, Sablotron, Saxon, Xalan-J, Xalan-C, XSLTC und XT. Das folgende Kapitel gibt einen Überblick über die aktuell verfügbaren XSLT Prozessoren am Markt, wobei jeder Prozessor kurz vorgestellt wird und die unterstützten Standards verglichen werden. Die folgende Tabelle fasst die Eigenschaften der jeweiligen Prozessoren zum besseren Überblick zunächst einmal kurz zusammen.

Name	Version	Plattform	Sprache	Standards	EXSLT
jd.xslt	1.5.5	Java	Java	XSLT 1.1	wenige
libxslt	2.6.0	Linux, Unix, Windows	Ansi C	XSLT 1.0	wenige
MSXML	3.0	Windows	C, C++	XSLT 1.0	nein
OracleXDK	9i	Java	Java	XSLT 1.0	k/A
Sablotron	1.0.1	Linux, Windows	C++	XSLT 1.0	nein
Saxon	6.2.2	Java	Java	teilweise XSLT 2.0	ja
SunXSLTC	k/A	Java	Java	k/A	k/A
Transformiix	1.6	Linux, Windows	C++	teilweise XSLT 1.0	nein
Xalan	1.8 / 2.6.0	Linux, Windows, Solaris, AIX, HP-UX	C++	XSLT 1.0	einige
XSL:P	1.0beta	Java	Java	k/A	k/A
XT	20020426a	Java	Java	teilweise XSLT 1.0	nein

3.1 jd.xslt

Der jd.xslt Prozessor ist ein OpenSource XSLT Prozessor, der in Java implementiert ist. Der von Johannes Döbler entwickelte Prozessor bringt bereits eine eigene Testum-

gebung mit. Allerdings enthält der Prozessor keinen eigenen XML Parser. Eine Vielzahl solcher Parser sind jedoch als OpenSource Projekte frei verfügbar. Dazu gehören unter anderem die bekannten XML Parser Expat und AElfired. In der hier vorliegenden Fachstudie kommt der von der Apache Software Foundation entwickelte Xerces Parser zum Einsatz, da dieser in Java implementiert und somit leicht in das Rahmenwerk einbindbar ist.

Lizenz	Plattform	Sprache	Standards	EXSLT
MPL	Java	Java	XSLT 1.1	Common, Math, Set

3.2 libxslt

Diese von Daniel Veillard in C implementierte Bibliothek wurde im Rahmen des Gnome Projektes entwickelt. Es existieren Schnittstellen für eine Reihe von Programmiersprachen. Die Bibliothek wurde auch bereits erfolgreich auf die Systeme Linux, Unix, Windows, MacOS, OS/2 und QNX portiert. Sie hält sich stark an die vorgegebenen Standards und kommt mit einem integrierten XML Parser. Was die Bibliothek von den meisten Prozessoren unterscheidet, ist die gute Unterstützung von EXSLT Funktionen, welche u.a. die Verarbeitung von Strings, Mengen, mathematischer Funktionen und regulärer Ausdrücke ermöglicht. Des weiteren existiert eine Kommandozeilen-Unterstützung für Unix Systeme.

Lizenz	Plattform	Sprache	Standards	EXSLT
MIT	Linux, Unix, Windows	Ansi C	XSLT 1.0	teilweise

3.3 MSXML

Von Microsoft wird diese DLL Bibliothek angeboten, welche in C++ implementiert ist. Die Dienste des MSXML Pakets umfassen einen XML Parser sowie einen XSLT Prozessor. Die Bibliothek kann unter anderem im Internet Explorer oder in anderen Anwendungen verwendet werden. Des weiteren gibt es die Möglichkeit, die Bibliothek über die Kommandozeile direkt anzusprechen und Transformationen durchzuführen.

Ein zu MSXML kompatibles Paket ist FastXML, welches im Rahmen der Master Thesis von Helena Kupkova entstand. Durch einige Optimierungsmaßnahmen erreicht dieser Prozessor eine hohe Performanz. Um diese zu erreichen, mußte jedoch auf eine umfassende Unterstützung des XSLT Standards verzichtet werden, weshalb dieser Prozessor auch nicht genauer untersucht wurde.

Lizenz	Plattform	Sprache	Standards	EXSLT
MS Lizenz	Windows	C, C++	XSLT 1.0	nein

3.4 Oracle XDK

Ein weiterer Prozessor, der auf Java basiert, jedoch kein OpenSource Projekt darstellt, ist das XML Development Kit von Oracle. Es enthält mehrere Komponenten,

die hier integriert wurden, um vor allem Nutzern von Oracle Datenbanken die Verarbeitung von XML Dokumenten einfach zu ermöglichen. Das XDK besteht neben dem XSLT Prozessor noch aus Komponenten für die Verarbeitung von XML-Dokumenten, die Generierung von Java-Klassen aus XML-Dokumenten sowie die Generierung von XML-Dokumenten aus SQL-Queries.

Lizenz	Plattform	Sprache	Standards	EXSLT
OTN	Java	Java	XSLT 1.0	k/A

3.5 Sablotron

Das OpenSource Projekt Sablotron der Firma Ginger Alliance zielt auf die Entwicklung eines vielseitigen XML Toolkits, das sehr konform zu den aktuellen Standards und gleichzeitig sehr übersichtlich und schnell sein soll. Der Prozessor ist in C++ implementiert. Es existieren Schnittstellen zu eine Vielzahl anderer Programmiersprachen, wie z.B. Perl, PHP, Python, Tcl und Ada. Ausführbare Versionen stehen für Linux und Windows zu Verfügung.

Lizenz	Plattform	Sprache	Standards	EXSLT
MPL/GPL	Linux, Windows	C++	XSLT 1.0	nein

3.6 Saxon

Der Saxon Prozessor ist in Java implementiert. Wie der Name bereits andeutet, verwendet dieser Prozessor im Gegensatz zur Mehrheit der XSLT Prozessoren für die Verarbeitung von XML-Dokumenten das SAX-Modell. Der Name liest sich daher „SAX-on“. Ursprünglich wurde der Prozessor von Michael Kay implementiert und ist nun bei der Software AG als OpenSource Projekt frei verfügbar. Der Prozessor zeichnet sich besonders durch die umfangreichen Erweiterungen und unterstützten Standards aus, was ihn für viele Aufgaben besonders interessant macht. Es ist z.B. einer der wenigen Prozessoren, die EXSLT Funktionen nutzen können. Für Windows ist zusätzlich eine Windows-Executable Version verfügbar, die sich InstantSaxon nennt. Hier sind weder Sourcecode noch Dokumentation erhältlich, weshalb der Prozessor auch nicht näher untersucht wurde.

Lizenz	Plattform	Sprache	Standards	EXSLT
MPL	Java	Java	teilweise XSLT 2.0	ja

3.7 Transformiix

Der Transformiix Prozessor entstand durch die Portierung einiger Teile des XSL:P Prozessors (s.u.) von Java nach C++. Das Hauptinteresse lag dabei auf der Entwicklung eines schnellen und effizienten XSLT Prozessors, der im Mozilla Browser eingesetzt werden kann. Der Prozessor steht unter Mozilla Public License. Dieser Prozessor wurde in der Studie nicht näher untersucht, da auf der Internetseite des Mozilla Projektes

keine eigenständige-Version des Prozessors verfügbar war, vielmehr wird der Prozessor als Modul in den Mozilla-Browser eingebunden.

Lizenz	Plattform	Sprache	Standards	EXSLT
MPL	Linux, Windows	C++	teilweise XSLT 1.0	nein

3.8 Xalan

Dieser Prozessor ist ein OpenSource Projekt und kann frei heruntergeladen werden. Er entstand als Projekt der Apache Software Foundation und liegt in zwei Varianten, einer in Java und einer in C++ vor. Bei beiden Prozessoren liegt das Hauptinteresse an der möglichst genauen Umsetzung der XSLT und XPath Standards. Dies bedingt momentan jedoch, dass die Performanz nicht optimal ist. Durch die Ersetzung des DOM Modells durch das sogenannte Document Table Model (DTM) konnte die Performanz allerdings schon verbessert werden. Für die Verarbeitung von XML Dateien ist ein externer Parser erforderlich. Hierfür wird typischerweise Xerces von der Apache Foundation verwendet.

Lizenz	Plattform	Sprache	Standards	EXSLT
ASF Lizenz	Java	Java	teilw. XSLT 2.0	ja
ASF Lizenz	Linux, Windows, Solaris, AIX, HP-UX	C++	XSLT 1.0	Common, Math, Set, String

3.9 XSLTC

Dieser Prozessor wurde ursprünglich von Sun Microsystems entwickelt, kurz nach der Entwicklung aber an die Apache Software Foundation weitergegeben und in das Jakarta Xalan Projekt eingegliedert. Innerhalb des Projektes steht er gleichberechtigt neben den XSLT Prozessoren XalanJ und XalanC. Eine Besonderheit dieses Prozessors ist die Möglichkeit, aus den XSL-Stylesheets vorkompilierte Javaklassen zu generieren, welche die spezifizierten Transformationen bei jedem Aufruf der Klassenmethoden durchführen. Dadurch entfällt das Parsen des Stylesheets bei jeder Transformation, was durchaus einen gewissen Gewinn an Performanz zu der gewöhnlichen Vorgehensweise ausmacht. Dieser Ansatz ist besonders sinnvoll, wenn genau festgelegte, invariante Transformationen auf viele XML-Dokumente angewandt werden sollen. In dieser Fachstudie wurde jedoch lediglich die Variante ohne vorkompilierte Stylesheets berücksichtigt.

Lizenz	Plattform	Sprache	Standards	EXSLT
k/A	Java	Java	k/A	k/A

3.10 XSL:P

Dieser Prozessor wird aus Gründen der Vollständigkeit hier aufgeführt. Er wurde ursprünglich von Keith Visco in Java implementiert. Der Prozessor wird heute nicht mehr

weiterentwickelt. Teile des Prozessors wurden in den Transformix Prozessor übernommen.

Lizenz	Plattform	Sprache	Standards	EXSLT
k/A	Java	Java	k/A	k/A

3.11 XT

Der von James Clark entwickelte XT Prozessor ist ebenfalls ein in Java implementierter OpenSource XSLT Prozessor. Er hat den Vorteil einer sehr hohen Verarbeitungsgeschwindigkeit. Auf der anderen Seite ist die Unterstützung des XSLT Standards nicht vollständig gegeben. Es fehlt hier z.B. die Element Extension und einige andere Funktionen. Die gute Performanz ist bei Nichtnutzung der fehlenden Mechanismen dennoch ein Argument für diesen Prozessor.

Lizenz	Plattform	Sprache	Standards	EXSLT
OpenSource	Java	Java	teilweise XSLT 1.0	nein

4 Kriterienkatalog

Wichtig bei der Bewertung der unterschiedlichen Prozessoren sind die Anforderungen der Anwender. Aus diesem Grund wurde ein Kriterienkatalog entwickelt, nach dem eine Nutzwertanalyse der verschiedenen Prozessoren durchgeführt werden kann. Anhand der Kriterien lässt sich ein Prozessor auswählen, der optimal auf die Bedürfnisse des Projektrahmenwerkes abgestimmt ist.

Für die Bewertung der einzelnen Kriterien wurde folgende Vorgehensweise gewählt:

Zunächst wurde die Erfüllung der einzelnen Kriterien über ein Punktesystem oder absolute Meßwerte festgehalten. Nach dieser Bewertung wurden die einzelnen erreichten Punkte normalisiert und gewichtet. Anhand der höchsten erreichten Gesamtpunktzahl lässt sich anschließend der Prozessor ermitteln, der die aufgestellten Kriterien am Besten von allen untersuchten Prozessoren erfüllt.

Um eine korrekte Gewichtung der Kriterien zu ermöglichen, müssen die Erfüllungsgrade der einzelnen Kriterien innerhalb des gleichen Intervalls liegen. Da die absoluten erreichten Werte jedoch in sehr unterschiedlichen Skalenbereichen liegen, muss vor der Kriteriengewichtung eine Normierung der Erfüllungsgrade vorgenommen werden. Der Erfüllungsgrad wird in den Ergebnistabellen auf zwei verschiedene Arten angegeben - entweder als absolute Zahl zwischen 0 und 1, wobei 0,5 einem Erfüllungsgrad von 50% und 1 einem Erfüllungsgrad von 100% entspricht, oder direkt als Prozentsatz.

