XML-Kompression und optische Codierung für den Transfer von Geschäftsdokumenten

Inaugural-Dissertation
zur
Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften
an der Wirtschaftswissenschaftlichen Fakultät
der Universität Passau

vorgelegt von
Dipl.-Kfm. Sebastian Schinkinger B.Sc.

Passau
Juni 2016

Erstgutachter: Prof. Dr. Peter Kleinschmidt

Zweitgutachter: Prof. Dr. Franz Lehner

Dissertationsort: Passau

Tag der letzten Fachprüfung des Rigorosums: 23. November 2016

Inhaltsverzeichnis

Inhaltsverzeichnis **I**

Abkürzungsverzeichnis **IV**

Abbildungsverzeichnis **VI**

Tabellenverzeichnis **VII**

1 Einleitung **1**
1.1 Motivation und Problemstellung 1
1.2 Zielsetzung und Methodik 4
1.3 Aufbau der Arbeit 5

2 Grundlagen und Begriffsbestimmung **8**
2.1 Grundlagen zu XML 8
2.1.1 Aufbau eines XML-Dokuments 9
2.1.2 Inhaltsmodelle für Dokumente 11
2.1.3 Themabezogener Vergleich relevanter Schemasprachen . . . 12
2.2 Elektronischer Austausch strukturierter Geschäftsdokumente . . . 16
2.3 Optische Codierung 18
2.3.1 Identifikation relevanter Codes 18
2.3.2 Vergleich der relevanten 2D-Codes 20
2.4 Grundlagen zur Datenkompression 24
2.4.1 Bezug zur Informations- und Codierungstheorie 24
2.4.2 Codierungs- und Kompressionstechniken 25

3 Identifizierung relevanter XML-Kompressionsverfahren **29**
3.1 Zur einschlägigen Literatur 30
3.2 Klassifikationssystem 32
3.2.1 Art und Ziel der Klassifikation 32
3.2.2 Facetten auf Anwendungsebene 34
3.2.2.1 Arten von Verfahren zur XML-Kompression . . . 34
3.2.2.2 Informationserhaltung im Kontext von XML . . . 36
3.2.2.3 Funktionale Perspektive 37
3.2.3 Technische Facetten 38
3.2.3.1 Art der Strukturkompression 39
3.2.3.2 Reduktion der Strukturredundanzen 43
3.2.3.3 Unterstützte Schemasprachen 44
3.2.3.4 Facetten der Nutzdatenkompression 46

3.3 Prozess zur Auswahl der Verfahren . . . 49
3.3.1 Grundlegende Anforderungen . . . 49
3.3.2 Eingrenzende Auswahlkriterien . . . 50
3.3.2.1 Funktionale Kriterien . . . 50
3.3.2.2 Ungeeignete Arten der Strukturkompression . . . 52
3.3.2.3 Varianten- und Bezeichnungscodierung im Vergleich 54
3.3.3 Analyse der eingegrenzten Verfahren . . . 55
3.4 Verfahren mit Bezeichnungscodierungen . . . 57
3.4.1 Archivierende Verfahren . . . 57
3.4.1.1 Implementierte Verfahren . . . 58
3.4.1.2 Nicht implementierte Verfahren . . . 61
3.4.2 Verfahren mit Abfragenunterstützung . . . 62
3.5 Verfahren mit Variantencodierungen . . . 66
3.5.1 Automatenbasierte Verfahren . . . 67
3.5.2 Baumbasierte Verfahren . . . 73
3.6 Untersuchungsergebnisse und deren Implikationen . . . 79
3.6.1 Bezeichnungscodierungen . . . 79
3.6.2 Variantencodierungen . . . 79

4 Parametrisiertes XML-Kompressionsverfahren 81
4.1 Schematischer Aufbau des Verfahrens . . . 82
4.2 Konzepte bestehender Verfahren . . . 84
4.2.1 Modellierung der Schemadefinition . . . 84
4.2.1.1 Modellierung als Baum . . . 84
4.2.1.2 Modellierung über Automaten . . . 86
4.2.2 Abbildung der Struktur eines XML-Dokuments . . . 88
4.2.2.1 Baumbasierte Abbildung . . . 89
4.2.2.2 Automatenbasierte Abbildung . . . 91
4.2.3 Behandlung von Redundanz in der Struktur . . . 92
4.2.4 Codierung der Strukturabbildung . . . 93
4.2.5 Kompression der Nutzdaten . . . 95
4.3 Gestaltung der Funktionsbausteine . . . 95
4.3.1 Modellierung der Schemadefinition . . . 96
4.3.2 Abbildung der Struktur eines XML-Dokuments . . . 97
4.3.3 Reduzierung von Redundanz in der Struktur . . . 102
4.3.4 Codierung der Strukturabbildung . . . 104
4.3.5 Kompression der Nutzdaten . . . 107
4.4 Prototypische Implementierung . . . 112
4.4.1 Die Verarbeitungsprozesse und deren Komponenten . . . 112
4.4.1.1 Der Kompressionsprozess . . . 112
4.4.1.2 Der Dekompressionsprozess . . . 117
4.4.2 Die objektorientierte Architektur . . . 118

5 Fallstudie zur optischen Codierung von Rechnungsdokumenten 127
5.1 Grundlagen zur Fallstudie . . . 127

5.1.1 Echtdaten als Basis 128
5.1.2 Technischer Rahmen 129
5.1.3 Verwendete Universalkompressoren 130
5.1.4 Ermittlung und Verdichtung von Messwerten 132
5.2 Leistungsvergleich der Konzeptkombinationen 136
5.2.1 Strukturverarbeitungskonzepte mit Einfluss auf nachgelagerte Universalkompressoren 136
5.2.1.1 Einfluss der Kürzung redundanter Strukturteile . 137
5.2.1.2 Einfluss der alternativen Häufigkeitsabbildung . . 141
5.2.2 Varianten der Strukturkompression 142
5.2.3 Varianten der Nutzdatenkompression 144
5.2.4 Ergebnis des Vergleichs der Konzepte 147
5.3 Leistungsvergleich ausgewählter Kompressionsverfahren 148
5.3.1 Überblick über die untersuchten Verfahren 148
5.3.1.1 Schemaneutrale XML-Kompressoren 149
5.3.1.2 Schemabasierte XML-Kompressoren 150
5.3.2 Ergebnis des Vergleichs der Kompressionsverfahren 153
5.4 Auswirkung von Kompressionsleistungsunterschieden auf die optische Codierung 155
5.4.1 Bestandteile des Vergleichs 156
5.4.2 Die Auswertung 157
5.4.3 Zusammenfassung 159

6 Fazit und Implikationen für weitere Untersuchungen 160

Anhang VIII
A Aggregierte Messwerte für Konfigurationen der Strukturkompression VIII
B Aggregierte Messwerte für Konfigurationen der Nutzdatenkompression XII

Quellenverzeichnis XVI

Abkürzungsverzeichnis

ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
B2B	Business-to-Business
BWT	Burrows-Wheller-Transformation
CCTS	Core Components Technical Specification
CD	Compact Disk
CSV	Comma-separated values
DAG	Directed Acyclic Graph
DEA	deterministischer endlicher Automat
DOM	Document Object Model
DTD	Document Type Definition
DVD	Digital Versatile Disc
EA	endlicher Automat
ebXML	Electronic Business using XML
ECC	Error Correcting Code
EDI	Electronic Data Interchange
EDIFACT	EDI for Administration, Commerce and Transport
EXI	Efficient XML Interchange
FB WI	Fachbereich Wirtschaftsinformatik
FeRD	Forum elektronische Rechnung Deutschland
GI	Gesellschaft für Informatik
GS1	Global Standard One
GS1 XML	Global Standard One XML
HTML	Hypertext Markup Language
ISO	Internationale Organisation für Normung
IT	Informationstechnologie
JAVA SE	Java Plattform, Standard Edition
JAXP	Java API for XML Processing

KST	Kleene Size Tree
LZMA	Lempel-Ziv-Markow-Algorithmus
MDT	Model Development Tools
MHM	Multiplexed Hierarchical Modelling
MPM	Multilevel Pattern Matching
OAGIS	Open Applications Group Integration Specification
OASIS	Organization for the Advancement of Structured Information Standards
PDF	Portable Document Format
PMC	PaperMatic-Code
PPM	Prediction by Partial Matching
PPMII	PPM with Information Inheritance
PPMVC	PPM with variable-length contexts
PSXC	Parametric Schema-based XML Compression
QR Code	Quick-Response Code
RELAX NG	Regular Language Description for XML New Generation
SEPA	Single European Payment Area
SGML	Standard Generalized Markup Language
SIT	Structure Index Tree
UBL	Universal Business Language
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WBXML	WAP Binar XML Content Format
WKWI	Wissenschaftliche Kommission Wirtschaftsinformatik
XML	Extensible Markup Language
xmlsn	XML-Namensraum
XSD	XML Schema Definition
ZUGFeRD	Zentraler User Guide des Forums elektronische Rechnung Deutschland

Abbildungsverzeichnis

2.1 Beispielhaftes XML-Dokument in gekürzter Form 10
2.2 Platzbedarf ausgewählter Maxtrixcodes in Anzahl an Zellen . . . 23

3.1 Facetten auf Anwendungsebene zur Unterscheidung von Kompressionsverfahren für XML-Dokumente 35
3.2 Technische Facetten zur Unterscheidung von Kompressionsverfahren für XML-Dokumente 40

4.1 Schematischer Kompressionsprozess mit Variantencodierung . . . 83
4.2 Modellierung als Baum bei *XCQ* 85
4.3 Ausschnitt eines Schemabaums bei *XSDS* 86
4.4 Automat des Inhaltsmodells eines Adressbucheintrags 87
4.5 Beispielhaftes Inhaltsmodell eines Adresseintrags in XSD 99
4.6 Modellierung des Beispiels als Baum 100
4.7 Modellierung des Beispiels als Automat 100
4.8 Adresseintrag Beispiel 1 101
4.9 Adresseintrag Beispiel 2 101
4.10 Hierarchische Gruppierung der Nutzdaten 109
4.11 Hierarchie der vordefinierten Datentypen bei XSD 110
4.12 Prozess der Kompression eines XML-Dokuments 113
4.13 Prozess der Dekompression eines XML-Dokuments 117
4.14 Klassendiagramm des Prototyps 119

5.1 Beschaffenheit der Echtdaten 129
5.2 Kantenlängen in Zentimeter von Aztec, Data Matrix und QR Code für unterschiedliche Kompressionsverfahren 158

Tabellenverzeichnis

2.1 Vergleich der Notationen von Kompositoren in DTD, XSD und RELAX NG 14
2.2 Vergleich der Notationen der Häufigkeitsdefinitionen in DTD und RELAX NG 15

5.1 In der Fallstudie berücksichtigte Universalkompressoren 131
5.2 Einstellungen konfigurierbarer Universalkompressoren 132
5.3 Auswirkungen der Kürzung redundanter Strukturteile bei automatenbasierter Modellierung 138
5.4 Auswirkungen der Kürzung redundanter Strukurteile bei baumbasierter Modellierung 139
5.5 Effekt der Kürzung redundanter Strukturteile 140
5.6 Auswirkungen der alternativen Häufigkeitsabbildung bei baumbasierter Modellierung 141
5.7 Effekt der alternativen Häufigkeitsabbildung 142
5.8 Aggregierte Messwerte relevanter Konfigurationsvarianten der Strukturkompression 143
5.9 Aggregierte Messwerte relevanter Konfigurationsvarianten der Nutzdatenkompression 145
5.10 Parameterwerte von PSXC für optimale Konzeptkombinationen . 148
5.11 Bezugsquellen der untersuchten XML-Kompressoren 149
5.12 Simulierte Verfahren und die entsprechenden Konfigurationen von *PSXC* 151
5.13 Vergleich ausgewählter Kompressoren 154
5.14 Kantenlängen in Zentimeter von Aztec, Data Matrix und QR Code für unterschiedliche Kompressionsverfahren 157

1 Einleitung

1.1 Motivation und Problemstellung

Die Globalisierung und der damit einhergehende Wettbewerbsdruck führen zunehmend dazu, dass sich Unternehmen auf ihre Kernkompetenzen konzentrieren. Mit dieser Entwicklung wächst die Zahl der an einer Wertschöpfungskette beteiligten Unternehmen und in Folge dessen auch die Häufigkeit des Austauschs von Geschäftsdaten in unternehmensübergreifenden Geschäftsprozessen. Dies erfordert einen effizienten Datenaustausch über Unternehmensgrenzen hinweg und damit den verstärkten Einsatz von Informations- und Kommunikationstechnologien.[1] Dabei ist es wichtig, dass die elektronischen Technologien nicht isoliert verwendet, sondern so kombiniert werden, dass eine durchgängig automatisierte Verarbeitung über die Unternehmensgrenzen hinweg möglich ist. Die Sendung eines Geschäftsdokuments – wie beispielsweise einer Bestellung, Rechnung oder Lieferavisierung – als PDF per E-Mail beschleunigt zwar den Prozess im Vergleich zum papierbasierten Versand, ermöglicht aber keine automatisierte Weiterverarbeitung.

Die Bestrebungen, Daten in elektronischer Form zwischen Unternehmen auszutauschen, gehen in die 1970er Jahre zurück.[2] Dennoch ist aktuell der elektronische Austausch in einer Form, die eine automatische Verarbeitung ohne Medienbruch ermöglicht, die Ausnahme. Dies verdeutlicht eine groß angelegte Studie zur „Nutzung von Informations- und Kommunikationstechnologien in Unternehmen“ des Statistischen Bundesamts aus dem Jahr 2015.[3] Laut dieser tauschen weniger als 26 % der Unternehmen Informationen zur Organisation der Lieferkette elektronisch aus und

[1] Für diesen und den nächsten Satz vgl. Bächle/Lehmann (2010), S. 3 f.
[2] Fink/Schneidereit/Voss (2005) S. 247 u. Laudon/Laudon/Schoder (2016) S. 464
[3] Vgl. Statistisches Bundesamt (2015).

lediglich 16 % nutzen eine Form, die automatisiert weiterverarbeitet werden kann.[4] Bei der Rechnungsstellung und -verarbeitung bietet sich ein ähnliches Bild, wie die obige Studie in Bezug auf die B2B-Rechnungserstellung zeigt. Demnach versenden 41 % der Unternehmen elektronische Rechnungen und nur 10 % der Unternehmen tun dies in einer Form, die eine automatisierte Weiterverarbeitung ermöglicht.[5] Mit einer Verbreitung von 97 % kommt die Papierrechnung noch nahezu in allen Unternehmen zum Einsatz. Deshalb überrascht es nicht, dass 80 % der deutschen Unternehmen mehr als 50 % ihrer Rechnungen in Papierform oder einer elektronischen Form erhalten, die keine automatisierte elektronische Weiterverarbeitung ermöglicht.[6]

Dokumente dieser Art erzeugen einen Medienbruch, unabhängig davon, ob sie in papierbasierter oder elektronischer Form vorliegen. Durch deren Ausrichtung auf die Verarbeitung durch Menschen und den damit einhergehenden bildlichen Charakter der Darstellung geht die Semantik für eine automatische Weiterverarbeitung verloren.

Um dennoch eine maschinelle Verarbeitung zu ermöglichen, existieren Produkte, die versuchen, aus der bildlichen Darstellung des Dokuments die nötigen Daten zu extrahieren und deren Semantik zu rekonstruieren. Im Bereich der Rechnungseingangsverarbeitung sind diese Lösungen aufgrund ihrer Kosten jedoch nicht für alle Unternehmen rentabel.[7] Zudem liegt der Durchschnitt der Quote der erkannten Rechnungen derzeit bei nur rund 80 %[8] und ist nur bei selten wechselnden Geschäftspartnern effizient genug[9].

Eine aktuelle Initiative zur Förderung der elektronischen Rechnung in Deutschland bildet das *Forum elektronische Rechnung Deutschland (FeRD)*.[10] Als Resultat seiner Arbeit entstand ein einheitliches Datenformat mit dem Namen ZUGFeRD

[4] Vgl. Statistisches Bundesamt (2015), S. 22.
[5] Für diesen und den nächsten Satz vgl. Statistisches Bundesamt (2015), S. 27.
[6] Vgl. Statistisches Bundesamt (2015), S. 28.
[7] Vgl. Baterip (2013), S. 41-48.
[8] Vgl. Schulz (2014), S. 67.
[9] Vgl. Bergmann u. a.(2014), S. 13.
[10] Vgl. Bergmann u. a.(2014), S. 9.

(Zentraler User Guide des Forums elektronische Rechnung Deutschland).[11] Es wurde mit dem Ziel entwickelt, entsprechend leicht handhabbar zu sein wie Papierrechnungen und dennoch ohne Medienbruch verarbeitet werden zu können.[12] Da daher der Austausch ohne vorherige Absprache mit den Geschäftspartnern ermöglicht werden soll, ist es nötig, dass Rechnungen sowohl herkömmlich manuell als auch automatisiert elektronisch weiterverarbeitet werden können.[13] Das ZUGFeRD-Format kombiniert dafür das Rechnungsbild mit den strukturierten Rechnungsdaten, indem in einem PDF-Dokument eine XML-Datei eingebettet wird.

Das ZUGFeRD-Format verhindert Medienbrüche jedoch lediglich im elektronischen Bereich. Eine Möglichkeit, strukturierte Daten und bildliche Darstellung auch bei papierbasierten Dokumenten zu kombinieren, liefert die Verwendung von optischen Codes, wie beispielsweise Aztec, Data Matrix oder QR Code. Diese ermöglichen die Speicherung und damit den Transport digitaler Daten auf Papier.

Im Jahr 2007 veröffentlichte das Unternehmen UnITeK GmbH einen auf dem Data Matrix Code basierenden Vorschlag namens PaperMatic-Code (PMC).[14] Ihr Ansatz konnte sich offensichtlich nicht durchsetzen, da die Firma ihre Bestrebungen in diese Richtung mittlerweile aufgegeben hat und abgesehen von der initialen Veröffentlichung keine Anhaltspunkte für einen Einsatz des Codes gefunden werden konnten. Einer der Nachteile der Lösung ist die Verwendung eines eigenen, nicht standardisierten Datenformats, das ein Problem für die Adaption in Unternehmen darstellt. Des Weiteren wird der PMC aufgrund fehlender Datenkompression sehr groß, wodurch sich besondere Anforderungen an das Layout eines Dokuments ergeben und die Nutzungsmöglichkeiten des Codes eingeschränkt werden.

Das Ziel, optische Codes zur Speicherung von Geschäftsdokumenten zu verwenden, um Prozesse in der unternehmerischen Praxis zu unterstützen, erfordert es, die Anforderungen dieses Umfelds zu berücksichtigen. In erster Linie betrifft dies das zu verwendende Datenformat. Hier empfiehlt es sich, XML zu verwenden, da

[11] Vgl. Bergmann u. a.(2014), S. 10.
[12] Für diesen und den nächsten Satz vgl. Bergmann u. a.(2014), S. 12 f.
[13] Für diesen und den nächsten Satz vgl. Bergmann u. a.(2014), S. 12, 15 u. 18-20.
[14] Vgl. UnITeK GmbH (2007) S. 1-22.

dies zum einen der aktuell wichtigste Standard zum Austausch von Daten zwischen Informationssystemen ist[15] und zum anderen im Bereich des elektronischen Datenaustauschs etabliert ist[16].

Durch die Verwendung von XML benötigen bereits kleine Geschäftsdokumente mehr als 3 kB Speicherbedarf und übersteigen damit die Kapazität aktueller 2D-Codes[17]. Stärker als diese technische Obergrenze wirkt sich jedoch die Notwendigkeit aus, den optischen Code klein zu halten und damit möglichst wenig Daten darin zu speichern, um eine problemlose Integration in das Rechnungsbild zu ermöglichen.

Folglich ist die zentrale Herausforderung und damit auch der zentrale Untersuchungsaspekt dieser Arbeit die Kompression XML-basierter Geschäftsdokumente. Hierzu wurde ein Verfahren entwickelt, mit dem Konzepte bestehender Verfahren rekombiniert werden können, um damit Verbesserungspotentiale zu identifizieren. Zur Verifizierung des Verfahrens wurde im Rahmen einer umfangreichen Fallstudie gezeigt, dass die Leistung bestehender Kompressionsverfahren übertroffen werden kann.

1.2 Zielsetzung und Methodik

Ausgehend von der im vorherigen Abschnitt dargelegten Problemstellung widmet sich diese Arbeit der folgenden zentralen Forschungsfrage:

> Inwiefern kann die Leistung bestehender Kompressionsverfahren für die optische Codierung XML-basierter Geschäftsdokumente verbessert werden?

Zur Beantwortung dieser Frage wird in der Arbeit das Ziel verfolgt, ein Verfahren zu entwickeln, mit dem für ein bestimmtes Anwendungsszenario die bestmögliche Kompression ermittelt werden kann.

[15] Vgl. Hansen/Mendling/Neumann (2015), S. 466.
[16] Kabak/Dogac (2010) S. 2 u. Bächle/Lehmann (2010) S. 52.
[17] Vgl. dazu die Ausführungen über optische Codierung in Abschnitt 2.3.

Mit der Zielsetzung folgt die Arbeit dem Leitgedanken der Wirtschaftsinformatik als praktisch-normativ ausgerichtete Wissenschaft.[18] Im interdisziplinären Spannungsfeld zwischen Betriebswirtschaftslehre und Informatik greift sie eine Problemstellung der betrieblichen Realität auf und verwendet Techniken der Informatik, um unternehmerische Prozesse zu unterstützen und einen Nutzen für die Praxis zu schaffen.[19] Da in der Arbeit ein Werkzeug erstellt wird, mit dem die Nutzung von Informationssystemen durch einen gesteigerten Grad der Automatisierung verbessert werden kann, verfolgt sie ferner einen konstruktions- bzw. gestaltungsorientierten Forschungsansatz.[20]

Das Verfahren zur Ermittlung der maximal möglichen Kompression wird in einem dreistufigen Prozess[21] über argumentativ-deduktive Analysen[22] entwickelt. Den ersten Schritt bildet die Erstellung einer Klassifikation für Verfahren zur XML-Kompression. Diese wird im zweiten Schritt verwendet, um eine Gruppe von relevanten Verfahren identifizieren zu können. Daraus werden im letzten Schritt sinnvolle Konzepte extrahiert, um sie in einem Verfahren zu vereinen, das dazu verwendet werden kann, die beste Kombination der Konzepte zu bestimmen. Zur Evaluation des entwickelten Verfahrens wird ein Prototyp[23] implementiert und auf Basis einer Fallstudie die Verfahrensqualität im Sinne der Kompressionsleistung dynamisch analysiert[24].

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in sechs Kapitel unterteilt. In einem **einleitenden Kapitel** wird zunächst die Problemstellung aus anwendungsorientierter Sicht dargelegt

[18] Vgl. Ulrich (1984), S. 33.
[19] Vgl. WKWI/GI FB WI (2011).
[20] Vgl. Wilde/Hess (2007) S. 281 i.V.m. Design Science Research Guidelines Hevner u. a.(2004) S. 83.
[21] Vgl. Vorgehensweise anwendungsorientierter Forschung nach Ulrich (1985) S. 2-32.
[22] Zur Definition der argumentativ-deduktive Analyse im Methodenspektrum der Wirtschaftsinformatik vgl. Wilde/Hess (2007) S. 282.
[23] Zur Definition von Prototyping im Methodenspektrum der Wirtschaftsinformatik vgl. Wilde/Hess (2007) S. 282.
[24] Zur Definition der Fallstudie und dynamischen Analyse im Rahmen der Evaluierung eines IT-Artefakts vgl. Hevner u. a.(2004) S. 85-87.

und zeigt auf, wie diese mit XML-Kompression und optischer Codierung umgegangen werden kann. Anschließend wird die sich daraus ergebende, zentrale Forschungsfrage dieser Arbeit formuliert und das Forschungsdesign beschrieben.

Im **zweiten Kapitel** werden die technischen Grundlagen dieser Arbeit behandelt. Nach der Beschreibung themarelevanter Konzepte der Extensible Markup Language (XML) im ersten Teil, folgt ein Überblick über die Technologien, die im Rahmen des elektronischen Austauschs strukturierter Geschäftsdokumente eingesetzt werden. Im Anschluss daran wird die optische Codierung betrachtet und identifiziert, welche Typen optischer Codes für den anfangs dargelegten Anwendungshintergrund geeignet sind. Der letzte Teil liefert eine kurze Einführung in die Datenkompression und gibt einen Überblick über die in dieser Arbeit eingesetzten Codierungs- und Kompressionstechniken.

Das **dritte Kapitel** behandelt den Stand der Forschung zur XML-Kompression und identifiziert im Zuge dessen Verfahren, die sich zur Kompression von XML-basierten Geschäftsdokumenten eignen. Dabei wird zunächst erklärt, dass die Literatur für die Kompression XML-basierter Geschäftsdokumente keinen ausreichenden Vergleich der Leistungsfähigkeit bestehender Verfahren bietet und daher eine eigene Untersuchung nötig ist. Da das Feld der XML-Kompression sehr umfangreich ist, wird im Abschnitt 3.2 ein Klassifikationssystem entwickelt, auf dessen Basis die zu untersuchenden Verfahren eingegrenzt werden können.

Im Abschnitt 3.3 folgt die Beschreibung des dreistufigen Prozesses, der das Klassifikationssystem instrumentalisiert und damit Verfahren identifiziert, die sich potentiell zur Kompression XML-basierter Geschäftsdokumente eignen. Das Ergebnis der ersten beiden Prozessschritte sind zwei Gruppen von Verfahren, die in den Abschnitten 3.4 und 3.5 vorgestellt und geprüft werden, inwiefern sie für den quantitativen Vergleich der späteren Fallstudie relevant sind. Der letzte Teil des Kapitels fasst die Erkenntnisse zusammen und liefert damit die Grundlage für die beiden folgenden Kapitel.

Ausgehend von dem im vorherigen Kapitel aufgedeckten Bedarf, die Rekombination von Konzepten der schemabasierten XML-Kompression zu untersuchen, be-

handelt das **vierte Kapitel** die Entwicklung eines parametrisierten Verfahrens für diese Aufgabe. Dazu werden im ersten Abschnitt der grundsätzliche Ablauf des Verfahrens und die technischen Konzepte aus den im dritten Kapitel behandelten Verfahren extrahiert. Im anschließenden Teil wird technisch analysiert, welche Konzepte in dem zu entwickelnden Verfahren zu berücksichtigen sind. Zudem wird dort die Systematik der Parameter des Verfahrens beschrieben, die dem Verfahren ermöglicht, verschiedene Konzeptkombinationen zu realisieren. Im letzten Teil wird die prototypische Implementierung des Verfahrens beschrieben.

Im **fünften Kapitel** wird das parametrisierte Verfahren im Rahmen einer Fallstudie evaluiert. Im ersten Teil dieses Kapitels werden die Grundlagen der Fallstudie beschrieben, zu denen beispielsweise die verwendeten Echtdaten und die Untersuchungsmodalitäten gehören. Im zweiten Teil wird die optimale Konfiguration des parametrisierten Verfahrens für den Anwendungsfall bestimmt, um diese im darauf folgenden Abschnitt 5.3 mit den im dritten Kapitel ausgewählten Verfahren zu vergleichen. Der letzte Teil der Fallstudie beleuchtet die Auswirkungen unterschiedlicher Kompressionsleistungen auf die Größe der resultierenden optischen Codierung XML-basierter Geschäftsdokumente.

Das abschließende **sechste Kapitel** beginnt mit einer Zusammenfassung der Arbeit und geht dabei auf den wissenschaftlichen Beitrag im Rahmen der Zielsetzung ein. Bevor letztlich weiterführende Forschungsarbeiten angeregt werden, wird der potentielle Einfluss der Arbeit auf die Praxis beschrieben.

2 Grundlagen und Begriffsbestimmung

2.1 Grundlagen zu XML

> „XML ist heute der mit Abstand wichtigste Standard für den Austausch von Daten zwischen Informationssystemen“[1]

XML steht für Extensible Markup Language und ist eine Metasprache zur Definition anwendungsspezifischer Auszeichnungssprachen.[2] Eine Auszeichnungssprache ermöglicht es, Daten auf textueller Ebene so zu repräsentieren, dass textuellen Elementen auf deklarative Weise eine Bedeutung und somit eine Semantik zugeordnet werden kann.[3] Mit XML lassen sich somit Klassen von Dokumenten, wie beispielsweise Bestellungen, so beschreiben, dass sie maschinell verarbeitet werden können.[4]

Vor der Entwicklung von XML wurde seit 1969 die Standard Generalized Markup Language (SGML) zur Beschreibung von Dokumentenstrukturen verwendet.[5] Eine populäre Auszeichnungssprache auf Basis von SGML ist die Hypertext Markup Language (HTML). Die Komplexität von SGML einerseits und die mangelnde Erweiterbarkeit von Auszeichnungssprachen wie HTML andererseits, begünstigten die Entwicklung von XML. Dies führte dazu, dass das World Wide Web Consortium (W3C) 1998 eine Empfehlung (Recommendation) zu XML veröffentlichte.[6] Die XML-Syntax wurde als leichter beherrschbare Teilmenge der SGML-Syntax gestaltet. Das W3C ist ein inoffizielles Gremium und kann daher im offiziellen Sinn

[1] Hansen/Mendling/Neumann (2015), S. 466.
[2] Vgl. Vonhoegen (2015), S. 31.
[3] Vgl. Hansen/Mendling/Neumann (2015), S. 466.
[4] Hansen/Mendling/Neumann (2015) S. 466, Vonhoegen (2015) S. 32 u. Eckstein/Eckstein (2004) S. 6
[5] Für diesen und die nächsten zwei Sätze vgl. Vonhoegen (2015), S. 30.
[6] Für diesen und den nächsten Satz vgl. Eckstein/Eckstein (2004), S. 5.

keine Standards definieren.[7] Aufgrund der breiten Akzeptanz seiner Empfehlungen und der Gestaltung des Prozesses, in dem diese entwickelt werden, wirken die Empfehlungen als De-facto-Standard. Aktuell ist XML in der zweiten Ausfertigung der Version 1.1 vom 29. September 2006 gültig.[8]

2.1.1 Aufbau eines XML-Dokuments

In einem XML-Dokument werden Daten grundsätzlich textuell repräsentiert und strukturiert. Damit ein XML-Dokument verarbeitet werden kann, muss es *wohlgeformt* sein. Diese Eigenschaft besitzt jedes Dokument, das der von XML geforderten Syntax entspricht. Im Folgenden wird der in dieser Arbeit wichtige Teil des Aufbaus von XML-Dokumenten beschrieben.[9]

Bei XML wird der Inhalt, der in dieser Arbeit als **Nutzdaten** bezeichnet wird, über Markierungen (englisch Markup) von der beschreibenden logischen Struktur, die hier als **Strukturinformation** oder **Strukturdaten** bezeichnet wird, getrennt. Die Marken – im Englischen als Tags bezeichnet – zeichnen sich dadurch aus, dass sie in spitze Klammern eingeschlossen sind. Jedes Element mit Inhalt benötigt eine öffnende und eine schließende Marke, auch Start- und End-Tag genannt. Bei dem Start-Tag beginnt direkt hinter der öffnenden Klammer der Name des Elements, der mehr eine Bezeichnung für dessen Typ darstellt, da mehrere Elemente den gleichen Namen haben können und damit für gleiche Datentypen stehen. Bei einem End-Tag folgt nach der öffnenden spitzen Klammer ein Slash und anschließend der Name des Elements. Bei leeren Elementen entfällt der End-Tag und wird durch einen Slash vor der schließenden spitzen Klammer ersetzt.

Ein XML-Dokument besteht aus elf verschiedenen Arten von Informationen, von denen hier nur die in Abbildung 2.1 dargestellten relevant sind. In der ersten Zeile jedes XML-Dokuments steht der Prolog. Hier wird definiert, nach welcher Version von XML das Dokument gültig ist. Zudem wird der Zeichensatz angegeben, in dem

[7] Für diesen und den nächsten Satz vgl. Eckstein/Eckstein (2004), S. 9 f.
[8] Vgl. Bray u. a.(2006).
[9] Für die Basis der nachfolgenden Ausführungen vgl. Vonhoegen (2015) S. 47-70. Da auf syntaktische und technische Details gezielt verzichtet wurde, sei für eine detailliertere Einführung die zuvor genannte Quelle sowie entsprechend weitere Fachliteratur verwiesen.

das Dokument codiert wurde. Nach dem Prolog folgt das Wurzelelement, das die gesamte Dokumenteninstanz beinhaltet.