4.1 Gewichtung

Da nicht jedes Kriterium dieselbe Bedeutung für die Bewertung der Prozessoren besitzt, werden die Kriterien unterschiedlich gewichtet. Diese Gewichtung erfolgt innerhalb der zugehörigen Unterkategorie (Performanz(kleine/große Dateien), Dokumentation, Integrierbarkeit).

4.2 Hauptkriterien

Die folgende Tabelle zeigt die Hauptkriterien und deren Gewichtung. Da die Performanz in dieser Studie als wichtigster Bewertungsmaßstab angesetzt wird, erhält das Kriterium ein Gewicht von 3.

Kategorie	Gewicht	Erklärung
Performanz	3	Wie hoch ist der Datendurchsatz des Prozessors?
Dokumentation	1	Welchen Umfang hat die vorhandene Dokumentation zum Prozessor?
Integrierbarkeit	2	Wie einfach lässt sich der Prozessor in das Projektrahmenwerk einbinden?

4.3 Dokumentation

Die Dokumentation dient der leichten Einarbeitung und korrekten Verwendung des Prozessors. Aktive Diskussionsforen zum Prozessor bieten eine wertvolle Hilfestellung zusätzlich zu Anleitung, Installationsanweisung, Beispielen und Tutorial. Die Gesamtpunktzahl, die ein Prozessor für die Dokumentation erhält, ergibt sich aus der Summe der Punkte pro vorhandenem Dokument. Die einzelnen überprüften Dokumente sind in der folgenden Tabelle mit den zugeordneten Punkten aufgeführt.

Punkte	Kriterium
1	Installationsanweisung vorhanden
1	Beispielanwendungen vorhanden
2	Tutorial vorhanden
2	Newsgroup/Forum (mit Aktivität) vorhanden
3	API-Dokumentation (Javadoc, Doxydoc) vorhanden

4.4 Integrierbarkeit (Implementierungssprache)

Die Implementierungssprache des Prozessors muß betrachtet werden, um die Integrierbarkeit zu beurteilen. Die Sprache Java ist für die Integrierbarkeit von Vorteil, da das Projektrahmenwerk vorrangig aus Java-Komponenten besteht.

Punkte	Kriterium
1	Kein Java
2	Java

4.5 Performanz

Der wichtigste Faktor für die Bewertung der Prozessoren innerhalb dieser Fachstudie ist die Performanz bei der Transformation von XML-Dokumenten. Dieses Kriterium wurde anhand des Datendurchsatz bewertet.

Hierfür wird zunächst der Durchsatz jedes Prozessoren pro Testfall und bei unterschiedlichen Dateigrößen bestimmt. Der Durchsatz wird in kB/s gemessen. Es werden kleine und große Eingabedateien unterschieden. Als kleine Eingabedateien werden dabei Dateien bis 3kB bezeichnet. Große Eingabedateien haben einem Umfang von 15kB bis 25kB. Da kleine Dateien häufiger durch das Projektrahmenwerk verarbeitet werden als große Eingabedateien, wird die Performanz, die Prozessoren bei deren Transformation erreichen, doppelt so stark gewichtet, wie die Performanz bei der Verarbeitung großer Dateien.

Gewicht	Kriterium
2	Performanz bei Transformationen kleiner Dateien
1	Performanz bei Transformationen großer Dateien

4.6 Testfälle

Die Messung der Performanz wurde anhand von acht Testfällen durchgeführt, wie zuvor beschrieben, jeweils mit kleinen Eingabedateien (Dateigrößen zwischen 1 und 3 kB) und mit großen Eingabedateien (Dateigrößen zwischen 15 und 25 kB).

Um die einzelnen Bedeutungen der Testfälle in die Beurteilung der Performanz einfließen lassen zu können, erhält jeder Testfall eine Gewichtung, die seiner Bedeutung für das Projektrahmenwerk entspricht. Dabei sind folgende Gewichtungen möglich.

Gewicht	Kriterium (x von 100 Transformationen)
0	Nie verwendet (0 von 100)
1	Selten verwendet (1-25 von 100)
2	Regelmäßig verwendet (26-50 von 100)
3	Häufig verwendet (51-75 von 100)
4	Sehr häufig verwendet (76-100 von 100)

Die nachfolgende Tabelle listet die definierten Testfälle mit Ihrer Gewichtung und einer Kurzbeschreibung auf. Eine ausführliche Beschreibung der Testfälle findet sich in Kapitel 5.

Testfallname	Gewicht	Funktionalität
AggregateXML	1	Aggregation (hier: Summenbildung) mit XML-Output
AggregateHTML	0 (nicht gewertet)	Aggregation (hier: Summenbildung) mit HTML-Output
AttrToElem	2	Strukturveränderungen (Attribute in Elemente umwandeln)
CreateElemSubset	2	Strukturveränderungen (Elemente umsordern)
FilterElement	4	Filterung einzelner Elemente
IncludeDocs	3	Integration mehrerer Eingabedateien
RenameElement	4	Umbenennen von Elementen
ReplaceElementValue	2	Verändern von Elementinhalten

4.7 Normalisierung

Da die maximal erreichbare Punktzahl von Kriterium zu Kriterium unterschiedlich groß ist, müssen die erreichten Punktzahlen normalisiert werden, um ein korrektes Vergleichen und Gewichten der Kriterien zuzulassen.

Die Normalisierung erfolgt nach Kriterien getrennt. Die normalisierten Werte werden in Prozent angegeben. Nach der Bewertung des Kriteriums anhand von Punkten beziehungsweise gemessenen Durchsätzen erhält der Prozessor mit dem höchsten erreichten

Wert im jeweiligen Kriterium den Wert 100 Prozent für dieses Kriterium zugewiesen. Der Erreichungsgrad der restlichen Prozessoren (für dieses Kriterium) wird an diesen 100 Prozent orientiert. Die Erreichungsgrade der restlichen Prozessoren sind demnach entweder gleich diesen 100 Prozent oder kleiner.

Eine Beispielnormierung:

Prozessor A erreicht bei Kriterium X insgesamt 5 Punkte.

Prozessor B erreicht im gleichen Kriterium insgesamt 10 Punkte.

Prozessor C erreicht 8 Punkte.

Prozessor B hat die höchste Punktzahl von allen untersuchten Prozessoren. Er bekommt 100% als normierten Erfüllungsgrad im Kriterium X zugewiesen. Gemessen am Erfüllungsgrad von Prozessor B hat Prozessor A 50% und Prozessor C 80% erreicht.

Die Performanz wird aufgrund des Datendurchsatzes des Prozessors in kB/s bestimmt. Die folgende beschriebene Auswertung findet je einmal für kleine Eingabedateien und einmal für große Eingabedateien statt.

Die gemessenen Durchsätze werden pro Testfall für alle Prozessoren normiert. Die normierten Durchsätze werden anschließend mit dem Gewicht des jeweiligen Testfalles multipliziert (gewichtet). Die normierten und gewichteten Durchsätze für alle Testfälle werden pro Prozessor addiert (gewichtete Performanz). Das Ergebnis wird wiederum unter allen Prozessoren normiert, so dass ein Prozessor bei kleinen beziehungsweise großen Testdateien maximal 100% Erfüllungsgrad erreichen kann.

Liegen die normierten Performanzen für kleine und große Eingabedateien vor, so werden diese Ergebnisse nochmals mit den Gewichten für die Transformation kleiner beziehungsweise großer Dateien gewichtet und anschließend normiert. Der errechnete Wert drückt die normierte Gesamtperformanz des Prozessors aus.

5 Beschreibung der Testfälle

Wie schon im Kapitel 2.6 kurz angedeutet wurde, besteht ein Testfall für das Rahmenwerk XSLTMark aus mehreren Eingabe- und Ausgabedateien. Die Auswahl und Steuerung dieser Testfälle geschieht über eine „plain text“ Konfigurationsdatei, in diesem Falle `my.conf`.

Das folgende Kapitel widmet sich nun den einzelnen Testfällen, die hier genauer beschrieben werden. Hierbei ist zu beachten, dass ein Testfall immer eine ganze Klasse von häufig auftretenden Transformationen abdeckt. Die jeweilige Klasse wird bei den entsprechenden Testfallbeschreibungen angegeben. Des weiteren werden die Testfälle innerhalb einer Klasse durch ein weiteres Merkmal aufgeteilt, um eine höhere Aussagekraft der Messdaten zu ermöglichen. Dieses Merkmal ist die Größe der zu verarbeitenden XML-Eingabedatei. Jede Transformationsklasse wird also auf je eine sehr kleine und eine deutlich umfangreichere XML-Datei angewandt. Zur besseren Unterscheidung wurde den Bezeichnern der Testfälle für kleine Eingabedateien das Suffix `-small` angehängt.

Das Kernstück eines jeden Testfalls bildet das sog. XSL Stylesheet, in dem die Transformationsschritte definiert werden. Um die Funktionsweise der Stylesheets nachvollziehen zu können, werden diese im Kapitel des zugehörigen Testfalls mit angegeben. Zum Verständnis der Transformationsschritte sind dort auch kurze Auszüge der zugrundeliegenden XML-Dateien aufgeführt. Auf die XML Headerinformationen wurde dabei jedoch aus Platzgründen verzichtet.

5.1 AggregateHTML und AggregateXML

Transformationsklasse: Funktionen zur Datenaggregation

Mit XSLT ist es möglich, aus einem beliebigen XML Datensatz einzelne Werte auszulesen und die gewünschte Information in bestimmter Weise zu verdichten, sprich zu aggregieren. Eine sehr einfache, jedoch oft benötigte Funktion ist das automatische Aufsummieren mehrerer Zahlen. Am Beispiel der vorliegenden Testfälle ist dies die Entwicklung von Verkaufszahlen einer fiktiven Firma in verschiedenen globalen Regionen im Verlauf mehrerer Jahre. Die Verkaufszahlen der Regionen USA, Europa und Asien werden dabei für jedes Jahr aufsummiert und in die Ausgabedatei geschrieben. Zusätzlich wird am Ende noch die Gesamtsumme der Verkaufszahlen aller Jahre ausgegeben. Dazu dient die `sum()` Funktion, welche in einer Iteration über alle `<year>` Elemente die gewünschten Daten sammelt. Die Iteration wird mit dem Ausdruck `xsl:for-each` angestoßen.

Die beiden Testfälle unterscheiden sich durch die Art der Ausgabe. Einmal wird ein HTML-Dokument erstellt und beim anderen Testfall eine XML-Datei (vgl. Namen der Testfälle). Diese Unterscheidung ist deshalb interessant, da ein eventueller Performanzverlust bei unterschiedlichen Ausgabeformaten auftreten könnte. Die Klasse der Aggregationsfunktionen ist jedoch die einzige, bei der dies berücksichtigt wurde, da die Ausgabe in HTML für das Projekt innerhalb des SFB 467 keine Bedeutung hat.

Deshalb ist auch die Gewichtung des Testfalles AggregateHTML gleich null (siehe Kap. 4.6).