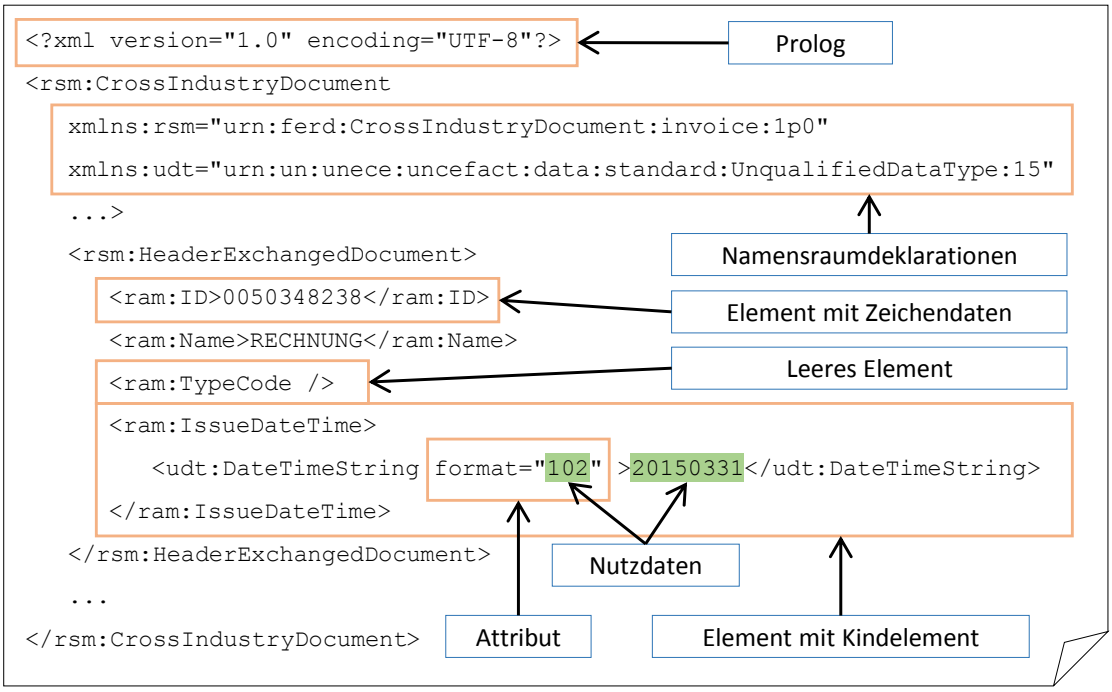

Abbildung 2.1: Beispielhaftes XML-Dokument in gekürzter Form (Eigene Darstellung)

Nicht leere Elemente können Zeichendaten und damit textuell repräsentierte Daten oder andere Elemente, die dann als Kindelemente bezeichnet werden, enthalten. Grundsätzlich ist es auch möglich, dass ein Element beides enthält, was aber im Kontext dieser Arbeit nicht zu erwarten ist. Im Start-Tag von Elementen können mehrere Attribute, bestehend aus Name und Wert, enthalten sein. Elemente und Attribute werden in dieser Arbeit auch zusammenfassend als **Bausteine** des XML-Dokuments bezeichnet.

Der hierarchische Aufbau eines XML-Dokuments ermöglicht seine Abbildung als gewurzelter Baum. Dies ist ein Graph, der einen ausgezeichneten Knoten besitzt, von dem aus alle anderen Knoten über einen eindeutigen, gerichteten Pfad erreichbar sind. Die inneren Knoten des Baums stehen für Elemente, die Kindelemente enthalten. Attribute und Elemente mit Zeichendaten bilden die Blattknoten. Die sequentielle Reihenfolge der Elemente im Dokument wird als Dokumentenreihenfolge bezeichnet und entspricht dem Ergebnis einer Tiefensuche im Baum.

Bezüglich der Bezeichnung von Elementen ist hier auch auf das Konzept der Namensräume einzugehen. Es wurde eingeführt, um für identische Bezeichnungen in unterschiedlichen Kontexten, wie beispielsweise Adresse als Postadresse oder als Adresse im Speicher eines Rechners, keine künstlichen Namen erschaffen zu müssen. Jeder Namensraum ist durch einen eindeutigen Uniform Resource Identifier (URI) definiert. Um die Zugehörigkeit eines Namens zu einem Namensraum zu kennzeichnen, wird dem jeweiligen Namen ein Präfix vorangestellt, das mit einem Doppelpunkt abgetrennt wird. Dadurch entsteht der sogenannte qualifizierte Name eines Bausteins. Der Name ohne Präfix ist hingegen der lokale Name. Wird in der Arbeit der Name oder die Bezeichnung eines Elements angesprochen, so ist damit immer der qualifizierte Name gemeint.
Bei der Deklaration eines Namensraums wird ein Präfix für dessen URI definiert. Wie aus Abbildung 2.1 ersichtlich, erfolgt dies über spezielle Attribute. Dem Namen dieser Attribute ist das Präfix *xmlsn* vorangestellt. Der lokale Name des Attributs definiert das Präfix für den Namensraum, der durch den URI im Wert des Attributs gegeben ist.

2.1.2 Inhaltsmodelle für Dokumente

Bei der Nutzung von XML als Datenaustauschformat im betriebswirtschaftlichen Umfeld kann davon ausgegangen werden, dass neben der Wohlgeformtheit des Dokuments dieses auch *gültig* sein muss.[10] Diese Eigenschaft ist erfüllt, wenn das Dokument einer definierten Struktur und damit einem Inhaltsmodell entspricht. Es gehört damit zu einem Dokumententyp bzw. einer Klasse von Dokumenten. Die Spezifikation der Struktur kann abstrakt als Schema bezeichnet werden, weswegen die verschiedenen, dafür existierenden Techniken hier als Schemasprachen zusammengefasst werden.
Ursprünglich wurde zur Beschreibung eines Dokumententyps die **Document Type Definition (DTD)** verwendet, die bereits im Rahmen der Standardisierung

[10] Für diesen und die nächsten zwei Sätze vgl. Vonhoegen (2015), S. 71-74.

von XML festgelegt wurde.[11] Da die DTD für manche Belange nicht ausreichend war, entwickelte die W3C die Schemasprache **XML-Schema**, die mit **XSD** abgekürzt wird, was für *XML schema definition* steht.[12] Eine weitere, in dieser Arbeit relevante Schemasprache ist **Regular Language Description for XML New Generation (RELAX NG)**, die von der *Organization for the Advancement of Structured Information Standards (OASIS)* standardisiert wurde.[13]

In dieser Arbeit wird ein konkretes Schema für einen bestimmten Dokumententyp entweder als **Schemainstanz** oder **Schemadefinition** und die Bestandteile einer konkreten Instanz als **Regeln eines Schemas** oder **Schemainformationen** bezeichnet. Entsprechend wird eine konkrete Schemainstanz in der Schemasprache XSD hier als **XML-Schema-Definition** bezeichnet, welche ebenfalls mit **XSD** abgekürzt wird. Aus dem jeweiligen Kontext der Verwendung ist zu erkennen, ob mit XSD die Schemasprache oder die konkrete Schemainstanz angesprochen wird.

2.1.3 Themabezogener Vergleich relevanter Schemasprachen

Die folgenden Ausführungen beinhalten einen auf die hier relevanten Aspekte reduzierten Vergleich der Schemasprachen DTD, XSD und RELAX NG.[14]

In einer Schemainstanz werden für die benötigten Elemente Inhaltsmodelle definiert. Die Zusammenführung aller Regeln für die einzelnen Elemente bildet den Rahmen für die erlaubte Struktur des jeweiligen XML-Dokuments. Zur Erstellung der Inhaltsmodelle bieten die drei Sprachen einen großen Umfang unterschiedlicher Konzepte, von denen hier nur Teilbereiche von Bedeutung sind. Darunter die grundlegenden Konzepte zur Spezifikation der Beschaffenheit der Nutzdaten sowie die des Aufbaus der Elemente aus Kindelementen. Bei Letzteren ist grundlegend relevant, welche Kindelemente in welcher Reihenfolge erlaubt sind. Die entsprechenden Sprachelemente werden in dieser Arbeit, in Anlehnung an XSD, als **Kompositoren** und

[11] Vgl. Bray u. a.(2006).

[12] Vgl. Gao/Sperberg-McQueen/Thompson (2012) und Gao/Sperberg-McQueen/Thompson (2012)

[13] Vgl. Clark/Murata (2001).

[14] Für die nachfolgenden Ausführungen zu den Sprachen vgl. Vonhoegen (2015) S. 71-184 für XSD und DTD sowie Vlist (2011) für RELAX NG. Für eine detaillierte Einführung sei auf diese beiden Werke sowie entsprechend weitere Fachliteratur verwiesen.

die Anzahl, in der ein Element in Folge auftreten darf, als **Häufigkeitsdefinition** bezeichnet.

Bei der Erklärung der konzeptionellen Unterschiede wird auf die Syntax der Schemasprachen Bezug genommen, weshalb zunächst die Unterschiede in diesem Punkt aufgezeigt werden. Die Notation der Regeln für Inhaltsmodelle mit Kindelementen gleicht bei DTD der Notation von regulären Ausdrücken mit Elementnamen, Operatoren und Klammern. In XSD erfolgt die Beschreibung hingegen in XML und damit auf Basis unterschiedlicher Elementtypen. Schemata in RELAX NG können wahlweise über eine XML-basierte oder eine kompakte, an DTD angelehnte Syntax beschrieben werden. Damit existieren zwei grundlegend unterschiedliche Notationsformen, die im Folgenden verallgemeinernd als **XML- und Operatorennotation** bezeichnet werden.

In Bezug auf die Erstellung von Inhaltsmodellen aus Kindelementen bieten alle drei Schemasprachen die Möglichkeit, Teilmodelle als Gruppen zu formulieren, die wie Elemente behandelt werden können.[15] Diese können auch mehrfach geschachtelt werden. Bei der Operatorennotation wird die Definition des Inhalts einer Gruppe in runde Klammern eingeschlossen, wohingegen sich bei der XML-Notation diese Gruppen direkt aus den Kompositoren ergeben. Des Weiteren existiert bei XSD und RELAX NG die hier nicht weiter thematisierte Möglichkeit, eigenständige Gruppen zu definieren, die in mehreren Inhaltsmodellen verwendet werden können.

Zur Komposition von Kindelementen finden sich in allen drei Schemasprachen Konzepte, um Sequenzen von Bausteinen und alternativen Bausteinen zu definieren. Bei einer Sequenz müssen die Bausteine in der gegebenen Reihenfolge im XML-Dokument auftreten. Im Fall von alternativen Bausteinen wird eine Liste von Alternativen angegeben, aus der eine im konkreten Dokument auftreten darf. XSD bietet mit dem `<xsd:all>`-Element zudem die Möglichkeit, eine Gruppe von Elementen zu definieren, für die keine Reihenfolge vorgeschrieben ist. Diese Modellierung ist jedoch nur auf oberster Ebene des Inhaltsmodells eines Elements erlaubt und darf

[15] In dieser Arbeit wird, im Zusammenhang mit Inhaltsmodellen, der Begriff **Baustein** als Oberbegriff für ein Element oder eine Elementgruppe verwendet.

ausschließlich sich nicht wiederholende Kindelemente und damit keine Gruppen enthalten. RELAX NG hebt mit dem `interleave`-Kompositor nicht nur diese Einschränkung auf, sondern erlaubt es auch, die Elemente verschiedener Gruppen zu mischen, solange dabei keine Reihenfolgebedingungen auf Gruppenebene verletzt werden.

Bei der Operatorennotation erfolgt die Zusammenstellung von Elementen und Gruppen zu einem Inhaltsmodell durch die Verknüpfung der Elementnamen sowie geklammerten Ausdrücken für Gruppen durch die binären Kompositionsoperatoren. In der XML-Notation treten Elemente als spezielle XML-Elementtypen (`<xsd:element>`) auf. Ein Inhaltsmodell besteht damit aus einem Kompostionselement, das als Kindelemente Elemente vom Typ `xsd:element` und weitere Kompositionselemente enthalten kann. Die verschiedenen Notationen der Kompositoren sind in Abbildung 2.1 aufgeführt.

Kompositor	DTD	XSD	RELAX NG	
			XML	kompakt
Sequenz	`,`	`<xsd:sequence>`	`<group>`	`,`
Alternativen	`|`	`<xsd:choice>`	`<choice>`	`|`
Elementpermutation	*fehlt*	`<xsd:all>`	`<interleave>`	`&`
freie Kombination	*fehlt*	*fehlt*	`<interleave>`	`&`

Tabelle 2.1: Vergleich der Notationen von Kompositoren in DTD, XSD und RELAX NG

Im Hinblick auf die Häufigkeitsdefinitionen sind DTD und RELAX NG identisch. Bei beiden sind vier Ausprägungen möglich: verpflichtend, optional, mindestens einmal und beliebig oft. Die Notationen der letzten drei Ausprägungen sind in Tabelle 2.2 zusammengefasst. Alle Elemente oder Gruppen, für die keine spezielle Häufigkeit definiert wurde, sind verpflichtend.

Die Definition der Häufigkeit erfolgt bei XSD über die Attribute `minOccure` und `maxOccure`, durch die für ein Element bzw. eine Gruppe eine Untergrenze und Obergrenze definiert werden kann. Indem `maxOccure = „unbounded"` gesetzt wird,

Häufigkeit	DTD	RELAX NG	
		XML	kompakt
optional	`?`	`<optional>`	`?`
mindestens einmal	`+`	`<oneOrMore>`	`+`
beliebig oft	`*`	`<zeroOrMore>`	`*`

Tabelle 2.2: Vergleich der Notationen der Häufigkeitsdefinitionen in DTD und RELAX NG

kann die Obergrenze aufgehoben werden. Werden die beiden Attribute nicht angegeben, ist das Element oder die Gruppe verpflichtend, da der Standardwert für beide Attribute `1` ist.

Zur Definition der erlaubten Beschaffenheit der Nutzdaten liefert XSD das umfangreichste Konzept. Die 47 vordefinierte Datentypen werden als einfache Datentypen bezeichnet. Des Weiteren können aus einfachen Datentypen neue einfache Datentypen abgeleitet werden. DTD ist nicht in der Lage, verschiedene Typen zu unterscheiden und fasst alle Nutzdaten als Text auf. RELAX NG hat ebenfalls kein eigenes System für unterschiedliche Nutzdatentypen. Es bietet aber die Möglichkeit, die einfachen Datentypen von XSD zu übernehmen, ohne die Erweiterungen auf eigene Typen zu unterstützen.

Neben Datentypen verwenden Schemasprachen weitere Konzepte, um die Beschaffenheit der Nutzdaten zu spezifizieren. XSD und RELAX NG ermöglichen die Beschränkung des Wertebereichs auf eine vorgegebene Liste von Werten. Neben den bisher betrachteten atomaren Datentypen, die für einen einzelnen Wert stehen, bieten beide Sprachen die Möglichkeit, zusammengesetzte Nutzdaten als Liste von Werten eines atomaren Datentyps zu definieren. Ferner ist es möglich, einen Wertebereich durch Zusammenführung mehrerer einfacher Datentypen zu erzeugen. DTD bietet von diesen Konzepten nur die Vorgabe von Wertelisten und dies nur bei Attributen.

2.2 Elektronischer Austausch strukturierter Geschäftsdokumente

Der zwischenbetriebliche elektronische Austausch strukturierter Geschäftsdokumente wird als *Electronic Data Interchange (EDI)* bezeichnet.[16] Ansätze dazu gehen in die 1970er Jahre zurück. EDI verfolgt das Ziel, die Weiterverarbeitung von Informationen möglichst stark zu automatisieren, um Prozesse zu beschleunigen und Fehler zu reduzieren. Die medienbruchfreie Übertragung digitaler Geschäftsdokumente setzt voraus, dass Sender und Empfänger das jeweils verwendete Datenformat kennen und verarbeiten können. Um den dadurch entstehenden Abstimmungsaufwand zu reduzieren, bildeten sich zunächst branchenspezifische Standardformate und später branchenübergreifende und internationale Standards wie EDIFACT (EDI for Administration, Commerce und Transport).

Zur Datenübertragung im Rahmen von EDI wurde anfangs nicht das Internet verwendet, sondern proprietäre Mehrwertdienste bzw. Stand- oder Wahlleitungen. Im Gegensatz zu diesem klassischen EDI ermöglicht das Internet-EDI eine flexiblere und kostengünstigere Kommunikation. Neben dieser bildete sich auf Basis des Internets zudem das Web-EDI, bei dem ein Unternehmen seinen Geschäftspartnern Geschäftsdokumente als webbasierte Formulare zur Verfügung stellt. Sie sind mit den betrieblichen Anwendungssystemen, mit denen sie im anbietenden Unternehmen weiterverarbeitet werden, verknüpft. Diese Form von EDI ermöglicht zwar die kostengünstige Einbindung kleiner Unternehmen in die EDI-basierten Prozesse großer Unternehmen, reduziert jedoch den manuellen Aufwand sowie die Fehleranfälligkeit kaum, sondern verlagert sie lediglich zu den Geschäftspartnern.

Eine Erweiterung des Internet-EDI ist XML-EDI, bei dem statt klassischen EDI-Formaten wie EDIFACT die flexiblere Auszeichnungssprache XML verwendet wird. Das Konzept verfolgt das Ziel, die Automatisierbarkeit von EDI mit der Einfachheit von Web-EDI zu verbinden. Auf der Basis von XML können Dokumente mit semantischen Strukturen erstellt werden, wodurch die enthaltenen Informationen

[16] Für die folgenden Ausführungen zu EDI vgl. Nomikos (2002) S. 151-156.

vollständig automatisiert verarbeitet werden können.[17] Die Vorteile von XML, auch für kleine und mittlere Unternehmen, haben dazu geführt, dass aktuellere Bestrebungen für Standardisierungen im Bereich des elektronischen Austausches auf XML basieren und dass XML der wichtigste Standard für den Datenaustausch zwischen Informationssystemen ist.[18]

Anfangs wurden Standards auf Dokumentenebene entwickelt[19], welche im Hinblick auf die Wiederverwendung in verschiedenen Einsatzgebieten unflexibel waren. In den frühen 2000er Jahren wurden deshalb von der UN/CEFACT Bestrebungen unternommen, eine Methodik zu entwickeln, mit der aus wiederverwendbaren Bausteinen Geschäftsdokumente gebildet werden können. Als Ergebnis entstand die UN/CEFACT Core Components Technical Specification (CCTS). Diese ging in das Framework der von UN/CEFACT und OASIS gegründeten Initiative Electronic Business using XML (ebXML) ein und wurde von der Internationalen Organisation für Normung (ISO) durch die Norm ISO 15000-5 anerkannt.

Die CCTS hat viele aktuell populäre, branchen- und länderübergreifende Standards beeinflusst.[20] Auf Ebene der Transaktionsstandards für Dokumente verwenden beispielsweise die Universal Business Language (UBL) und Open Applications Group Integration Specification (OAGIS) die in CCTS definierten Bausteine. Ferner übernimmt Global Standard One (GS1) XML (GS1 XML) die Methodik zur Komposition von wiederverwendbaren Bausteinen, definiert diese allerdings selbst. In Deutschland wurde 2014 vom *Forum elektronische Rechnung Deutschland* das ZUGFeRD-Format veröffentlicht, das ebenfalls auf der ISO-Norm 15000-5 basiert.[21] Dabei handelt es sich um ein Format für elektronische Rechnungen, das vor allem auf kleine und mittlere Unternehmen ausgerichtet ist und die Lücke zwischen rein bildlich dargestellten Rechnungen und EDI-Austauschformaten schließen soll. Dazu werden die Rechnungsdaten, in strukturierter Form als XML-Datei, in ein PDF-

[17] Vgl. Fink/Schneidereit/Voss (2005), S. 252.
[18] Kabak/Dogac (2010) S. 2, Bächle/Lehmann (2010) S. 52. u. Hansen/Mendling/Neumann (2015) S. 466.
[19] Gesamter Absatz vgl. Kabak/Dogac (2010), S. 2 u. 8.
[20] Gesamter Absatz vgl. Kabak/Dogac (2010), S. 4 u. 23 f.
[21] Gesamter Absatz vgl. Bergmann u. a.(2014), S. 10-21.

Dokument eingebettet, das die Rechnung bildlich darstellt. Auf diese Weise muss der Rechnungsversender nicht unterscheiden, ob der Empfänger die Rechnung automatisiert oder manuell weiterverarbeitet. Bei der manuellen Verarbeitung wird die bildliche Darstellung des PDF-Dokuments genutzt und bei der automatischen Verarbeitung die XML-Datei extrahiert.
Bei allen Standards ist es notwendig, den Aufbau einer Dokumentenart, wie beispielsweise einer Rechnung oder Bestellung spezifizieren zu können. Bei XML-basieren Ansätzen wird zu diesem Zweck ein Inhaltsmodell für die jeweilige Dokumentenart definiert. Zur Beschreibung von Inhaltsmodellen für XML-Dokumente hat sich die Schemasprache XML-Schema etabliert, wie ihre Verwendung in den aktuell weit verbreiteten Standards wie UBL, OAGIS, GS1 XML sowie dem ZUGFeRD-Format zeigt.[22]

2.3 Optische Codierung

Die optische Codierung ist der Oberbegriff für 1D-, 2D- und 3D-Codes.[23] Die älteste Form ist der 1D-Code, der auch als Strichcode bezeichnet wird. Grundsätzlich wird die optische Codierung zur optischen Identifikation von Objekten eingesetzt.[24] Dort fungiert sie als maschinell lesbarer optischer Datenträger für die zur Identifikation nötigen Informationen.[25] Darüber hinaus werden vor allem 2D-Codes zur Datenübertragung im Bereich weniger Kilobyte in Geschäftsprozessen verwendet.[26] Eben zu diesem Zweck soll die optische Codierung in dieser Arbeit eingesetzt werden.

2.3.1 Identifikation relevanter Codes

Zum Transport von Geschäftsdokumenten ist nicht jeder optische Code geeignet. Aus diesem Grund werden im Folgenden Anforderungen beschrieben und entsprechend dieser passende Codes ausgewählt. Die betrachteten Kriterien sind die Da-

[22] Vgl. Kabak/Dogac (2010) S. 11, 15 u. 19-21 sowie Bergmann u. a.(2014) S. 93
[23] Für diesen und den nächsten Satz vgl. Lenk (2003), S. 100.
[24] Vgl. Lenk (2003), S. 70.
[25] Vgl. Lenk (2005a), S. 47.
[26] Vgl. Lenk (2005b), S. 20 f.

tenkapazität, die Verfügbarkeit der Codespezifikation, die Verbreitung in der unternehmerischen Praxis sowie die technischen Rahmenbedingungen zur Erstellung und Verarbeitung der Codes.

Die optisch zu codierende Datenmenge ergibt sich aus der konkreten Größe des jeweils komprimierten Geschäftsdokuments. Bei der in Abschnitt 5.3 beschriebenen Fallstudie ist beispielsweise eine Kapazität von rund 800 Byte ausreichend. In anderen Szenarien können dennoch größere Datenmengen benötigt werden.

Aufgrund der erwarteten Datenmenge ist die Verwendung von 1D-Codes nicht möglich.[27] Wenngleich die Kapazität von 2D-Codes größer als die von 1D-Codes ist, ist diese dennoch nicht bei allen Typen ausreichend. 2D-Codes unterteilen sich grundsätzlich in Stapel- und Matrixcodes.[28] Stapelcodes sind weniger kompakt als Matrixcodes[29], weswegen sie mehr Platz benötigen und damit für den vorliegenden Einsatzzweck nicht brauchbar erscheinen. Im Bereich der Matrixcodes finden sich eine Reihe von Typen, die in der Lage sind, binäre Daten mit mehr als 1 Kilobyte Größe zu speichern: Aztec, Code One, Data Glyphs, Data Matrix, QR Code, Tag Code und Vericode.[30]

Neben einer ausreichenden Kapazität ist auch entscheidend, dass die Spezifikation des Codes zumindest öffentlich verfügbar und bestenfalls standardisiert ist. Dies begründet sich darin, dass ein Code im Kontext dieser Arbeit einfach und kostengünstig implementierbar sein muss. Nur auf diese Weise wird er von möglichst vielen Unternehmen adaptiert und kann reibungslose unternehmensübergreifende Geschäftsprozesse ermöglichen.

Codes ohne öffentliche Spezifikationen sind meist durch Patente geschützt und damit für den Einsatz zur optischen Codierung von Geschäftsdokumenten nicht sinnvoll. Von den Codes mit ausreichender Kapazität ist diese bei den folgenden

[27] Vgl. Lenk (2005a), S. 168.
[28] Vgl. Lenk (2002), S. 18.
[29] Vgl. Lenk (2004), S. 25-26.
[30] Vgl. Lenk (2002), S. 256-494.

der Fall: Data Glyphs[31], Vericode[32], Color Code[33] und Color Ultra Code[34]. Die beiden letzteren sind 3D-Codes und als solche Matrixcodes, bei denen durch die Verwendung von Farbe eine weitere Dimension der Codierung geschaffen wird. Diese Codes sind aufgrund der zusätzlichen Dimension kompakter als 2D-Codes und damit im vorliegenden Anwendungsfeld grundsätzlich von Vorteil. Es konnte jedoch kein 3D-Code gefunden werden, der eine öffentliche Spezifikation besitzt.
Von den 2D-Codes finden nur der Aztec, Data Matrix und QR Code eine nennenswerte Anwendung in der unternehmerischen Praxis.[35] Diese sind zudem von den zuvor genannten 2D-Codes die einzig standardisierten Codes. Im Hinblick auf die Verwendung von Farben zur Codierung besteht derzeit auch die Einschränkung, dass zum Drucken von Dokumenten vorwiegend Laserdrucker mit Graustufen eingesetzt werden.
Für die drei relevanten 2D-Codes besteht eine breite Palette an Programmen und Programmbibliotheken zur Erstellung und Verarbeitung der als Bild vorliegenden Codes. Damit können Dokumentenscanner, Kameras und im Speziellen mobile Geräte, wie beispielsweise Smartphones oder Tablets, zur Erfassung der Codes verwendet werden. Ferner existieren mobile und stationäre Geräte, die speziell auf das Lesen und Verarbeiten optischer Codes ausgelegt sind[36], was ihre Verwendung im Einsatzgebiet dieser Arbeit begünstigt.

2.3.2 Vergleich der relevanten 2D-Codes

Im Folgenden werden die hier relevanten Aspekte des Aztec, Data Matrix und QR Code verglichen. Zu diesen gehören die typischen Einsatzgebiete der Codes sowie die maximale Datenmenge, die Kompaktheit und die Fehlertoleranz gegenüber Beschädigungen. Die drei Codes sind in den folgenden Normen standardisiert[37]:

[31] Vgl. Lenk (2002), S. 336.
[32] Vgl. Lenk (2002), S. 491.
[33] Vgl. Lenk (2002), S. 325.
[34] Vgl. Lenk (2002), S. 327.
[35] Vgl. Lenk (2012), S. 5.
[36] Vgl. Lenk (2012), S. 279, 359 u. 456.
[37] Vgl. Lenk (2012), S. 93, 129 u. 149.

- QR Code 2005: ISO/IEC 18004:2006(E), Corrigendum 1:2009
- Data Matrix Code ECC 200: ISO/IEC 16022:2006
- Aztec Code: ISO/IEC 24778:2008

Die drei betrachteten Codes haben sich in unterschiedlichen Anwendungsgebieten etabliert. Der Data Matrix Code wird primär in der Logistik und Produktion zur direkten Markierung von Bauteilen verwendet.[38] Zudem wird er im Dokumentenmanagement und im Bereich der Logistik eingesetzt[39], wie zum Beispiel von der Deutschen Post[40] und GLS[41]. Im industriellen Bereich wird der QR Code, vorwiegend auf Papier oder Etiketten gedruckt, in der Produktionssteuerung und Logistik eingesetzt.[42] Zudem findet er breite Anwendung im öffentlichen Bereich für Marketing und Mobile Tagging.[43] Der Aztec Code ist im Vergleich zu den beiden anderen weniger stark verbreitet.[44] Sein übliches Anwendungsgebiet liegt im Bereich von elektronischen oder gedruckten Fahr- und Boardkarten[45], wie beispielsweise bei den Unternehmen Deutsche Bahn[46] oder Air Berlin[47].

Zur Korrektur von Lesefehlern bei beschädigten Codes wird bei allen drei Codetypen der Fehlerkorrekturalgorithmus Reed Solomon[48] verwendet[49], der beispielsweise auch bei CDs, DVDs und ADSL eingesetzt wird.[50] Mit dieser Technik ist es möglich, die Daten bei Beschädigungen bis zu einem bestimmten Prozentsatz der Codefläche zu rekonstruieren. Bei QR Codes sind vier verschiedene Fehlerkorrekturstufen möglich, die unterschiedliche maximale Beschädigungsgrade kompensieren: L bis 7 %, M bis 15 %, Q bis 25 % und H bis 30 %.[51] Der Aztec Code deckt in der

[38] Vgl. Knuchel u. a.(2011) S. 50 f. und Lenk (2012) S. 6.
[39] Vgl. Lenk (2012), S. 319.
[40] Vgl. Lenk (2005b), S. 14.
[41] Vgl. Lenk (2012), S. 351.
[42] Vgl. Lenk (2012), S. 6.
[43] Vgl. Knuchel u. a.(2011) S. 50 f. und Lenk (2012) S. 5 f.
[44] Vgl. Lenk (2012), S. 6.
[45] Vgl. Knuchel u. a.(2011) S. 50 f. und Lenk (2012) S. 5 u. 158-160.
[46] Vgl. Lenk (2012), S. 158.
[47] Vgl. Lenk (2012), S. 159.
[48] Für Details zu diesem Verfahren sei auf Lenk (2012) S. 31-35 verwiesen.
[49] Vgl. Lenk (2012), S. 93, 129 u. 149.
[50] Vgl. Lenk (2012), S. 31.
[51] Vgl. Lenk (2012), S. 126 f.

Standardeinstellung eine Fehlertoleranz von 23 % ab, ist aber so flexibel konfigurierbar, dass Beschädigungsgrade zwischen 5 % und 95 % frei wählbar sind.[52] Im Gegensatz dazu ist die Fehlerkorrektur bei Data Matrix ECC 200 nicht variabel und entspricht der Stufe M des QR Codes.

Die maximale Datenkapazität ist von der gewünschten Fehlerkorrekturstufe abhängig. Entsprechend der Kapazitätsvergleiche bei Lenk (2012)[53] wird auch hier beim Aztec Code die Standardstufe und beim QR Code die Stufe M verwendet. In dieser Konfiguration ist die Kapazität des QR Codes mit 2.331 Byte am höchsten.[54] Danach folgt der Aztec Code mit 1.914 Byte[55] und schließlich der Data Matrix Code mit 1.556 Byte.[56]

Wichtiger als die maximale Datenkapazität ist der Platzbedarf für eine bestimmte Datenmenge. An dieser Stelle sei erwähnt, dass die drei betrachteten Matrixcodes aus einem Raster gleich großer quadratischer Zellen bestehen.[57] Die gedruckte Größe eines Codes ist damit von der Kantenlänge abhängig, mit der eine Zelle gedruckt wird. Diese ist wiederum einerseits von der Auflösung der eingesetzten Drucker und andererseits von der der Lesegeräte abhängig.[58] Bei einer Druckauflösung von 300 dpi ergibt sich eine minimale Kantenlänge von 0,085 mm und bei 100 dpi von 0,254 mm. Als Lesegeräte für die Codes sind im Zusammenhang dieser Arbeit Dokumentenscanner, Kameras an Desktoparbeitsplätzen, Smartphones und spezielle tragbare und stationäre Lesegeräte denkbar. Welche Kantenlänge im vorliegenden Umfeld dieser heterogenen Möglichkeiten sinnvoll ist, muss eine Untersuchung klären, die nicht Teil dieser Arbeit ist.

Die Kompaktheit der Codes kann unabhängig von der Druckdichte verglichen werden. Dazu wird statt der konkreten Druckgröße die Anzahl der nötigen Zellen verglichen. In diesem Zusammenhang muss auf die sogenannten Ruhezonen der Codes eingegangen werden. Dabei handelt es sich um Ränder, die leer bleiben müssen, um

[52] Für diesen und den nächsten Satz vgl. Lenk (2012), S. 128.
[53] Vgl. Lenk (2012), S. 128 u. 134.
[54] Vgl. Lenk (2012), S. 108.
[55] Vgl. Lenk (2012), S. 149.
[56] Vgl. Lenk (2002), S. 368.
[57] Vgl. Lenk (2005a) S. 47 u. Lenk (2002) S. 22.
[58] Vgl. Lenk (2004), S. 16 f.

den eigentlichen Code von seiner Umgebung abzugrenzen.[59] Beim QR Code beträgt die Größe dieser Zone umlaufend die vierfache Kantenlänge einer Zelle, bei Data Matrix umlaufend eine Kantenlänge und beim Aztec Code ist sie nicht nötig.[60]
Die Graphik in Abbildung 2.2 vergleicht den Aztec[61], Data Matrix ECC 200[62] und QR Code[63] bezüglich ihres Platzbedarfs. Dabei wird die Codegröße in Anzahl an Zellen inklusive Ruhezone angegeben. Der Kapazitätsbereich wurde von 500 und 1.500 Byte gewählt, weil nicht zu erwarten ist, dass Geschäftsdokumente unter 500 Byte komprimiert werden können und die Kapazitätsgrenze des Data Matrix Codes knapp über 1.500 Byte liegt. Die Graphik verdeutlicht, dass der Aztec Code am kompaktesten ist und der QR Code den höchsten Platzbedarf hat.

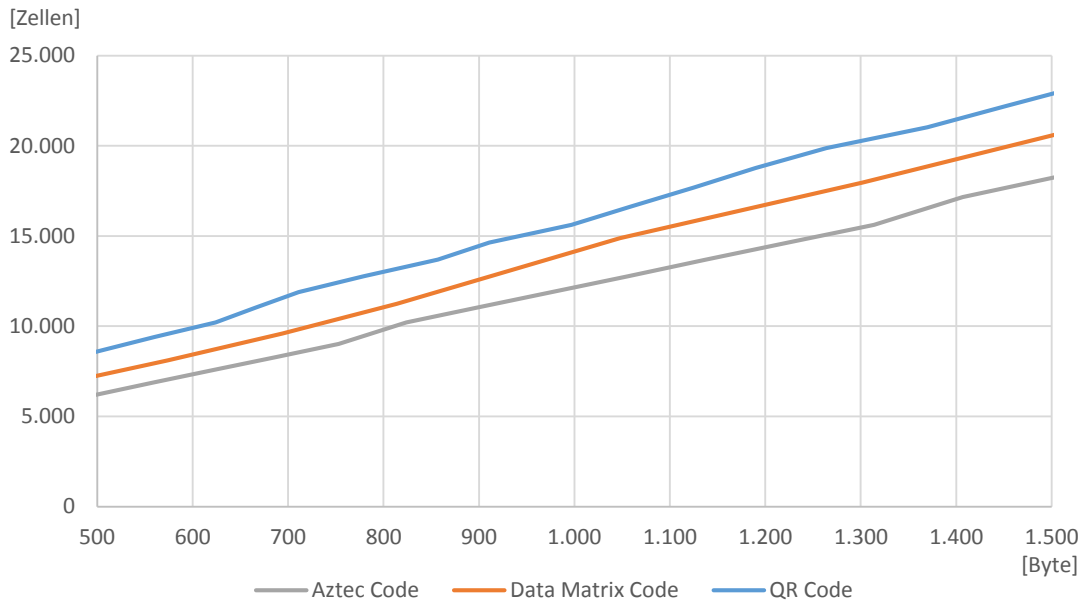

Abbildung 2.2: Platzbedarf ausgewählter Maxtrixcodes in Anzahl an Zellen (Eigene Darstellung)

[59] Vgl. Uitz/Harnisch (2012), S. 341.
[60] Vgl. Lenk (2012), S. 150.
[61] Vgl. Lenk (2002), S. 282.
[62] Vgl. Lenk (2002), S. 368.
[63] Vgl. Lenk (2012), S. 100 u. 105 f.

2.4 Grundlagen zur Datenkompression

Datenkompression ist nach Salomon (2007a) der "process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, the bitstream, or the compressed stream) that has a smaller size."[64] Wenn die Information der erzeugten Daten mit der der Quelldaten identisch ist, ist die Datenkompression *verlustfrei*, andernfalls ist sie *verlustbehaftet*.[65] Der Verlust von Information kann bei manchen Daten, wie beispielsweise Audio oder Video, ein sinnvolles Mittel sein, um deren Größe zu reduzieren. Im Kontext dieser Arbeit ist jedoch eine verlustfreie Datenkompression nötig.

2.4.1 Bezug zur Informations- und Codierungstheorie

Im Rahmen der Codierungstheorie entspricht die Datenkompression der Quellencodierung.[66] Dieser Theorie liegt das allgemeine Modell der Nachrichtenübertragung von Shannon (1948) zugrunde, bei dem eine Informationsquelle Nachrichten über einen Kanal an ein Ziel sendet.[67] Die Quelle besitzt eine endliche Menge von Symbolen bzw. Zeichen, die als Zeichenvorrat oder Alphabet bezeichnet wird.[68] Eine Nachricht wird als Folge von Zeichen aus dem Alphabet gebildet.

Die Auswahl eines Zeichens aus dem Zeichenvorrat der Quelle wird als Ereignis bezeichnet. Welche Nachricht und damit Abfolge von Ereignissen eine Quelle produziert, ist nur ihr bekannt. Für einen externen Beobachter sind die Ereignisse daher das Ergebnis eines Zufallsprozesses. Somit herrscht bei ihm Ungewissheit über das als nächstes von der Quelle gesendete Symbol. Die Beseitigung dieser Unsicherheit durch das Eintreten eines Ereignisses wird als Information bezeichnet.

Das Maß an Unbestimmtheit definiert die mit einem Ereignis verbundene Informationsmenge und kann wie folgt formuliert werden. Sei $X = x_1, x_2, ...x_N$ eine Menge von Ereignissen und $p(x_i)$ die Wahrscheinlichkeit, mit der ein Ereignis x_i

[64] Salomon (2007a), S. 2.
[65] Vgl. Herold/Lurz/Wohlrab (2012), S. 750.
[66] Vgl. Salomon (2007b), S. 13 f.
[67] Vgl. Schönfeld/Klimant/Piotraschke (2012) S. 9 und Shannon (1948) S. 379 f.
[68] Die nachfolgenden Ausführungen zur Informations- und Codierungstheorie basieren, soweit nicht anders angegeben, auf Schönfeld/Klimant/Piotraschke (2012) S. 9-14.

für $i = 1, 2, ...N$ auftritt, dann ist das Maß H_i für die Unbestimmtheit über das Ereignis x_i mit $H_i = -log(p(x_i))$ definiert. Somit ist die Informationsmenge bei einem Ergebnis umso größer, je unwahrscheinlicher sein Eintreten ist.
Die Übertragung einer Nachricht wird in diesem Sinn als Informationsübertragung bezeichnet. Damit diese erfolgen kann, muss die Nachricht auf das Alphabet des Kanals abgebildet und somit codiert werden. Eine Codierung ist im Allgemeinen eine injektive Abbildung, die jedem Element eines Alphabets ein Element oder eine Folge von Elementen – ein sogenanntes Wort – eines anderen Alphabets zuordnet.[69]
Die Abbildung des Quellenalphabets in das Kanalalphabet wird als Quellencodierung bezeichnet. Da der Kanal im vorliegenden Kontext grundsätzlich die digitale Abbildung von Informationen darstellt, besteht das Kanalalphabet U aus zwei Symbolen: $U = \{0, 1\}$. Dementsprechend ist die Quellencodierung eine Binärcodierung. Bei der Quellencodierung wird der Informationsgehalt der Elemente des Quellenalphabets berücksichtigt, um eine möglichst kurze Codierung der Information zu erzeugen. Symbolen mit hoher Eintrittswahrscheinlichkeit werden dabei kürzere binäre Codewörter zugeordnet und Symbolen, die weniger häufig auftreten, längere. Je besser die Codierung den Informationsgehalt der Symbole berücksichtigt, desto kürzer ist die resultierende Nachricht. Da Informationen im digitalen Kontext immer codiert als Daten vorliegen, sind Quellencodierung und Datenkompression gleichzusetzen.

2.4.2 Codierungs- und Kompressionstechniken

In der Quellencodierung bzw. Datenkompression kommen verschiedene Arten von Codierungstechniken zum Einsatz. Salomon (2007b) unterscheidet dafür vier Klassen[70]: *block-to-block*, *block-to-variable*, *variable-to-block* und *variable-to-variable*. Die Techniken der ersten Klasse wandeln die Eingabe in eine Ausgabe der gleichen Länge um und sind daher keine Kompressionstechniken, sondern dienen der Vorverarbeitung für nachfolgende Codierungen.

[69] Vgl. Heise/Quattrocchi (1989), S. 31.
[70] Zur Beschreibung der Klassen vgl. Salomon (2007b) S. 1-4.

Die zweite Klasse beinhaltet Techniken, bei denen jedem Symbol des Eingabealphabets ein Code variabler Länge zugeordnet wird. Dabei wird der Code für ein Symbol umso länger, je seltener es verwendet wird. In dieser Gruppe sind drei Ansätze zu unterscheiden: Zum einen statische Codes, bei denen die Zuordnung bzw. Abbildungsvorschrift zwischen Symbol und zugehörigem Code bei dem Kompressor und Dekompressor fest eingebaut sind und schon vor der Kompression feststehen. Die anderen beiden Ansätze bilden die Codes für die Symbole dynamisch auf Basis der zu komprimierenden Daten. Zu den beiden letzteren gehört die Gruppe der statistischen Kompressionsverfahren.
Die dritte Klasse von Codierungen besteht aus solchen, die unterschiedlich lange Zeichenfolgen des Eingabealphabets auf Codes fester Länge abbilden. Zu diesen sind die Verfahren mit Wörterbuchcodierung zuzuordnen. Die letzte Klasse umfasst Methoden, die aus mehreren Schritten bestehen und auf diese Weise verschieden lange Sequenzen des Eingabealphabets auf unterschiedlich lange Codes abbilden. In dieser Gruppe finden sich häufig universale Kompressionsverfahren.
Im Rahmen dieser Arbeit wird zwischen statischer und dynamischer Codierung unterschieden. Zu Ersteren zählen neben statischen Codes variabler Länge auch Blockcodes mit fester Länge.[71] Alle anderen Verfahren, bei denen die Zuordnung zwischen Quellensymbolen und Code nicht vordefiniert ist, werden als dynamische Codierungen zusammengefasst.

Relevante statische Codierungen

Die wichtigste statische Codierungstechnik für Codes variabler Länge ist die Huffmann-Codierung. Bei dieser werden für die Codes für die Symbole des Quellenalphabets auf Basis ihrer Auftrittswahrscheinlichkeit so gebildet, dass die durchschnittliche Länge der Codes minimiert wird.[72]
Im Rahmen der Arbeit wird des Weiteren der Beta-Code verwendet.[73] Dieser bildet Zahlen ab, indem er sie als Dualzahl darstellt. Da es sich dabei um einem Block-

[71] Vgl. Heise/Quattrocchi (1989), S. 31.
[72] Für weitere Details zur Huffmann-Codierung vgl. beispielsweise Salomon (2007b) S. 42-49
[73] Vgl. Salomon (2007b), S. 69.

code handelt, muss die maximal abbildbare Zahl n bekannt sein, um dessen Länge $l = \lceil \log_2 n \rceil$ bestimmen zu können. Mit ihm ist es möglich, beliebige Alphabete zu codieren. Dazu werden die Symbole durchnummeriert und die dem auftretenden Symbol entsprechende Nummer codiert.

Zur Codierung von Zahlen, deren Wertebereich nicht bekannt ist, existiert eine Vielzahl von Methoden.[74] In der vorliegenden Arbeit sind davon drei Codierungen relevant. Der Zählcode bildet eine positive Zahl n in einer Folge aus $n - 1$ 1-Bits, gefolgt von einem einzelnen 0-Bit ab.[75] Ein Code, der kleine Zahlen kürzer abbildet, ist der Fibonacci Code. Er verwendet zur Abbildung von Zahlen die Fibonacci-Reihe als Grundlage für ein Stellenwertsystem.[76]

Die dritte Variante zur Codierung von Zahlen bildet eine Zahl in mehrere Blöcke zu je acht Bit ab. Dabei wird die Zahl als Dualzahl dargestellt und in je sieben Bit lange Blöcke zerlegt. Dem letzten dieser Blöcke wird ein 1-Bit vorangestellt, dem Rest ein 0-Bit. Die blockweise Dekodierung erkennt auf diese Art das Ende des Codes. Der Code wird in der Arbeit für die bytebasierte Codierung von Zahlen verwendet.

Relevante dynamische Codierungen

Bei den Kompressionsverfahren können, wie oben beschrieben, drei Gruppen unterschieden werden. Aus der Gruppe der Kompressoren, die einen *variable-to-block*-Ansatz mit Wörterbüchern verfolgen, ist zum einen der *Deflate*-Algorithmus relevant, wie er in den Kompressionsprogrammen *Zip* und *gzip* zum Einsatz kommt.[77] Zum anderen wird auch der *Lempel-Ziv-Markow-Algorithmus* (*LZMA*) in Form des Kompressionsprogramms *7-Zip* verwendet.[78]

Aus dem Bereich der statistischen Verfahren sind die arithmetische Codierung[79] und die *Prediction by Partial Matching (PPM)* Technik hier von Bedeutung. Bei

[74] Vgl. Salomon (2007b), S. 69-142.
[75] Vgl. Salomon (2007b), S. 70.
[76] Vgl. Salomon (2007b), S. 112-115.
[77] Vgl. Salomon (2007a), S. 230-241.
[78] Vgl. Salomon (2007a), S. 241-246.
[79] Vgl. Salomon (2007a), S. 112-124.

PPM-Verfahren wird ein Modell der zu komprimierenden Daten erstellt, wodurch für die Wahrscheinlichkeit eines Symbols der Kontext berücksichtigt werden kann.[80] Die Codierung des Symbols erfolgt adaptiv arithmetisch. Eine häufig, wie auch hier, verwendete Implementierung eines PPM-Verfahrens ist der von Shkarin entwickelte Kompressor *PPMd*. Bei diesem wird die in Shkarin (2001) vorgestellte PPM-Variante namens *PPM with Information Inheritance (PPMII)* implementiert.[81] Eine weitere PPM-Variante, die auf PPMII aufbaut und hier verwendet wird, ist *PPM with variable-length contexts (PPMVC)*.[82]

Von den mehrstufigen Verfahren der *variable-to-variable*-Klasse kommen in dieser Arbeit vier Verfahren zum Einsatz. Eines davon ist das auf der *Burrows-Wheeler-Transformation (BWT)*[83] und der Huffmann-Codierung basierende Kompressionsprogramm *bzip2*.[84] Zudem wird *Sequitur* verwendet, das aus den zu komprimierenden Daten eine Grammatik ableitet, die arithmetisch codiert wird.[85] Letztlich werden mit *LPAQ* und *PAQ8* auch zwei Vertreter der PAQ-Kompressoren eingesetzt, in denen verschiedenste Kompressionskonzepte kombiniert werden, wodurch sie sehr leistungsstark werden.[86]

Alle dynamischen Codierungen werden gemeinhin als Kompressionsverfahren bezeichnet. Die in diesem Abschnitt genannten Kompressoren werden im Vorgang der Arbeit unter der Bezeichnung Universalkompressoren zusammengefasst.

[80] Für diesen und den nächsten Satz vgl. Salomon (2007a), S. 139-160.
[81] Vgl. Shkarin (2001), S. 226-234.
[82] Vgl. Skibinski/Grabowski (2004), S. 409-418.
[83] Vgl. Salomon (2007a), S. 853-858.
[84] Vgl. Seward (2007), S. 3 f.
[85] Vgl. Salomon (2007a), S. 906-911.
[86] Vgl. Mahoney (2002) S. 1-12 u. Mahoney (2005) S. 1-6.

3 Identifizierung relevanter XML-Kompressionsverfahren

Zur Kompression von XML-Dokumenten ist ein breites Spektrum an Ansätzen und Verfahren verfügbar. Auf Basis der Literatur ist es jedoch nicht möglich, das Verfahren zu identifizieren, das XML-Geschäftsdokumente am besten komprimiert. Daher ist es nötig, die vorhandenen Verfahren in Bezug auf deren Leistungsfähigkeit zur Kompression von XML-Geschäftsdokumenten zu untersuchen.
Wenngleich die Basis einer fundierten Analyse ein quantitativer Vergleich der bestehenden Verfahren ist, ist es nicht sinnvoll, alle vorhandenen Verfahren in diesen einzubeziehen. Denn einerseits liegen nicht alle Verfahren in implementierter Form vor und andererseits verfolgen nicht alle Verfahren einzig die Optimierung der Kompressionsrate, sondern zusätzliche, meist konkurrierende Zielsetzungen. Zudem setzen manche Verfahren Techniken ein, die für die Kompression von Geschäftsdokumenten unpassend sind. Ein nicht in implementierter Form vorliegendes Verfahren zu implementieren, ist nicht sinnvoll, wenn bereits aus seiner Zielsetzung oder der technischen Konzeption abgeleitet werden kann, dass es im Vergleich zu anderen Verfahren geringere Kompressionsraten erzielen wird. Aus diesem Grund ist eine Eingrenzung der quantitativ zu vergleichenden Verfahren auf qualitativer Ebene nötig. Als Basis wird dafür eine eigene Klassifikation von Verfahren zur XML-Kompression entwickelt.
Das vorliegende Kapitel behandelt die qualitative Auswahl der Verfahren, die im quantitativen Vergleich der in Kapitel 5 beschriebenen Fallstudie berücksichtigt werden. Der erste Abschnitt beschreibt, dass in der Literatur keine ausreichenden Informationen vorhanden sind, um die Leistungsfähigkeit von Verfahren zur Kompression von XML-basierten Geschäftsdokumenten vergleichen zu können und

rechtfertigt damit die qualitative Voruntersuchung. Im Anschluss daran wird eine eigene Klassifikation für Verfahren zur XML-Kompression entwickelt, die die Basis für den Auswahlprozess bildet. Die darauf folgenden Teile erklären die Auswahl der quantitativ zu vergleichenden Verfahren. Zunächst wird das sehr weite Feld der XML-Kompression anhand funktionaler Gesichtspunkte auf die im Zusammenhang dieser Arbeit relevanten Verfahren eingegrenzt. Anschließend werden, unterteilt in zwei Gruppen, alle zu untersuchenden Verfahren vorgestellt und auf Basis technischer Eigenschaften beurteilt, ob der Aufwand gerechtfertigt ist, ein nicht in implementierter Form verfügbares Verfahren zu implementieren. Abschließend werden die Untersuchungsergebnisse zusammengefasst.

3.1 Zur einschlägigen Literatur

Eine Literaturrecherche ergab über 50 Veröffentlichungen, in denen einzelne XML-spezifische Kompressionsverfahren vorgestellt werden, sowie sechs Vergleichsstudien. Leistungsvergleiche von Verfahren finden sich sowohl in Vergleichsstudien als auch in den Arbeiten zu den Verfahren selbst. Die verwendeten Untersuchungsdaten weichen jedoch in deren Aufbau und Größe von den hier zu verarbeitenden Geschäftsdokumenten ab, weshalb keine validen Rückschlüsse möglich sind. Zudem decken einzelne Vergleiche jeweils nur einen Teil der Verfahren ab und können nicht zu einem sinnvollen Gesamtvergleich kombiniert werden.

Sakr berücksichtigt in seiner ersten Vergleichsstudie[1] 22 Verfahren und erweitert diesen Umfang in seiner zweiten Studie[2] um drei Verfahren. Auf qualitativer Ebene liefern beide Arbeiten gute Unterscheidungsmerkmale für die betrachteten Verfahren. Einem quantitativen Leistungsvergleich wurden jedoch lediglich neun Verfahren unterzogen.[3] Wenngleich die Datenbasis dafür aus 30 verschiedenen Datensätzen besteht, wird die Charakteristik der im Kontext dieser Arbeit zu erwartenden Dokumente dennoch nicht abgedeckt.[4] Insgesamt wurden 14 Verfahren aufgrund

[1] Vgl. Sakr (2009a).
[2] Vgl. Sakr (2009b).
[3] Vgl. Sakr (2009a) S. 51 f. und Sakr (2009b) S. 311 f.
[4] Vgl. Sakr (2009a) S. 53-56 und Sakr (2009b) S. 310.

fehlender bzw. fehlerhafter Implementierungen nicht in die quantitative Untersuchung aufgenommen.[5] Zudem wurden drei Verfahren ausgeschlossen, weil sie Schemainformationen nutzen, und dies als zu spezifisch angesehen wird.[6] Im Kontext dieser Arbeit ist dieser Ausschluss unnötig, da für Geschäftsdokumente Schemata definiert sind.

Wie die beiden Arbeiten von Sakr liefert auch Augeri u. a.(2007) auf quantitativer Ebene grundsätzlich gute Ergebnisse. Hier werden 14 Verfahren auf einer Datenbasis von 44 Datensätzen aus unterschiedlichen Anwendungsgebieten verglichen. Ebenfalls nicht enthalten sind jedoch Dokumente, die Geschäftsdokumenten mit weniger als 100 kB ähnlich sind.

Die Studien von Ng/Lam/Cheng (2006) und Nair (2007) enthalten Leistungsvergleiche von kleinem Umfang. Obwohl in Ng/Lam/Cheng (2006) elf Verfahren beschrieben werden, werden nur fünf auf der Basis von lediglich sechs Datensätzen ausgewertet.[7] Nair (2007) thematisiert fünf Verfahren und untersucht davon drei mit drei verschiedenen Datensätzen.[8] Bei den anderen beiden werden mangels lauffähiger Binärprogramme die Werte aus Veröffentlichungen der jeweiligen Urheber zitiert.

Die aktuellste Vergleichsstudie enthält keine Leistungsvergleiche und ist qualitativer Natur. [9] Sie behandelt ein Standardverfahren, das schon durch die zuvor genannten Studien besser erfasst ist, und drei Verfahren der Autoren Alkhatib und Scholl.

Neben den Vergleichsstudien liefern auch viele Veröffentlichungen einzelner Verfahren Messwerte für deren Leistungsfähigkeit. Diese zu einem gesamten quantitativen Vergleich zu kombinieren, ist jedoch nicht möglich, da die Messungen nicht auf einem einheitlichen Testdatenkorpus beruhen und daher nicht direkt vergleichbar sind. Zudem berücksichtigt keine der Analysen Daten, die mit Geschäftsdokumenten vergleichbar sind. Wenngleich strukturell vergleichbare Dokumente betrachtet

[5] Vgl. Sakr (2009b) S. 311.
[6] Vgl. Sakr (2009a) S. 51.
[7] Vgl. Ng/Lam/Cheng (2006), S. 1-28.
[8] Für diesen und den nächsten Satz vgl. Nair (2007), S. 1-23.
[9] Für diesen und den nächsten Satz vgl. Mahalakshmi/Hanchate (2013), S. 165-172.

werden, sind diese um ein Vielfaches größer. Da bei großen Dokumenten Redundanzen entstehen, die bei kleinen Dokumenten nicht vorhanden sind, ist die Kompressionsrate nicht vergleichbar.

3.2 Klassifikationssystem

Die Kompression von XML-Dokumenten ist ein breites Feld, bei dem unterschiedliche Techniken zum Einsatz kommen und verschiedene Zielsetzungen verfolgt werden. Wie in der Einleitung zu Kapitel 3 beschrieben ist, ist der Aufwand einer quantitativen Untersuchung aller verfügbaren Verfahren nicht sinnvoll. Aus diesem Grund wird in dieser Arbeit eine eigene Klassifikation entwickelt, um die Verfahren auf qualitativer Ebene einzugrenzen.
Keine in der Literatur gefundene Systematik zur Unterscheidung von Kompressionsverfahren für XML-Daten liefert die nötige Trennschärfe, um Verfahren im Hinblick auf ihre Kompressionsleistung zu unterscheiden. Diese Erkenntnis wurde gewonnen, indem die in implementierter Form vorliegenden Verfahren auf eine Reihe von Testdaten angewandt wurden. Dabei zeigte sich, dass keine vorhandene Klassifikation alleine klare Rückschlüsse auf das Leistungsniveau ermöglicht. Daher werden in diesem Abschnitt Klassifikationsmerkmale erarbeitet, um diese Lücke zu schließen.

3.2.1 Art und Ziel der Klassifikation

In der Literatur finden sich Klassifikationen für unterschiedliche Blickwinkel auf Kompressionsverfahren. Da auch in dieser Arbeit verschiedene Aspekte relevant sind, wird eine monohierarchische Klassifikation nicht für sinnvoll erachtet und stattdessen eine facettierte Ordnung verwendet. Bei einer sogenannten Facettenklassifikation werden Untersuchungsgegenstände nach unterschiedlichen Blickwinkel, sogenannten Facetten, untergliedert.[10] In jeder Facette erfolgt dabei eine eigenständige Klassifikation, beschränkt auf den jeweiligen Blickwinkel.[11]

[10] Vgl. Stock/Stock (2008), S. 273.
[11] Vgl. Kwasnik (1999), S. 39 f.

Die Facettenklassifikation geht auf Ranganathan zurück, der 1933 eine universale Klassifikation für Wissen entwickelte.[12] Diese wurde in vielen Bereichen adaptiert und ist heute beispielsweise Teil von Suchwerkzeugen im World Wide Web. Auch Sakr verwendet in seinen beiden Vergleichsstudien Sakr (2009b) und Sakr (2009a) implizit diese Form der Klassifikation, wenngleich er sie nicht explizit als solche bezeichnet.[13]

Die hier verfolgte Zielsetzung ist die Schaffung eines kontrollierten Vokabulars, auf dessen Basis die Verfahren beschrieben werden können. Entsprechende Ansätze finden sich auch in der inhaltlichen Erschließung von Dokumenten.[14] Das Vokabular ermöglicht eine klare Abgrenzung der zu untersuchenden Verfahren. Auf diese Weise kann der Auswahlprozess abstrakt definiert werden und kann dadurch auch mit hier nicht betrachteten Verfahren reproduziert werden. Daraus ergibt sich der Vorteil, nicht alle auszuschließenden Verfahren explizit aufführen zu müssen.

Bei der Facettenklassifikation werden zunächst die grundlegenden Kategorien bzw. Facetten bestimmt, anhand derer der Untersuchungsgegenstand betrachtet wird.[15] Diese können auch hierarchisch gruppiert werden.[16] Anschließend wird in jeder Kategorie eine Klassifikation aus Einfachklassen erstellt.[17] Diese charakterisieren sich dadurch, dass sie sich nur aus einem einzelnen Merkmal ableiten.[18] Auf die erzeugten Einfachklassen, die in der Literatur auch als Foci bezeichnet werden[19], wird in dieser Arbeit als Ausprägungen der Facetten Bezug genommen.

Die weiteren Schritte der Facettenklassifikation, zu denen das Ordnen der Facetten und deren Ausprägungen sowie das Erstellen einer Notation gehört[20], sind hier nicht nötig, da die Klassifizierung der Verfahren nicht explizit durchgeführt wird. Das System wird lediglich dazu verwendet, um den Auswahlprozess abstrakt beschreiben zu können.

[12] Für diesen und den nächsten Satz vgl. Stock/Stock (2008), S. 275.
[13] Sakr (2009b) S.304 und Sakr (2009a) S. 50 f.
[14] Vgl. Buchanan (1989), S. 12 f.
[15] Vgl. Kwasnik (1999) S. 39.
[16] Vgl. Buchanan (1989), S. 48.
[17] Vgl. Kwasnik (1999) S. 39 u. Buchanan (1989) S.48.
[18] Vgl. Buchanan (1989), S. 18.
[19] Vgl. Buchanan (1989), S. 47.
[20] Vgl. Kwasnik (1999) S. 40 u. Buchanan (1989) S. 48 f.

Die Gliederungen der nachfolgenden beiden Abschnitte basiert auf der hierarchischen Gruppierung der aus der Literatur abgeleiteten Facetten. Die in dieser Arbeit vorgenommene Aufteilung in eine anwendungsorientierte und eine technische Gruppe hat mehrere Gründe. Erstens lassen sich alle Einteilungen der Literatur diesen beiden Ebenen zuordnen. Zweitens wird damit die Funktionalität eines Verfahrens von dessen technischer Realisierung getrennt. Drittens sind die anwendungsorientierten Merkmale einfacher aus den Verfahrensbeschreibungen abzulesen als die technischen, bei denen mehr Analyseaufwand nötig ist. Viertens dienen die anwendungsorientierten Facetten der Abgrenzung des Untersuchungsraums, wohingegen die technischen im Rahmen der Untersuchung verwendet werden.

3.2.2 Facetten auf Anwendungsebene

Die einzelnen Klassifikationen auf Anwendungsebene verwenden Merkmale, die Verhalten und Funktionalität von Kompressionsverfahren beschreiben. Abbildung 3.1 liefert einen Überblick über die identifizierten Facetten und Merkmale der Anwendungsebene. In den ersten beiden Abschnitten werden grundlegende Unterscheidungskriterien thematisiert, die bei jeder Art von Kompression relevant sind. Diese fließen beispielsweise auch bei Malaka/Butz/Hussmann (2009) zur allgemeinen Unterscheidung von Kompressionsverfahren ein.[21] In der funktionalen Perspektive im dritten Abschnitt werden alle in der Literatur thematisierten Funktionen von Kompressoren für XML-Daten behandelt.

3.2.2.1 Arten von Verfahren zur XML-Kompression

Ein grundlegendes Unterscheidungsmerkmal für Kompressionsverfahren, die zur Verarbeitung von XML-Dokumenten eingesetzt werden können, findet sich in den beiden Vergleichsstudien Sakr (2009b) und Sakr (2009a). Dort werden Verfahren grundlegend in solche, die spezielle Eigenschaften von XML-Dokumenten berücksichtigen und universelle Textkompressoren unterteilt[22] Dies begründet sich darin,

[21] Vgl. Malaka/Butz/Hussmann (2009), S. 72 f.
[22] Vgl. Sakr (2009b) S. 304 und Sakr (2009a) S. 50

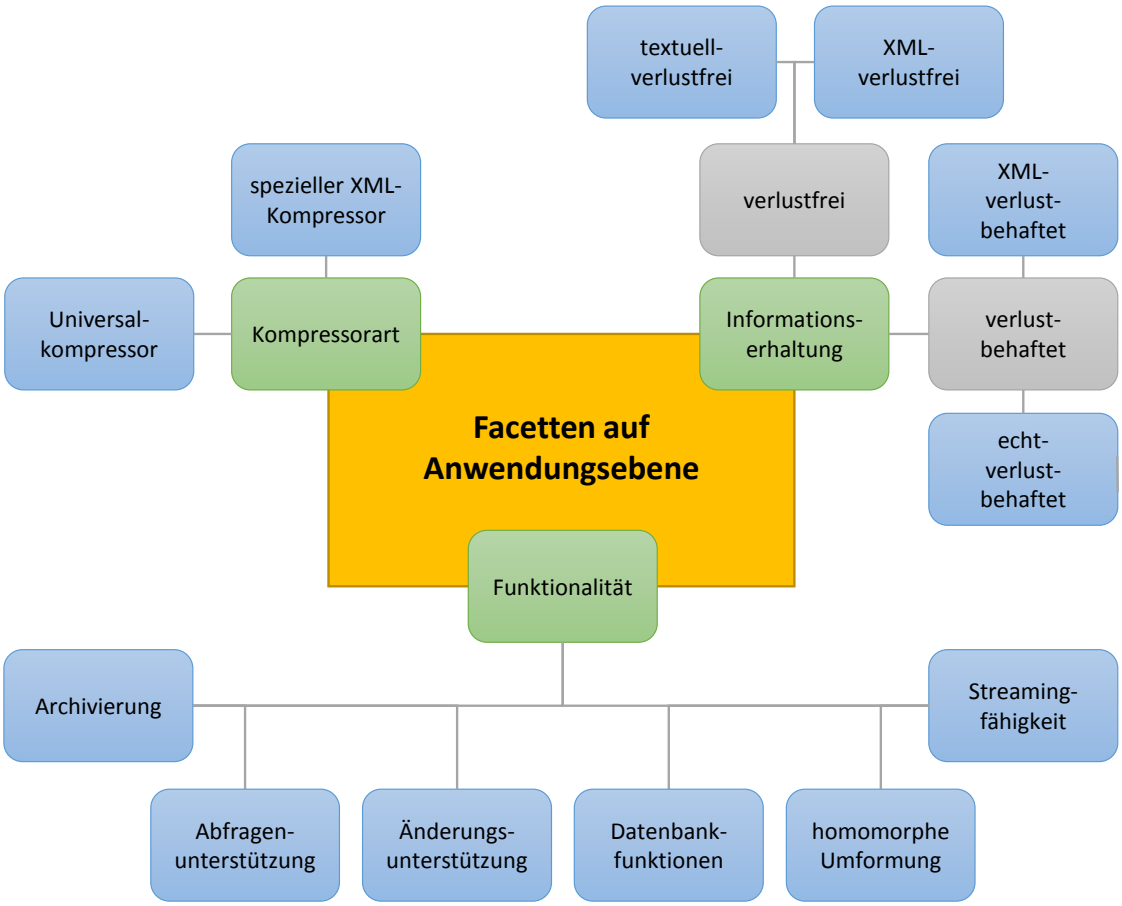

Abbildung 3.1: Facetten auf Anwendungsebene zur Unterscheidung von Kompressionsverfahren für XML-Dokumente (Eigene Darstellung)

dass jedes XML-Dokument als Text aufgefasst und von Universalkompressoren verarbeitet werden kann. Die semantische Strukturierung von XML-Dokumenten wird dabei jedoch nicht berücksichtigt. Durch die Verwendung dieser zusätzlichen Informationen können Verfahren besser an ihren Zweck angepasst werden, wodurch im Speziellen auch höhere Kompressionsraten möglich sein sollen.[23] Die soeben beschriebene Unterteilung wird hier als **Kompressorsart** mit den Ausprägungen **Universalkompressor** und **spezieller XML-Kompressor** übernommen.

[23] Vgl. Sakr (2009b), S. 304.

3.2.2.2 Informationserhaltung im Kontext von XML

Die Kompression von Daten kann entweder verlustfrei oder verlustbehaftet erfolgen. Im Kontext allgemeiner Textdokumente bedeutet die Verlustfreiheit, dass das dekomprimierte Dokument eine exakte Kopie des Originals ist. Für die Verlustfreiheit von XML-Dokumenten verwenden Müldner u. a.(2008) eine aus textueller Sicht weniger strikte Anforderung. Demzufolge sind zwei Dokumente identisch, wenn deren kanonische Form übereinstimmt.[24] Diese Form entfernt beispielsweise mehrfache Leerzeichen zwischen Attributen und sortiert diese alphabetisch nach deren Bezeichner.[25] Ferner definieren Müldner u. a.(2008) XML-spezifische Verfahren als verlustbehaftet, wenn sich die kanonische Form des Originals und die des wiederhergestellten Dokuments nur dann entsprechen, wenn alle Leerzeichensymbole und Zeilenumbrüche entfernt wurden, die nicht Teil der Nutzdaten, sondern zur optischen Aufbereitung für einen menschlichen Leser vorgesehen sind.[26]
Aufgrund der speziellen Anforderungen bei XML-Dokumenten ergeben sich für die Facette **Informationserhaltung** folgende Ausprägungen: **textuell-verlustfrei**, **XML-verlustfrei**, **XML-verlustbehaftet** und **echt-verlustbehaftet**. Als textuell-verlustfrei werden Verfahren bezeichnet, die entsprechend der allgemeinen Definition verlustfrei sind. Die nach allgemeinen Gesichtspunkten verlustbehafteten Verfahren untergliedern sich des Weiteren in drei Ausprägungen, von denen sich die beiden XML-spezifischen direkt aus den oben beschrieben Eigenschaften ergeben. Alle Verfahren, die keine der bisher beschriebenen Merkmale aufweisen, bilden die Gruppe der echt-verlustbehafteten Verfahren. Ein Beispiel für diese Gruppe liefert die von Cannataro/Comito/Pugliese (2002) verwendete Vorgehensweise, die Nutzdaten zu komprimieren, indem die Werte aggregiert werden, wodurch die Einzelwerte verloren gehen.[27]

[24] Vgl. Müldner u. a.(2008), S. 2.
[25] eine detaillierte Beschreibung der kanonischen Form liefert http://www.w3.org/TR/xml-c14n
[26] Vgl. Müldner u. a.(2008), S. 2.
[27] Vgl. Cannataro/Comito/Pugliese (2002), S. 328 f.