XSL Stylesheet für XML Ausgabe:

```
<xsl:template match="/">
  <xsl:element name="totalSales">
    <xsl:element name="perYear">
      <xsl:for-each select="salesdata/year">
        <xsl:element name="year">
          <xsl:attribute name="value">
            <xsl:value-of select="year"/>
          </xsl:attribute>
          <xsl:element name="sales">
            <xsl:value-of select="sum(region/sales)"/>
          </xsl:element>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
    <xsl:element name="grandTotal">
      <xsl:value-of select="sum(salesdata/year/region/sales)"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

XSL Stylesheet für HTML Ausgabe:

```
<xsl:template match="/">
  <html>
    <table border="1">
      <tr>
        <td colspan="2">Total Sales</td>
      </tr>
      <xsl:for-each select="salesdata/year">
        <tr>
          <td>
            <xsl:value-of select="year"/>
          </td>
          <td align="right">
            <xsl:value-of select="sum(region/sales)"/>
          </td>
        </tr>
      </xsl:for-each>
      <tr>
        <td>Grand Total</td>
        <td align="right">
          <xsl:value-of select="sum(salesdata/year/region/sales)"/>
        </td>
      </tr>
    </table>
  </html>
</xsl:template>
```

XML Eingabe:

```
<salesdata>
  <year>
    <year>1991</year>
    <region>
      <name>USA</name>
      <sales unit="millions">32</sales>
    </region>
    <region>
      <name>Europe</name>
      <sales unit="millions">11</sales>
    </region>
    <region>
      <name>Asia</name>
      <sales unit="millions">19</sales>
    </region>
  </year>
</salesdata>
```

5.2 AttrToElem

Transformationsklasse: Strukturveränderungen

Bei diesem Testfall wird die in die hierarchische Struktur der XML-Eingabedatei eingegriffen. Die ursprüngliche Datei wird durch einen flachen, aber relativ breiten Baum repräsentiert. Diese Datenstruktur, die sich durch die starke Verwendung von attributierten Elementen auszeichnet, ist jedoch für menschliche Leser eher unpraktisch. Der Testfall formt die Struktur dergestalt um, dass in der Ausgabedatei für jedes Attribut ein eigenes Element angelegt wird. Dazu dient der Ausdruck `xsl:element`, der auch beliebig verschachtelt werden kann. So können auch sehr komplexe Datenstrukturen automatisch erstellt werden. Das Stylesheet iteriert, wie auch das vorhergehende, über alle Elemente mit Hilfe von `xsl:for-each`.

XSL Stylesheet:

```
<xsl:template match="/">
  <customers>
    <xsl:for-each select="/customers/customer">
      <xsl:element name="{name()}">
        <xsl:for-each select="@*">
          <xsl:element name="{name()}">
            <xsl:value-of select="."/>
          </xsl:element>
        </xsl:for-each>
      </xsl:element>
    </xsl:for-each>
  </customers>
</xsl:template>
```

XML Eingabe:

```

<customers>
  <customer CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"
    ContactName="Maria Anders" ContactTitle="Sales Representative"
    Address="Obere Str. 57" City="Berlin" PostalCode="12209"
    Country="Germany" Phone="030-0074321" Fax="030-0076545"/>
</customers>

```

5.3 CreateElemSubset

Transformationsklasse: Untermengen von Elementen

Dieser Testfall ähnelt in seiner Funktionsweise dem vorhergehenden derart, dass aus einer sehr flachen und breiten Datenstruktur von stark attributierten Elementen eine schmale und tiefe Struktur von nicht-attributierten Elementen hervorgeht. Hier werden allerdings nicht alle Attribute in die Ausgabedatei übernommen, sondern nur eine kleine Untermenge, die den Namen, die ID und den Landessitz aller Elemente, in diesem Falle Kunden einer fiktiven Firma, enthält. Zu diesem Zweck wird nicht über alle Elemente iteriert, sondern über die Templaterregel `match='customer'` zu jedem darauf passenden Element in der Eingabedatei ein entsprechendes Element in der Ausgabedatei erzeugt und mit dem zugehörigen Wert verknüpft.

XSL Stylesheet:

```

<xsl:template match="/">
  <customers>
    <xsl:apply-templates select="/customers/customer" />
  </customers>
</xsl:template>

<xsl:template match="customers">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="customer">
  <customer>
    <CompanyName>
      <xsl:value-of select="@CompanyName" />
    </CompanyName>
    <CustomerID>
      <xsl:value-of select="@CustomerID" />
    </CustomerID>
    <Country>
      <xsl:value-of select="@Country" />
    </Country>
  </customer>
  <xsl:apply-templates/>
</xsl:template>

```

XML Eingabe: siehe *AttrToElem*

5.4 FilterElement

Transformationsklasse: Filterung/Projektion einzelner Elemente

Bei dieser Transformationsklasse geht es um die Reduktion eines beliebigen Datensatzes, um z.B. die Übersichtlichkeit zu erhöhen oder lediglich die momentan relevanten Daten herauszufiltern. Im Beispiel dieses Testfalles wird die Projektion einer relativ umfangreichen Datenstruktur auf nur ein einziges Element erreicht. Zu jedem Kunden in dem Datensatz wird die interne ID herausgefiltert und in die Ausgabedatei geschrieben. Dies geschieht wiederum mit der `match` Regel, welche zu jedem Kunden den `CustomerID` Tag findet und ein entsprechendes Element mit dem zugehörigen Wert erzeugt.

XSL Stylesheet:

```
<xsl:template match="customer">
  <xsl:apply-templates select="CustomerID" />
</xsl:template>

<xsl:template match="CustomerID">
  <xsl:element name="CustomerID">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
```

XML Eingabe (vereinfacht):

```
<customers>
  <customer>
    <CustomerID>
      ALFKI
    </CustomerID>
    <CompanyName>
      Alfreds Futterkiste
    </CompanyName>
    <Address>
      Obere Str.57
    </Address>
    <City>
      Berlin
    </City>
    <PostalCode>
      12209
    </PostalCode>
  </customer>
</customers>
```

5.5 IncludeDocs

Transformationsklasse: Einbinden mehrerer XML-Dateien

Bisher wurden lediglich Transformationen betrachtet, welche auf einer einzigen XML-Eingabedatei operieren. Im professionellen Einsatz müssen aber evtl. Daten aus mehreren XML Datensätzen zusammengeführt werden. Deshalb gibt es in XSL Stylesheets die Möglichkeit, mit multiplen Eingabedateien zu arbeiten. In diesem Testfall dient zunächst `IncludeCustomer.xml` als primäre Eingabedatei. Für jeden Kunden, der dort aufgelistet ist, wird in der Templaterregel `match='Customer'` zuerst ein Element für die ID angelegt und der entsprechende Attributwert darin geschrieben. Gleichzeitig wird dieser Wert auch in der Variable `Customer-Id` abgelegt. Anschließend wird für die weiteren Verarbeitungsschritte durch den Funktionsaufruf von `document('IncludeNames.xml')` diese Datei betrachtet. Mit Hilfe des Ausdrucks `xsl:test` wird der Kunde in der Datei gefunden, dessen ID mit dem Wert in der zuvor angelegten Variable übereinstimmt. Dann wird der Name des Kunden aus dem Attribut `Name` ausgelesen und in dem entsprechenden Element in der Ausgabedatei abgelegt. Als letztes werden aus der Datei `IncludeAddresses.xml` die weiteren Daten des Kunden (Typ, Adresse) herausgelesen und in eigens dafür angelegte Elemente geschrieben. Im vorliegenden Beispiel handelt es sich dabei um fiktive Händleradressen. Um nun dieses Verhalten zu erreichen, wird aus der aktuellen Regel heraus mit Hilfe von `xsl:apply-templates select='Address'` eine andere Regel aufgerufen, die diese Aufgabe erledigt.

Die Variation der Dateigröße betrifft in diesem Testfall lediglich die primäre „Kunden-datenbank“, welche durch die Datei `IncludeCustomer.xml` repräsentiert wird.

XSL Stylesheet:

```
<xsl:template match="/CustomerOrder">
  <xsl:copy>
    <xsl:apply-templates select="Customer"/>
    <xsl:copy-of select="Items"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="Customer">
  <Customer>
    <Id><xsl:value-of select="@Id"/></Id>
    <xsl:variable name="Customer-Id" select="@Id"/>
    <xsl:for-each select="document('IncludeNames.xml')
      /CustomerNames/CustomerName">
      <xsl:if test="@Id=$Customer-Id">
        <Name><xsl:value-of select="@Name"/></Name>
      </xsl:if>
    </xsl:for-each>

    <xsl:for-each select="document('IncludeAddresses.xml')
      /Addresses">
      <xsl:apply-templates select="Address"/>
    </xsl:for-each>
  </Customer>
</xsl:template>

<xsl:template match="Address">
```

```

<Address>
  <Type><xsl:value-of select="@type"/></Type>
  <Street><xsl:value-of select="Street"/></Street>
  <ZIP><xsl:value-of select="ZIP"/></ZIP>
  <City><xsl:value-of select="City"/></City>
</Address>
</xsl:template>

```

XML Eingabe 1:

```

<Addresses>
  <Address type="deliver">
    <Street>Hauptstr.8</Street>
    <ZIP>70176</ZIP>
    <City>Stuttgart</City>
  </Address>
</Addresses>

```

XML Eingabe 2:

```

<CustomerOrder>
  <Customer Id="2"/>
  <Items>
    <Item Product="Television"/>
  </Items>
</CustomerOrder>

```

XML Eingabe 3:

```

<CustomerNames>
  <CustomerName Id="204" Name="Scholz AG"/>
  <CustomerName Id="205" Name="Bayer AG"/>
  <CustomerName Id="23" Name="Genia GmbH"/>
</CustomerNames>

```

5.6 RenameElement

Transformationsklasse: Umbenennen von Elementen/Tags

Um den Informationsaustausch zwischen heterogenen Systemen mit XSLT automatisieren zu können, besteht nicht nur der Bedarf an einer Möglichkeit, Elemente in XML Dokumenten beliebig umzustellen oder herauszufiltern. Zusätzlich benötigt man häufig einen Mechanismus zum Umbenennen von XML Elementen bzw. Tags. In dem vorliegenden Testfall sollen die englischsprachigen Elementbezeichner ins Deutsche übersetzt werden. Zu diesem Zweck wird an oberster Stelle die Templaterregel `match='*'` verwendet, welche als allgemeinstes Muster lediglich die Eingabedatei in die Ausgabedatei kopiert. Der Ausdruck `xsl:apply-templates`, welcher auch in den vorigen Testfällen schon auftritt, bewirkt, dass die nachfolgenden Regeln rekursiv auf die Eingabedatei angewandt werden. Diese spezifischeren Regeln kommen nur

dann zur Anwendung, wenn ein entsprechendes Muster gefunden wird. In jeder dieser Regeln wird dann ein Element mit dem passenden deutschen Bezeichner angelegt und mit dem Wert der Eingabedatei belegt.

XSL Stylesheet (vereinfacht):

```
<xsl:template match="*">
  <xsl:copy>
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>

<xsl:template match="customer">
  <xsl:element name="kunde">
    <xsl:value-of select="./text()" />
  </xsl:element>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="CustomerID">
  <xsl:element name="KundenID">
    <xsl:value-of select="./text()" />
  </xsl:element>
</xsl:template>

<xsl:template match="CompanyName">
  <xsl:element name="Firmenname">
    <xsl:value-of select="./text()" />
  </xsl:element>
</xsl:template>

<xsl:template match="Address">
  <xsl:element name="Adresse">
    <xsl:value-of select="./text()" />
  </xsl:element>
</xsl:template>

<xsl:template match="City">
  <xsl:element name="Stadt">
    <xsl:value-of select="./text()" />
  </xsl:element>
</xsl:template>

<xsl:template match="PostalCode">
  <xsl:element name="Postleitzahl">
    <xsl:value-of select="./text()" />
  </xsl:element>
</xsl:template>
```

XML Eingabe: siehe *FilterElement*

5.7 ReplaceElementValue

Transformationsklasse: Ändern von Elementwerten/Daten

Im Gegensatz zum vorhergehenden Testfall, werden in diesem bei der Transformation nicht die Bezeichner der Strukturelemente, sondern deren tatsächlichen Werte verändert. In diesem Beispiel werden alle Elementwerte der einzelnen Kunden mit dem selben Wert überschrieben, um möglichst viele Transformationsschritte für eine sinnvolle Messung zu erzielen. Im normalen Gebrauch würde sinnigerweise nur ein einzelner Kundeneintrag verändert werden, um z.B. eine Adress- oder Namensänderung im Datensatz wiederzuspiegeln. Prinzipiell arbeitet dieser Testfall genau so, wie auch der vorige. Durch die allgemeinste Regel `match='*'` wird einfach die Eingabedatei kopiert. In jeder spezielleren Regel wird dann in das jeweilige Element der neue Wert geschrieben. Dies geschieht mit dem Ausdruck `xsl:text`.

XSL Stylesheet:

```
<xsl:template match="*">
  <xsl:copy>
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>

<xsl:template match="CustomerID">
  <xsl:element name="CustomerID">
    <xsl:text>newCustomerID</xsl:text>
  </xsl:element>
</xsl:template>

<xsl:template match="CompanyName">
  <xsl:element name="CompanyName">
    <xsl:text>newCompanyName</xsl:text>
  </xsl:element>
</xsl:template>

<xsl:template match="Address">
  <xsl:element name="Address">
    <xsl:text>newAddress</xsl:text>
  </xsl:element>
</xsl:template>

<xsl:template match="City">
  <xsl:element name="City">
    <xsl:text>newCity</xsl:text>
  </xsl:element>
</xsl:template>