3.2.2.3 Funktionale Perspektive

In der Literatur finden sich sechs funktionale Eigenschaften, anhand derer Verfahren zur Kompression von XML-Dokumenten unterschieden werden. Vier davon beschränken sich auf XML-spezifische Verfahren. Lediglich die Archivierung sowie das Streaming sind auch mit universellen Kompressoren möglich.
Für die Facette **Funktionalität** konnten folgende Ausprägungen identifiziert werden:

Abfragenunterstützung Diese, im Englischen als *queryable* bezeichnete Funktionalität steht für die Möglichkeit, über eine Abfragesprache auf einzelne Daten eines XML-Dokuments zugreifen zu können, ohne das gesamte Dokument dekomprimieren zu müssen. Ihre häufige Thematisierung in der Literatur zeigt die hohe Bedeutung für XML-spezifische Kompressionsverfahren. Zielsetzung der Verfahren mit Abfragenunterstützung ist die Minimierung der Verarbeitungsgeschwindigkeit und Ressourcennutzung auf dekomprimierender Seite.[28]

Änderungsunterstützung Die im Englischen als *updateable* bezeichnete Funktionalität ermöglicht es, komprimierte XML-Dokumente zu verändern, ohne diese vollständig neu komprimieren zu müssen.[29]

Datenbankfunktionen Die Datenbankfunktionalität ist ein Merkmal, das weiter differenziert betrachtet werden muss. Manche Autoren, wie beispielsweise Müldner u. a.(2008), fordern lediglich umfangreiche Abfragefunktionen und hohe Geschwindigkeit bei der Dekompression.[30] Da diese Definition keine ausreichende Abgrenzung gegenüber der abfragenunterstützenden Gruppe liefert, wird hier dem Verständnis von Wong/Lam/Shui (2007) gefolgt, die neben schnellen Abfragemöglichkeiten auch Änderungs- und Navigationsfunktionen erwarten.[31] Hingegen wurden Verfahren, die sich als XML-Datenbanken im

[28] Vgl. Sakr (2009b) S. 304 und Müldner u. a.(2008) S. 2
[29] Vgl. Wong/Lam/Shui (2007), S. 1073 S. 1073, Müldner/Miziołek/Fry (2012), S. 131, Tolani/Haritsa (2002), S. 228 und Alkhatib/Scholl (2008a), S. 605
[30] Vgl. Müldner u. a.(2008), S. 2.
[31] Vgl. Wong/Lam/Shui (2007), S. 1073 f.

Sinne von Müldner u. a.(2008) oder noch freier verstehen, als abfragenunterstützend klassifiziert.

Archivierung Bei Verfahren, die diese Eigenschaft erfüllen, steht die Kompressionsrate und -geschwindigkeit im Vordergrund.[32] Sie wird nur in wenigen untersuchten Veröffentlichungen explizit genannt. Implizit ist sie allerdings immer dann vorhanden, wenn keine Abfrage-, Änderungs- oder Navigationsfunktionalität realisiert ist. Im Speziellen sind alle Universalkompressoren dieser Kategorie zuzuordnen.

Streamingfähigkeit Diese deutsche Umschreibung der im Englischen verwendeten Begriffe *streamable* oder *online* steht für die Eigenschaft, dass ein Dokument nicht am Stück, sondern abschnittsweise komprimiert bzw. dekomprimiert werden kann.[33] Dass dies gerade für große Dokumente wichtig ist, zeigt ihre häufige Nennung in der Literatur.

homomorphe Umformung Diese Eigenschaft erfordert, dass das komprimierte XML-Dokument auf vergleichbare Art und Weise wie das unkomprimierte Original verwendet werden kann. Dies beinhaltet beispielsweise die Navigation im Dokument zum Zugriff auf Nutzdaten sowie den Vergleich des Inhalts von Elementen und Attributen.[34]

3.2.3 Technische Facetten

Bei der Kompression von XML-Dokumenten sind einerseits dessen Strukturdaten und andererseits dessen Nutzdaten zu verarbeiten. Im Bereich der Strukturverarbeitung sind drei Facetten zu unterscheiden: die Arten der Strukturkompression, die Verarbeitung der Redundanz in der Struktur und die unterstützten Schemasprachen bei schemabasierten Verfahren. Bei Nutzdatenverarbeitung unterscheiden sich die Verfahren in Bezug auf die Strategie der Aufteilung der Nutzdaten in Da-

[32] Vgl. Sakr (2009a) S. 51 und Müldner u. a.(2008) S. 2
[33] Vgl. Böttcher/Steinmetz/Klein (2007), S. 93.
[34] Vgl. Sakr (2009b), S. 304.

tencontainer, die Reihenfolge, in der dies geschieht und letztlich die Art und Weise, in der die Container komprimiert werden. Einen Überblick über alle technischen Facetten und Merkmale stellt Abbildung 3.2 dar.

3.2.3.1 Art der Strukturkompression

Die explizite Unterscheidung von Verfahren nach der Art der Strukturkompression findet sich in zwei unterschiedlich detaillierten Ausprägungen. Bei der gröberen Einteilung unterscheiden die Autoren nur zwischen schemabasierten und schemaneutralen Verfahren.[35] Böttcher und Hartel unterscheiden in der schemaneutralen Gruppe weiter zwischen Verfahren zur Beseitigung textueller sowie struktureller Redundanz.[36]

Die Aufteilung kann hier nicht bestätigt werden, da sich sowohl in der Gruppe der Verfahren mit Beseitigung der textuellen Redundanz als auch in der schemabasierten Gruppe Verfahren finden, die die Redundanz in der Struktur berücksichtigen. In der schemaneutralen Gruppe mit Entfernung textueller Redundanz ist das Verfahren von Lin u. a.(2005) als Beispiel zu nennen. Bei diesem werden zunächst die Bezeichner der Struktur in ein Wörterbuch ausgelagert und anschließend die Redundanz in der Struktur auf Basis einer Grammatik entfernt.[37] Ferner liefern Böttcher/Hartel/Messinger (2009) ein schemabasiertes Verfahren, bei dem zusätzlich strukturelle Wiederholungen entfernt werden.[38] Folglich wird in dieser Arbeit die Entfernung der Redundanz in der Struktur als zusätzliche Unterscheidungsebene identifiziert, die später in einem gesonderten Abschnitt betrachtet wird. Die Entfernung der textuellen Redundanz auf Bezeichnungsebene findet sich bei allen schemaneutralen Verfahren. Im Allgemeinen werden dabei die Bezeichnungen der XML-Bausteine durch möglichst kurze Schlüssel ersetzt. Bei manchen abfragenunterstützenden Verfahren, wie beispielsweise bei Hu/Zhang/Yuan (2012),

[35] Vgl. beispielsweise Sakr (2009b) S.50, Sakr (2009a) S.304 oder League/Eng (2007a) S. 10.
[36] Vgl. Böttcher/Hartel/Heinzemann (2008) S. 19, Böttcher/Hartel/Heinzemann (2009) S. 76, Böttcher/Hartel/Messinger (2010) S. 110 und Böttcher/Hartel/Messinger (2011) S. 460.
[37] Vgl. Lin u. a.(2005), S. 661-663.
[38] Vgl. Böttcher/Hartel/Messinger (2009), S. 2-5.

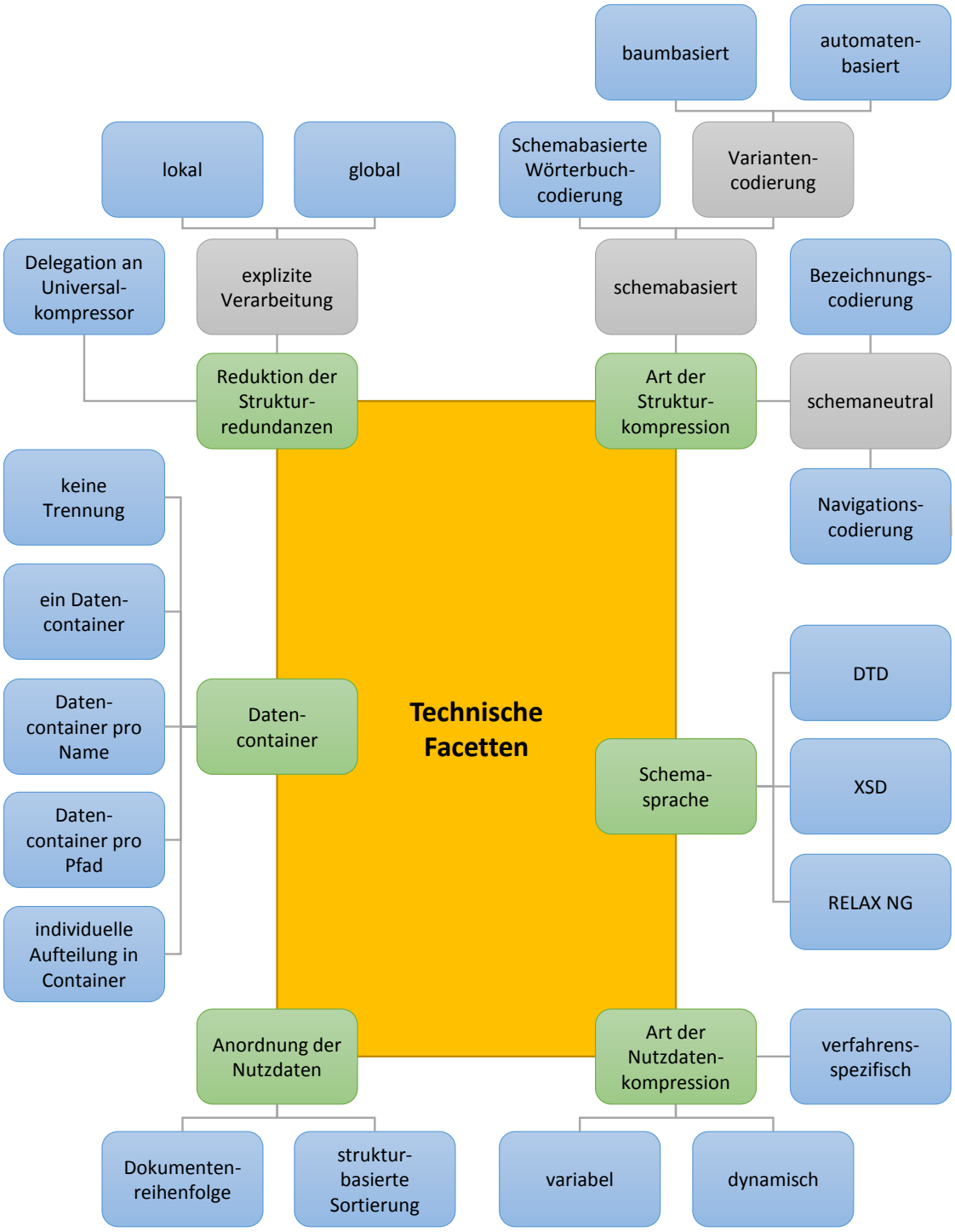

Abbildung 3.2: Technische Facetten zur Unterscheidung von Kompressionsverfahren für XML-Dokumente (Eigene Darstellung)

werden diese Schlüssel so gestaltet, dass sie zusätzlich die hierarchischen Beziehungen zwischen den XML-Elementen abbilden, um eine Navigation auf der komprimierten Struktur zu ermöglichen.[39] Aus diesem Grund wird die Gruppe hier in **Bezeichnungscodierungen** und **Navigationscodierungen** unterteilt. Erstere zielen einzig auf kurze Identifikatoren ab. Bei letzteren konkurriert das Ziel, die Identifikatoren möglichst kurz zu halten, mit der Aufgabe, darin auch strukturelle Informationen abzubilden.

Wie die Gruppe der schemaneutralen Verfahren ist auch die der schemabasierten weiter zu unterteilen. Nach dem Grade der Nutzung der Schemainformationen ergeben sich zwei Gruppen: **schemabasierte Wörterbuchcodierungen** und **Variantencodierungen**. Bei ersterer werden aus den Schemata lediglich Wörterbücher abgeleitet, mit denen die Struktur der XML-Dokumente komprimiert wird. Ein Beispiel dafür liefert der Ansatz von Serin (2003).[40] In der zweiten Gruppe wird die Schemainformation intensiver genutzt. Wie beispielsweise bei Levene/ Wood (2002) wird dazu das Regelwerk der Schemainstanz instrumentalisiert, um eine kompaktere Abbildung der Dokumentenstruktur zu erzeugen.[41] Diese Abbildung ist als Sammlung von Informationen zu sehen, die es ermöglicht, aus dem Regelwerk die originale Dokumentenstruktur zu rekonstruieren. Der Name ergibt sich aus der Tatsache, dass ein Schema eine Menge von Strukturvarianten definiert, von denen genau eine die Struktur eines konkreten XML-Dokuments darstellt.

Die Variantencodierung nutzt die Eigenschaft der Schemaregeln, einen Rahmen für die Gestaltung der Struktur vorzugeben. Durch dieses zusätzliche Wissen ist eine kompaktere Codierung möglich. Sind beispielsweise in einem Schema bestimmte Elemente eines XML-Dokuments als verpflichtend definiert, müssen diese nicht in einer komprimierten Darstellung der Struktur enthalten sein, da sie ohnehin direkt aus dem Schema abgeleitet werden können. Zudem wird bei der Codierung eines Bausteins die Tatsache genutzt, dass das Schema nur eine bestimmte Menge an

[39] Vgl. Hu/Zhang/Yuan (2012), S. 713-715.
[40] Vgl. Serin (2003), S. 42-47.
[41] Vgl. Levene/Wood (2002), S. 56-60.

Auswahlmöglichkeiten bietet. Da nur die vorliegende Alternative zu codieren ist, sind die Codewörter sehr kompakt.[42]

Die Gruppe der Variantencodierungen ist ferner nach der Art der Modellierung der Schemainformationen in **baumbasierte** und **automatenbasierte** Verfahren zu untergliedern. In der ersten Gruppe wird eine Schemainstanz in einen Baum mit verschiedenen Knotentypen überführt. Die unterschiedlichen Typen resultieren aus den verschiedenen Arten von Informationen, die ein Schema liefert. Im Wesentlichen ergeben sich Knoten für Elemente, Attribute, Häufigkeitsdefinitionen und Elementkompositoren. Bei der Abbildung der Struktur eines XML-Dokuments wird für jeden Knoten, der eine variable Gestaltung ermöglicht, ein Wert gespeichert, der die konkrete Gestaltung symbolisiert und damit in Verbindung mit der Schemainstanz zur Rekonstruktion der Struktur ausreicht.

In der Gruppe der automatenbasierten Verfahren wird die Tatsache genutzt, dass ein Schema als formale Grammatik aufgefasst werden kann. Dementsprechend wird aus der jeweiligen Schemainstanz ein Automat gebildet. Über die Zustandsübergänge wird somit modelliert, welche XML-Bausteine aufeinander folgen dürfen. Der erzeugte Automat fungiert zur Abbildung der Struktur eines XML-Dokuments als Transduktor. Dieser ist so gestaltet, dass er bei seiner Ausführung Ausgaben erzeugt, die für jeden Zustand den gewählten Zustandsübergang identifizieren. Auf Basis des Automaten kann damit aus diesen Ausgaben die Struktur des Dokuments rekonstruiert werden.

Zusammenfassend können Verfahren bezüglich der **Art der Strukturkompression** auf oberster Ebene in **schemaneutrale** und **schemabasierte** untergliedert werden. Erstere können zudem in **Navigationscodierungen** und **Bezeichnungscodierungen** unterteilt werden. Letztere unterteilen sind in **schemabasierte Wörterbuchcodierungen** und **Variantencodierungen**. Letztlich können innerhalb der **Variantencodierungen** noch **baumbasierte** und **automatenbasierte** Ansätze unterschieden werden.

[42] Für eine detaillierte Beschreibung der Variantencodierung sei auf Kapitel 4 und im Speziellen auf das Beispiel im zweiten Teil des Abschnitts 4.3.2 verwiesen.

3.2.3.2 Reduktion der Strukturredundanzen

Verfahren, die die Redundanz in der Struktur beseitigen, sind nach Böttcher/Hartel/Messinger (2010) diejenigen, die Struktur auf Basis einer Grammatik verarbeiten.[43] Diese Definition umfasst jedoch nicht alle Herangehensweisen, mit denen Redundanz in der Struktur reduziert werden kann. Beispielsweise vergleichen Cheng/Ng (2004) den strukturellen Aufbau benachbarter Elemente und komprimieren identische Strukturen.[44] Aus diesem Grund ist hier eine detailliertere Einteilung nötig.

Zur Unterscheidung von Verfahren, die Redundanz in der Struktur eliminieren, konnten in dieser Arbeit drei Klassen identifiziert werden. Eine Klasse bilden die Verfahren, die die Struktur aufbereiten und an einen nachgelagerten Universalkompressor senden. Zu diesen gehört beispielsweise der Ansatz von Ferragina u. a.(2006).[45]

Die beiden weiteren Klassen, bei denen jeweils eigenständige Techniken ohne Universalkompressor realisiert werden, unterscheiden sich dadurch, dass in einer die Redundanz global und in der anderen nur lokal beseitigt wird. Bei Verfahren mit lokalen Ansätzen werden Wiederholungen in der Struktur nur in benachbarten Elementen gekürzt. Dadurch werden redundante Strukturen nur dann kompakter repräsentiert, wenn sie auf einer Ebene und nicht in unterschiedlicher Strukturtiefe im Dokument auftreten. Ein Beispiel für diese Herangehensweise liefert das bereits erwähnte Verfahren von Cheng/Ng (2004).[46]

Die Definition von Böttcher/Hartel/Messinger (2010) bildet die Basis für die dritte Klasse von Ansätzen. Die globale Entfernung von Redundanzen ist allerdings nicht nur auf Basis einer Grammatik möglich. Böttcher/Hartel/Heinzemann (2009) bilden beispielsweise die Struktur als Baum ab und erweitern diesen zu einem gerichteten Graphen, indem redundante Teilstrukturen durch Zeiger ersetzt werden.[47]

[43] Vgl. Böttcher/Hartel/Messinger (2010), S. 110.
[44] Vgl. Cheng/Ng (2004), S. 219-226.
[45] Vgl. Ferragina u. a.(2006), S. 753 f.
[46] Vgl. Cheng/Ng (2004), S. 219-226.
[47] Vgl. Böttcher/Hartel/Heinzemann (2009), S. 67 f.

Zusammenfassend ergeben sich für die Facette **Reduktion der Strukturredundanzen** einerseits die explizite Verarbeitung von Redundanz in der Struktur auf **lokaler** oder **globaler** Ebene und andererseits die Delegation dieser Aufgabe an einen **nachgelagerten Universalkompressor**.

3.2.3.3 Unterstützte Schemasprachen

Die **Schemasprache** ist ein Merkmal, das bisher noch nicht zur Unterscheidung von Verfahren verwendet wurde, jedoch insofern charakteristisch ist, als dass alle schemabasierten Verfahren primär für eine Schemasprache konzipiert sind. Am häufigsten wird hierbei **DTD** verwendet, gefolgt von **XSD** und letztlich **RELAX NG**.

Informationen zur Erweiterbarkeit eines Verfahrens auf andere Schemasprachen sind in der Literatur nicht für alle Verfahren vorhanden. Zudem sind manche Ausführungen oberflächlich gestaltet, wie beispielsweise bei Böttcher/Steinmetz/Klein (2007), wo ohne Erklärung angegeben wird, dass ihr Verfahren auf alle Schemasprachen erweiterbar sei.[48] Harrusi/Averbuch/Yehudai (2006a) definieren hingegen formal, welche Schemasprachen möglich sind.[49] Die als *einfach* bezeichnete Anpassung wird hingegen nicht beschrieben.

Die Unterscheidung nach der verwendeten Schemasprache ist sinnvoll, da die Schemasprachen unterschiedliche Konzepte bieten. Die Unterschiede führen dazu, dass die Erweiterbarkeit eines Verfahrens auf andere Schemasprachen nicht immer möglich ist und daher geprüft werden muss. Zudem führt die Erweiterung auf eine Sprache mit größerem Konzeptumfang zu der Frage, wie diese zusätzlichen Konzepte berücksichtigt werden.

Der Vergleich der relevanten Schemasprachen in Kapitel 2.1.3 legt bereits grundsätzliche Unterschiede der Sprachen offen. Bevor im Folgenden auf die Erweiterbarkeit eingegangen wird, werden zunächst einzelne Unterschiede aus Abschnitt 2.1.3 thematisiert. Beispielsweise bietet XSD im Vergleich zu DTD und RELAX

[48] Vgl. Böttcher/Steinmetz/Klein (2007), S. 94.
[49] Für diesen und den nächsten Satz vgl. Harrusi/Averbuch/Yehudai (2006a), S. 11.

NG eine wesentlich feinere Typisierung für Nutzdaten. Bei der Erweiterung eines Verfahrens auf XSD würde dieses Potential ungenutzt bleiben. Zudem besitzt XSD mit `<xsd:all>` einen Kompositor, der bei DTD nicht vorhanden ist. Ähnliches gilt für den `interleave`-Kompositor von RELAX NG, der in keiner der beiden anderen Sprachen zu finden ist. Im Gegensatz zu Datentypen für Nutzdaten können zusätzliche Kompositoren nicht unberücksichtigt bleiben, ohne die Funktion des Verfahrens zu beeinträchtigen.

Die Erweiterbarkeit eines Verfahrens auf eine Sprache ist gegeben, wenn das Verfahren in der Lage ist, die Ausdrucksstärke der jeweiligen Sprache zu erfassen. Zur Unterscheidung der Ausdrucksstärke der Schemasprachen wird in der Literatur das Konzept der formalen Grammatiken verwendet.[50] Eine konkrete Schemadefinition legt fest, wie ein XML-Dokument beschaffen sein darf, damit es zu der von dieser Definition beschriebenen Gruppe gehört. Dies ist analog zur Festlegung einer Sprache durch eine Grammatik zu sehen.[51] In diesem Sinne entspricht eine Schemasprache einer Gruppe von Grammatiken. In Murata u. a.(2005) wird genau für die drei hier relevanten Schemasprachen jeweils eine Klasse von Grammatiken formal definiert und diese in eine hierarchische Beziehung gebracht. Zu diesem Zweck werden Baumgrammatiken verwendet, da diese die Struktur von XML-Dokumenten berücksichtigen und damit besser geeignet sind als Grammatiken für Zeichenketten.[52]

Die allgemeinste Klasse, die der Ausdrucksstärke von RELAX NG entspricht, ist die der regulären Baumgrammatiken.[53] Für XSD und DTD muss die Klasse weiter eingeschränkt werden, da für diese die Inhaltsmodelle deterministisch sein müssen. Das bedeutet, dass alleine das nächste Element ausreichend sein muss, um eine Ableitungsregel eindeutig zu bestimmen. Diese Eigenschaft wird von Single-Type Baumgrammatiken erfüllt.[54] Sie decken damit die Ausdrucksstärke von XSD größtenteils ab. Einzig der Einsatz von Platzhaltern, die es ermöglichen, an deren Stelle

[50] vgl. Harrusi/Averbuch/Yehudai (2006b) S. 402 und Murata u. a.(2005) S. 661
[51] Vgl. Harrusi/Averbuch/Yehudai (2006b), S. 402.
[52] Vgl. Murata u. a.(2005), S. 662.
[53] Für diesen und die nächsten zwei Sätze vgl. Murata u. a.(2005), S. 676 u. 690.
[54] Für diesen und den nächsten Satz vgl. Murata u. a.(2005), S. 665.

beliebige Elemente oder Attribute zu verwenden, fällt nicht darunter.[55] Deren Verwendung ist im Kontext von XML-basierten Standards für Geschäftsdokumente jedoch nicht zu erwarten. Für DTD sind weitere Einschränkungen nötig, da es hier im Gegensatz zu XSD nicht möglich ist, Inhaltsmodelle zu definieren, die mehrfach verwendet werden können.[56] Damit ist DTD der Klasse der lokalen Baumgrammatiken zuzuordnen.[57]

Jede lokale Baumgrammatik ist auch immer eine Single-Type Grammatik, die wiederum eine Spezialisierung einer regulären Baumgrammatik ist.[58] Daraus kann abgeleitet werden, dass ein Verfahren, dass mächtig genug ist, RELAX NG zu verarbeiten, auch grundsätzlich auf XSD und DTD erweitert werden kann. Ob ein Verfahren für DTD auf die anderen beiden Schemasprachen erweiterbar ist, muss jeweils geprüft werden. Außerdem ist die Erweiterbarkeit von Verfahren für XSD auf DTD immer gegeben und muss lediglich für RELAX NG im Einzelfall untersucht werden.

3.2.3.4 Facetten der Nutzdatenkompression

XML-spezifische Kompressionsverfahren lassen sich hinsichtlich der Nutzdatenkompression anhand von drei Merkmalen charakterisieren, die im Folgenden beschrieben werden.

Datencontainer

XML-Dokumente bestehen aus Struktur- und Nutzdaten in jeweils textueller Form. Aus diesem Grund gibt es einerseits Verfahren, die einen Kompressionsansatz für das gesamte Dokument verwenden, wie dies beispielsweise bei Toman (2003) der Fall ist[59], und solche, die beide Teile getrennt verarbeiten.

Die getrennte Verarbeitung von Struktur- und Nutzdaten ist eine Vorgehensweise, die bereits Liefke/Suciu (2000) bei dem ersten XML-spezifischen Kompressions-

[55] Vgl. Murata u. a.(2005), S. 675 f.
[56] Vgl. Murata u. a.(2005), S. 671.
[57] Vgl. Murata u. a.(2005), S. 664.
[58] Vgl. Murata u. a.(2005), S. 666.
[59] Vgl. Toman (2003), S. 53-73.

verfahren verwendeten.[60] Bei Verfahren dieser Art werden die textuellen Bestandteile eines XML-Dokuments in sogenannte Container aufgeteilt, die jeweils einzeln komprimiert werden. Manche der Verfahren legen alle Nutzdaten in einen Container und andere verteilen sie auf mehrere Container. Ein einzelner Datencontainer wird beispielsweise bei dem Verfahren von Levene/Wood (2002) genutzt.[61]
Die Verteilung der Nutzdaten auf Container erfolgt auf Basis von Informationen aus der Struktur. Häufig werden dafür der Name (Cheng/Ng (2004)) und der Pfad (Skibiński/Swacha (2007)) des Elements oder Attributs, das die Nutzdaten enthält, verwendet.[62] Der Container, in den die Nutzdaten gelegt werden, wird somit nach dem Namen oder dem Pfad des Bausteins, der die Daten enthält, ausgewählt.
In wenigen Beispielen wird die Aufteilung der Daten in Container sehr individuell gestaltet. Beispielsweise erfolgt die Zuordnung der Daten bei Liefke/Suciu (2000) auf Basis sehr fein und individuell gestaltbarer Regeln. Sie decken das ganze Spektrum von einem einzelnen Datencontainer bis zur Aufteilung nach Strukturpfaden ab. Bei Weimin (2003) werden die Nutzdaten von Bausteinen gleicher Namen nur dann in einen Container gruppiert, wenn die Bausteine innerhalb der Struktur auf der gleichen Ebene liegen.[63] Cheney (2001) unterscheidet nur zwischen Nutzdaten in XML-Elementen und solchen in Attributen und verwendet dementsprechend lediglich zwei Datencontainer.[64] Die Aufteilung nach Namen wird bei Schneider u. a.(2014) insofern eingeschränkt, dass die resultierenden Container erst ab einer bestimmten Größe separat komprimiert und darunter zusammengefasst werden.[65]
Zusammenfassend ergeben sich für die Facette **Datencontainer** folgende Ausprägungen:

- **keine Trennung**
- **ein Datencontainer**

[60] Vgl. Liefke/Suciu (2000), S. 156 f.
[61] Vgl. Levene/Wood (2002), S. 59.
[62] Vgl. Cheng/Ng (2004), S. 226 und Skibiński/Swacha (2007), S. 336
[63] Vgl. Weimin (2003), S. 77.
[64] Vgl. Cheney (2001), S. 167-170.
[65] Vgl. Schneider u. a.(2014).

- **Datencontainer pro Name**
- **Datencontainer pro Pfad**
- **individuelle Aufteilung in Container**

Anordnung der Nutzdaten

Die **Anordnung der Nutzdaten** in den Datencontainern erfolgt bei nahezu allen Verfahren in der Reihenfolge ihres Auftretens im Dokument, was hier als **Dokumentenreihenfolge** bezeichnet wird. Es gibt aber auch Verfahren, die die Nutzdaten auf Basis ihres Strukturpfades im Dokument sortieren. Den Ansatz der **strukturbasierten Sortierung** nutzen beispielsweise Ferragina u. a.(2006) und Skibiński/Grabowski/Swacha (2008).[66]

Art der Nutzdatenkompression

Als letzte Eigenschaft zur Unterscheidung der Verfahren wird die **Art der Nutzdatenkompression** herangezogen. Dabei wird unterschieden, ob zur Kompression der Nutzdaten eine Technik verwendet wird, die **verfahrensspezifisch** ist und daher nicht durch andere Techniken ersetzt werden kann oder ob universale Algorithmen, im Speziellen Universalkompressoren, zum Einsatz kommen. Verfahrensspezifische Techniken zur Nutzdatenkompression werden beispielsweise bei Toman (2004) oder Harrusi/Averbuch/Yehudai (2006b) eingesetzt.[67]

Von den Verfahren, die universale Kompressionsalgorithmen verwenden, nutzt eine Gruppe für alle Nutzdatencontainer den gleichen Universalkompressor. Ein Beispiel dafür ist der Ansatz von Ng u. a.(2006). Die andere Gruppe setzt hingegen verschiedene universale Techniken ein.[68] Da ein Universalkompressor ausgetauscht werden kann, wird der Ansatz der ersten Gruppe als **variable** Nutzdatencodierung bezeichnet. Sie unterscheidet sich von der **dynamischen** Nutzdatencodierung der zweiten Gruppe in der Weise, dass bei letzterer verschiedene Universaltechniken kombiniert werden. So werden Struktur- und Typinformationen genutzt, um pro

[66] Vgl. Ferragina u. a.(2006), S. 751-754 und Skibiński/Grabowski/Swacha (2008), S. 1035.
[67] Vgl. Toman (2004), S. 4-6 und Harrusi/Averbuch/Yehudai (2006b), S. 404-408.
[68] Vgl. Ng u. a.(2006) S. 424.

Nutzdatencontainer eine Kompressionstechnik zu bestimmen. Ein Beispiel dafür liefert der Ansatz von Wang u. a.(2004).[69]

3.3 Prozess zur Auswahl der Verfahren

Die Auswahl der Verfahren, die in die quantitative Untersuchung der Fallstudie in Kapitel 5 eingehen, erfolgt in drei Stufen. In der ersten Stufe werden die zu berücksichtigenden Verfahren grundlegend eingegrenzt. In der nächsten Stufe dienen funktionale und technische Kriterien zur weiteren Auswahl. In der letzten Stufe wird für alle verbleibenden Verfahren, die nicht implementiert vorliegen, auf technischer Ebene analysiert, ob der Aufwand der Implementierung gerechtfertigt ist.

3.3.1 Grundlegende Anforderungen

Eines der Ziele XML-spezifischer Kompressionsverfahren ist es, höhere Kompressionsraten als Universalkompressoren zu ermöglichen.[70] Aus diesem Grund werden in der qualitativen Auswahl nur XML-spezifische Verfahren berücksichtigt. Um diese Entscheidung zu bekräftigen, werden bei der Fallstudie in Abschnitt 5.3 diejenigen Universalkompressoren eingeschlossen, die bei den Veröffentlichungen zu XML-spezifischen Verfahren für einen Leistungsvergleich herangezogen wurden.
Hinsichtlich der Informationserhaltung wird die im Folgenden beschriebene Eingrenzung vorgenommen. Da alle echt-verlustbehafteten Verfahren dazu führen, dass die Nutzdaten und deren Semantik nicht exakt erhalten bleiben, werden Verfahren dieser Gruppe hier ausgeschlossen. Ein Beispiel dafür stellt *SqueezeX*[71] dar. XML-verlustbehaftete Lösungen stellen hingegen den Idealfall dar, da auf diese Weise die höchste Kompression erreicht werden kann, ohne relevante Information zu verlieren. Eine Unterscheidung der Verfahren nach textuell-verlustfrei, XML-verlustfrei und XML-verlustbehaftet ist dennoch überflüssig. Zum einen können in einer Vorverarbeitungsstufe alle überflüssigen Leerzeichen eines XML-Dokuments

[69] Vgl. Wang u. a.(2004), S. S. 356.
[70] Vgl. Sakr (2009b), S. 304.
[71] Vgl. Cannataro/Comito/Pugliese (2002), S. 326-331.

entfernen werden, wodurch der Unterschied zwischen XML-verlustbehafteten und XML-verlustfreien Verfahren eliminiert wird. Zum anderen kann der Unterschied zwischen textuell-verlustfrei und XML-verlustfrei im Hinblick auf die Kompressionsleistung vernachlässigt werden.
Neben diesen beiden anwendungsorientierten Kriterien sind die zu untersuchenden Verfahren anhand dreier Anforderungen einzuschränken, die den Inhalt der jeweils zugehörigen Veröffentlichung betreffen. So werden alle Ansätze ausgeschlossen, die nur die Grundlagen eines Verfahrens behandeln, deren Beschreibungen jedoch nicht ausreichen, um das jeweilige Verfahren zu implementieren. Entsprechend wird mit Ansätzen verfahren, die jeweils nur die Kompression der Struktur behandeln und die Nutzdaten unberücksichtigt lassen, wie beispielsweise *BPLEX*[72] und *TreeRePair*[73]. Des Weiteren werden keine Ansätze untersucht, die auf speziellen Programmierparadigmen beruhen, die nicht auf die imperative Programmierung übertragbar sind, wie dies bei *XComprez*[74] der Fall ist.

3.3.2 Eingrenzende Auswahlkriterien

Aufbauend auf Kapitel 3.3.1 beschreibt dieser Abschnitt, welche funktionalen und technischen Anforderungen erfüllt sein müssen, damit ein Verfahren für die quantitative Untersuchung in Betracht kommt. Bei den auf diese Weise eingegrenzten Verfahren muss individuell untersucht werden, ob ein Verfahren in der Fallstudie zu berücksichtigen ist. Die Beschreibung dieser Analyse folgt in Abschnitt 3.3.3.

3.3.2.1 Funktionale Kriterien

Im Hinblick auf die Funktionalität sind in jedem Fall archivierende Verfahren für die Kompression von XML-basierten Geschäftsdokumenten relevant. Die Annahme, dass Verfahren mit anderen funktionalen Zielsetzungen (z.B. Abfragenunterstützung) im Hinblick auf die Kompressionsrate nicht konkurrenzfähig sind, gilt

[72] Vgl. Maneth/Mihaylov/Sakr (2008), S. 243-247.
[73] Vgl. Lohrey/Maneth/Mennicke (2011), S. 353-362.
[74] Vgl. Jeuring/Haag (2002), S. 1-12.

nicht pauschal. Es zeigt sich, dass manche Verfahren Abfragen ermöglichen und dennoch sehr gute Kompressionsraten erzeugen. Die Unterstützung von Abfragen ist damit kein Ausschlusskriterium.

Die Unterstützung von Änderungen ist im Gegensatz zur Abfragenunterstützung eine Zielsetzung, bei der mit hoher Wahrscheinlichkeit die Kompressionsrate eingeschränkt wird. Dies begründet sich damit, dass durch die Vorbereitung auf Änderungen die gegebene Redundanz weniger intensiv berücksichtigt wird, wodurch sich die Kompressionsrate verschlechtert. Diese Argumentation wird zudem damit bekräftigt, dass in der später folgenden Untersuchung kein Verfahren einzig wegen der Unterstützung von Änderungen ausgeschlossen wurde, sondern jeweils klare technische oder funktionale Nachteile vorhanden waren.

Die Datenbankfunktionalität in der oben definierten Form ermöglicht keine konkurrenzfähige Kompressionsrate. Aus diesem Grund werden Konzepte dieser Art, wie dies bei *ISX*[75] der Fall ist, ausgeschlossen.

Die Eigenschaft der homomorphen Verfahren, die Daten unabhängig von ihrer Position im Dokument zu codieren, führt zu deutlichen Leistungseinbußen, da die Beziehungen der Nutzdaten untereinander und auch die Ähnlichkeiten der Daten nicht ausgenutzt werden können. Die Verfahren dieser Gruppe sind daher im Hinblick auf die Kompressionsleistung nicht konkurrenzfähig und werden daher nicht betrachtet. Vertreter dieser Gruppe sind mitunter *XGrind*[76], *XCpaqs*[77] und *XPress*[78] sowie das mit *WBXML* abgekürzte *WAP Binary XML Content Format*[79].

Eine Eigenschaft, die nur sehr geringe Auswirkungen auf die Kompressionsrate hat, ist die Streamingfähigkeit. Viele Verfahren dieser Art verarbeiten die Quelldaten in Blöcken, die regelmäßig größer als die in dieser Arbeit relevanten Daten sind. Da nicht davon auszugehen ist, dass dieses Merkmal einen signifikanten Einfluss auf die Kompressionsleistung bei XML-basierten Geschäftsdokumenten nehmen wird, kann es nicht als Ausschlusskriterium verwendet werden.

[75] Vgl. Wong/Lam/Shui (2007), S. 1073-1082.
[76] Vgl. Tolani/Haritsa (2002), S. 225-234.
[77] Vgl. Wang u. a.(2004), S. 354-358.
[78] Vgl. Min/Park/Chung (2003), S. 122-133.
[79] Vgl. Martin/Basher (1999).

3.3.2.2 Ungeeignete Arten der Strukturkompression

Die Voraussetzung für den Einsatz schemabasierter Verfahren ist, dass für das zu komprimierende XML-Dokument ein Schema definiert und dieses sowohl bei der Kompression als auch bei der Dekompression verfügbar ist. In dieser Arbeit wird vorausgesetzt, dass die Geschäftsdokumente auf Basis von gängigen Standards mit öffentlich zugänglichem Schema erstellt werden und dieses somit sowohl bei der Kompression als auch bei der Dekompression verfügbar ist. Somit sind schemabasierte Verfahren nicht auszuschließen und in Form der Variantencodierung vielmehr als potentiell leistungsstärkste Form der Kompression XML-basierter Geschäftsdokumente zu sehen.

Navigationscodierungen

In technischer Hinsicht ist die Unterscheidung der Verfahren nach der Art der Strukturkompression für die Eingrenzung entscheidend. Verfahren, die Navigationscodierungen realisieren, zielen bei der Kompression der Struktur nicht einzig darauf ab, die Struktur möglichst kompakt zu codieren, wie dies bei Bezeichnungscodierungen der Fall ist, sondern zudem darauf, dass eine Navigation auf der komprimierten Struktur möglich ist. Dadurch müssen an den Knoten des Strukturbaums zusätzliche und damit redundante Daten eingebettet werden, die die Kompressionsrate negativ beeinflussen. Da diese Zielsetzung erheblich mit der hier verfolgten konkurriert, ist es nicht sinnvoll, Verfahren dieser Art zu berücksichtigen.

Es wurden eine Reihe von Verfahren mit Navigationscodierungen gefunden, von denen keines archivierender Natur ist. Vielmehr unterstützen alle Verfahren Abfragen und einige auch Änderungen. Häufig wird bei diesen Verfahren ein spezielles Schema zur Erstellung der Codewörter für die Elemente verwendet. Die Autoren von *RRZip* verwenden beispielsweise ein Bezeichnungsschema, das sie *Relative Region Labeling Scheme* nennen.[80] Ebenso wird bei *TREECHOP* ein Schema entwickelt, bei dem ein Codewort für ein Element gebildet wird, indem dem Codewort des

[80] Vgl. Hu/Zhang/Yuan (2012), S. 713.

Elternelements ein Suffix hinzugefügt wird.[81] Auf diese Weise wird jeweils der Pfad redundant pro Element gespeichert.

Eine vergleichbare Redundanz ist auch bei dem homomorphen Verfahren *XPress* vorhanden.[82] Jedoch wird diese hier in einem Vorgang namens *reverse arithmetic encoding* verpackt. Die Elemente werden dabei auf Basis der arithmetischen Codierung so verarbeitet, dass sich daraus die gesamten Pfade ableiten lassen. Die Autoren von *XQueC* gehen zur Unterstützung von Abfragen einen Schritt weiter und speichern zusätzlich zur Navigationscodierung Daten, um Abfragen gezielt zu beschleunigen.[83]

Zu erwähnen ist auch, dass die Autoren Alkhatib und Scholl drei Verfahren mit Navigationscodierungen vorschlagen, die Redundanzen der Struktur auf lokaler Ebene entfernen. Ihre Grundidee führte zu dem Verfahren *SCQX*[84], das im Gegensatz zu den beiden Erweiterungen *CXQU*[85] und *CXDLS*[86] nur Abfragen und noch keine Änderungen unterstützt. Jedoch wird in keinem der drei Verfahren die Kompression der Nutzdaten berücksichtigt.

Schemabasierte Wörterbuchcodierungen

Unter den schemabasierten Verfahren sind diejenigen mit schemabasierten Wörterbuchcodierungen im Hinblick auf die vorliegende Anforderung ebenfalls als unbrauchbar einzustufen. Diese verwenden das Schema lediglich als Datenquelle für Wörterbücher, die dazu dienen, die Elementbezeichnungen zu kürzen. Im Vergleich zu Variantencodierungen ist die erzeugte Einsparung aber sehr gering zu bewerten, denn Variantencodierungen nutzen die Informationen des Schemas wesentlich intensiver. So werden keine verpflichtenden Elemente abgebildet, da sie ohnehin durch das Schema vorgegeben sind. Des Weiteren werden Elemente und Attribute kompakter codiert, da das Schema Wahlmöglichkeiten für diese vorgibt und damit die Entropie reduziert wird.

[81] Vgl. Müldner/Leighton/Diamond (2005), S. 462.
[82] Für diesen und die nächsten zwei Sätze vgl. Min/Park/Chung (2003), S. 122 f.
[83] Vgl. Arion u. a.(2007), S. 7-12.
[84] Vgl. Alkhatib/Scholl (2008b), S. 365-369.
[85] Vgl. Alkhatib/Scholl (2008a), S. 605-612.
[86] Vgl. Alkhatib/Scholl (2009), S. 158-170.

Das älteste Verfahren zur schemabasierten Wörterbuchcodierung ist *Millau*. Es verfolgt den Ansatz die Struktur eines XML-Dokuments zu komprimieren, indem die Element- und Attributnamen durch Nummern ersetzt werden.[87] Die Nummerierung wird generiert, indem die zugehörige DTD durchlaufen wird und eigene Nummernräume für Elemente, Attribute und Werte von Attributwertlisten verwendet werden.

Das Verfahren von Serin (2003) ersetzt die textuellen Werte der Struktur ebenfalls durch Nummern, die aus dem Schema (hier in Form einer XSD) gewonnen werden.[88] Unterschiedliche Bausteine werden in diesem Verfahren nicht unterschieden, wodurch nur ein Nummernkreis existiert und die Codes für die Bezeichner im Durchschnitt länger werden. Das bereits erwähnte Verfahren *XGrind* ist vom Ansatz her mit *Millau* zu vergleichen, realisiert jedoch eine homomorphe Kompression, weswegen Nutzdaten und Struktur nicht getrennt werden.[89]

3.3.2.3 Varianten- und Bezeichnungscodierung im Vergleich

Die Verwendung zusätzlicher Information aus dem Schema wird in der Literatur als potentieller Vorteil in Bezug auf die Kompressionsleistung gewertet.[90] Der Vergleich von Verfahren beider Arten bei Harrusi/Averbuch/Yehudai (2006a) deutet jedoch eher darauf hin, dass Bezeichnungscodierungen leistungsstärker sind.[91] Vergleiche wie der von League/Eng (2007b) lassen vermuten, dass es von der Beschaffenheit der zu komprimierenden Daten abhängt, welcher Ansatz besser geeignet ist.[92]

Eine abstrakte Abschätzung der Unterschiede ist mit einem einfachen Beispiel möglich. Betrachtet wird zunächst ein Dokument, in dem sich wenige verschiedene Elemente oft wiederholen und das zugehörige Schema eine große Anzahl unterschiedlicher Dokumentenvarianten ermöglicht. In diesem Szenario kann die binäre

[87] Für diesen und die nächsten zwei Sätze vgl. Girardot/Sundaresan (2000), S. 747-765.
[88] Für diesen und den nächsten Satz vgl. Serin (2003), S. 33-63.
[89] Vgl. Tolani/Haritsa (2002), S. 225-234.
[90] Vgl. Augeri u. a.(2007) S. 2 und Sakr (2009b) S. 304.
[91] Vgl. Harrusi/Averbuch/Yehudai (2006a), S. 32.
[92] Vgl. League/Eng (2007b), S. 279.

Codierung eines Bausteins bei der Variantencodierung im Durchschnitt länger als bei der Bezeichnungscodierung sein, da die Entropie der Bausteine innerhalb des konkreten XML-Dokuments geringer sein wird als die durch das Schema erzeugte Entropie.
Bei allen Bezeichungscodierungen ist es nötig, die Namen der Elemente und Attribute zu speichern, was bei der Variantencodierung nicht nötig ist. Andererseits kann es, wie oben ausgeführt, dazu kommen, dass die binären Codewörter der Elemente und Attribute bei der Bezeichnungscodierung kompakter sind als bei der Variantencodierung. Ist dieser letzte Effekt in Summe größer als der Speicherplatz, der für die Bezeichnungen nötig ist, ist die Bezeichnungscodierung von Vorteil. Je kleiner ein Dokument ist, desto geringer ist die Einsparung durch kürzere Codewörter und desto stärker wirkt sich der Nachteil der zusätzlichen Daten für die Namen der Elemente und Attribute aus. Somit kann gefolgert werden, dass die Variantencodierung für kleine Dokumente effektiver ist als die Bezeichnungscodierung und sich dieser Zusammenhang ab einer bestimmten Dokumentengröße umkehrt.
Die Größe der Dokumente, ab der sich der Vorteil umkehrt, kann nicht als Wert bestimmt werden, da sie, wie das obige Beispiel zeigt, stark vom Schema und dem Aufbau des Dokuments abhängt. Aufgrund des Beispiels und der Beschaffenheit von Geschäftsdokumenten liegt die Vermutung nahe, dass die Variantencodierung vorteilhaft ist. Um diese Vermutung auch quantitativ zu überprüfen, werden in diesem Kapitel zu untersuchende Verfahren aus beiden Bereichen ausgewählt. Der Leistungsvergleich in Abschnitt 5.3 wird letztlich die hier aufgestellte Vermutung bestätigen.

3.3.3 Analyse der eingegrenzten Verfahren

Aufbauend auf den Kapiteln 3.3.1 und 3.3.2 beschreibt dieser Abschnitt das Vorgehen, mit dem entschieden wird, ob ein Verfahren, das nicht über die bisherigen Kriterien ausgeschlossen wurde, in der Fallstudie zu berücksichtigen ist. Betrachtet werden dabei technische Eignung, Aufwand der Implementierung bzw. Anpassung und Leistungsfähigkeit im Vergleich zu anderen Verfahren.

Implementierte Verfahren gehen grundsätzlich in den quantitativen Vergleich der Fallstudie ein, abgesehen von drei Ausnahmen: Die erste betrifft Verfahren mit mehreren Varianten. Hier wird das am besten geeignete Verfahren bestimmt und nur dieses berücksichtigt. Die zweite Ausnahme bezieht sich auf schemabasierte Verfahren, denn hier kann eine Implementierung nur dann verwendet werden, wenn sie die Schemasprache XSD unterstützt, da sich diese bei XML-basierten Standards für Geschäftsdokumente etabliert hat.[93] Andernfalls muss das entsprechende Verfahren so behandelt werden, als ob keine Implementierung verfügbar wäre. Die letzte Ausnahme stellen Verfahren mit Abfragenunterstützung dar. Da diese regelmäßig für die reine Archivierung unnötige Daten speichern, müssen sie wie Verfahren geprüft werden, die nicht in implementierter Form vorliegen.

Schemabasierte Verfahren können nur dann in die Fallstudie eingehen, wenn sie in der Lage sind XSD zu verarbeiten. Ist ein Verfahren nicht entsprechend erweiterbar, wird es ausgeschlossen. Ansonsten ist zu prüfen, ob der Aufwand der Erweiterung rentabel ist oder ob ähnliche Verfahren vorhanden sind, die nicht erweitert werden müssen.

Bei abfragenunterstützenden Verfahren ist zu untersuchen, ob speziell für diese Funktionalität benötigte Daten gespeichert werden. Ist dies der Fall, wird geprüft, ob das Verfahren unter Verzicht auf die Abfragenunterstützung in der Art implementiert werden kann, dass die zusätzlichen Daten nicht gespeichert werden. Die entsprechenden Änderungen dürfen nur die Abfragenunterstützung beeinträchtigen, nicht aber die grundsätzliche Dekompression. Verfahren, bei denen dies nicht möglich ist, werden nicht weiter berücksichtigt.

Bei nicht implementierten Verfahren und solchen, die Änderungen erfordern, wird untersucht, ob der damit verbundene Aufwand gerechtfertigt ist. So ist es bei technisch ähnlichen Verfahren nicht sinnvoll, alle zu implementieren, sondern nur den Vertreter der Gruppe, der die besten technischen Eigenschaften besitzt.

Bei einem technischen Vergleich von Verfahren sind die Aufteilung der Nutzdaten auf Container und die Reduzierung der Redundanz in der Struktur besonders

[93] Siehe dazu Abschnitt 2.2.

relevant. Bezüglich der Datencontainer ist die Aufteilung nach dem Namen des Bausteins, wie auch nach dessen Pfad, als Nachteil zu werten. In Geschäftsdokumenten sind bei diesen Arten der Aufteilung Container mit sehr wenigen Daten zu erwarten, da die Dokumente selbst nicht groß sind und viele verschiedene Elemente beinhalten. Die Folge davon ist, dass die Daten in verschiedenen Containern nicht unterschiedlich genug sind, wodurch Redundanzen in den Daten nicht ausreichend entfernt werden können. Die Bestätigung dieser Annahme liefert die Fallstudie in Abschnitt 5.3.
Bei technisch ähnlichen Verfahren sind solche vorzuziehen, die Redundanzen in der Struktur entfernen. Bei der Entfernung der Strukturredundanz ist der globale Ansatz besser als der lokale. Inwiefern die Nutzung von nachgelagerten Universalkompressoren zur Entfernung von Strukturredundanz besser oder schlechter geeignet ist als die beiden zuvor genannten Ansätze, kann nicht allgemeingültig beantwortet werden, da dies von dem konkret verwendeten Universalkompressor abhängig ist.

3.4 Verfahren mit Bezeichnungscodierungen

Aus der Eingrenzung über funktionale und technische Kriterien resultieren Verfahren, die entweder Bezeichnungs- oder Variantencodierungen verwenden. Die Betrachtung der Verfahren mit Bezeichnungscodierungen ist Gegenstand dieses Abschnitts und ist in archivierende und abfragenunterstützende Verfahren unterteilt.

3.4.1 Archivierende Verfahren

Im folgenden Abschnitt werden acht archivierende Verfahren analysiert, von denen sechs Verfahren in implementierter Form verfügbar sind und in die quantitative Untersuchung eingehen. Für die beiden nicht in implementierter Form vorliegenden Verfahren wird beschrieben, wie sie über qualitative Merkmale ausgeschlossen werden können.

3.4.1.1 Implementierte Verfahren

Alle Verfahren dieses Kapitels liegen in implementierter Form vor und unterstützen keine Abfragen. Da zudem keine Varianten davon existieren, von denen bessere Leistungen zu erwarten sind, gehen alle sechs Verfahren in die Fallstudie ein.

XMill

Das erste Verfahren, das XML-Dokumente nicht als reinen Text auffasste, sondern deren Strukturierung zur Kompression explizit berücksichtigte, war *XMill*[94]. Die darin enthaltenen Ideen wurden von vielen nachfolgenden Verfahren aufgegriffen. Eine dieser Ideen ist die Zerlegung eines XML-Dokuments in verschiedene Teile, die jeweils durch Universalkompressoren komprimiert werden. Bei der Aufteilung wird die Struktur extrahiert und die verbleibenden Nutzdaten anhand von Strukturinformationen gruppiert. Das Standardkriterium der Gruppierung ist der Name des Elements bzw. Attributs, das die jeweiligen Daten enthält. Sowohl die Aufteilung als auch die zu verwendenden Universalkompressoren können bei *XMill* durch den Anwender konfiguriert werden. So kann die Aufteilung der Nutzdaten in Container durch Regeln, die auf Strukturinformationen beruhen, gesteuert und pro Container ein Kompressionsverfahren definiert werden.
Die Abbildung der Struktur ist bei *XMill* eine Abfolge von Zahlen. Positive fungieren als Symbole für Element- bzw. Attributnamen und negative als Kennzeichner für Datencontainer. Jeder schließende Tag eines Elements wird durch eine 0 ersetzt. Da Elemente in der Reihenfolge geschlossen werden müssen, in der sie geöffnet wurden, kann so jederzeit das konkret zu schließende Element rekonstruiert werden.

XWRT

XWRT[95] verbessert den Ansatz von *XMill* in mehreren Aspekten. Eine zentrale Erweiterung ist die Beseitigung textueller Wiederholungen im gesamten Dokument. Dazu werden in einem ersten Schritt häufig wiederkehrende Zeichenketten erfasst

[94] Für die nachfolgende Beschreibung des Verfahrens vgl. Liefke/Suciu (2000).
[95] Für die nachfolgende Beschreibung des Verfahrens vgl. Skibinski/Grabowski/Swacha (2007) und Skibiński/Grabowski/Swacha (2008).

und durch Verweise auf ein dynamisch aufgebautes Wörterbuch ersetzt. Bei dessen Erstellung wird der spezielle Aufbau der XML-Markups berücksichtigt. Zudem integriert *XWRT* Optimierungen, die in *XMill* nur durch Anwenderwissen und der passenden Konfiguration erreicht werden können. So werden spezielle Datentypen, wie Nummern, Datumsangaben und Uhrzeiten automatisch erkannt und gesondert verarbeitet. Zur Kompression der weiteren Nutzdaten bietet *XWRT* 14 verschiedene Leistungsstufen. Bei diesen kommen vier Universalkompressoren, basierend auf *Deflate*, *LZMA*, *PPMII* und *PAQ*, in unterschiedlichen Konfigurationen zum Einsatz. Zudem werden die Nutzdaten, dem jeweiligen Kompressor entsprechend, optimiert angeordnet.

XMLPPM

Mit *XMLPPM*[96] zeigte Cheney eine völlig neue, auf PPM basierende Herangehensweise. Er verwendet dazu vier Kontextmodelle, je eines für Element- und Attributname sowie Element- und Attributwerte. Über eine als *multiplexed hierarchical modelling (MHM)* bezeichnete Technik werden die Modelle um strukturelle Informationen angereichert. Das auf PPMII aufbauende Kompressionsverfahren berücksichtigt die zusätzlichen Daten zur Verbesserung der Vorhersagen, komprimiert diese jedoch nicht mit.[97]

SCMPPM

SCMPPM[98] greift die Grundidee von *XMLPPM* auf, verwendet aber eine andere Strategie zur Bildung von Kontextmodellen.[99] Als Zuordnungskriterium von textuellen Daten zu einem Modell wird, ähnlich zu *XMill*, der Elementname verwendet. Dem Modell des umgebenden Elements wird dabei noch der Name eines Elements hinzugefügt. Danach wird zu einem neuen Modell gewechselt, in welchem anschlie-

[96] Für die nachfolgende Beschreibung des Verfahrens vgl. Cheney (2001).

[97] In Cheney (2001) werden die PPM-Varianten PPMD+ und PPM* angesprochen, wohingegen die unter https://sourceforge.net/projects/xmlppm/files verfügbare Implementierung auf dem von Shkarin entwickelten Universalkompressor *PPMd* und damit auf PPMII basiert.

[98] Für die nachfolgende Beschreibung des Verfahrens vgl. Adiego/La Fuente/Navarro (2005).

[99] Ähnlich wie Cheney (2001) beziehen sich auch in ihren Ausführungen auf PPMD+, wobei in der unter http://www.infor.uva.es/ jadiego/files/scmppm-0.93.3.zip verfügbaren Implementierung wieder der von Skahrin entwickelte Universalkompressor *PPMd* verwendet wird.

ßend die Namen und Werte der Attribute, die Nutzdaten und die Namen der Kindelemente erfasst werden.

XBzip

Wie schon bei *XMLPPM* wird auch bei *XBzip*[100] eine völlig neue Idee realisiert. Inspiriert von der Burrows-Wheller-Transformation (BWT)[101] wird das XML-Dokument in zwei Listen zerlegt, die jeweils auf Basis von PPMII komprimiert werden. Die eine Liste enthält die Namen und Werte der Elemente bzw. Attribute. Die andere Liste enthält für jeden Eintrag der ersten Liste den Pfad zum Wurzelknoten. Indem die zweite Liste sortiert wird, kann der nachgelagerte Kompressor die lokalen Redundanzen in der Struktur entfernen. Die Liste der Namen und Werte wird bei der Sortierung ebenfalls angepasst, um die Zuordnung zwischen den Listeneinträgen zu erhalten. Auf diese Weise werden auch die Nutzdaten mit gleichen Pfaden gruppiert, wodurch die Kompressionsrate gesteigert werden soll.

Exalt

Bei *Exalt*[102] werden Daten und Struktur im Gegensatz zu den anderen bisher betrachteten Verfahren nicht getrennt voneinander komprimiert. Stattdessen wird in einem ersten Schritt ein nummeriertes Wörterbuch aus Element- und Attributnamen angelegt und ihre Vorkommen durch die entsprechenden Nummern ersetzt. In einem zweiten Schritt wird das Dokument wiederholt durchlaufen und für jedes Element ein endlicher Automat angelegt, der dessen Aufbau abbildet. Da dieser bei jeder Instanz des Elements anders sein kann, wird auch die Wahrscheinlichkeit jeder Variante gepflegt. Dies erfolgt sukzessive im Rahmen des zweiten Durchlaufs. Auf Basis der Automaten wird eine kontextfreie Grammatik abgeleitet, die letztlich arithmetisch codiert wird. Die globale Entfernung struktureller Redundanzen wird in diesem Prozess von den endlichen Automaten realisiert.

[100] Für die nachfolgende Beschreibung des Verfahrens vgl. Ferragina u. a.(2006) und Ferragina u. a.(2009).
[101] Vgl. Burrows/Wheeler (1994).
[102] Für die nachfolgende Beschreibung des Verfahrens vgl. Toman (2003) und Toman (2004).

3.4.1.2 Nicht implementierte Verfahren

Von den archivierenden Verfahren liegen zwei in nicht implementierter Form vor. Im Rahmen der nachfolgenden Ausführungen wird dargestellt, warum keines der beiden Verfahren quantitativ zu untersuchen ist.

XComp

Das Verfahren *XComp*[103] ist sehr ähnlich zu *XMill*. Aus den Element- und Attributnamen wird ein Wörterbuch aufgebaut, das zur Kürzung der Strukturdaten verwendet wird. Zudem werden spezielle Informationen in der Struktur, wie beispielsweise schließende Tags und das Vorhandensein von Nutzdaten durch Zahlen identifiziert. Zur Aufteilung der Nutzdaten auf Container verwendet *XComp* ein Konzept, das eine feinere Unterteilung als *XMill* erzeugt. Bei diesem werden Nutzdaten nur dann demselben Container zugeordnet, wenn das jeweilige Element oder Attribut gleich benannt ist und zudem in der gleichen Tiefe in der Struktur auftritt. Die Leistungsvergleiche von Weimin (2003) zeigen, dass *XComp* auf dem Niveau von *XMill* liegt, wenn beide Verfahren als Backendkompressor Deflate verwenden.[104] Da nicht genannt ist, welche Version von *XMill* betrachtet wurde, ist ein Quervergleich zu Sakr (2009a) nötig, um zu erkennen, dass die ursprüngliche Version mit *gzip* zum Einsatz kam.[105] Aufgrund der Ähnlichkeit von *XComp* zu *XMill* und der ähnlichen Kompressionsleistung wird eine Implementierung von *XComp* als nicht rentabel erachtet.

AXECHOP

Bei *AXECHOP*[106] werden die Struktur- und Nutzdaten getrennt und letztere anhand der zugehörigen Pfade im Dokument in Container aufgeteilt. Diese werden mit *bzip2* komprimiert und die Struktur von einem grammatikbasierten Kompressi-

[103] Für die nachfolgende Beschreibung des Verfahrens vgl. Weimin (2003).
[104] Vgl. Weimin (2003), S. 71.
[105] Vgl. Sakr (2009a), S. 54 und 59.
[106] Für die nachfolgende Beschreibung des Verfahrens vgl. Leighton (2005) und Leighton/Diamond/Müldner (2005).

onsverfahren namens *Multilevel Pattern Matching (MPM)*[107] verarbeitet. Auf diese Weise werden Redundanzen in der Struktur auf globaler Ebene entfernt. Die Auswertung von Leighton/Diamond/Müldner (2005) bescheinigt *AXECHOP* eine Leistung, die bei kleinen Dokumenten in etwa der von *XMill* in der *bzip2*-Variante entspricht. Da sie damit auch auf dem Niveau des ebenfalls grammatikbasierten Verfahrens *Exalt* liegt und die Zerlegung der Nutzdaten nach Pfaden bei kleinen Dokumenten von Nachteil ist, wird *AXECHOP* nicht implementiert.

3.4.2 Verfahren mit Abfragenunterstützung

Bei der nachfolgenden Beschreibung und Analyse der Verfahren mit Abfragenunterstützung wird *XBzipIndex* nicht explizit beschrieben, da es sich um eine Variante von *XBzip* handelt, die lediglich um zusätzliche Indexdaten zur Abfragenunterstützung ergänzt ist.[108]

XSAQCT

Bei *XSAQCT*[109] wird der baumartige Aufbau der Dokumentenstruktur genutzt und ein Strukturbaum aufgebaut, bei dem sukzessiv lokale Wiederholungen in der Struktur entfernt werden. Auf diese Weise entsteht eine minimale Repräsentation der Dokumentenstruktur. Für jeden Knoten wird eine Liste mit Anzahlen gepflegt, die angibt, wie häufig das Element des Knotens in der originalen Struktur enthalten ist. Die Bezeichnungen der Attribute und Elemente wird in ein Wörterbuch ausgelagert. Die Nutzdaten werden in verschiedene Container zerlegt, wobei der Pfad des zugehörigen Elements bzw. Attributs das Zuordnungskriterium bildet. Für die Kompression der einzelnen Container kann ein beliebiger Universalkompressor verwendet werden.

[107] Vgl. Kieffer u. a.(2000), S. 1227-1245.
[108] Vgl. Ferragina u. a.(2009), S. 15.
[109] Für die nachfolgende Beschreibung des Verfahrens vgl. Müldner u. a.(2008) und Müldner u. a.(2009).

BSBC und DAG+BSBC

Mit *BSBC*[110] und *DAG+BSBC*[111] erfolgt eine Analyse zweier Verfahren der Autoren Böttcher, Hartel und Heinzemann. Sie erklären *DAG+BSBC* als Erweiterung von *BSBC*[112]. Jedoch lassen die Ausführungen zu beiden Verfahren keine Unterschiede erkennen. Obwohl in Böttcher/Hartel/Heinzemann (2009) angegeben wird, dass nur bei *DAG+BSBC* aus der Struktur ein gerichteter Graph aufgebaut wird, bei dem gleiche Subgraphen durch Zeiger ersetzt werden, und diese Methodik bei *BSBC* fehlen soll, wird sie in Böttcher/Hartel/Heinzemann (2008) im nahezu identischen Wortlaut erläutert.[113]

Ferner weist das spätere Werk der Autoren für *BSBC* bei den gleichen Analysedatensätzen eine schlechtere Leistung aus, als dies in der Veröffentlichung des Verfahrens der Fall ist. Aufgrund der Unklarheiten bezüglich *BSBC* und der Tatsache, dass *DAG+BSBC* ohnehin eine Verbesserung davon sein soll, wird *BSBC* nicht weiter berücksichtigt.

XQzip

Bei *XQzip*[114] wird, wie auch bei *DAG+BSBC* und *XSAQCT*, explizit die Struktur als Baum verarbeitet. Die dazu verwendete Datenstruktur nennen die Autoren *Structure Index Tree (SIT)*. Ähnlich zur Vorgehensweise bei *XSAQCT* wird SIT benutzt, um lokale Wiederholungen in der Struktur zu kürzen. Statt einer Liste von Häufigkeiten wird pro Knoten eine Liste von Nummern gespeichert, die angeben, an welcher Stelle im Dokument der jeweilige Knoten auftritt. Die entsprechende Nummerierung berücksichtigt die öffnenden und schließenden Tags der originalen Struktur in Dokumentenreihenfolge.

Die grundsätzliche Bedeutung des SIT wird bei Cheng/Ng (2004) widersprüchlich dargestellt. Einerseits wird er nicht als zusätzlicher Index, sondern als Abbild der

[110] Für die nachfolgende Beschreibung des Verfahrens vgl. Böttcher/Hartel/Heinzemann (2008).
[111] Für die nachfolgende Beschreibung des Verfahrens vgl. Böttcher/Hartel/Heinzemann (2009).
[112] Vgl. Böttcher/Hartel/Heinzemann (2009), S. 65.
[113] Vgl. Böttcher/Hartel/Heinzemann (2009), S. 67 und Böttcher/Hartel/Heinzemann (2008), S. 13-16.
[114] Für die nachfolgende Beschreibung des Verfahrens vgl. Cheng/Ng (2004).

Dokumentenstruktur beschrieben und andererseits werden Kompressionsraten mit und ohne SIT angeben. Würde der Baum die grundsätzliche Repräsentation der Struktur sein, könnte ohne ihn das Dokument nicht rekonstruiert werden. Daher ist anzunehmen, dass SIT doch ein zusätzlicher Index ist.

DAG+BSBC, XQzip und XSAQCT im Vergleich

Die Strukturverarbeitung der Verfahren *DAG+BSBC*, *XQzip* und *XSQACT* ist sehr ähnlich. Die textuellen Redundanzen werden jeweils durch ein Wörterbuch und Wiederholungen in der Struktur auf Basis eines Baumes entfernt. Der Ansatz von *DAG+BSBC*, identische Teilbäume durch einen Zeiger zu ersetzen und damit den Strukturbaum in einen gerichteten Graphen zu verwandeln, ist als effektivste Lösung einzustufen. Bei den beiden anderen Ansätzen wird die Abbildung redundanter Teilstrukturen zwar gekürzt, dennoch werden auch bei redundanten Teilbäumen pro Knoten Daten gespeichert.