<xsl:template match="PostalCode">
  <xsl:element name="PostalCode">
    <xsl:text>newPostalCode</xsl:text>
  </xsl:element>
</xsl:template>
```

XML Eingabe: siehe *FilterElement*

6 Die Testumgebung

Die Prozessoren wurden auf einem Rechner mit dem Betriebssystem Windows 2000 (SP4) getestet. Als Java Laufzeitumgebung wurde die SUN Java Runtime Environment 1.3.1_1 verwendet. Der Rechner arbeitete mit einem AMD Athlon 900 MHz Prozessor und 256 MB RAM.

7 Evaluationsergebnisse

In den Tabellen 1 und 2 werden die Meßergebnisse bei großen und kleinen Testdateien aufgelistet, die in der beschriebenen Testumgebung gemessen wurden. Diese Aufstellung bietet die Möglichkeit, bei späterer Änderung der Kriteriengewichte eine komplette Nutzwertanalyse durchzuführen, ohne nochmals die Testfälle messen zu müssen.

Die einzelnen Ergebnisse werden im Anhang B nochmals nach Prozessoren getrennt aufgelistet.

XSLT-Prozessor	AggregateXML	AggregateHTML	AttrToElem	CreateElemSubset	FilterElement	IncludeDocs	RenameElement	ReplaceElemValue
jdxml	159,91	167,52	551,77	521,38	701,62	43,18	726,39	752,27
msxml	152,74	167,48	730,18	536,37	684,62	66,63	675,62	662,48
oracle-xdk	112,09	269,01	376,45	896,42	1143,79	9,95	386,79	558,59
sablot	116,04	119,96	289,89	319,35	381,20	20,66	301,15	324,05
saxon	139,14	44,29	387,64	367,76	421,40	8,14	406,42	470,05
xalan-c	127,90	104,02	518,39	454,98	746,71	22,7	640,79	673,28
xalan-j	144,11	161,41	377,44	478,09	420,66	8,14	399,66	479,43
xsltc	224,18	192,15	340,68	341,49	470,97	4,26	399,66	532,58
xt	118,26	126,45	468,75	404,33	407,72	39,8	617,00	593,65

Tabelle 1: Ergebnistabelle Durchsatz in KByte/s große Dateien

XSLT-Prozessor	AggregateXML	AggregateHTML	AttrToElem	CreateElemSubset	FilterElement	IncludeDocs	RenameElement	ReplaceElemValue
jdxml	44,57	42,72	153,98	90,11	131,62	15,45	149,31	151,43
msxml	43,95	47,42	196,59	101,12	138,71	25,04	134,44	131,11
oracle-xdk	65,56	60,19	72,90	103,26	199,43	7,16	185,62	149,88
sablot	36,31	35,60	116,34	70,99	101,75	8,95	103,19	103,25
saxon	24,43	26,38	94,99	59,57	92,46	2,98	126,34	70,97
xalan-c	36,53	38,88	145,45	77,87	115,37	14,22	131,76	124,74
xalan-j	24,43	41,12	68,57	61,95	92,46	2,82	133,32	67,03
xsltc	25,47	60,19	56,12	140,8	201,09	1,88	201,09	159,81
xt	36,68	38,88	141,14	77,87	91,58	16,58	121,88	118,04

Tabelle 2: Ergebnistabelle Durchsatz in KByte/s kleine Dateien

8 Auswertung der Ergebnisse

8.1 Übersicht der Durchsätze

Die Abbildungen 2 und 3 visualisieren, mit welchem Durchsatz die einzelnen Prozessoren die jeweiligen Testfälle abgearbeitet haben. Die Säulen in den Diagrammen stellen dabei den Durchsatz eines jeden Prozessors dar. Die Diagramme sollen zeigen, bei welchen Testfällen die Prozessoren, relativ gesehen, besonders leistungsschwach oder leistungsstark sind. Eine kleine Fläche auf den Säulen bedeutet dabei, daß der Prozessor wenig Durchsatz geleistet hat.

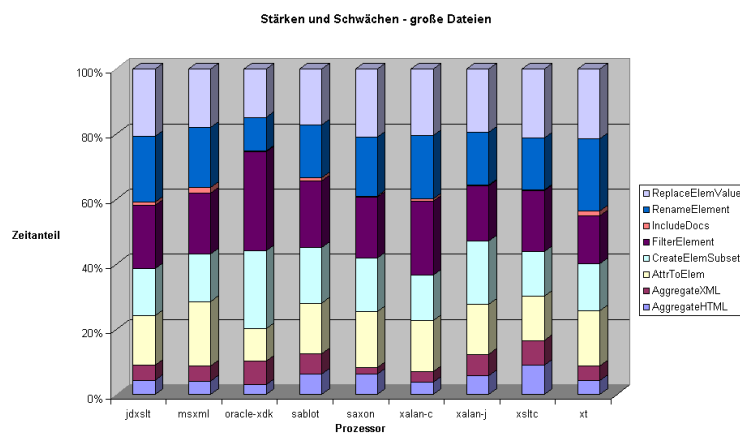


Abbildung 2: Stärken und Schwächen (große Testdateien)

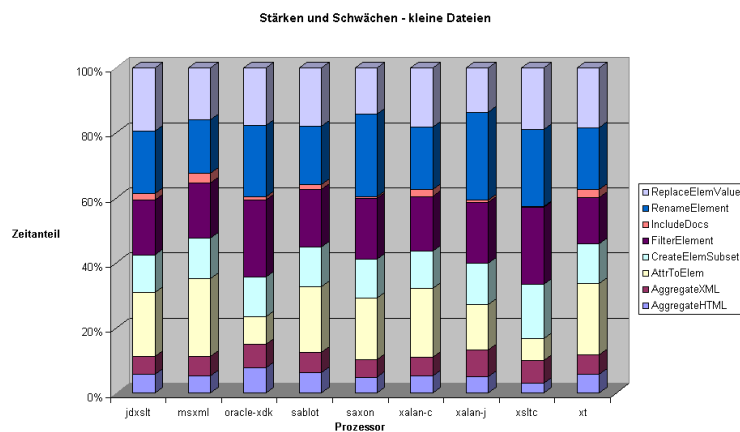


Abbildung 3: Stärken und Schwächen (kleine Testdateien)

Man kann beispielsweise erkennen, daß zwar beim Durchsatz des Testfalls Include-Docs große Unterschiede festzustellen sind, diese außerdem recht klein sind im Vergleich zum Durchsatz der übrigen Testfälle.

An der Säule des Prozessors xslt kann man erkennen, daß dieser Prozessor einen

relativ großen Anteil des gesamten Durchsatzes des Testfalles AggregateHTML liefert, d.h. sein Durchsatz recht groß ist.

Die Grafik 4 ermöglicht den Vergleich der Durchsätze pro Testfall zwischen allen Prozessoren. Die Durchsätze aller Prozessoren wurden für jeden Testfall addiert. Diese Summe entspricht der Gesamthöhe der Säule. Bezogen auf diese Gesamtsumme wurde der prozentuale Anteil des Durchsatzes, die jeder Prozessor für den dargestellten Testfall benötigte, farblich markiert. Dieses Diagramm kann nicht dazu verwendet werden, um die absolute Laufzeit der Prozessoren zu vergleichen, sondern soll zeigen, bei welchen Testfällen die Prozessoren relativ und im Vergleich zueinander gesehen besonders schnell oder langsam sind. Eine kleine Fläche auf den Säulen bedeutet dabei, daß der Prozessor wenig Zeit zur Abarbeitung gebraucht hat.

Beispielsweise kann man in der Säule IncludeDocs sehen, daß msxml einen großen Zeitanteil zur Abarbeitung gebraucht hat, z.B. oracle-xdk jedoch einen recht kleinen Anteil.

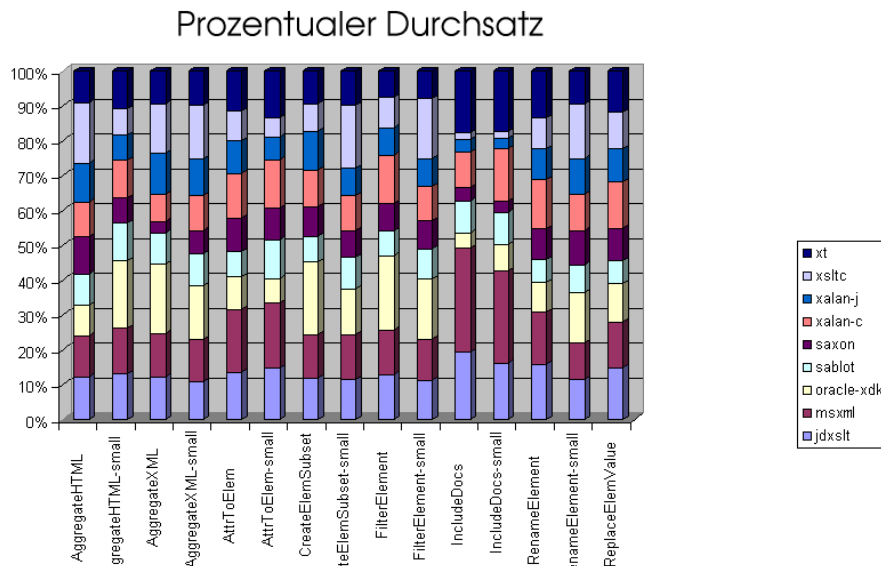


Abbildung 4: Prozentuale (normalisierte) Durchsätze

8.2 Auswertung der Performanz bei großen Testdateien

Die Tabellen im Anhang C zeigen die Auswertung der Testläufe für die großen Testdateien (15-25 kb).

Jede Tabelle zeigt die Ergebnisse aller Testfälle pro Prozessor. Die Spalte „Testfall“ enthält jeweils den Namen des gemessenen Testfalles. In der Spalte „Gewicht“ enthält den Wert, mit dem der Testfall gewichtet ist (4.6). Der Durchsatz stellt den gemessenen Wert dar. Die Spalte „norm.Durchsatz“ wird der Durchsatz normiert dargestellt. Normiert bedeutet in diesem Fall, dass der Prozessor mit dem höchsten bei diesem Testfall gemessenen Durchsatz den normierten Durchsatz 1,0 erhält. Beim Testfall Include-

Docs ist dies zum Beispiel der Prozessor msxml. Der Prozessor jdxslt erhält dagegen als normierten Durchsatz nur 0,64 , da der bei ihm gemessene Durchsatz nur 64% des gemessenen Wertes des Prozessors msxml ausmachte. Die Spalte „norm. Durchsatz * Gewicht“ enthält den normierten Durchsatz, der mit dem Gewicht des Testfalles multipliziert (gewichtet) wurde.

Der normierte Durchsatz wirkt sich durch die Gewichtung bei stärker gewichteten Testfällen mehr aus als bei weniger stark gewichteten Testfällen.

8.2.1 Zusammenfassung der Ergebnisse bei großen Testdateien

Die folgende Tabelle enthält die Summe der gewichteten, normierten Durchsätze jedes Prozessors. Diese Werte werden wiederum nach der bekannten Vorgehensweise normiert, so dass deutlich wird, dass der Prozessor msxml insgesamt die beste Performanz bei der Bearbeitung der großen Testdateien erreichte.

Prozessor	Summe der gewichteten Performanz	norm.Performanz
msxml	14,75	100,00%
jdxslt	13,78	93,43%
xalan-c	11,95	81,04%
oracle-xdk	11,59	78,58%
xt	10,90	73,92%
xsltc	8,15	55,24%
xalan-j	8,05	54,60%
saxon	7,83	53,07%
sablotron	6,80	46,13%

8.3 Graphischer Vergleich der gemessenen Performanz bei großen Dateien

Die folgenden Grafiken geben pro Testfall den Durchsatz in kB/s an, welche die jeweiligen Prozessoren im Durchschnitt (bei jeweils 20 Testläufen) erreicht haben.

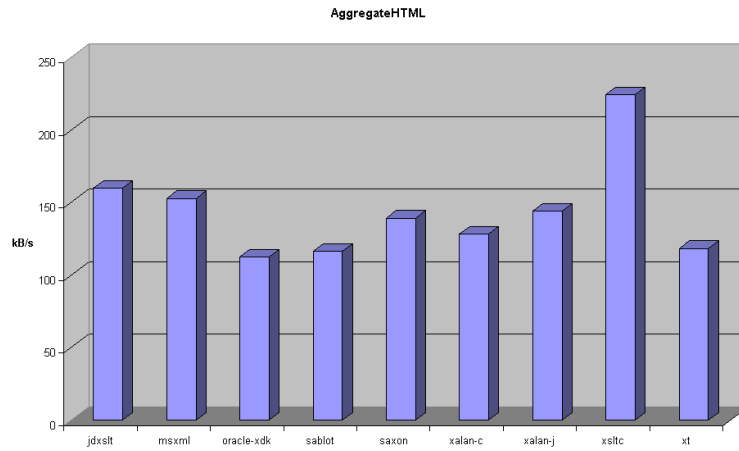


Abbildung 5: Performanzvergleich AggregateHTML

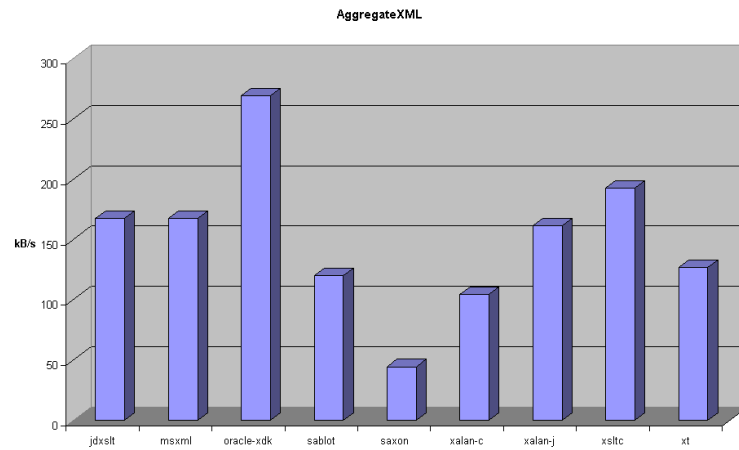


Abbildung 6: Performanzvergleich AggregateXML

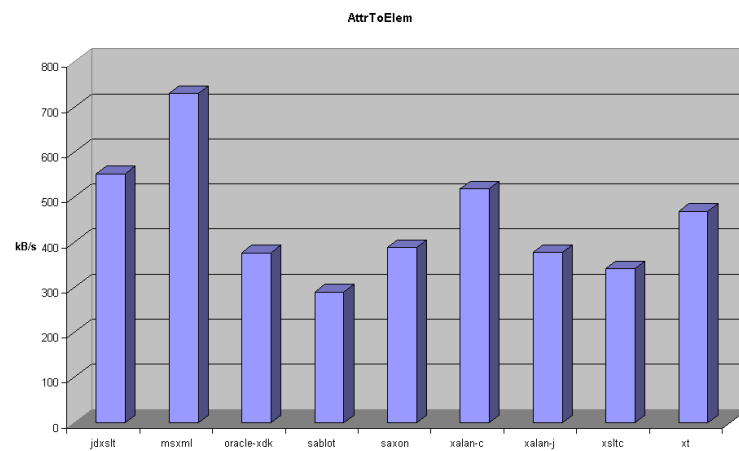


Abbildung 7: Performanzvergleich AttrToElem

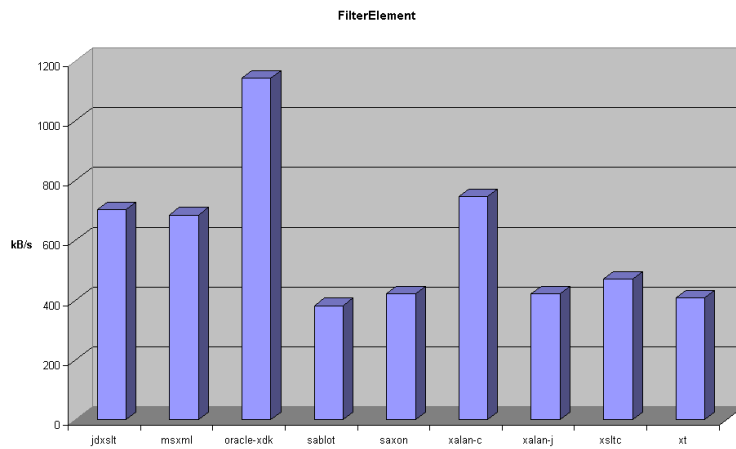


Abbildung 8: Performanzvergleich FilterElement