Die Aufteilung der Nutzdaten auf Container erfolgt bei *DAG+BSBC* und *XQzip* nach den Bezeichnungen der Elemente und Attribute. Bei *XAQCT* wird hingegen die in diesem Zusammenhang als leistungsschwächer einzustufende Aufteilung nach Pfaden verwendet. Als Nutzdatenkompressor kommen bei allen drei Verfahren Universalkompressoren zum Einsatz.

Keines der drei Verfahren liegt in implementierter Form vor. Aufgrund der Nachteile von *XQzip* und *XSAQCT* gegenüber *DAG+BSBC* ist es nicht sinnvoll, die beiden ersten zu implementieren, weshalb nur letzteres in die quantitative Untersuchung eingeht. Von *XSAQCT* existiert zudem eine Version, die Streaming und Änderungen unterstützt.[115] Aufgrund der zusätzlichen Zielsetzungen wird davon ausgegangen, dass die Kompressionsrate sinkt, weswegen dieser Ansatz hier ebenfalls ausgeschlossen wird.

[115] Vgl. Müldner/Miziołek/Fry (2012).

QXT

Wie *XMill* unterstützt auch *XWRT* keine Abfragen, weshalb Skibinski und Swacha mit *QXT*[116] ihren ersten Vorschlag um eine Abfragenunterstützung erweiterten. Dabei änderten sie auch die Verarbeitung der Nutzdaten. Daten spezieller Typen werden nicht mehr in eigene Container gelegt, sondern in einem vorgelagerten Schritt in eine kürzere Repräsentation überführt und anschließend, wie auch der Rest der Nutzdaten, anhand ihrer vollständigen Pfade auf Container verteilt. Als Universalkompressoren sind nur *gzip* und *LZMA* vorgesehen. Die Codierung des Wörterbuchs und die Vorverarbeitung der speziellen Datentypen werden entsprechend des gewählten Universalkompressors angepasst.
Da *QXT* eine auf Abfragen ausgerichtete Variante von *XWRT* ist, wird es im quantitativen Vergleich nicht berücksichtigt. Dies wird auch dadurch bekräftigt, dass zur Aufteilung auf Container die Pfade der Bausteine verwendet werden.

XSeq

XSeq[117] basiert auf dem universellen Kompressionsverfahren *Sequitur*. Bei diesem werden die Daten komprimiert, indem aus ihnen eine kontextfreie Grammatik abgeleitet und diese arithmetisch codiert wird.[118] Die Zerlegung des XML-Dokuments entspricht der Vorgehensweise von *XMill*. Indem die Strukturdaten separat grammatikbasiert komprimiert werden, werden neben textueller Redundanz auch Wiederholungen in der Struktur auf globaler Ebene entfernt. Zur Unterstützung von Abfragen werden zusätzliche Indexstrukturen verwendet. Da diese nur für die Abfragenunterstützung, nicht aber für die eigentliche Dekompression des gesamten Dokuments nötig sind, können sie entfernt werden.
XSeq liegt nicht in implementierter Form vor, weswegen abzuschätzen ist, ob eine Implementierung gerechtfertigt ist. Entsprechend der Ausführungen bei Lin u. a.(2005) entspricht die Leistung von *XSeq* der von *XMill* und übertrifft dieses, je

[116] Für die nachfolgende Beschreibung des Verfahrens vgl. Skibiński/Swacha (2007).
[117] Für die nachfolgende Beschreibung des Verfahrens vgl. Lin u. a.(2005).
[118] Vgl. Nevill-Manning/Witten (1997), S. 3-11.

regelmäßiger die Daten sind.[119] Bei einer Implementierung würden die unnötigen Indexdaten nicht gespeichert werden, wodurch sich die Leistung noch verbessern würde.

XSeq ist technisch ähnlich zu *Exalt* und *AXECHOP*. Die Aufteilung der Nutzdaten erfolgt bei *XSeq* nach dem Namen des Elements bzw. Attributes. Dies ist als Vorteil gegenüber *AXECHOP* zu bewerten, bei dem der Strukturpfad verwendet wird. *Exalt* vollzieht keine explizite Trennung von Struktur und Nutzdaten, was für einen grammatikbasierten Ansatz aber kein grundsätzlicher Nachteil ist. *XSeq* befindet sich aus technischer Sicht zwischen den beiden anderen Verfahren. Da der Implementierungsaufwand bei *XSeq* gering ist, wird es in die quantitative Auswertung aufgenommen. Auf diese Weise kann in der Untersuchung in Abschnitt 5.3 gezeigt werden, dass schon die Aufteilung der Nutzdaten nach Namen nicht konkurrenzfähig ist, wodurch der Ausschluss von *AXECHOP* bekräftigt wird.

XPack

Bei *XPack*[120] sind Zusatzdaten für die Abfragenunterstützung in der Form enthalten, dass pro Knoten sowohl der Nachfolger als auch der Vorgänger gespeichert werden. Zur Rekonstruktion der Struktur ist jedoch einer von beiden ausreichend. Die Nutzdaten werden auf Basis ihres Pfads im Dokument auf Container verteilt. Als Backend-Kompressor wird *gzip* verwendet. Aufgrund der redundanten Daten in der Strukturabbildung und der Zerlegung der Nutzdaten nach Pfaden wird das Verfahren als nicht leistungsstark genug eingestuft. Zudem sind die Erläuterungen des Verfahrens nicht für eine Implementierung ausreichend. Das Verfahren muss daher ausgeschlossen werden.

3.5 Verfahren mit Variantencodierungen

In diesem Abschnitt werden die zwölf in der Literatur gefundenen Verfahren mit Variantencodierung betrachtet, die nicht bereits durch funktionale oder technische

[119] Vgl. Lin u. a.(2005), S. 663 f.
[120] Für die nachfolgende Beschreibung des Verfahrens vgl. Rocco/Caverlee/Liu (2005).

Kriterien ausgeschlossen werden konnten. Im ersten Teil werden fünf Verfahren untersucht, die Automaten zur Modellierung von Schemainformationen verwenden. Die im zweiten Teil analysierten sieben Verfahren bilden Schemainstanzen als Bäume ab.

3.5.1 Automatenbasierte Verfahren

Alle automatenbasierten Verfahren sind auf Archivierung ausgerichtet. Die letzten beiden in diesem Kapitel untersuchten Verfahren sind die einzigen in implementierter Form vorliegenden Verfahren mit Variantencodierung. Zunächst werden drei PPM-basierte Verfahren betrachtet.

XAUST

XAUST[121] folgt dem Vorgehen bei *XMill* und zerlegt das XML-Dokument in Struktur- und Nutzdaten, wobei die Nutzdaten nach ihrem Namen in Container aufgeteilt werden. Die Strukturdaten werden auf der Grundlage von Schemainformationen gekürzt und anschließend, wie auch die Nutzdatencontainer, mit einem PPM-Verfahren komprimiert. Zur Verarbeitung der Struktur wird aus dem Schema, mit dem das zu komprimierende XML-Dokument verknüpft ist, ein Kellerautomat erstellt. Dieser besteht aus endlichen Automaten, von denen jeder ein Element des Schemas repräsentiert, und einem Kellerspeicher, der die Schachtelung der Elemente ermöglicht. Bei der Verarbeitung der Struktur mit dem Kellerautomaten wird von diesem, bei Zuständen mit mehr als einem Folgezustand, die Bezeichnung des gerade verarbeiteten Bausteins ausgegeben. Ist nur ein Folgezustand vorhanden, bedeutet dies, dass das Element verpflichtend und dadurch bei der Dekompression durch den Automaten eindeutig rekonstruierbar ist.

DTDPPM

DTDPPM[122] ist eine Erweiterung von *XMLPPM*, wobei der Grundaufbau und die eigentliche Kompression durch PPM unverändert bleibt. Die eingebauten Verbes-

[121] Für die nachfolgende Beschreibung des Verfahrens vgl. Subramanian/Shankar (2006).
[122] Für die nachfolgende Beschreibung des Verfahrens vgl. Cheney (2005).

serungen betreffen mitunter fünf Bereiche: Erstens werden überflüssige Leerzeichen entfernt, wodurch der PPM-Codierer, der immer eine bestimmte Anzahl an Stellen als Kontext für ein zu codierendes Zeichen verwendet, nicht mehr durch eine Folge bedeutungsloser Leerzeichen gestört wird. Zweitens wird kein Wörterbuch für die Bezeichner in das komprimierte Dokument integriert, da dies aus dem Schema generiert werden kann. Drittens wird zur Repräsentation von Attributlisten ein Bit-Array verwendet, in dem für jedes optionale Attribut eine 1 gesetzt wird, falls es in dem zu verarbeitenden XML-Dokument vorhanden ist, andernfalls eine 0. Bei der letzten Verbesserung wird Speicherplatz dadurch eingespart, dass die Struktur auf die Elemente reduziert wird, die nicht durch das Schema eindeutig bestimmt sind. Somit werden nur für die Bausteine der Struktur Daten gespeichert, an denen das Schema Wahlmöglichkeiten bietet. Bei Cheney (2005) wird nicht erklärt, wie bestimmt wird, ob eine Wahlmöglichkeit besteht. Es ist aber naheliegend, dass dazu, entsprechend der Vorgehensweise bei *XAUST*, ein Automat verwendet wird.

DPDT-L

DPDT-L[123] arbeitet ähnlich zu *DTDPPM* und verwendet je ein PPM-Modell für Struktur- und Nutzdaten. Es findet keine explizite Zerlegung des Dokuments statt, sondern die XML-basierte Darstellung der Struktur wird durch spezielle Zeichen ersetzt. Dies ermöglicht zum einen die Trennung von Struktur und Daten in zwei Modelle und zum anderen wird damit für das Nutzdatenmodell ein struktureller Kontext bereitgestellt. Die Verarbeitung der Struktur auf Basis einer Schemadefinition ist sehr ähnlich zu der bei *XAUST* und erfolgt in drei Schritten. Zunächst wird aus den Regeln des Schemas eine Grammatik abgeleitet, aus der anschließend ein dynamischer Parser erzeugt wird. Dabei wird jede Regel als endlicher Automat abgebildet und alle endlichen Automaten mit einem Kellerspeicher zu einem einzigen Kellerautomaten vereint. Dieser wird letztlich in einen Transduktor verwandelt, der bei jedem Zustandsübergang ein dem Übergang zugeordnetes Symbol ausgibt.

[123] Für die nachfolgende Beschreibung des Verfahrens vgl. Harrusi/Averbuch/Yehudai (2006b).

Der Transduktor übernimmt die Aufgabe, die Bausteine der XML-Struktur durch spezielle Symbole zu ersetzen. Die Symbole sind dabei so gestaltet, dass aus ihnen ersichtlich ist, ob der entsprechende Zustandsübergang durch das Schema eindeutig bestimmt ist und das Symbol somit für einen verpflichtenden Baustein steht oder ob es für einen von mehreren möglichen Zustandsübergängen steht und das Element somit in definierten Grenzen variabel ist. In beiden Fällen wird das Symbol als Kontext in den PPM-Modellen berücksichtigt, aber nur im zweiten Fall codiert. Mit *DPDT-G* existiert auch eine Variante dieses Verfahrens, bei dem beide Arten der zuvor beschriebenen Symbole codiert werden. Da bei dieser die Abbildung der Struktur weniger kompakt ist, bleibt diese Variante im weiteren Verlauf unberücksichtigt.

Zusammenfassung der PPM-basierten Verfahren

Die drei eben beschriebenen Verfahren liegen nicht in implementierter Form vor und sind in ihrer Technik, die Struktur zu komprimieren, sehr ähnlich. In den Aufsätzen zu den Verfahren wird durchgehend die Schemasprache DTD verwendet. Nur bei *DPDT-L* wird erklärt, dass das Verfahren mächtig genug ist, XSD verarbeiten zu können.[124] Diese Aussage ist auch auf *XAUST* erweiterbar, weil dieses Verfahren ebenfalls einen Kellerautomaten verwendet. Da bei *DTDPPM* nur das Ergebnis der Strukturverarbeitung, nicht aber eine Vorgehensweise dafür beschrieben ist, stellt sich das Problem der Mächtigkeit nicht.

Im Hinblick auf die reine Verarbeitung der Struktur sind *DTDPPM* und *XAUST* identisch. Der Nachteil von *XAUST* liegt in der Verarbeitung der Nutzdaten, die nach dem Namen des beinhaltenden Elements in Container aufgeteilt werden. Streng genommen stellt *DTDPPM* nur eine Idee dar, da keine konkrete Vorgehensweise für die Strukturverarbeitung beschrieben ist. Aufgrund der Einschränkungen beider Verfahren ist für die quantitative Untersuchung nur eine Verbindung beider Verfahren in der Form relevant, dass *DTDPPM* mit der Vorgehensweise von *XAUST* zur Strukturabbildung berücksichtigt wird. Außerdem kann *DPDT-L*, wie

[124] Vgl. Harrusi/Averbuch/Yehudai (2006a), S. 11.

eben begründet, nicht ausgeschlossen werden und ist somit ebenfalls quantitativ zu untersuchen.

rngzip

Ein weiteres automatenbasiertes Verfahren ist *rngzip*[125]. Zur Abbildung des Schemas werden hier Baumautomaten verwendet. Wie bei allen Verfahren dieser Art fungiert der Automat auch hier in Form eines Transduktors. Bei der Verarbeitung eines XML-Dokuments wird für ein Element bzw. Attribut genau dann ein Symbol ausgegeben, wenn es nicht durch das Schema ohnehin erzwungen wird. Bei mehreren Alternativen werden die Zustandsübergänge so durchnummeriert, dass die Nummer als Zuordnung eines Bausteins zu einem Übergang fungiert. Die so geschaffene Abbildung wird nicht wie bei den bisher betrachteten Verfahren durch ein nachgelagertes Kompressionsverfahren verarbeitet, sondern direkt binär codiert. Die Anzahl der dafür nötigen Stellen kann aus der Anzahl der alternativen Übergänge aus dem Automaten abgeleitet werden.
Der Leistungsvergleich dieser Lösung in League/Eng (2007a) zeigt, dass *rngzip* bei Dokumenten unter 200 kB besser als *XWRT* ist. Besonders zu berücksichtigen ist dabei, dass *rngzip* nur *gzip* als Textkompressor verwendet, wohingegen *XWRT* ein wesentlich leistungsstärkeres PAQ-Verfahren einsetzt. Daraus folgt, dass die Leistungsfähigkeit des Verfahrens quantitativ untersucht werden soll. *rngzip* liegt zwar in implementierter Form vor, ermöglicht aber in dieser nur die Verwendung von Schemata in RELAX NG. Entsprechend der Ausführungen in Abschnitt 3.2.3.3 ist ein Verfahren für RELAX NG grundsätzlich mächtig genug, auf XSD erweitert zu werden, weshalb *rngzip* in den quantitativen Vergleich der Fallstudie eingeht.

EXI

In Schneider u. a.(2014) wird ein Format namens *Efficient XML Interchange (EXI)* spezifiziert, mit dem XML-Daten auf binärer Basis kompakter repräsentiert

[125] Für die nachfolgende Beschreibung des Verfahrens vgl. League/Eng (2007a) und League/Eng (2007b).

werden können.[126] Im Gegensatz zu nahezu allen anderen hier betrachteten Werken handelt es sich bei *EXI* nicht um ein im wissenschaftlichen Kontext entwickeltes Verfahren, sondern um ein auf den praktischen Einsatz ausgelegtes Format, das ein De-facto-Standard des World Wide Web Consortiums (W3C) ist. Es kann nicht eindeutig charakterisiert werden, da es in verschiedenen Ausprägungen auftritt. Dies beginnt bei einer homomorphen Umformung ohne Kompression der Nutzdaten und reicht bis hin zur Abbildung der Struktur in Form einer Variantencodierung sowie der Nutzdatenkompression auf Basis einer dynamischen Containerstrategie.
Die in Bezug auf die Kompressionsrate offensichtlich effektivste und damit als einzige betrachtete Variante des *EXI*-Formats ist diejenige, die das Schema in Form einer Variantencodierung nutzt und die Nutzdaten komprimiert. Die Ausführungen in Schneider u. a.(2014) enthalten Definitionen und Erklärungen, die die Umwandlung von XML-Daten in das *EXI*-Format und umgekehrt spezifizieren. Die Implementierung dieser Anforderungen wird nicht betrachtet. Es wird jedoch definiert, dass eine Anwendung, die die Spezifikationen umsetzt, als *EXI Processor* bezeichnet wird.
Um zu beschreiben, wie die Regeln eines Schemas zur Abbildung der Struktur zu nutzen sind, wird in Schneider u. a.(2014) keine konkrete Schemasprache verwendet, sondern eine formale Grammatik. Durch diese Entkopplung kann jede Schemasprache verwendet werden. Es ist lediglich eine Vorschrift nötig, wie ein konkretes Schema in der jeweiligen Sprache in eine der von *EXI* erwarteten Form entsprechende, formale Grammatik überführt werden kann. Wenngleich als Beispiele für mögliche Schemasprachen DTD, XML-Schema und RELAX NG angegeben werden, wird bei Schneider u. a.(2014) nur die Abbildung von XML-Schema-Definitionen behandelt.
Bei der Abbildung eines XML-Dokuments wird ermittelt, welche Produktionsregeln der Grammatiken zu verwenden sind, um die Struktur des XML-Dokuments zu erstellen. Indem jeder Regel ein eindeutig reproduzierbarer Schlüssel zugewiesen wird, kann aus der Abfolge der Schlüssel und der Grammatik die Struktur

[126] Für die nachfolgende Beschreibung des Verfahrens vgl. Schneider u. a.(2014).

rekonstruiert werden. Im Hinblick auf die hier relevanten Bausteine eines XML-Dokuments sind diese Schlüssel ausschließlich Zahlenwerte.[127] Da formale Grammatiken über Automaten implementiert werden, ist *EXI* den automatenbasierten Ansätzen zuzuordnen. Die Verknüpfung von Werten mit Produktionsregeln entspricht in diesem Sinn dem Prinzip eines Transduktors.

Entsprechend der anderen automatenbasierten Verfahren werden auch hier keine Daten für eindeutig bestimmbare Elemente gespeichert. Die Zahlenwerte, die einen Zustandsübergang und damit eine Produktionsregel identifizieren, werden als Dualzahl codiert. Die dafür verwendeten Stellen sind immer Vielfache von acht, damit die Strukturabbildung von dem nachgelagerten Universalkompressor sinnvoll verarbeitet werden kann.

Grundsätzlich ist *EXI* streamingfähig, indem die Eingabe in Blöcken verarbeitet wird, deren Größe einstellbar ist und in der Standardkonfiguration bereits deutlich über der Größe der hier zu erwartenden Dokumente liegt. Aus diesem Grund wird im Folgenden davon ausgegangen, dass keine Unterteilung in Blöcke vollzogen wird.

Die Verarbeitung der Nutzdaten erfolgt nach einem dynamischen Containerkonzept. Grundsätzlich werden die Nutzdaten entsprechend des Namens ihres beinhaltenden Bausteins in sogenannte *value channels* zerlegt. Die Strukturabbildung wird in einen *structure channel* gespeichert. Enthalten alle *value channels* in Summe weniger als 100 Werte, werden alle *channels* gemeinsam komprimiert. Andernfalls wird der *structure channel* einzeln komprimiert. Von den *value channels* werden diejenigen einzeln komprimiert, die mehr als 100 Werte beinhalten. Alle weiteren werden zusammenfasst und gemeinsam komprimiert. Als Kompressor wird im Standard der Deflate-Algorithmus verwendet, der aber durch beliebige Universalkompressoren ersetzt werden kann.

EXI ist im Hinblick auf die Strukturverarbeitung dem Ansatz von *DPDT-L* am ähnlichsten. Als W3C-Spezifikation ist es im Gegensatz zu den anderen schema-

[127] Die im Kontext dieser Arbeit nicht relevanten Bestandteile, wie beispielsweise nicht im Schema definierte Elemente oder Verarbeitungsanweisungen, werden über Schlüssel aus bis zu drei Zahlenwerten abgebildet.

basierten Verfahren kein wissenschaftliches Konzept eines Verfahrens, sondern eine fein ausgearbeitete Empfehlung für ein breites Einsatzspektrum anwendungsbezogener Natur. Dadurch weist es von allen betrachteten Verfahren die deutlich höchste Relevanz für den praktischen Einsatz auf. Aus diesem Grund gibt es eine ganze Reihe von Anwendungen, die die Implementierung eines EXI-Processors darstellen.[128] Da diese Programme im Hinblick auf die Kompressionsleistung identische Ergebnisse liefern müssen, ist es ausreichend, eine dieser Implementierungen im quantitativen Vergleich zu berücksichtigen.

3.5.2 Baumbasierte Verfahren

Bei den baumbasierten Verfahren werden zunächst zwei archivierende Verfahren und im Anschluss fünf Verfahren mit Abfragenunterstützung betrachtet. Keines der untersuchten Verfahren dieses Abschnitts liegt in implementierter Form vor.

DDT

In Sundaresan/Moussa (2002) wird eine Erweiterung von *Millau* vorgestellt, die als *differential DTD tree compression* bezeichnet und hier mit *DDT* abgekürzt wird. Wie der Name ausdrückt, sieht das Verfahren als Schema eine DTD vor.[129] Aus den Elementen, Attributen und Operatoren der DTD wird ein Baum aufgebaut. Ähnlich wie bei einem validierenden Parser wird bei der Kompression eines Dokuments der Baum parallel zu dem zu komprimierenden XML-Dokument durchlaufen. An den Knoten, die die Operatoren `?`, `*`, `+` und `|` darstellen und damit eine variable Gestaltung des Dokuments ermöglichen, werden Daten generiert, die einen Rückschluss auf die Struktur des XML-Dokuments ermöglichen. Die Gesamtheit der generierten Daten beschreibt einen Pfad durch den Baum und stellt als solches ein Strukturabbild dar.
Bei der binären Codierung der Strukturabbildung wird bei dem Auswahloperator `|` ein 1-Bit gespeichert, wenn aus dem XML-Dokument folgt, dass die linke

[128] Vgl. Kamiya (2016).
[129] Für die nachfolgende Beschreibung des Verfahrens vgl. Sundaresan/Moussa (2002).

Seite zu wählen ist und andernfalls ein 0-Bit. Bei den Häufigkeitsoperatoren `?`, `*` und `+` wird die konkret vorhandene Anzahl an Wiederholungen bestimmt und als Zahl gespeichert. Die binäre Abbildung der Zahl wird nicht definiert. Während die Nutzdatenkompression bis zu einer Dokumentengröße von 10 kB in einem Container erfolgt, werden die Nutzdaten bei größeren Dokumenten nach dem Namen ihres Bausteins in Container aufgeteilt. Zur Codierung der Container kann ein beliebiger Universalkompressor verwendet werden.

Bei der Dekompression wird der Baum der DTD auf dem von der Strukturabbildung beschriebenen Pfad durchlaufen und aus den Elementen und Attributen dieses Pfades die Struktur des Dokuments rekonstruiert. Bei allen Operatoren, die Verzweigungen im Baum generieren, wird die zur Rekonstruktion nötige Information aus dem Strukturabbild gelesen. An den Konten des Baumes, die für Nutzdaten stehen, werden die Daten aus dem entsprechenden Nutzdatencontainer geladen.

SCA

Bei Levene/Wood (2002) wird ein Verfahren zur schemabasierten XML-Kompression vorgestellt, das hier nach Ng/Lam/Cheng (2006) mit *SCA* bezeichnet wird.[130] Die Autoren geben an, dass der Ansatz im Grunde dem von *DDT* entspricht, bei ihnen aber ein eigener Algorithmus im Fokus steht.[131] Dabei wird aus den Elementen der DTD ein Baum erstellt, der das zu komprimierende XML-Dokument abbildet.[132] Dieser wird anschließend auf die Knoten reduziert, an denen die DTD Wahlmöglichkeiten eröffnet. Indem bei jedem Knoten Daten gespeichert werden, die seine konkrete Gestaltung beschreiben, entsteht ein kompaktes Abbild der Struktur, das zu dem Abbild bei *DDT* identisch ist. Die Verarbeitung der Nutzdaten erfolgt, wie bei *DDT*, über einen Universalkompressor in einem einzelnen Nutzdatencontainer. Aufgrund der Ähnlichkeit zwischen *DDT* und *SCA* repräsentieren beide Ansätze im Grunde ein Verfahren, das in Levene/Wood (2002) technisch detaillierter beschrieben wird.

[130] Vgl. Ng/Lam/Cheng (2006), S. 2.
[131] Vgl. Levene/Wood (2002), S. 57.
[132] Für die nachfolgende Beschreibung des Verfahrens vgl. Levene/Wood (2002).

XSAQCT

Müldner u. a.(2008) beschreibt mit *SXSAQCT* eine schemabasierte Version von *XSAQCT*. Die Ausführungen dazu beziehen sich auf die Schemasprache XSD.[133] Bei dem Ansatz wird eine Schemainstanz als Baum modelliert, indem die einzelnen Bäume der Inhaltsmodelle der Elemente kombiniert werden. Knoten, die für Elemente mit variabler Anzahl stehen, werden mit einer Markierung versehen. Bei der Abbildung der Struktur eines XML-Dokuments werden für die markierten Knoten die konkreten Anzahlen bestimmt. Die so gebildete Liste von Zahlen wird, wie auch die nach Pfaden zerlegten Nutzdaten, mit Hilfe eines Universalkompressors verarbeitet. Die Beschreibung des Verfahrens ist insofern unvollständig, als dass die Abbildung von alternativen Bausteinen und Elementpermutationen nicht behandelt wird. Aus diesem Grund muss das Verfahren ausgeschlossen werden.

XSDS

Ein Verfahren, dass mit der Schemasprache XSD eng verknüpft ist, ist *XML Schema Subtraction (XSDS)*.[134] Es wird erstmals in Böttcher/Hartel/Messinger (2010) zur Kompression des Office Open XML Dateiformats skizziert und in Böttcher/Hartel/Messinger (2011) für XML-basierte Zahlungsanweisungen der Single European Payment Area (SEPA) vertieft beschrieben. Ähnlich wie bei den bisher beschriebenen Verfahren wird aus einer XSD ein Baum gebildet und die Struktur eines XML-Dokuments auf Basis dieses Baums abgebildet. Da bei XSD gegenüber DTD wesentlich mehr Sprachelemente zu berücksichtigen sind, besteht der Baum aus bis zu zwölf verschiedenen Knotenarten. Die Abbildung der Struktur ist wie bei anderen baumbasierten Verfahren ein Satz von Daten, der einen Pfad durch den XSD-Baum darstellt und anhand dessen die Struktur rekonstruiert werden kann.

Die binäre Codierung der Strukturabbildung ist der in *KST+DAG* sehr ähnlich. Anzahlen bis 24 werden mit einer vorberechneten Huffmann-Codierung und An-

[133] Für die nachfolgende Beschreibung des Verfahrens vgl. Müldner u. a.(2008).

[134] Für die nachfolgende Beschreibung des Verfahrens vgl. Böttcher/Hartel/Messinger (2010) u. Böttcher/Hartel/Messinger (2011).

zahlen ab 25 als bytebasierte Codewörter variabler Länge codiert. Ein Element aus einer Gruppe alternativer Elemente oder aus einer Elementpermutationsgruppe wird gespeichert, indem die Elemente der jeweiligen Gruppe durchnummeriert und die Nummer des jeweiligen Elements über den Beta-Code binär codiert wird. Die Anzahl der dazu benötigten Stellen für den Beta-Code ergibt sich aus der Anzahl der möglichen Elemente.
Die Nutzdaten werden anhand des datentragenden Bausteinnamens in Container zerlegt und abhängig von ihrem Typ auf drei verschiedene Arten binär codiert. Zahlen werden über eine statische Codierung als bytebasierte Codewörter variabler Länge gespeichert. Werte aus Aufzählungen werden wie Elemente in Elementauswahlgruppen verarbeitet und als binäre Darstellung der Positionsnummer des Wertes gespeichert. Die übrigen Daten werden als Text aufgefasst und können von einem Universalkompressor verarbeitet werden, für den als Beispiele *gzip* und *bzip2* genannt werden.

DTDSub und KST+DAG

Böttcher/Steinmetz/Klein (2007) beschreiben ein Verfahren namens *DTD subtration*[135] (*DTDsub*), das sehr ähnlich zu *DDT* und *SCA* ist. Die Beschreibung des Verfahrens beruht auf der Schemasprache DTD, wenngleich das Verfahren, nach Aussage der Autoren, auf weitere Schemasprachen erweiterbar ist. Im Rahmen dieses Verfahrens wird die DTD-Instanz über binäre Syntaxbäume in eine Attributgrammatik überführt. Mit deren Regeln wird ein XML-Dokument verarbeitet und daraus ein sogenannter *Kleene Size Tree (KST)* erzeugt. Dieser enthält nur die Anzahlen bei Wiederholungen und Daten für Auswahlentscheidungen bei alternativen Elementen. Bei der binären Abbildung des KST wird jeder Eintrag mit mindestens einem Byte gespeichert. Die Nutzdaten werden in einem Block von einem Universalkompressor verarbeitet.
Das Verfahren *KST+DAG*[136] ist eine Verbesserung von *DTDsub*, die eine kompak-

[135] Für die nachfolgende Beschreibung des Verfahrens vgl. Böttcher/Steinmetz/Klein (2007).
[136] Für die nachfolgende Beschreibung des Verfahrens vgl. Böttcher/Hartel/Messinger (2009).

tere binäre Codierung des KST und eine Entfernung redundanter Strukturteile auf globaler Ebene realisiert. Bei der Strukturabbildung werden nicht einzelne Auswahloperatoren codiert, sondern die Alternativen einer Gruppe durchnummeriert und eine konkrete Auswahl als Binärzahl gespeichert. Zur Entfernung von Wiederholungen in der Struktur werden redundante Teilbäume des KST durch Zeiger ersetzt, was der Vorgehensweise bei *DAG+BSBC* entspricht. Sowohl *DTDsub* als auch *KST+DAG* betten in die komprimierten Daten einen Satz von Daten ein, der Abfragen beschleunigt, aber zur vollständigen Dekompression nicht nötig ist.

XCQ

Bei *XCQ* wird ebenfalls eine Baumstruktur zur Abbildung einer Schemainstanz verwendet.[137] Die Beschreibung bezieht sich auf die Schemasprache DTD, wobei die Erweiterbarkeit auf andere Sprachen nicht thematisiert wird. Im Gegensatz zu den bisher betrachteten baumbasierten Verfahren werden hier Knoten nur für Elemente, Attribute, Nutzdaten und den Auswahloperator angelegt. Eine Sequenz wird abgebildet, indem dem jeweiligen Elternknoten mehrere Kindknoten zugeordnet werden. Bei allen Elementen, die nicht verpflichtend, sondern optional sind oder wiederholt werden können, wird die mögliche Anzahl konkreter Ausprägungen dem Knoten als Annotation hinzugefügt. Auswahlknoten werden, wie bei *KST+DAG*, als binäre Codierung der Nummer der gewählten Alternative gespeichert. Wiederholungen werden hingegen nicht als Zahlen, sondern mit Bit-Markierungen pro Wiederholung abgebildet.

Die Nutzdaten werden anhand ihres Strukturpfades in Container aufgeteilt und zur Beschleunigung von Abfragen weiter in Blöcke zerlegt, die mit *gzip* komprimiert werden. Pro Block wird zudem eine Zusammenfassung seines Inhalts gespeichert, um bei Abfragen die relevanten Blöcke identifizieren zu können. Es handelt sich somit um rein zusätzliche Daten, die zur vollständigen Dekompression unnötig sind.

[137] Für die nachfolgende Beschreibung des Verfahrens vgl. Ng u. a.(2006).

Zusammenfassung der baumbasierten Verfahren

Zusammenfassend ist festzuhalten, dass alle baumbasierten Verfahren die Struktur auf sehr ähnliche Art und Weise abbilden und sich meist nur in der binären Codierung dieser Abbildung unterscheiden. Dass dazu sowohl Verfahren zählen, die auf die Schemasprache DTD ausgerichtet sind, als auch solche, die auf XSD ausgelegt sind, zeigt, dass die benötigte Erweiterbarkeit der DTD-basierten Verfahren auf XSD gegeben ist.

Die Erweiterbarkeit der baumbasierten Verfahren kann damit begründet werden, dass bei diesen jeweils das nächste Element im XML-Dokument ausreicht, um den Pfad innerhalb eines schemabeschreibenden Baums zu bestimmen. Dies ist sowohl bei DTD als auch bei XSD gegeben, da in beiden Schemasprachen aus der Abfolge zweier Elemente ohne weiteren Kontext eindeutig die Regel ableitbar sein muss, die die konkrete Gestaltung eines XML-Dokuments beschreibt.[138] Bei RELAX NG ist diese Einschränkung nicht vorhanden, weswegen entweder der Schemabaum entsprechend gestaltet sein muss oder der Kontext auf eine andere Art modelliert werden muss.

Von den Verfahren dieses Abschnitts ist grundsätzlich *KST+DAG* den anderen überlegen. Es ist in der Struktur- und Nutzdatenverarbeitung sehr ähnlich zu *SCA* und damit auch zu *DDT*, berücksichtigt jedoch Wiederholungen in der Struktur. Der Vorteil gegenüber *XSDS* und *XCQ* liegt in der Verarbeitung der Nutzdaten, da bei diesen beiden Verfahren diese nach Namen bzw. Pfaden in Container verteilt werden.

Wenngleich alle Verfahren auf XSD erweiterbar sind, stellt nur *XSDS* ein Verfahren dar, das alle relevanten Konzepte von XSD abbildet. Eine sinnvolle Implementierung eines baumbasierten Verfahrens für XSD ist damit eine Kombination aus *KST+DAG* und *XSDS*.

[138] Siehe dazu Kapitel 2.1.3.

3.6 Untersuchungsergebnisse und deren Implikationen

Im Rahmen der Auswahluntersuchung dieses Kapitels wurden anhand qualitativer Kriterien von den in der Literatur vorhandenen Kompressionsverfahren für XML-Dokumente 13 XML-spezifische Verfahren identifiziert, die quantitativ zu untersuchen sind. Sie unterteilen sich in acht Verfahren mit Bezeichnungs- und fünf mit Variantencodierungen.

3.6.1 Bezeichnungscodierungen

Von den Verfahren mit Bezeichnungscodierungen liegen die folgenden sechs in implementierter Form vor: *XWRT*, *XBzip*, *XMLPPM*, *Exalt*, *XMill* und *SCMPPM*. Indem die beiden Letzteren in der Fallstudie in Abschnitt 5.3 berücksichtigt werden, kann die bisher nur durch eine Argumentation dargelegte Aussage, dass die Aufteilung der Nutzdaten nach dem Namen der umgebenden Bausteine für die Kompression von Geschäftsdokumenten von Nachteil ist, bekräftigt werden.
Die Verfahren *XSeq* und *DAG+BSBC* wurden trotz fehlender Implementierung in die vertiefte Untersuchung aufgenommen. Begründet werden kann dies damit, dass *DAG+BSBC* einen vielversprechenden Ansatz zur Entfernung der Redundanz in der Struktur verfolgt. *XSeq* wurde aufgenommen, da der Implementierungsaufwand gering ist und auf diese Weise neben *Exalt* ein zweites grammatikbasiertes Verfahren evaluiert werden kann.

3.6.2 Variantencodierungen

Von den fünf Verfahren mit Variantencodierungen liegt lediglich *EXI* in implementierter Form vor. Bei den baumbasierten Verfahren ist eine Mischung aus *KST+DAG* und *XSDS* zu implementieren. Im Bereich der automatenbasierten Verfahren wurde ferner ermittelt, dass *rngzip* und *DPDT-L* sowie *DTDPPM* mit der Schemamodellierung von *XAUST* zu implementieren sind.
Im Rahmen der Untersuchungen wurde festgestellt, dass die Verfahren nicht nur innerhalb der baumbasierten, sondern auch in der automatenbasierten Gruppe starke

Ähnlichkeiten im Hinblick auf die logische Abbildung der Struktur aufweisen. Die Konzepte zur Modellierung des Schemas und zur Abbildung der Struktur unterscheiden sich zwar, dennoch sind die Ergebnisse der Strukturabbildung, abstrahiert von Unterschieden in der binären Codierung, sehr ähnlich.
Aus diesem Grund können Verfahren, die aufgrund ihrer Gesamtleistung ausgeschlossen wurden, Techniken enthalten, die eine Teilaufgabe der Kompression besser lösen als dies die entsprechenden Konzepte in anderen Verfahren tun. Beispielsweise könnte die Herangehensweise von *SXSAQCT*, die Strukturabbildung durch einen Universalkompressor binär codieren zu lassen, eine kompaktere Darstellung erzeugen als dies bei *KST+DAG* der Fall ist.
Diese Erkenntnisse führen zu der Schlussfolgerung, dass es nicht sinnvoll ist, die identifizierten Verfahren mit Variantencodierungen einzeln zu implementieren und als geschlossene Einheiten zu vergleichen. Vielmehr sind diese in einzelne Schritte zu zerlegen, um untersuchen zu können, welche Kombinationen der Verarbeitungsschritte und welche Technik pro Schritt zur höchsten Kompressionsleistung führt. Auf diese Weise kann analysiert werden, ob eine Kombination von Techniken existiert, die eine höhere Kompressionsleistung als die vorhandenen Verfahren ermöglicht. Um dies untersuchen zu können, wurde ein parametrisiertes Verfahren zur schemabasierten XML-Kompression entwickelt, dessen Konzeption und prototypische Implementierung im nächsten Kapitel beschrieben wird.

.

4 Parametrisiertes XML-Kompressionsverfahren

Im vorhergehenden Kapitel wurde der Bedarf identifiziert, die Verfahren mit Variantencodierung in Teilschritte zu zerlegen und die verschiedenen Techniken dieser Schritte sowie deren unterschiedliche Kombinationsmöglichkeiten zu vergleichen. Zur Bewältigung dieser Aufgabe wurde ein parametrisiertes Verfahren zur schemabasierten XML-Kompression (kurz *PSXC* für *Parametric Schema-based XML Compression*) entwickelt, dessen Konzeption und prototypische Implementierung in diesem Kapitel beschrieben wird. *PSXC* ist nur auf die Schemasprache XSD ausgelegt, da sie sich im Bereich von Geschäftsdokumenten etabliert hat.[1]
Die Teilschritte bestehender Verfahren werden im parametrisierten Verfahren über Funktionsbausteine abgebildet, deren Verhalten über Parameter gesteuert werden kann. Auf diese Weise ist es möglich, unterschiedliche Konzepte pro Teilschritt und verschiedene Kombinationen von Teilschritten zu realisieren. Die Grundstruktur des Verfahrens wurde dazu aus dem gemeinsamen Aufbau aller Verfahren mit Variantencodierung abgeleitet. Die berücksichtigten Techniken stammen ebenfalls grundlegend aus den vorhandenen Verfahren, wobei nur die leistungsfähigsten Konzepte herausgefiltert und einzelne optimierende Modifizierungen vorgenommen wurden.[2]
Das Kapitel ist wie folgt aufgebaut: Im ersten Abschnitt wird der schematische Aufbau des parametrisierten Verfahrens als Verallgemeinerung der Vorgehensweise bestehender Verfahren mit Variantencodierung entwickelt. Im Anschluss daran (Abschnitt 4.2) folgt ein systematischer Überblick der in den bestehenden Verfah-

[1] Siehe hierzu die Ausführungen im Grundlagenkapitel 2.2.
[2] Bei allgemeinen Aussagen zur Verwendung von Techniken in Verfahren werden im Folgenden nicht an jeder Stelle explizit Quellangaben beigefügt, da sie auf den bereits in den Abschnitten 3.4 und 3.5 hinreichend ausführlichen Darstellung und den ebenda angegebenen Quellen basieren.

ren eingesetzten Konzepte. Aufbauend darauf wird in Abschnitt 4.3 die Gestaltung der Funktionsbausteine sowie deren Parameter dargelegt. In diesem Rahmen werden die zuvor beschriebenen Konzepte bewertet und die daraus resultierende Berücksichtigung im parametrisierten Verfahren beschrieben. Der letzte Abschnitt behandelt die prototypische Implementierung des Verfahrens.

4.1 Schematischer Aufbau des Verfahrens

Alle Verfahren mit Variantencodierung folgen auf abstrakter Ebene einem gemeinsamen Ablauf, wie er in Abbildung 4.1 schematisch dargestellt ist. Die auf der linken Seite abgebildeten Verarbeitungsschritte (grüne Boxen) sind nur in Verfahren mit Variantencodierungen enthalten. Die Schritte der rechten Seite (gelbe Boxen) finden sich hingegen in allen XML-spezifischen Verfahren. Der erste Schritt jedes Verfahrens ist das Einlesen des XML-Dokuments. Bei der anschließenden Verarbeitung folgen alle Verfahren mit Variantencodierung dem bei *XMill* eingeführten Konzept der getrennten Kompression von Struktur- und Nutzdaten.
Bei der Variantencodierung wird das Schema genutzt, um die Struktur eines XML-Dokuments zu komprimieren. Aus diesem Grund muss das zu verwendende Schema eingelesen und in ein internes Modell überführt werden. Im ersten der drei Verarbeitungsschritte der Variantencodierung wird die Struktur eines XML-Dokuments in Relation zum Schemamodell abgebildet. Bei dieser Abbildung wird eine Folge von Daten generiert, die eine Rekonstruktion der Struktur des ursprünglichen XML-Dokuments ermöglichen. Beispielsweise wird für ein sich wiederholendes Element nur seine Anzahl gespeichert. Diese ist bei der Dekompression ausreichend, da sich aus dem Schema ergibt, welches Element zu wiederholen ist. Im nächsten Schritt, der nur in einem Verfahren zu finden ist, werden sich wiederholende Strukturteile durch Verweise auf das erste Auftreten der Teilstruktur ersetzt, wodurch die redundante Abbildung der jeweiligen Strukturteile eingespart wird. Im letzten Schritt werden die Daten der Strukturabbildung binär codiert.
In der ersten Stufe der Nutzdatenverarbeitung werden die Nutzdaten für die Kom-

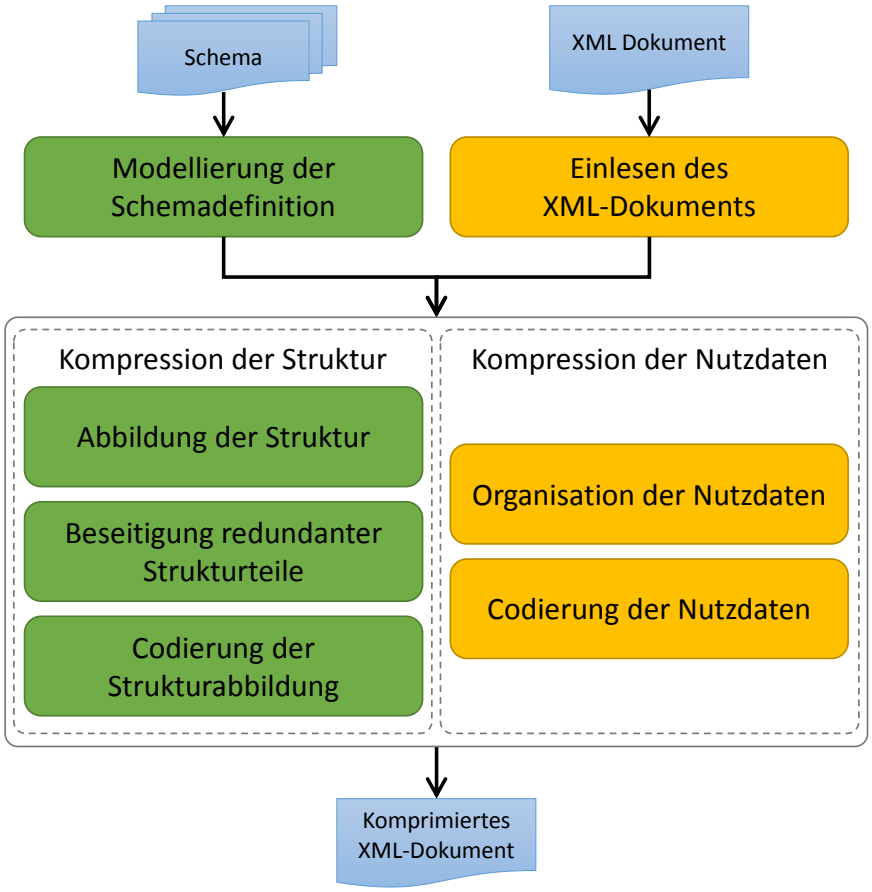

Abbildung 4.1: Schematischer Kompressionsprozess mit Variantencodierung (Eigene Darstellung)

pression vorbereitet. Im Zuge dessen werden sie, je nach Verfahren, in Container aufgeteilt und deren Reihenfolge geändert. Zudem werden hier Schemainformationen verarbeitet und beispielsweise textuelle Werte, für die in der Schemainstanz eine Werteliste definiert ist, durch die jeweilige Positionsnummer ersetzt. In der zweiten Stufe erfolgt die eigentliche Kompression. Falls Container gebildet wurden, werden diese einzeln verarbeitet. Dazu werden entweder eigene Varianten bestehender Kompressionsalgorithmen oder Universalkompressionen eingesetzt.

Die Darstellung des Prozesses als sequenzielle Abfolge ist eine vereinfachte Modellierung, da in vielen Verfahren die Verarbeitung blockweise in einer sich mehrfach wiederholenden Abfolge von Einlesen und Komprimieren der Struktur- und Nutzdaten erfolgt. Auf der hier betrachteten Abstraktionsebene ist die Modellvorstel-

lung ausreichend, zumal ausreichend kleine Dokumente ohnehin in einem Block verarbeitet werden.

4.2 Konzepte bestehender Verfahren

Im folgenden Abschnitt werden die Techniken der bestehenden Verfahren systematisch beschrieben. Bei der Erhebung der Techniken im Bereich der Strukturverarbeitung werden alle Verfahren mit Variantencodierung berücksichtigt, die in Abschnitt 3.5 behandelt wurden. Für die Methoden der Nutzdatenkompression werden zusätzlich die Verfahren mit Bezeichnungscodierungen aus Abschnitt 3.4 herangezogen. Die Untergliederung der Ausführungen orientiert sich an den Verfahrensschritten des im Abschnitt 4.1 aus den bestehenden Verfahren extrahierten Aufbaus des parametrisierten Verfahrens.
Bei den folgenden Ausführungen wird die Verarbeitung von XML-Elementen beschrieben, ohne explizit auf deren Attribute einzugehen. Die Verarbeitung von Attributen erfolgt bei allen Verfahren analog zur Behandlung einer Sequenz von XML-Elementen.

4.2.1 Modellierung der Schemadefinition

Die Nutzung von Schemainformationen erfordert eine geeignete Modellierung der Schemainstanz. Bei den bestehenden Verfahren erfolgt diese entweder als Baum oder in Form eines für die Verarbeitung des XML-Dokuments verwendeten Automaten.

4.2.1.1 Modellierung als Baum

Wenngleich die konkrete Gestaltung des aus dem Schema erzeugten Baumes je nach Verfahren unterschiedlich ist, ist die Grundidee bei allen identisch. Verschiedene Arten von Knoten definieren die unterschiedlichen Eigenschaften des erlaubten Aufbaus der Struktur eines XML-Dokuments. Dabei werden immer Knoten für XML-Elemente über Knoten verbunden, die deren Beziehung zueinander definieren.

Bei der Gestaltung der Bäume sind verschiedene Ansätze zu unterscheiden. Bei dem Ansatz, dem alle DTD-basierten Verfahren außer *XCQ* folgen, wird jeder Kompositions- und Häufigkeitsoperator als separater Knoten abgebildet.[3] Die beiden Kompositionsoperatoren `,` und `|` stellen, entsprechend ihrer Stelligkeit, Knoten mit zwei Nachfolgern dar. Die Knoten der drei Häufigkeitsoperatoren `?`, `+` und `*` haben hingegen nur jeweils einen einzelnen Nachfolger, der entweder ein XML-Element oder ein Kombinationsoperator sein kann.

Bei *XCQ* werden nicht die Operatoren direkt als Knoten verwendet, sondern alle Bausteine, die mit dem gleichen Operator in einer Reihe verknüpft sind, zusammengefasst.[4] So werden alle Knoten einer Sequenz als Kindknoten eines XML-Elements zusammengefasst. Für eine Gruppe von Alternativen wird hingegen ein expliziter Knoten verwendet, dessen Kinder die Alternativen darstellen. Die Häufigkeitsdefinitionen werden als Operatoren den zugehörigen Knoten beigefügt. Ein entsprechendes Beispiel zeigt Abbildung 4.2.

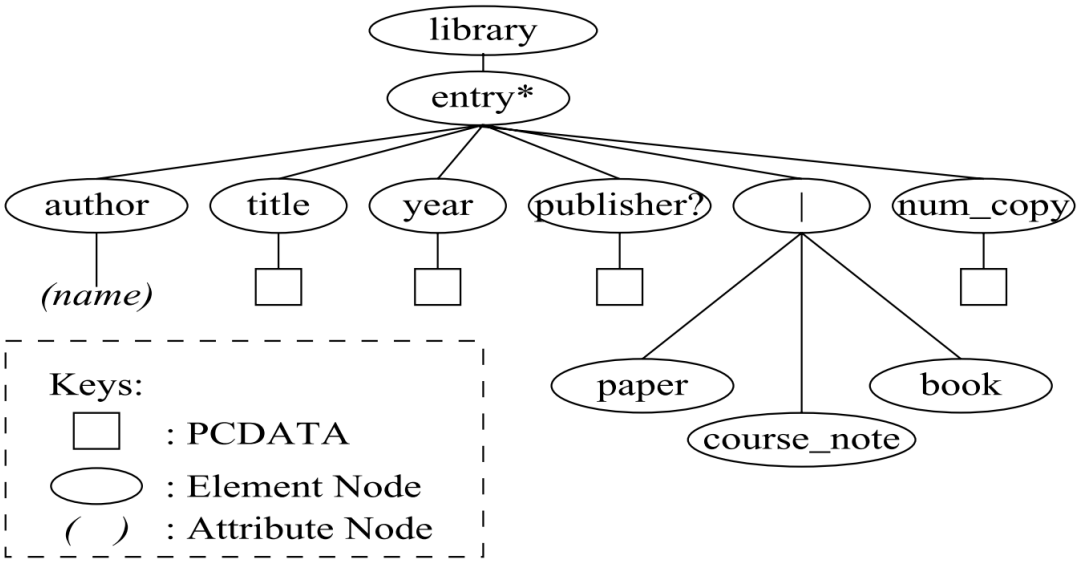

Abbildung 4.2: Modellierung als Baum bei *XCQ* (Ng u. a.(2006) S. 424)

[3] Für die folgenden Ausführungen zu diesem Ansatz vgl. beispielsweise Böttcher/Steinmetz/Klein (2007) S. 87.

[4] Für diesen und die nächsten drei Sätze vgl. Ng u. a.(2006), S. 424.

Der dritte Ansatz betrifft die Modellierung von Schemainstanzen in der umfangreicheren Schemasprache XML-Schema, wie es am Beispiel von *XSDS* in Abbildung 4.3 dargestellt ist. Sequenzen und Alternativen werden hier nicht über binäre Operatoren verknüpft, sondern über Gruppen und entsprechende Kompositoren definiert.[5] Neben den in DTD ebenfalls vorhandenen Sequenzen und Alternativen gibt es bei XSD zusätzlich noch die Elementpermutation des `<xsd:all>`-Kompositors. Bei der Abbildung der Kompositionsgruppen wird der Kompositor als Elternknoten verwendet und diesem für jedes Gruppenelement ein Kindknoten angefügt.[6] Im Hinblick auf die Häufigkeitsdefinition gibt es nur einen Knotentyp. Er enthält die minimale und maximale Anzahl, mit der sein einziger Nachfolger, der entweder ein XML-Element oder eine Gruppe sein kann, auftreten darf.

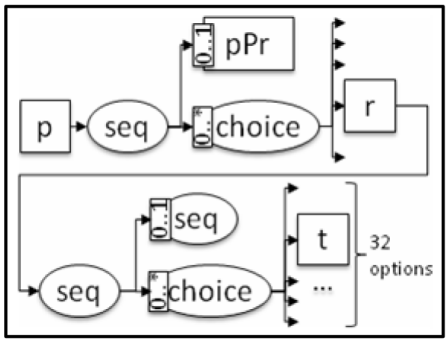

Abbildung 4.3: Ausschnitt eines Schemabaums bei *XSDS* (Böttcher/Hartel/Messinger (2010) S. 105)

4.2.1.2 Modellierung über Automaten

Die automatenbasierte Modellierung verwendet eine Schemainstanz als Definition einer formalen Grammatik und erstellt auf dieser Basis einen Automaten. Bei DTD-basierten Verfahren wird für jedes XML-Element ein deterministischer endlicher Automat (DEA) angelegt, der das Inhaltsmodell des Elements abbildet.[7] Ein beispielhafter DEA ist in Abbildung 4.4 dargestellt. Das entsprechende Inhalts-

[5] Siehe dazu Abschnitt 2.1.3.

[6] Für diesen und die nächsten zwei Sätze vgl. beispielsweise Böttcher/Hartel/Messinger (2011) S. 453

[7] Zu den folgenden Ausführungen zur automatenbasierten Modellierung bei DTD-basierten Verfahren vgl. beispielsweise Subramanian/Shankar (2006) S. 285-288.

modell erlaubt als Name entweder einen einzelnen Eintrag oder die Angabe des Vornamens und anschließend des Nachnamens. Die Angabe der E-Mail-Adresse ist verpflichtend und die einer Notiz optional.

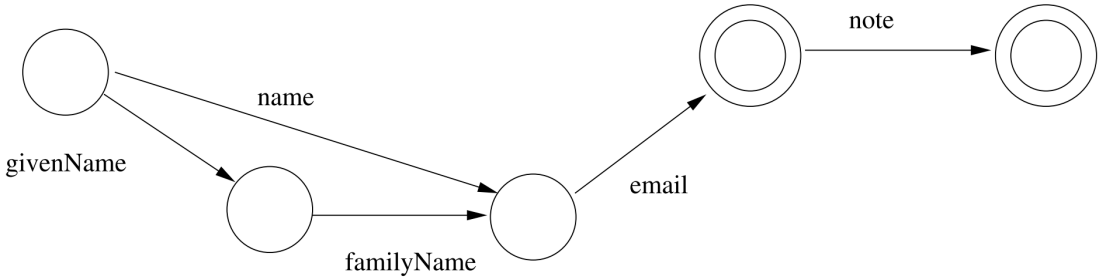

Abbildung 4.4: Automat des Inhaltsmodells eines Adressbucheintrags (Subramanian/ Shankar (2006) S. 287)

Die Zustände der Automaten sind als abstrakte Zustände des Verarbeitungsprozesses eines XML-Dokuments zu sehen. Die Zustandsübergänge geben jeweils an, welche Elemente an der dem Zustand entsprechenden Stelle des Verarbeitungsprozesses als Nächstes erlaubt sind. Im Startzustand eines Automaten geben die Zustandsübergänge somit an, welche Elemente als erstes Kindelement eines Elements auftreten dürfen. Im Beispiel der Abbildung 4.4 sind dies die Elemente *givenName* und *name*.

Im Zuge eines Zustandsübergangs muss auch das dem Zustandsübergang entsprechende Element verarbeitet und damit ein weiterer DEA durchlaufen werden. Dadurch müssen bei der Verarbeitung eines XML-Dokuments mehrfach geschachtelte DEAs abgearbeitet werden. Dies führt dazu, dass nach jeder Abarbeitung eines DEAs zum passenden Zustand des jeweils übergeordneten DEA zurückgekehrt werden muss. Um dies zu ermöglichen, wird ein Kellerspeicher verwendet, auf den bei jedem Absprung in einen neuen DEA ein Wert gelegt wird, der den als nächstes zu verarbeitenden Zustand des aktuellen DEAs identifiziert. Vom Kellerspeicher wird immer dann gelesen, wenn ein Automat einen Endzustand erreicht hat.

Die Lösung von *rngzip* ist sehr ähnlich zum Ansatz der DTD-basierten Verfahren, ermöglicht jedoch die Verarbeitung der mächtigeren Schemasprache RELAX NG. Mit dieser kann ein XML-Element abhängig von dessen Kontext, der durch die

umgebenden Elemente definiert ist, verschiedene Inhaltsmodelle besitzen.[8] Dies zu modellieren ist mit der zuvor beschriebenen Kombination aus DEAs und einem Kellerspeicher nicht möglich, da dort jedes XML-Element von genau einem DEA abgebildet wird und damit kein kontextsensitiver Wechsel der Inhaltsmodelle möglich ist.

Aus diesem Grund verwendet *rngzip* Baumautomaten.[9] Die in *rngzip* verwendete Methodik ist der bereits beschriebenen dennoch ähnlich, da hier ebenfalls eine Sammlung von DEA-ähnlichen Teilautomaten in Kombination mit einem Kellerspeicher vorliegt. Der Unterschied ist jedoch der, dass für ein XML-Element nicht nur ein einzelner Teilautomat, sondern für jedes der verschiedenen möglichen Inhaltsmodelle eines XML-Elements ein eigener Teilautomat existiert. Da ein Teilautomat folglich nicht eindeutig durch ein XML-Element identifizierbar ist, ist bei jedem Zustandsübergang der Startzustand des Teilautomaten angegeben, der im Zuge des Zustandsübergangs abzuarbeiten ist. Um nach der Verarbeitung des jeweils untergeordneten Teilautomaten in den korrekten Zustand überzugehen, wird ebenfalls ein Kellerspeicher verwendet.

4.2.2 Abbildung der Struktur eines XML-Dokuments

Die Abbildung der Struktur als Variante ist die Überführung der Struktur in einen Satz von Daten, der so beschaffen ist, dass daraus die ursprüngliche Struktur auf Basis des modellierten Schemas rekonstruiert werden kann. Um möglichst wenig Speicherplatz zu benötigen, werden nur Daten für variable Teile einer Schemainstanz gespeichert, da die verpflichtenden Elemente ohnehin eindeutig rekonstruierbar sind. Wenngleich die binäre Codierung der Daten je nach Verfahren sehr unterschiedlich ist, werden in allen Verfahren einer Modellierungsart die gleichen Arten von Daten ermittelt.

[8] Vgl. Murata u. a.(2005), S. 476 f.
[9] Für die Ausführungen zu diesem Verfahren vgl. League/Eng (2007b) S. 276-278

4.2.2.1 Baumbasierte Abbildung

Bei der baumbasierten Abbildung wird das XML-Dokument parallel zum modellierten Baum der Schemainstanz durchlaufen. In diesem Prozess wird bei allen Knoten, die keine feste Gestaltung des Dokuments erzwingen, ermittelt, welche der von dem Knoten erlaubten Varianten in dem konkreten XML-Dokument vorliegt. In den Knoten werden daraufhin Daten gespeichert, die einen Rückschluss auf die Variante ermöglichen. Bei allen anderen Knoten ist dies nicht nötig, da die Gestaltung des Dokuments an diesen Stellen eindeutig rekonstruierbar ist.
Inwiefern ein Knoten für fixe und damit eindeutig rekonstruierbare Strukturteile oder variable Teile eines Dokuments steht, wird im Folgenden anhand der Bestandteile Element, Sequenz, Alternative, Häufigkeitsdefinition und Elementpermutation (`<xsd:all>`) näher betrachtet. Die Abbildung von Attributen kann, wie bereits erklärt, analog zur Abbildung von Elementsequenzen betrachtet werden.

Feste Bestandteile

Knoten für XML-Elemente stehen nur für das konkrete Element und sind somit fest. Sequenzen sind ebenfalls eindeutig rekonstruierbar, unabhängig davon, ob sie als Knoten mit mehreren Nachfolgern oder über einen binären Teilbaum abgebildet werden. Per Definition müssen alle in einer Sequenz beteiligten Bestandteile in der angegebenen Reihenfolge auftreten. Der Fall, dass in einem XML-Dokument ein Bestandteil der Sequenz fehlt, bedeutet daher nur, dass dieser mit der Häufigkeit 0 auftritt, was gesondert abgebildet wird. Bei der Rekonstruktion eines XML-Dokuments müssen folglich immer alle Kindknoten eines Sequenzknotens abgearbeitet werden. Alle weiteren Bestandteile einer Schemadefinition sind nicht eindeutig rekonstruierbar.

Alternativen und Elementpermutationen

Bei Alternativen, unabhängig von deren Modellierung als Knoten mit mehreren Nachfolgern oder als binärer Teilbaum, wird bei deren Abbildung gespeichert, welche Alternative im konkreten Dokument vorliegt. Sowohl bei der Abbildung als

auch bei der Rekonstruktion wird nur der Nachfolgeknoten verarbeitet, der für die jeweils vorliegende Alternative steht.

Zur Identifikation der Alternativen wird bei allen Verfahren ein Zeichenvorrat pro Auswahlknoten gebildet, dessen Symbole den Alternativen umkehrbar eindeutig zugeordnet sind. Es wird jeweils das Symbol gespeichert, das der zu wählenden Alternative entspricht. Der Zeichenvorrat fungiert in der binären Codierung als Quellenalphabet. Im Fall der Modellierung über binäre Bäume enthält jeder Zeichenvorrat nur zwei Symbole und kann daher direkt binär codiert werden. Die einzige Ausnahme dazu bildet *KST+DAG*, da hier alle Nachfolger verbundener binärer Auswahlknoten zusammengefasst werden. Damit werden sie als Alternativen eines Auswahlknotens behandelt und wie bei nicht binären Bäumen gespeichert.

Ein Kompositor, der nur bei XSD vorkommt und nur bei dem Verfahren *XSDS* berücksichtigt wird, ist der einer Elementpermutation. Bei dieser dürfen die Elemente der Gruppe in beliebiger Reihenfolge, aber maximal einmal auftreten. Es sind somit die in einem XML-Dokument konkret vorkommenden XML-Elemente sowie deren Reihenfolge zu speichern. Die gespeicherten Daten entsprechen denen der mehrfachen Wiederholung einer Auswahl aus Alternativen.

Wiederholungen

Wenngleich sich die Modellierung der Häufigkeitsdefinitionen bei Verfahren für DTD und RELAX NG von denen für XSD geringfügig unterscheidet, ist die Abbildung identisch. Bei Knoten, die keine feste Häufigkeit definieren, wird die konkrete Anzahl der Wiederholungen des Elements bzw. der Gruppe ermittelt und gespeichert. Als fixe Häufigkeitsdefinitionen sind bei DTD und RELAX NG nur das einmalige Auftreten eines Elements möglich, was dadurch gekennzeichnet wird, dass kein Häufigkeitsoperator verwendet wird. Da bei XSD die minimale und maximale Anzahl frei definiert werden kann, ist die Angabe einer beliebigen festen Häufigkeit möglich.

Bei der Abbildung von Häufigkeitsinformationen treten im Allgemeinen zwei verschiedene Szenarien auf. Die beiden Sonderfälle *XCQ* und *DDT* werden später ein-

zeln betrachtet. In einem Szenario ist das Vorhandensein eines optionalen Bausteins und im anderen die Anzahl der konkret vorhandenen Elemente zu erfassen. Ein optionaler Baustein wird grundsätzlich über einen binären Zeichenvorrat abgebildet. Beim Speichern der Häufigkeit eines Bausteins wird im Allgemeinen, mit Ausnahmen von *SCA*, von der vorhandenen Anzahl an Bausteinen die minimal erlaubte Anzahl subtrahiert und dann als Zahlenwert zur Weiterverarbeitung gespeichert. Die mit XSD definierbare Obergrenze wird von keinem Verfahren genutzt.
Der Unterschied bei *DDT* zu den oben beschriebenen Methoden besteht in der Nichtberücksichtigung von optionalen Bausteinen und minimalen Anzahlen, die stattdessen immer als direkt auftretende Anzahl an Bausteinen gespeichert werden.
Bei *XCQ* ist die Herangehensweise eine grundlegend andere. Hier wird bei jedem Baustein, der variabel oft auftreten kann, eine binäre Markierung und somit ein Symbol aus einem binärer Zeichenvorrat gespeichert, um pro Baustein anzugeben, ob ein weiterer Baustein vorhanden ist.

4.2.2.2 Automatenbasierte Abbildung

Bei den automatenbasierten Verfahren wird die Struktur eines XML-Dokuments mit dem aus dem Schema gebildeten Automaten verarbeitet, wobei dieser als Transduktor fungiert und bei Zustandsübergängen Ausgaben produziert, die als Abbildung der Struktur gespeichert werden. Auf Basis der gespeicherten Ausgaben kann der Ablauf des Automaten nachgebildet und die Struktur des XML-Dokuments rekonstruiert werden.
Ist an einer Stelle der Verarbeitung der jeweils nächste Baustein verpflichtend, bietet der Automat nur einen Zustandsübergang und damit keine Wahlmöglichkeit, weshalb in diesem Fall keine Ausgabe zu speichern ist. Anders ist dies bei den Zuständen, deren nächster Zustand nicht eindeutig durch den Automaten vorgegeben ist. Dies ist nicht nur dann der Fall, wenn mehr als ein Zustandsübergang vorhanden ist, sondern auch dann, wenn der Automat in einem Zustand ist, der einen potentiellen Endzustand darstellt und dennoch über mindestens einen Zustandsübergang verlassen werden kann.

Im Beispiel der Abbildung 4.4 ist die Angabe der E-Mail-Adresse verpflichtend, weshalb hier nur ein Zustandsübergang vorhanden ist. Die optionale Notiz stellt den zuletzt beschriebenen Fall dar. Trotz eines einzelnen Zustandsübergangs vom vorletzten zum letzten Zustand ist nicht eindeutig, dass dieser gewählt werden muss, da der vorletzte Zustand ohne einen Notizeintrag der Endzustand des DEA ist.

Es existieren zwei unterschiedliche Ansätze, wie die Daten, die ein Automat bei einem Zustandsübergang zur Identifizierung des konkreten Übergangs ausgibt, beschaffen sind. Eine Vorgehensweise, wie sie beispielsweise *XAUST* verfolgt, ist, die Bezeichnung des jeweils verarbeiteten XML-Elements auszugeben. Der andere Ansatz, wie er beispielsweise bei *rngzip* zu finden ist, bildet einen Zeichenvorrat pro Zustand, dessen Symbole den Zustandsübergängen des Zustands umkehrbar eindeutig zugeordnet sind. Es wird jeweils das Symbol gespeichert, das dem zu wählenden Zustandsübergang entspricht. Der Zeichenvorrat fungiert in der binären Codierung als Quellenalphabet.

4.2.3 Behandlung von Redundanz in der Struktur

Redundanz in der Struktur wird nicht bei allen Verfahren behandelt. *KST+DAG* ist das einzige Verfahren, das redundante Teile der Struktur eines XML-Dokuments auf globaler Ebene identifiziert und entfernt. Eine zweite Herangehensweise ist die Delegation der Strukturredundanzverarbeitung an nachgelagerte Universalkompressoren, wie es beispielsweise bei *XAUST* der Fall ist.