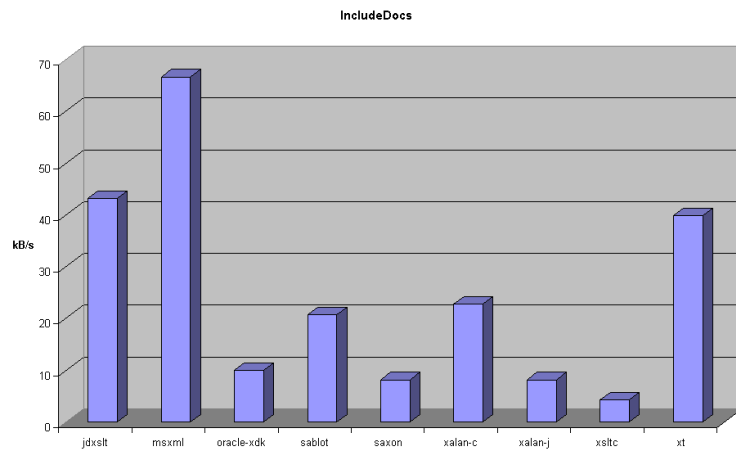


Abbildung 9: Performanzvergleich IncludeDocs

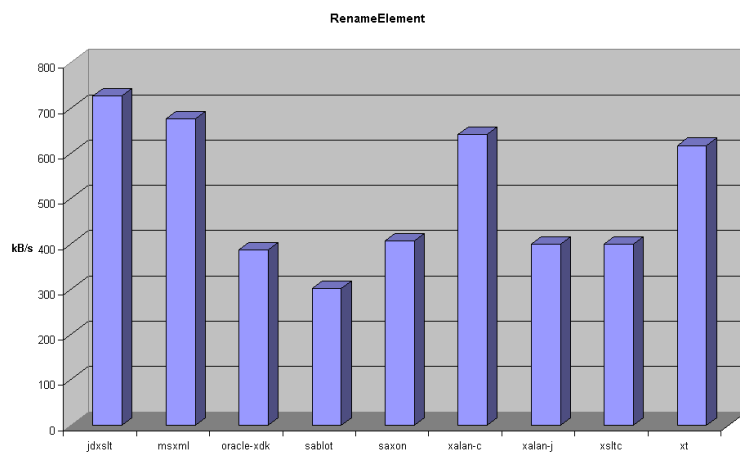


Abbildung 10: Performanzvergleich RenameElement

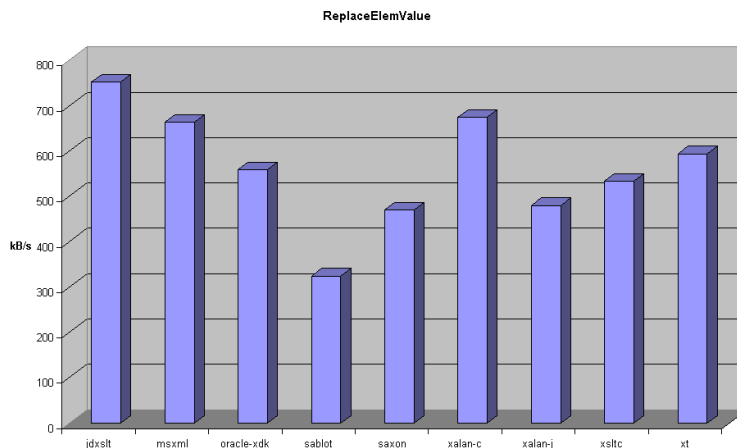


Abbildung 11: Performanzvergleich RenameElemValue

8.4 Auswertung der Performanz bei kleinen Testdateien

Die Tabellen im Anhang werten die gemessenen Ergebnisse pro Prozessor bei der Bearbeitung kleiner Testdateien (1-3kB) aus. Die Spalten der Tabellen haben die gleiche Bedeutung wie die Spalten der Auswertungen bei großen Testdateien im Kapitel 8.2.

8.4.1 Zusammenfassung der Ergebnisse bei kleinen Testdateien

Die folgende Tabelle zeigt die Summen der gewichteten, normierten Durchsätze für die Bearbeitung von kleinen Testdateien pro Prozessor. Die Spalte „norm.Performanz“ enthält die normierten Summen als Endwert für die Performanz bei kleinen Testdateien. Aus der Tabelle geht deutlich hervor, dass auch bei den kleinen Testdateien der Prozessor msxml der Prozessor mit der besten Performanz war.

Prozessor	Summe der gewichteten Performanz	norm.Performanz
msxml	14,18	100,00%
oracle-xdk	13,60	95,91%
xslt	13,18	92,97%
jdxslt	12,86	90,68%
xalan-c	11,32	79,85%
xt	10,81	76,23%
sablotron	9,18	64,78%
saxon	7,78	54,88%
xalan-j	7,61	53,72%

8.5 Graphischer Vergleich der gemessenen Performanz bei kleinen Dateien

Die folgenden Grafiken geben pro Testfall den Durchsatz in kB/s an, welche die jeweiligen Prozessoren im Durchschnitt (bei jeweils 20 Testläufen) erreicht haben.

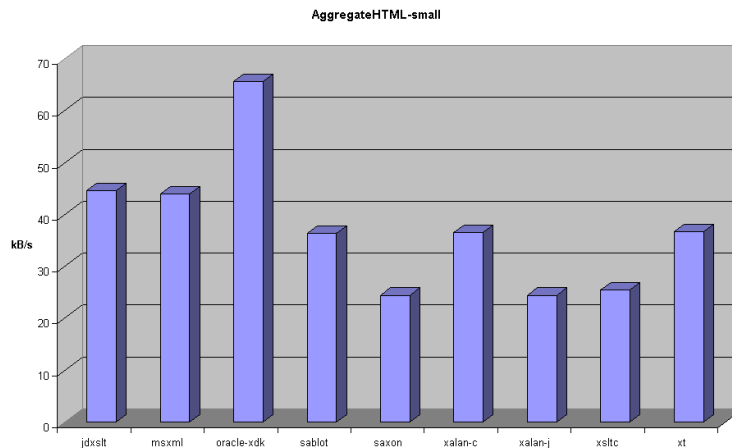


Abbildung 12: Performanzvergleich AggregateHTML-small

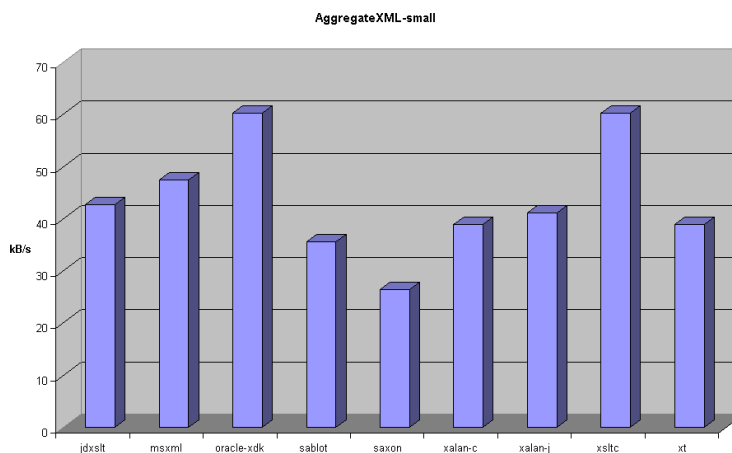


Abbildung 13: Performanzvergleich AggregateXML-small

8.6 Markante Unterschiede der gemessenen Performanz bei großen und kleinen Dateien

Einige markante Unterschiede fallen beim Vergleich der Durchsatz-Meßwerte der Prozessoren aller Testfälle jeweils gruppiert nach großen und kleinen Dateien auf:

1. Der Meßwert für AggregateHTML ist beim Prozessor xslt bei den kleinen Dateien überproportional klein.

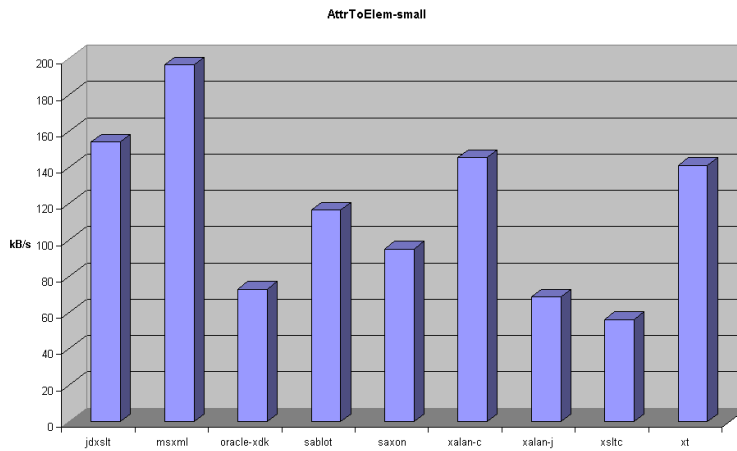


Abbildung 14: Performanzvergleich AttrToElem-small

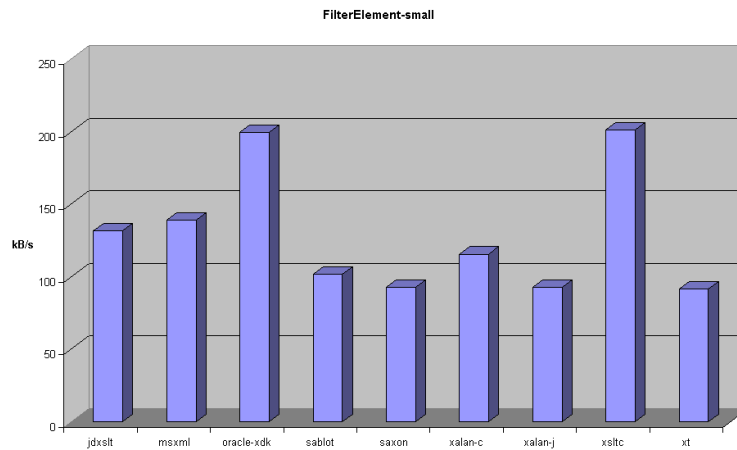


Abbildung 15: Performanzvergleich FilterElement-small

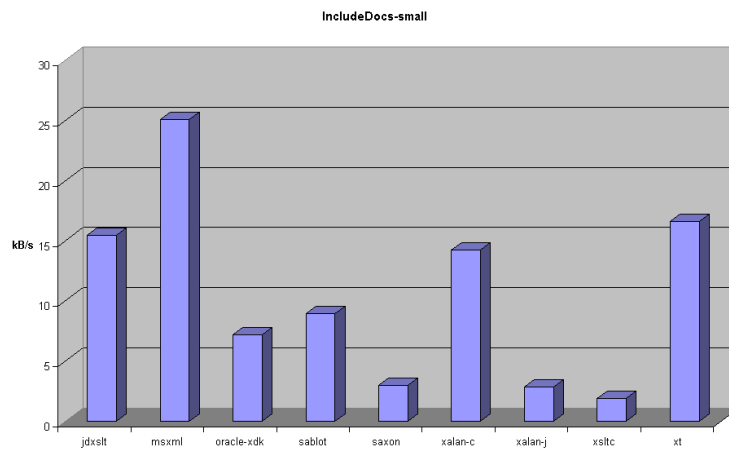


Abbildung 16: Performanzvergleich IncludeDocs-small

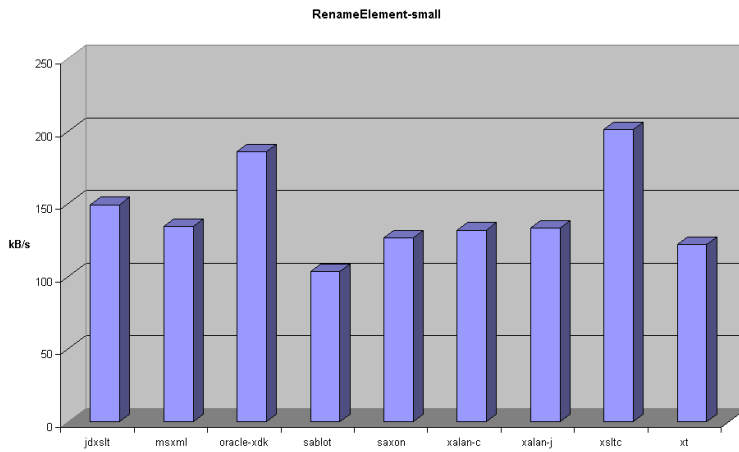


Abbildung 17: Performanzvergleich RenameElement-small

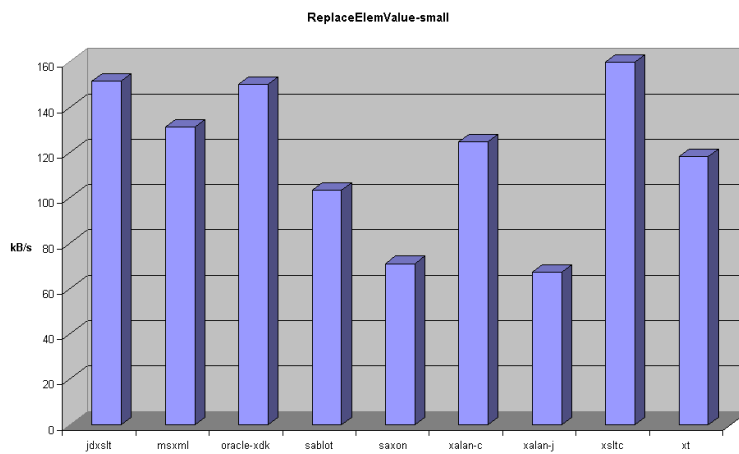


Abbildung 18: Performanzvergleich RenameElemValue-small

2. Der Meßwert für AggregateXML ist beim Prozessor saxon bei den großen Dateien überproportional klein.
3. Der Meßwert für FilterElement ist beim Prozessor xalan-c bei den kleinen Dateien überproportional klein.
4. Der Meßwert für FilterElement ist beim Prozessor saxon bei den kleinen Dateien überproportional groß.
5. Die Meßwerte für RenameElement sind bei den Prozessoren jdxslt, msxml, xalan-c und xt bei den kleinen Dateien überproportional klein. Bei xslt und oracle-xdk ist diese Relation jedoch genau umgekehrt.
6. Der Meßwert für ReplaceElemValue ist beim Prozessor xslt bei den kleinen Dateien überproportional groß.
7. Der Meßwert für ReplaceElemValue ist beim Prozessor xalan-j bei den kleinen Dateien überproportional klein.

8.7 Zusammenfassung der Performanz-Messungen für große und kleine Testdateien

Die Performanzen bei der Bearbeitung von kleinen und großen Dateien haben unterschiedliche Gewichtungen bei der Berechnung der Gesamtperformanz. Die Tabelle „Gewichtung und anschließende Addition der gewichteten Performanzen“ enthält pro Prozessor die normierte Performanz für große und für kleine Dateien, die zugehörigen Gewichte, die sich daraus ergebende gewichtete Performanz und die Summe der gewichteten Performanzen.

Die Summen der gewichteten Performanzen müssen anschließend wiederum normiert werden, um eine korrekte Gewichtung zwischen dem Hauptkriterium Performanz und den Kriterien Integrierbarkeit und Dokumentation zu ermöglichen. Dies geschieht in der Tabelle „Normierung der Summe der gewichteten Performanz“.

Legende

NP(gD)	-	normierte Performanz(große Dateien)
GgD	-	Gewicht für große Testdateien
gew NP(gD)	-	gewichtete Performanz für große Dateien(=NP(gD)*GgD)
NP(kD)	-	normierte Performanz(kleine Dateien)
GkD	-	Gewicht für kleine Testdateien
gew NP(kD)	-	gewichtete Performanz für kleine Dateien(=NP(kD)*GkD)
Summe	-	gew NP(gD)+gew NP(kD)

Gewichtung und anschließende Addition der gewichteten Performanzen

Prozessor	NP(gD)	GgD	gew NP(gD)	NP(kD)	GkD	gew NP(kD)	Summe
msxml	100,00	1	100,00	100,00	2	200,00	300,00
jdxml	93,44	1	93,44	90,69	2	181,38	274,82
oracle-xdk	78,58	1	78,58	95,91	2	191,82	270,41
xslt	55,24	1	55,24	92,97	2	185,95	241,19
xalan-c	81,05	1	81,05	79,85	2	159,70	240,75
xt	73,93	1	73,93	76,24	2	152,48	226,40
saxon	53,08	1	53,08	54,88	2	109,76	162,84
xalan-j	54,60	1	54,60	53,72	2	107,44	162,05
sablotron	46,14	1	46,14	64,78	2	129,57	175,70

Normierung der Summe der gewichteten Performanz

Prozessor	Summe	norm. Performanz	Maximum
msxml	300,00	100,00	300,00
jdxml	274,82	91,61	300,00
oracle-xdk	270,41	90,14	300,00
xslt	241,19	80,40	300,00
xalan-c	240,75	80,25	300,00
xt	226,40	75,47	300,00
saxon	162,84	54,28	300,00
xalan-j	162,05	54,02	300,00
sablotron	175,70	58,57	300,00

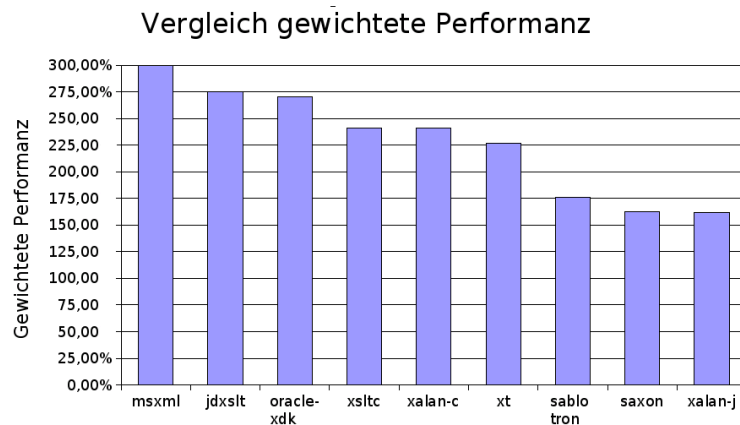


Abbildung 19: Gewichtete Performanzen (große und kleine Dateien zusammen)

8.8 Auswertung der Integrierbarkeit

Die Implementierung des Prozessors in der Programmiersprache Java erleichtert die Integrierbarkeit in das Java-basierte Projektrahmenwerk, während Implementierungen in anderen Sprachen die Integration des Prozessor erschweren aber zumindestens zusätzlichen Aufwand bedeuten. Deshalb erhalten in Java implementierte Prozessoren die Punktzahl 2, während Prozessoren, die in anderen Programmiersprachen umgesetzt

sind nur die Punktzahl 1 erhalten.

Prozessor	Integrierbarkeit	Sprache	norm.Integrierbarkeit
jdxslt	2	Java	100,0
msxml	1	C++	50,0
oracle-xdk	2	Java	100,0
sablot	2	Java	100,0
saxon	2	Java	100,0
xalan-c	1	C++	50,0
xalan-j	2	Java	100,0
xsltc	2	Java	100,0
xt	2	Java	100,0

8.9 Auswertung der Dokumentation

Die Homepage des jdxslt-Projektes war nicht mehr erreichbar, deshalb konnte nur die vorliegende Dokumentation gewertet werden.

Da der XSLTC-Prozessor ein Kommandozeilenprogramm ist, existiert keine eigene API-Dokumentation. Der Prozessor kann aber über die TrAX/JAXP-API einfach in Java-Programme eingebunden werden. Diese APIs sind sehr gut dokumentiert. Aus diesem Grund erhält XSLTC ebenfalls die Punkte für eine vorhandene API-Dokumentation.

Gewicht	Kriterium	Kürzel
1	Installationsanweisung vorhanden	I
1	Beispielanwendungen vorhanden	B
2	Tutorial vorhanden	T
2	Newsgroup/Forum/Mailingliste vorhanden	M
3	API-Dokumentation (Javadoc, Doxydoc) vorhanden	A

In der Spalte „Umfang“ der folgenden Tabelle stehen jeweils die Kürzel der vorhandenen Dokumentationen und Informationsmöglichkeiten.

Prozessor	Dokumentation	norm.Dokumentation	Umfang
jdxslt	3	33,33	I + T
msxml	8	88,89	I + T + M + A
oracle-xdk	3	33,33	B + T
sablot	9	100,00	I + B + T + M + A
saxon	8	88,89	I + T + M + A
xalan-c	9	100,00	I + B + T + M + A
xalan-j	9	100,00	I + B + T + M + A
xsltc	9	100,00	I + B + T + M + A
xt	4	44,44	B + A

8.10 Gesamtauswertung

Die Gesamtauswertung dient dazu, die drei Hauptkriterien Performanz, Integrierbarkeit und Dokumentation zusammenzufassen und daraus ein Gesamtergebnis zu ermitteln. Die Tabelle „Übersicht der normierten Ergebnisse“ stellt nochmals die normierten Ergebnisse eines jeden Prozessor in den drei Hauptkategorien zur besseren Übersicht dar.

Übersicht der normierten Ergebnisse

Prozessor	Performanz	Dokumentation	Integrierbarkeit
msxml	100,00%	88,90%	50,00%
jdxml	91,61%	33,33%	100,00%
oracle-xdk	90,14%	33,33%	100,00%
xsltc	80,40%	100,00%	100,00%
xalan-c	80,25%	100,00%	50,00%
xt	75,47%	44,44%	100,00%
saxon	54,28%	88,90%	100,00%
xalan-j	54,02%	100,00%	100,00%
sablotron	58,57%	100,00%	100,00%

Die errechneten Punktzahlen der drei Hauptkriterien werden nun mit dem festgelegten Gewicht jedes Kriteriums multipliziert. Im Kriterienkatalog wurden folgende Gewichte vergeben: Die Performanz erhielt das Gewicht 3, die Dokumentation das Gewicht 1 und die Integrierbarkeit das Gewicht 2.

Daraus ergeben sich die Werte in der Tabelle „Gewichtete Kriterien, Gesamtsumme und Platzierung“. Die Summe der gewichteten Kriterien ergibt die Gesamtpunktzahl, die jeder Prozessor nach dem aufgestellten Kriterienkatalog erhält. Die Prozessoren mit der höchsten Prozentzahl erfüllen die Anforderungen des Projektrahmenwerkes am Besten und sind somit für den Einsatz innerhalb des Projektes gut geeignet.

Gewichtete Kriterien, Gesamtsumme und Platzierung

Prozessor	gew. Performanz	gew. Dokumentation	gew. Integrierbarkeit	Summe	Platz
msxml	300,00	88,89	100,00	488,89	4
jdxml	274,82	33,33	200,00	508,15	2
oracle-xdk	270,41	33,33	200,00	503,74	3
xsltc	241,19	100,00	200,00	541,19	1
xalan-c	240,75	100,00	100,00	440,75	9
xt	226,40	44,44	200,00	470,84	6
saxon	162,84	88,89	200,00	451,73	8
xalan-j	162,05	100,00	200,00	462,05	7
sablotron	175,70	100,00	200,00	475,70	5

9 Zusammenfassung

Ziel der Fachstudie war es, den für das Projekt „Modellierung von und Exploration in komplexen Unternehmensinformationen“ am Besten geeigneten XSLT-Prozessor zu ermitteln. Um eine faire und nachvollziehbare Auswertung zu ermöglichen, die auch nach dem Ende der Studie bei geänderten Kriteriengewichten benutzbare Ergebnisse liefern kann, wurde eine Nutzwertanalyse durchgeführt.

Die verwendeten Kriterien wurden im Kriterienkatalog erläutert. Der Grad der Bedeutung der einzelnen Kriterien für die Bewertung der Prozessoren wurde durch eine Gewichtung der Kriterien festgelegt. Ein hohes Gewicht ist dabei gleichbedeutend mit einem starken Einfluß des Kriteriums auf den Ausgang der Bewertung und einer großen Bedeutung bezüglich der Anforderungen an den auszuwählenden Prozessor. Bei gleichen Kriterien und geänderten Gewichten kann daher ein komplett anderes Ergebnis entstehen.

9.1 Interpretation der Gesamtauswertung

Im Kapitel 8.10 werden die untersuchten Prozessoren mit den erreichten Punkten pro Kriterium und der Gesamtpunktzahl aufgelistet. Die zwei Prozessoren mit den höchsten Punktzahlen heißen `xsltc` und `jdxslt`. Diese Prozessoren stellen auf Basis des Kriterienkataloges die am Besten geeigneten Kandidaten für das Projektrahmenwerk dar. Die zwei Prozessoren, die bei der Gesamtauswertung am schlechtesten abschnitten, heißen `Saxon` und `Xalan-C`.

Der beste Prozessor `xsltc` ist nur der viertbeste bezüglich der Performanz, konnte aber durch eine sehr gute Dokumentation und die Implementierung in Java wichtige Punkte gutmachen und sich so von den bezüglich der Performanz besseren Prozessoren `msxml`, `jdxslt` und `Oracle-XDK` abheben.

Die Prozessoren `Saxon` und `Xalan-C` erreichten hauptsächlich wegen Ihrer relativ niedrigen Performanz eine schlechte Bewertung. Bei `Xalan-C` wirkte sich zusätzlich die Implementierung in der Programmiersprache `C++` negativ aus, da dies die Integration in das Java-Rahmenwerk erschweren könnte.

9.2 Empfehlung auf Basis des Ergebnisses der Evaluation

Auf der Basis des aufgestellten Kriterienkataloges empfehlen wir, den Prozessor `xsltc` zu benutzen. Ist es wichtig, bei auftretenden Problemen schnell eine Lösung zu finden, aber trotzdem mit einer hohen Performanz rechnen zu können, so ist dieser Prozessor eine sehr gute Wahl. Der Prozessor ist Teil des Open Source Projektes `Xalan-J` der Apache Foundation. Der Quelltext des Prozessors steht dem Benutzer zur Verfügung. Eigene Erweiterungen und die Fehlersuche im Quelltext sollten somit einfach möglich sein. Des weiteren existieren umfangreiche Beispiele und eine Mailingliste, so dass das Lösen von auftretenden Problemen schnell möglich sein sollte.

Wenn ausschließlich Performanzaspekte eine Rolle spielen, so empfehlen wir den Pro-

zessor Oracle-XDK. Der msxml-Prozessor lässt sich in das Projektrahmenwerk schwerer integrieren als der Oracle-XDK, so dass der relativ geringe Performanzunterschied größere Integrationsaufwände nur schwer rechtfertigt. Den Prozessor jdxslt empfehlen wir nicht, weil zum Abschluß der Studie die Internetseite des Prozessors über einen längeren Zeitraum nicht erreichbar war und deshalb nicht klar ist, wie sich die Zukunft dieses schnellen Prozessors weiter gestaltet.

Wird der Prozessor Oracle-XDK aufgrund seiner guten Performanz statt des Prozessors xsltc eingesetzt, muß sich das Projektteam allerdings darüber im Klaren sein, dass bei Problemen weniger schnell eine Lösung gefunden wird, da die Dokumentation nicht so umfangreich ist, wie die des Prozessors xsltc, und insbesondere keine Mailingliste beziehungsweise Newsgroup vorhanden ist. Desweiteren steht der Quelltext nicht zur Verfügung, da es sich nicht um ein Open Source Projekt handelt. Aus diesem Grund werden eigene Erweiterungen erschwert und die Fehlersuche im Quelltext unmöglich.

Literatur

- [1] Eric Armstrong , „Understanding XML“,
Zugriff am 10.12.2003 unter
<http://java.sun.com/webservices/docs/1.0/tutorial/doc/IntroXML.html>
- [2] Eric Armstrong , „The Java API for Xml Processing (JAXP) Tutorial“,
Zugriff am 12.12.2003 unter
<http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/index.html>
- [3] Stuart Halloway , „JDC Tech Tips: June 27, 2000“,
Zugriff am 12.12.2003 unter
<http://java.sun.com/developer/TechTips/2000/tt0627.html>
- [4] James Clark , „XSL Transformations (XSLT) Version 1.0
(W3C Recommendation 16 November 1999)“,
Zugriff am 15.12.2003 unter
<http://www.w3.org/TR/xslt>
- [5] Sharon Adler (IBM) et al. ,
„Extensible Stylesheet Language (XSL)Version 1.0
(W3C Recommendation 15 October 2001)“,
Zugriff am 15.12.2003 unter
<http://www.w3.org/TR/xsl>
- [6] James Clark, Steve DeRose,
„XML Path Language (XPath)Version 1.0
(W3C Recommendation 16 November 1999)“,
Zugriff am 15.12.2003 unter
<http://www.w3.org/TR/xpath>
- [7] XSLTMark-Homepage: Zugriff am 10.01.2004 unter
<http://www.datapower.com/xmldev>
- [8] jd.xslt-Homepage: Zugriff am 15.01.2004 unter
<http://www.aztecrider.com/xslt>
- [9] libxslt-Homepage: Zugriff am 15.01.2004 unter
<http://xmlsoft.org/XSLT>
- [10] MSXML-Homepage: Zugriff am 16.01.2004 unter
<http://msdn.microsoft.com/xml>
- [11] Oracle XDK-Homepage: Zugriff am 15.01.2004 unter
<http://otn.oracle.com/tech/xml>
- [12] Sablotron-Homepage: Zugriff am 16.01.2004 unter
http://www.gingerall.com/charlie/ga/xml/p_sab.xml
- [13] Saxon-Homepage: Zugriff am 15.01.2004 unter
<http://www.saxonica.com>

- [14] Transformiix-Homepage: Zugriff am 15.01.2004 unter
<http://www.mozilla.org/projects/xslt>
- [15] Xalan-J-Homepage: Zugriff am 15.01.2004 unter
<http://xml.apache.org/xalan-j>
- [16] Xalan-C-Homepage: Zugriff am 15.01.2004 unter
<http://xml.apache.org/xalan-c>
- [17] XSLTC-Homepage: Zugriff am 15.01.2004 unter
http://xml.apache.org/xalan-j/xsltc_usage.html
- [18] XT-Homepage: Zugriff am 15.01.2004 unter
<http://www.blz.com/xt>

A Aufgabenstellung

Bewertung von XSLT-Prozessoren

Prüfer : Prof. Dr. B. Mitschang
Bearbeiter : Manuel Früh, Philipp Häuser, Michael Marks
Betreuung : Dipl.Inf. Uwe Heinkel

A.1 Hintergrund

Das Teilprojekt A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ des Sonderforschungsbereiches 467 beschäftigt sich mit der Weiterleitung und Transformation von Datenänderungen aus einem Informationssystem zu abhängigen anderen Informationssystemen. Für die Transformationen wird ein XSLT-Prozessor verwendet, der anhand von XSL-Dokumenten XML-Dokumente umwandelt. Aus diesem Grund spielt der Prozessor eine zentrale Rolle innerhalb des Teilprojektes.

A.2 Aufgabe

Im Rahmen der Fachstudie soll eine Übersicht über die Eigenschaften und Unterschiede verschiedener XSLT-Prozessoren entstehen. Als erstes soll eine möglichst vollständige Liste von XSLT-Prozessoren erstellt werden. Diese Prozessoren sollen anhand eines zuvor erstellten Kriterienkataloges beurteilt werden, dabei soll besonderer Augenmerk auf die Performanz der Prozessoren gelegt werden.

A.3 Teilaufgaben

- Einarbeitung in den Themenbereich
- Aufstellen eines Kriterienkataloges
- Aufbauen einer Testumgebung zur Performanzmessung
- Festlegen relevanter Umwandlungsszenarien
- Festlegen von Testklassen für die Szenarien und Erstellen geeigneter Testfälle
- Vergleich und Bewertung der Prozessoren

B Messwerttabellen

B.1 jdxslt

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	159,91	AggregateHTML-small	44,57
AggregateXML	167,52	AggregateXML-small	42,72
AttrToElem	551,77	AttrToElem-small	153,98
CreateElemSubset	521,38	CreateElemSubset-small	90,11
FilterElement	701,62	FilterElement-small	131,62
IncludeDocs	43,18	IncludeDocs-small	15,45
RenameElement	726,39	RenameElement-small	149,31
ReplaceElemValue	752,27	ReplaceElemValue-small	151,43

B.2 msxslt

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	152,74	AggregateHTML-small	43,95
AggregateXML	167,48	AggregateXML-small	47,42
AttrToElem	730,18	AttrToElem-small	196,59
CreateElemSubset	536,37	CreateElemSubset-small	101,12
FilterElement	684,62	FilterElement-small	138,71
IncludeDocs	66,63	IncludeDocs-small	25,04
RenameElement	675,62	RenameElement-small	134,44
ReplaceElemValue	662,48	ReplaceElemValue-small	131,11

B.3 oracle-xdk

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	112,09	AggregateHTML-small	65,56
AggregateXML	269,01	AggregateXML-small	60,19
AttrToElem	376,45	AttrToElem-small	72,9
CreateElemSubset	896,42	CreateElemSubset-small	103,26
FilterElement	1143,79	FilterElement-small	199,43
IncludeDocs	9,95	IncludeDocs-small	7,16
RenameElement	386,79	RenameElement-small	185,62
ReplaceElemValue	558,59	ReplaceElemValue-small	149,88

B.4 sablotron

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	116,04	AggregateHTML-small	36,31
AggregateXML	119,96	AggregateXML-small	35,6
AttrToElem	289,89	AttrToElem-small	116,34
CreateElemSubset	319,35	CreateElemSubset-small	70,99
FilterElement	381,2	FilterElement-small	101,75
IncludeDocs	20,66	IncludeDocs-small	8,95
RenameElement	301,15	RenameElement-small	103,19
ReplaceElemValue	324,05	ReplaceElemValue-small	103,25

B.5 saxon

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	139,14	AggregateHTML-small	24,43
AggregateXML	44,29	AggregateXML-small	26,38
AttrToElem	387,64	AttrToElem-small	94,99
CreateElemSubset	367,76	CreateElemSubset-small	59,57
FilterElement	421,4	FilterElement-small	92,46
IncludeDocs	8,14	IncludeDocs-small	2,98
RenameElement	406,42	RenameElement-small	126,34
ReplaceElemValue	470,05	ReplaceElemValue-small	70,97

B.6 xalan-c

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	127,9	AggregateHTML-small	36,53
AggregateXML	104,02	AggregateXML-small	38,88
AttrToElem	518,39	AttrToElem-small	145,45
CreateElemSubset	454,98	CreateElemSubset-small	77,87
FilterElement	746,71	FilterElement-small	115,37
IncludeDocs	22,7	IncludeDocs-small	14,22
RenameElement	640,79	RenameElement-small	131,76
ReplaceElemValue	673,28	ReplaceElemValue-small	124,74

B.7 xalan-j

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	144,11	AggregateHTML-small	24,43
AggregateXML	161,41	AggregateXML-small	41,12
AttrToElem	377,44	AttrToElem-small	68,57
CreateElemSubset	478,09	CreateElemSubset-small	61,95
FilterElement	420,66	FilterElement-small	92,46
IncludeDocs	8,14	IncludeDocs-small	2,82
RenameElement	399,66	RenameElement-small	133,32
ReplaceElemValue	479,43	ReplaceElemValue-small	67,03

B.8 xsltc

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	224,18	AggregateHTML-small	25,47
AggregateXML	192,15	AggregateXML-small	60,19
AttrToElem	340,68	AttrToElem-small	56,12
CreateElemSubset	341,49	CreateElemSubset-small	140,8
FilterElement	470,97	FilterElement-small	201,09
IncludeDocs	4,26	IncludeDocs-small	1,88
RenameElement	399,66	RenameElement-small	201,09
ReplaceElemValue	532,58	ReplaceElemValue-small	159,81

B.9 xt

große Testdateien		kleine Testdateien	
Test	Durchsatz kB/s	Test	Durchsatz kB/s
AggregateHTML	118,26	AggregateHTML-small	36,68
AggregateXML	126,45	AggregateXML-small	38,88
AttrToElem	468,75	AttrToElem-small	141,14
CreateElemSubset	404,33	CreateElemSubset-small	77,87
FilterElement	407,72	FilterElement-small	91,58
IncludeDocs	39,8	IncludeDocs-small	16,58
RenameElement	617	RenameElement-small	121,88
ReplaceElemValue	593,65	ReplaceElemValue-small	118,04

C Auswertungstabellen

C.1 jdxslt (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	159,91	0,71	0,71
AggregateHTML	0	167,52	0,62	0,00
AttrToElem	2	551,77	0,75	1,51
CreateElemSubset	2	521,38	0,58	1,16
FilterElement:	4	701,62	0,61	2,45
IncludeDocs	3	43,18	0,64	1,94
RenameElement	4	726,39	1,00	4,00
ReplaceElemValue	2	752,27	1,00	2,00
Summe				13,78

C.2 msxml (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	152,74	0,68	0,68
AggregateHTML	0	167,48	0,62	0,00
AttrToElem	2	730,18	1,00	2,00
CreateElemSubset	2	536,37	0,59	1,19
FilterElement:	4	684,62	0,59	2,39
IncludeDocs	3	66,63	1,00	3,00
RenameElement	4	675,62	0,93	3,72
ReplaceElemValue	2	662,48	0,88	1,76
Summe				14,75

C.3 oracle-xdk (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	112,09	0,50	0,50
AggregateHTML	0	269,01	1,00	0,00
AttrToElem	2	376,45	0,51	1,03
CreateElemSubset	2	896,42	1,00	2,00
FilterElement:	4	1143,79	1,00	4,00
IncludeDocs	3	9,95	0,14	0,44
RenameElement	4	386,79	0,53	2,12
ReplaceElemValue	2	558,59	0,74	1,48
Summe				11,59

C.4 sablotron (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	116,04	0,51	0,51
AggregateHTML	0	119,96	0,44	0,00
AttrToElem	2	289,89	0,39	0,79
CreateElemSubset	2	319,35	0,35	0,71
FilterElement:	4	381,2	0,33	1,33
IncludeDocs	3	20,66	0,31	0,93
RenameElement	4	301,15	0,41	1,65
ReplaceElemValue	2	324,05	0,43	0,86
Summe				6,80

C.5 saxon (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	139,14	0,62	0,62
AggregateHTML	0	44,29	0,16	0,00
AttrToElem	2	387,64	0,53	1,06
CreateElemSubset	2	367,76	0,41	0,82
FilterElement:	4	421,4	0,36	1,47
IncludeDocs	3	8,14	0,12	0,36
RenameElement	4	406,42	0,55	2,23
ReplaceElemValue	2	470,05	0,62	1,24
Summe				7,83

C.6 xalan-c (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	127,9	0,57	0,57
AggregateHTML	0	104,02	0,38	0,00
AttrToElem	2	518,39	0,70	1,41
CreateElemSubset	2	454,98	0,50	1,01
FilterElement:	4	746,71	0,65	2,61
IncludeDocs	3	22,7	0,34	1,02
RenameElement	4	640,79	0,88	3,52
ReplaceElemValue	2	673,28	0,89	1,78
Summe				11,95

C.7 xalan-j (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	144,11	0,64	0,64
AggregateHTML	0	161,41	0,60	0,00
AttrToElem	2	377,44	0,51	1,03
CreateElemSubset	2	478,09	0,53	1,06
FilterElement:	4	420,66	0,36	1,47
IncludeDocs	3	8,14	0,12	0,36
RenameElement	4	399,66	0,55	2,20
ReplaceElemValue	2	479,43	0,63	1,27
Summe				8,05

C.8 xslt (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	224,18	1,00	1,00
AggregateHTML	0	192,15	0,71	0,00
AttrToElem	2	340,68	0,46	0,93
CreateElemSubset	2	341,49	0,38	0,76
FilterElement:	4	470,97	0,41	1,64
IncludeDocs	3	4,26	0,06	0,19
RenameElement	4	399,66	0,55	2,20
ReplaceElemValue	2	532,58	0,70	1,41
Summe				8,15

C.9 xt (große Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	118,26	0,52	0,52
AggregateHTML	0	126,45	0,47	0,00
AttrToElem	2	468,75	0,64	1,28
CreateElemSubset	2	404,33	0,45	0,90
FilterElement:	4	407,72	0,35	1,42
IncludeDocs	3	39,8	0,59	1,79
RenameElement	4	617	0,84	3,39
ReplaceElemValue	2	593,65	0,78	1,57
Summe				10,90

C.10 jdxslt (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	44,57	0,67	0,67
AggregateHTML	0	42,72	0,70	0,00
AttrToElem	2	153,98	0,78	1,56
CreateElemSubset	2	90,11	0,63	1,27
FilterElement:	4	131,62	0,65	2,61
IncludeDocs	3	15,45	0,61	1,85
RenameElement	4	149,31	0,74	2,97
ReplaceElemValue	2	151,43	0,94	1,89
Summe				12,86

C.11 msxml (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	43,95	0,67	0,67
AggregateHTML	0	47,42	0,78	0,00
AttrToElem	2	196,59	1,00	2,00
CreateElemSubset	2	101,12	0,71	1,43
FilterElement:	4	138,71	0,68	2,75
IncludeDocs	3	25,04	1,00	3,00
RenameElement	4	134,44	0,66	2,67
ReplaceElemValue	2	131,11	0,82	1,64
Summe				14,18

C.12 oracle-xdk (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	65,56	1,00	1,00
AggregateHTML	0	60,19	1,00	0,00
AttrToElem	2	72,9	0,37	0,74
CreateElemSubset	2	103,26	0,73	1,46
FilterElement:	4	199,43	0,99	3,96
IncludeDocs	3	7,16	0,28	0,85
RenameElement	4	185,62	0,92	3,69
ReplaceElemValue	2	149,88	0,93	1,87
Summe				13,60

C.13 sablotron (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	36,31	0,55	0,55
AggregateHTML	0	35,6	0,59	0,00
AttrToElem	2	116,34	0,59	1,18
CreateElemSubset	2	70,99	0,50	1,00
FilterElement:	4	101,75	0,50	2,02
IncludeDocs	3	8,95	0,35	1,07
RenameElement	4	103,19	0,51	2,05
ReplaceElemValue	2	103,25	0,64	1,29
Summe				9,18

C.14 saxon (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	24,43	0,37	0,37
AggregateHTML	0	26,38	0,43	0,00
AttrToElem	2	94,99	0,48	0,96
CreateElemSubset	2	59,57	0,42	0,84
FilterElement:	4	92,46	0,45	1,83
IncludeDocs	3	2,98	0,11	0,35
RenameElement	4	126,34	0,62	2,51
ReplaceElemValue	2	70,97	0,44	0,88
Summe				7,78

C.15 xalan-c (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	36,53	0,55	0,55
AggregateHTML	0	38,88	0,64	0,00
AttrToElem	2	145,45	0,73	1,47
CreateElemSubset	2	77,87	0,55	1,10
FilterElement:	4	115,37	0,57	2,29
IncludeDocs	3	14,22	0,56	1,70
RenameElement	4	131,76	0,65	2,62
ReplaceElemValue	2	124,74	0,78	1,56
Summe				11,32

C.16 xalan-j (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	24,43	0,37	0,37
AggregateHTML	0	41,12	0,68	0,00
AttrToElem	2	68,57	0,34	0,69
CreateElemSubset	2	61,95	0,43	0,87
FilterElement:	4	92,46	0,45	1,83
IncludeDocs	3	2,82	0,11	0,33
RenameElement	4	133,32	0,66	2,65
ReplaceElemValue	2	67,03	0,41	0,83
Summe				7,61

C.17 xsltc (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	25,47	0,38	0,38
AggregateHTML	0	60,19	1,00	0,00
AttrToElem	2	56,12	0,28	0,57
CreateElemSubset	2	140,8	1,00	2,00
FilterElement:	4	201,09	1,00	4,00
IncludeDocs	3	1,88	0,07	0,22
RenameElement	4	201,09	1,00	4,00
ReplaceElemValue	2	159,81	1,00	2,00
Summe				13,18

C.18 xt (kleine Dateien)

Testfall	Gewicht	Durchsatz	norm. Durchsatz	norm. Durchsatz * Gewicht
AggregateXML	1	36,68	0,55	0,55
AggregateHTML	0	38,88	0,64	0,00
AttrToElem	2	141,14	0,71	1,43
CreateElemSubset	2	77,87	0,55	1,10
FilterElement:	4	91,58	0,45	1,82
IncludeDocs	3	16,58	0,66	1,98
RenameElement	4	121,88	0,60	2,42
ReplaceElemValue	2	118,04	0,73	1,47
Summe				10,81

D Begriffslexikon

ASF:	Apache Software Foundation
DOM:	Document Object Model
DTD:	Document Type Definition
GPL:	GNU General Public License
MIT:	Massachusetts Institute of Technology
MPL:	Mozilla Public License
OTN:	Oracle Technology Network
SAX:	Simple API for XML
SFB:	Sonderforschungsbereich
Testfall:	Ein Testfall deckt eine Klasse von Transformationen ab, wie z.B. Filterung bzw. Projektion von Elementen und besteht aus mehreren zugehörigen Dateien. Dazu zählen eine XML Eingabedatei, ein XSL Stylesheet und eine Referenzausgabedatei.
XML:	Extensible Markup Language
XSL:	Extensible Stylesheet Language
Stylesheet:	Ein Stylesheet definiert die Transformationsschritte von einer XML Eingabedatei in eine beliebige Ausgabedatei. Dieses Stylesheet wird von dem jeweiligen XSLT Prozessor interpretiert, wobei die Transformationen umgesetzt werden.
XSLT:	Extensible Stylesheet Language Transformations

Erklärung

Hiermit versichern wir, diese Arbeit selbständig verfaßt und nur die angegebenen Quellen benutzt zu haben.

Michael Marks Manuel Früh Philipp Häuser