Bei *KST+DAG* ist das Ergebnis der Strukturabbildung ein Baum, der die Struktur eines XML-Dokuments darstellt.[10] Dieser Baum in Verbindung mit einer Signatur, die den Aufbau eines Teilbaums beschreibt, bildet die Basis für die Erkennung redundanter Strukturteile. Der Baum besteht, entsprechend der Struktur des XML-Dokuments, aus Knoten für feste und solche für variable XML-Elemente bzw. Attribute. An den Knoten für variable Bausteine werden die bei der Strukturabbildung erzeugten Daten annotiert.

[10] Zur Beschreibung dieser Technik vgl. Böttcher/Hartel/Messinger (2009) S. 2-5.

Zur Erkennung redundanter Strukturabschnitte wird immer dann, wenn ein Teilbaum unterhalb eines Knotens für variablen Inhalt abgebildet wurde, dessen Signatur bestimmt und überprüft, ob bereits ein Teilbaum mit dieser Signatur vorhanden ist. Falls dies nicht der Fall ist, wird der Teilbaum unverändert belassen. Falls doch, wird überprüft, ob es sinnvoll ist, den Teilbaum durch einen Zeiger auf einen bereits bestehenden Teilbaum zu ersetzen. Durch diese Zeiger wird aus dem Baum ein gerichteter azyklischer Graph (engl. *directed acyclic graph*, DAG). Ob ein Zeiger sinnvoll ist, hängt davon ab, ob er weniger Speicherplatz benötigt, als die in dem zu ersetzenden Teilbaum beinhalteten Symbole dies tun.
Zur Speicherung eines Zeigers muss die Quelle und das Ziel abgebildet werden. Zur Abbildung der Quelle schlagen Böttcher/Hartel/Messinger (2009) zwei Ansätze vor.[11] Entweder die separate Auflistung der Knoten, die Quellen von Zeigern sind, oder die Verwendung einer binären Markierung, die angibt, ob ein Knoten, der als Quelle fungieren kann, eine Quelle ist. In einem Vergleich der beiden Varianten ermittelten die Autoren, dass die Knotenmarkierung bei ihnen deutlich bessere Kompressionsraten liefert. Die Abbildung der Ziele erfolgt bei diesem Verfahren durch die Speicherung der Distanz zwischen den beiden Knoten. Als Maß für die Distanz verwenden die Autoren die Anzahl an Elementen zwischen den Knoten. Sie merken zudem an, das die Verwendung von Bits oder Bytes zur Adressierung innerhalb der codierten Strukturabbildung nicht ausreichend komprimierbar ist.

4.2.4 Codierung der Strukturabbildung

Der Schritt umfasst die binäre Codierung der Daten, die im Rahmen der Strukturabbildung erstellt wurden. Zu diesem Zweck werden sowohl statische als auch dynamische Codierungsverfahren verwendet.[12]
Alle betrachteten automatenbasierten Verfahren, die die Bezeichnung eines XML-Elements zur Identifikation eines Zustandsübergangs verwenden, codieren diese

[11] Für diesen und die nächsten fünf Sätze vgl. Böttcher/Hartel/Messinger (2009), S. 4-5.
[12] Für die verschiedenen Codierungs- bzw. Kompressionstechniken siehe Grundlagen Abschnitt 2.4.2.

über PPM-Verfahren und damit dynamisch.[13] Bei der anderen Identifikation der Zustandsübergänge über individuelle Zeichenvorräte pro Zustand kommen sowohl statische als auch dynamische Codierungen zum Einsatz. Bei der statischen Codierung wird, wie beispielsweise bei *rngzip*, der Beta-Code verwendet.[14] Bei der dynamischen Codierung werden die Zeichenvorräte der verschiedenen Zustände verschmolzen und mit einem Universalkompressor codiert. Bei *DPDT-L* wird dazu eine PPM-Variante verwendet.[15]

Im Bereich der baumbasierten Abbildung kommen vorwiegend statische Codierungen zum Einsatz. Die Codierung der Quellenalphabete, die im Rahmen der Auswahl aus Alternativen bei der Strukturabbildung verwendet werden, erfolgt in der Regel auf Basis des Beta-Codes. Nur bei *XCQ* und *DTDsub* werden binäre Auswahlentscheidungen mit einem ganzen Byte codiert. Symbole aus binären Zeichenvorräten bei optionalen Bausteinen und solchen, bei denen nur zwei unterschiedliche Anzahlen möglich sind, werden in einem einzelnen Bit gespeichert. Die einzige Ausnahme dazu bildet *DTDsub*, das hier ein ganzes Byte verwendet.

Bei der Codierung von Häufigkeiten und damit Zahlenwerten können vier Ansätze unterschieden werden. Eine Herangehensweise, die nur bei *SCA* gefunden wurde, ist, die größte Zahl zu bestimmen und auf deren Basis alle Zahlen als Beta-Code zu codieren. Der Ansatz von *XCQ*, jede Wiederholung als einzelnes Bit abzubilden, ist gleichbedeutend mit der Codierung der Anzahl als Zählcode. Bei der dritten Variante, die ebenfalls nur bei einem Verfahren (*SXSAQCT*) verwendet wird, werden die Zahlen über einen nachgelagerten Universalkompressor codiert.

Alle anderen Verfahren verwenden statische Abbildungsvorschriften, die Zahlenwerte in Codes variabler Länge überführen, die aus Blöcken zu je acht Bit bestehen.[16] Bei *KST+DAG* werden zudem häufig auftretende Zahlenwerte durch eine vorberechnete Huffman-Codierung verarbeitet.[17] Eine vergleichbare Vorgehenswei-

[13] Vgl. beispielsweise Subramanian/Shankar (2006) S. 285-288
[14] Vgl. League/Eng (2007b), S. 276-278.
[15] Vgl. Harrusi/Averbuch/Yehudai (2006b), S. 409.
[16] Vgl. beispielsweise Böttcher/Hartel/Messinger (2010), S. 106
[17] Vgl. Böttcher/Hartel/Messinger (2009), S. 3.

se wird auch bei *XSDS* realisiert, indem für die Zahlen von 0 bis 24 eine Huffmann-Codierung vorberechnet wird. [18].

4.2.5 Kompression der Nutzdaten

Im Gegensatz zur Betrachtung der Strukturverarbeitung ist die Analyse der Konzepte der Nutzdatenkompression nicht auf die Verfahren mit Variantencodierung beschränkt, da jeder Ansatz der Nutzdatenkompression, der nicht an eine bestimmte Strukturverarbeitung gebunden ist, relevant ist. Aus diesem Grund werden zur Identifikation potentiell sinnvoller Techniken die in den Abschnitten 3.5 und 3.4 behandelten Verfahren berücksichtigt.
Der erste Schritt der Nutzdatenkompression ist die Organisation der Nutzdaten. Im Zuge dessen werden die Nutzdaten je nach Verfahren von der Struktur getrennt und in einen oder mehrere Container zerlegt. Zudem wird ihre Reihenfolge innerhalb der Container bei manchen Verfahren verändert. Im nächsten Schritt werden die Nutzdaten getrennt nach Container komprimiert. Die verschiedenen Konzepte, die für die soeben beschriebenen drei Aufgaben existieren, wurden bereits im Rahmen der Klassifikation der Kompressionsverfahren für XML-Dokumente in Abschnitt 3.2.3.4 auf Seite 46 beschrieben, weshalb hier auf diesen Abschnitt verwiesen wird. Eine Technik, die nicht in dem Abschnitt zur Klassifikation angesprochen wird, ist die Verarbeitung von Werten, die aus einer in der Schemainstanz definierten Liste von erlaubten Werten stammen. Hier wird bei vereinzelten schemabasierten Verfahren eine Wörterbuchcodierung angewandt und die Werte durch die Positionsnummer in der vordefinierten Liste ersetzt.[19]

4.3 Gestaltung der Funktionsbausteine

Dieser Abschnitt baut auf der in Kapitel 4.1 beschriebenen Struktur des parametrisierten Verfahrens und dem in Kapitel 4.2 erbrachten Überblick über die Konzepte bestehender Verfahren auf. Auf dieser Basis wird im Folgenden die Gestaltung der

[18] Vgl. Böttcher/Hartel/Messinger (2010), S. 106.
[19] Vgl. beispielsweise Böttcher/Hartel/Messinger (2011) S. 455.

Funktionsbausteine, die die Verfahrensschritte des parametrisierten Verfahrens abbilden, entwickelt. Dabei wird für die zuvor identifizierten Techniken analysiert, inwiefern diese im parametrisierten Verfahren zu berücksichtigen sind. Im Zuge dessen werden zum einen Konzepte ausgeschlossen, die nicht leistungsstark genug sind und zum anderen Konzepte kombiniert sowie modifiziert.

4.3.1 Modellierung der Schemadefinition

Eine Schemainstanz kann, wie unter Abschnitt 4.2.1 beschrieben, entweder baum- oder automatenbasiert modelliert werden. Es gibt keine Anhaltspunkte, weshalb eine der beiden Arten der anderen überlegen ist, weshalb im parametrisierten Verfahren beide berücksichtigt werden. Wie im Folgenden aber gezeigt wird, ist pro Modellierungsart nur eine Vorgehensweise sinnvoll. Aus diesem Grund besitzt dieser Funktionsbaustein nur einen Parameter, der steuert, welche der beiden **Modellierungsarten** zu verwenden ist.

Für die **baumbasierte Modellierung** wird dem Ansatz von *XSDS* gefolgt[20], da, wie im Folgenden erklärt wird, der Ansatz von *XCQ* auf Modellierungsebene identisch ist und die Ansätze der DTD-basierten Verfahren nachteilig sind. Zudem werden von diesem alle Bestandteile von XSD vollständig berücksichtigt, wohingegen die anderen Ansätze auf diese Schemasprache erweitert werden müssten.

Der Ansatz von *XCQ* unterscheidet sich zu *XSDS* in dem Sinn, dass dort Bausteine, die in *XSDS* getrennt modelliert werden, in einem Knoten dargestellt werden. Beispielsweise wird ein XML-Element und dessen Häufigkeitsinformation in einem Knoten repräsentiert. Im Ergebnis unterscheiden sich die beiden Ansätze jedoch nicht.

Die Modellierungsvariante der DTD-basierten Verfahren über binäre Bäume ist weniger leistungsfähig als die von *XSDS*. Dies ist damit zu begründen, dass es von Vorteil ist, alternative Elemente nicht über binäre Teilbäume abzubilden, sondern zusammengefasst als Kinder eines Knotens[21], wie dies auch bei *XSDS* der Fall ist.

[20] Vgl. Böttcher/Hartel/Messinger (2011), S. 454 f.
[21] Vgl. Böttcher/Hartel/Messinger (2009), S. 3 f.

Die **automatenbasierte Modellierung** erfolgt im parametrisierten Verfahren über den Ansatz, der in den DTD-basierten Verfahren[22] verwendet wird. Dieser Ansatz ist mächtig genug, XSD abzubilden, weshalb es nicht nötig ist, ein mächtigeres Konzept, wie es beispielsweise von *rngzip* zur Verarbeitung von RELAX NG verfolgt wird, zu adaptieren.

4.3.2 Abbildung der Struktur eines XML-Dokuments

Der Funktionsbaustein zur Abbildung der Struktur eines XML-Dokuments erfordert keine Parameter. Das Resultat dieses Bausteins ist ein Baum, der die Struktur eines XML-Dokuments repräsentiert und die bei der Abbildung der Struktur erzeugten Daten in seinen Knoten speichert. Jeder Knoten enthält dabei einen Satz von Daten, der in Verbindung mit dem Inhaltsmodell des XML-Elements, für das der Knoten steht, definiert, welche Kindelemente der Knoten besitzt. Sind die Kindelemente nicht variabel, enthält der Knoten keine Daten.
Bei der automatenbasierten Modellierung enthält ein Knoten die Folge von Ausgaben, die bei der Abarbeitung des DEAs, der das Inhaltsmodell des jeweiligen Elements abbildet, erzeugt werden. Wie in Abschnitt 4.2.2 dargelegt, ist die Ausgabe bei bestehenden Verfahren entweder der Name des gerade verarbeiteten XML-Elements oder ein Symbol, das für den gewählten Zustandsübergang steht. Hier wird nur die zweite Variante verwendet, da ein Symbol kürzer codiert werden kann, als der Name des Elements, der auch Redundanzen auf textueller Ebene enthalten kann.
Die Abbildung von Zustandsübergängen über Symbole aus einem spezifischen Zeichenvorrat findet sich bei der baumbasierten Modellierung zur Identifizierung der konkreten Auswahl aus alternativen Bausteinen. Da alle bestehenden Verfahren diese Art der Abbildung nutzen[23], wird sie auch im hier vorgestellten parametrisierten Verfahren verwendet.

[22] Vgl. beispielsweise Subramanian/Shankar (2006) S. 286 f.
[23] Siehe dazu Abschnitt 4.2.2.

Die umkehrbar eindeutige Zuordnung von Zeichen eines Zeichenvorrats zu Zustandsübergängen bzw. möglichen Alternativen einer Auswahl ist beispielsweise eine reproduzierbare Nummerierung der Zustandsübergänge bzw. Alternativen. Ein Zeichenvorrat kann in diesem Zusammenhang durch seine Kardinalität eindeutig bestimmt werden. Im Rahmen der Strukturabbildung des parametrisierten Verfahrens wird daher die Nummer des Zustandsübergangs bzw. die der Alternative und zudem die Anzahl der jeweils möglichen Ausprägungen gespeichert. Diese Informationen sind nötig, um im Funktionsbaustein zur Strukturcodierung verschiedene Varianten der binären Codierung zu realisieren.
Bei der Abbildung von Wiederholungen wird die konkrete Anzahl an Wiederholungen des jeweiligen Elements bzw. der jeweiligen Gruppe ermittelt, was der Vorgehensweise aller untersuchten Verfahren entspricht. Für *XCQ* ist dies nicht direkt erkennbar, doch das Speichern einer binären Markierung pro Wiederholung[24] ist gleichbedeutend mit der Codierung der Anzahl über den Zählcode. Bei *PSXC* wird letztlich nicht die ermittelte Anzahl, sondern der um die minimal erlaubte Anzahl reduzierte Wert gespeichert, wie dies auch bei *XSDS* der Fall ist.

Beispielhafte Veranschaulichung der Strukturabbildung

Im Folgenden wird die Strukturabbildung für beide Modellierungsarten beispielhaft veranschaulicht. Inspiriert von dem Beispiel in Subramanian/Shankar (2006) wird ein Ausschnitt aus einem vereinfachten Adressbuch verwendet.[25] Die Gestaltung des betrachteten Adresseintrags wird als in XSD formuliertes Inhaltsmodell in Abbildung 4.5 dargestellt. Ein Adressbucheintrag besteht aus einem Namensteil, der entweder einen einzelnen Namen oder einen Vornamen und Nachnamen enthält, wobei der Vorname optional ist. Danach folgen mindestens eine E-Mail-Adresse und ein optionaler Block mit Notizen, dessen Aufbau nicht mehr Teil des Beispiels ist. Die Modellierung des Inhaltsmodells als Baum ist in Abbildung 4.6 und die als Automat in Abbildung 4.7 dargestellt.

[24] Vgl. Ng u. a.(2006), S. 430.
[25] Vgl. Subramanian/Shankar (2006), S. 286 f.

```
<xsd:element name='Adresseintrag'>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name='Name' type='xsd:string' />
        <xsd:sequence>
          <xsd:element name='Vorname' type='xsd:string' minOccurs='0' />
          <xsd:element name='Nachname' type='xsd:string' />
        </xsd:sequence>
      </xsd:choice>
      <xsd:element name='Email' type='xsd:string' maxOccurs='unbounded' />
      <xsd:element name='Notizen' type='tns:notes' minOccurs='0' />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Abbildung 4.5: Beispielhaftes Inhaltsmodell eines Adresseintrags in XSD (Eigene Darstellung)

Die Strukturabbildung der in Abbildungen 4.8 und 4.9 enthaltenen Beispiele für Adresseinträge wird im Folgenden für beide Modellierungsarten beschrieben.

Baumbasierte Modellierung

Die grundsätzliche Vorgehensweise wird anhand des ersten Beispieladresseintrags erklärt. Wie Abbildung 4.6 zeigt, besteht die baumbasierte Darstellung des Inhaltsmodells des Adresseintrags in der ersten Ebene aus einer Sequenz. Bei dieser sind keine Daten zu erstellen, jedoch alle Bestandteile abzuarbeiten. Der erste Baustein ist ein Auswahlknoten, der genau einmal vorkommen darf, weshalb keine Häufigkeitsinformation zu speichern ist. Jedoch ist zu ermitteln, welcher der alternativen Bausteine zu wählen ist. Da der erste Beispieladresseintrag einen einzelnen Namen verwendet, ist die erste Alternative zu wählen und daher eine `1` zu speichern. Damit ist das erste Kindelement des Adressbucheintrags abgebildet. Die anderen Alternativen der Gruppe müssen nicht weiter verarbeitet werden.

Es folgt die Verarbeitung des zweiten Bausteins der Sequenz auf oberster Modellebene. Hier ist zu erfassen, wie viele E-Mail-Adressen konkret vorliegen. Im ersten Beispiel ist dies eine einzelne E-Mail-Adresse. Folglich wird eine `0` gespeichert, da die konkret vorliegende Anzahl um die minimal erlaubte Anzahl reduziert wird. Da-

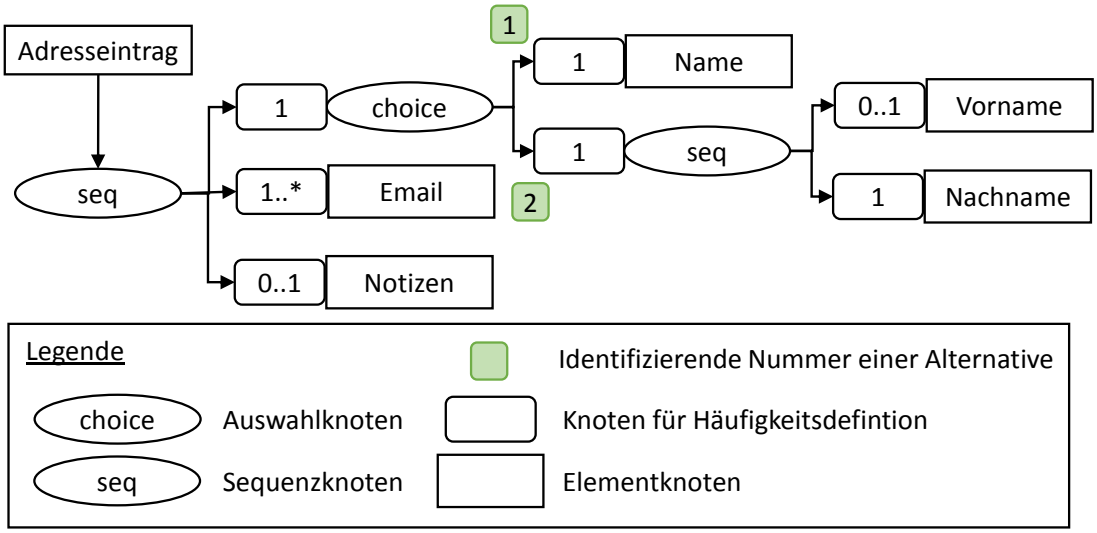

Abbildung 4.6: Modellierung des Beispiels als Baum (Eigene Darstellung)

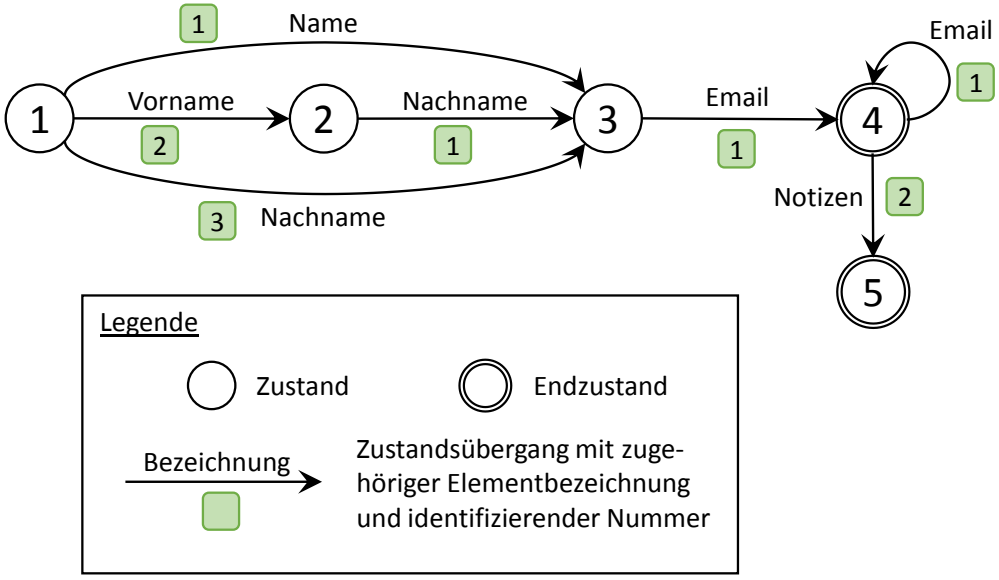

Abbildung 4.7: Modellierung des Beispiels als Automat (Eigene Darstellung)

```
<Adresseintrag>
   <Name>Webdesign Parallax</Name>
   <Email>info@webdesign-parallax.de</Email>
   <Notizen>
      <Notizeintrag datum='2016-02-15'>Angebot einholen</Notizeintrag>
   </Notizen>
</Adresseintrag>
```

Abbildung 4.8: Adresseintrag Beispiel 1 (Eigene Darstellung)

```
<Adresseintrag>
   <Vorname>Michael</Vorname>
   <Name>Muster</Name>
   <Email>michi-muster@mail.de</Email>
   <Email>michael.muster@hns-consulting.de</Email>
<Adresseintrag>
```

Abbildung 4.9: Adresseintrag Beispiel 2 (Eigene Darstellung)

mit ist die Verarbeitung des zweiten Bausteins der Sequenz und gleichzeitig die des zweiten Kindelements abgeschlossen. Es folgt die Behandlung des dritten Bausteins der Sequenz, welcher das optionale Notizen-Element darstellt. Da dieses Element im Beispiel vorhanden ist, wird eine `1` gespeichert. Die Abbildung des Inhalts des Notizenblocks wird in diesem Beispiel nicht betrachtet, da immer ein Inhaltsmodell am Stück verarbeitet wird und erst anschließend die Abbildung der Kindelemente in Bezug auf deren Inhaltsmodelle erfolgt.
Wird auch der zweite Beispieladresseintrag entsprechend abgebildet, ergeben sich die folgenden Daten:

- Abbildungsdaten des ersten Beispieladressbucheintrags: `1 0 1`
- Abbildungsdaten des zweiten Beispieladressbucheintrags: `2 1 1 0`

Automatenbasierte Modellierung

Zur Erläuterung der Vorgehensweise der Strukturabbildung bei der automatenbasierten Modellierung wird der zweite Beispieladresseintrag verwendet. Die Bearbeitung beginnt im Startzustand des Automaten, der das Inhaltsmodell des Adresseintrags repräsentiert (siehe Abbildung 4.7). Da im zweiten Beispiel Vor- und Nachnamen einzeln vorhanden sind, ist der zweite Zustandsübergang des Startzustandes

zu wählen und dementsprechend eine `2` zu speichern. Danach befindet sich der Automat im Zustand mit der Nummer zwei, indem verpflichtend der Nachname einzulesen ist. Da der Übergang in Zustand drei ohne Alternative ist, muss dessen identifizierende Nummer nicht gespeichert werden.
Im dritten Zustand wird verpflichtend die erste E-Mail-Adresse im Adresseintrag verarbeitet und zwangsläufig in den Zustand vier gewechselt, weswegen hier keine Daten zu speichern sind. Da der zweite Beispieladresseintrag eine zweite E-Mail-Adresse enthält, muss wiederum in den vierten Zustand übergegangen und damit eine `1` gespeichert werden. Mit diesem Schritt sind alle Kindelemente des Adresseintrags abgebildet und der Automat befindet sich in einem potentiellen Endzustand. Da dieser jedoch auch wieder verlassen werden könnte und nicht alternativlos als Endzustand fungiert, muss gekennzeichnet werden, dass er bei dieser Abbildung einen Endzustand darstellt. Zu diesem Zweck wird eine Zahl gespeichert, die um eins größer als die Anzahl der Zustandsübergänge ist. Im konkret vorliegenden Fall ist dies die Zahl `3`.
In analoger Weise kann auch der erste Beispieladresseintrag verarbeitet werden. Zusammenfassend ergeben sich für beide Beispiele die folgenden Daten:

- Abbildungsdaten des ersten Beispieladressbucheintrags: `1 2`
- Abbildungsdaten des zweiten Beispieladressbucheintrags: `2 1 3`

4.3.3 Reduzierung von Redundanz in der Struktur

Zur Reduzierung der Redundanz in der Struktur werden beide in Abschnitt 4.2.3 identifizierten Konzepte – die Kürzung redundanter Strukturteile durch Zeiger und die Delegation der Beseitigung der Strukturredundanz an einen Universalkompressor – berücksichtigt. Ihre Integration in das Verfahren erfolgt nicht im Sinne zweier Alternativen für eine Aufgabe, sondern ist so gestaltet, dass auch ihre Kombination ermöglicht wird. Dazu kann über einen Parameter die **Behandlung der Redundanz in der Struktur** aktiviert werden. Ist sie aktiviert, wird die Strukturabbildung vor ihrer binären Codierung entsprechend verarbeitet. Die Delegation

der Behandlung der strukturellen Redundanz an nachgelagerte Universalkompressoren wird immer dann durchgeführt, wenn zur Codierung der Strukturabbildung Universalkompressoren als Backend eingesetzt werden.[26]

Ist die Behandlung der Redundanz in der Struktur aktiviert, wird die Redundanz von einer leicht modifizierten Version der Vorgehensweise bei *KST+DAG* entfernt.[27] Um die Vorgehensweise für beide Modellierungsarten verwenden zu können, wird als Baum für die Redundanzermittlung der in Abschnitt 4.3.2 beschriebene Strukturbaum verwendet, da die baumbasierte Abbildung der Struktur nur bei der baumbasierten Modellierung verfügbar ist. Ein Vorteil dieser Herangehensweise ist, dass jedes Element bei der Redundanzverarbeitung berücksichtigt werden kann und nicht nur solche, die nicht verpflichtend sind, wie dies bei *KST+DAG* der Fall ist. Zur Erzeugung der Signatur, die zur Identifizierung von Redundanzen verwendet wird, werden die Daten aller Knoten des jeweiligen Teilbaums verknüpft und daraus ein Hashwert berechnet. Wie auch bei *KST+DAG* wird ein redundanter Teilbaum nur dann durch einen Zeiger ersetzt, wenn dieser weniger Speicherplatz benötigt, als dies die Daten der Knoten des jeweiligen Teilbaums tun.

Bei *KST+DAG* erhalten alle Knoten eine Markierung, die angibt, ob der Knoten durch einen Zeiger ersetzt wurde. Das alternative Konzept der Auflistung der Knoten, die durch Zeiger ersetzt wurden, wird dort als nachteilig identifiziert. Da dieser Nachteil nicht pauschal gültig ist, wird im hier vorgestellten Verfahren dynamisch bei jedem Kompressionsvorgang berechnet, welche der beiden Alternativen von Vorteil ist und die entsprechend bessere verwendet.

Für die Variante der Knotenmarkierung ist ein Bit pro Knoten nötig, weshalb sich der nötige Speicherplatz direkt aus der Anzahl der Knoten ergibt. Diesem Wert wird gegenübergestellt, wie viel Speicherplatz eine Liste von Nummern belegt, die die verknüpften Knoten identifiziert. Der Speicherplatz für die Zeigerziele, die als Anzahl der Knoten zwischen Quelle und Ziel gespeichert werden, ist in beiden Ansätzen identisch.

[26] Die Codierung der Strukturabbildung und deren Parameter wird in Abschnitt 4.3.4 behandelt.
[27] Eine kurze Erklärung dieser Vorgehensweise findet sich in Abschnitt 4.2.3.

Da beide Konzepte zur Behandlung von Redundanz in der Struktur getrennt aktiviert werden können, ist es mit dem parametrisierten Verfahren möglich, sowohl den alternativen, wie auch den kombinierten Einsatz der Konzepte zu untersuchen. Dabei ist zu berücksichtigen, dass das Ersetzen redundanter Strukturteile durch Zeiger negative Auswirkungen auf die Kompressionsleistung eines nachgelagerten Kompressors haben kann und die Struktur ohne Zeiger unter Umständen besser komprimiert werden könnte.

4.3.4 Codierung der Strukturabbildung

Der Funktionsbaustein der binären Codierung der Strukturabbildung besitzt drei Parameter: **Art der Codierung**, **Backendkompressor** und **Art der Häufigkeitsabbildung**. Die **Art der Codierung** hat die Einstellungsmöglichkeiten **statisch**, **dynamisch** und **gemischt**. Erstere ergeben sich aus den beiden unterschiedlichen Ansätzen, die Struktur entweder über statische oder über dynamische Codierungsverfahren zu komprimieren.[28] Bei der gemischten Codierungsart werden binäre Zeichenvorräte statisch und alle weiteren dynamisch codiert. Dieser Ansatz ist damit zu begründen, dass die statische Codierung von binären Zeichenvorräten in einem Bit erfolgt und Universalkompressoren für die entsprechenden Symbole nur unter günstigen Bedingungen eine durchschnittliche Codelänge von weniger als einem Bit erreichen.

Gestaltung der statischen Codierung

Die binäre Codierung von Zustandsübergängen und die Auswahl aus alternativen Bausteinen erfolgt bei Verfahren mit statischer Codierung immer über den Beta-Code, weshalb dieser auch im parametrisierten Verfahren verwendet wird. Die Nummer des Zustandsübergangs bzw. der gewählten Alternative sowie die für den Beta-Code nötige Anzahl ihrer Ausprägungsmöglichkeiten wurde bereits bei der Strukturabbildung ermittelt und gespeichert.

[28] Siehe dazu Abschnitt 4.2.4.

Zur statischen Codierung von Anzahlen wird bei dem parametrisierten Verfahren der Fibonacci-Code verwendet. Er wird als universaler Ersatz für die Kombination aus vorberechneter Huffmann-Codierung und bytebasierter Codierung variabler Länge, wie sie bei *XSDS* und *KST+DAG* zur Anwendung kommt, eingesetzt. Die beiden anderen Codierungen sind im Vergleich zum Fibonacci-Code weniger kompakt: Da bei der Verwendung des Beta-Codes die höchste Anzahl die Länge aller Codes bestimmt, wird hier unnötig Speicherplatz belegt. Der Zählcode ist nur bis zur Zahl 3 um ein Bit kürzer als der Fibonacci-Code und wird ab der Zahl 6 wesentlich länger.

Der Einsatz von Universalkompressoren

Im Falle der dynamischen oder gemischten Codierungsart regelt der zweite Parameter **welcher Universalkompressor** als **Backendkompressor** zur Codierung der Struktur verwendet wird. Bei der Abbildung der Struktur werden alle Daten in Form von Zahlen gespeichert. Diese werden bei der Nutzung eines nachgelagerten Universalkompressors in eine bytebasierte Darstellung variabler Länge überführt, wie sie in den Grundlagen in Abschnitt 2.4.2 zur Codierung von Zahlen beschrieben ist.

Alternative Häufigkeitsabbildung

Bei beiden Arten der Modellierung ist es möglich, bestimmte Häufigkeiten alternativ abzubilden. So kann bei der baumbasierten Modellierung die Abbildung von Sequenzen dahingehend erweitert werden, dass gespeichert wird, welche Bestandteile in der Sequenz häufiger als in ihrer minimal erlaubten Anzahl auftreten. Auf diese Weise wird es überflüssig, die konkrete Häufigkeit der Bausteine zu speichern, die in ihrer minimalen Anzahl auftreten. Zur Kennzeichnung, für welche Bausteine die konkret auftretende Anzahl zu speichern ist, wird eine Liste von Symbolen aus einem binären Zeichenvorrat angelegt. Für jeden Baustein der Sequenz, dessen Anzahl nicht ohnehin fixiert ist, enthält die Liste ein Symbol, das angibt, ob der jeweilige Baustein in seiner minimalen Anzahl oder häufiger auftritt. Wenn ein

Baustein nur in zwei verschiedenen Anzahlen auftreten darf, ergibt sich die konkret auftretende Häufigkeit aus der Information in der Sequenzliste, weshalb die Anzahl nicht zusätzlich gespeichert werden muss.
Bei der statischen und gemischten Codierung wird die Liste als Bitfolge mit einem Bit pro Symbol gespeichert. Bei der dynamischen Codierung wird die Liste als Bytefolge von je einem Byte an den nachgelagerten Universalkompressor übergeben. Für den statischen und gemischten Fall kann berechnet werden, ob die soeben beschriebene alternative Häufigkeitsabbildung zu einer kompakteren Strukturcodierung führt. Auf dieser Basis wird bei jedem Kompressionsvorgang individuell ermittelt, welche Art der Häufigkeitsabbildung zu wählen ist. Für den dynamischen Fall kann diese Entscheidung nicht eindeutig getroffen werden, da es von dem jeweils nachgelagerten Universalkompressor abhängig ist, welche Art der Häufigkeitsabbildung vorteilhaft ist. Daher wird dort die Berechnung für den statischen Fall als Heuristik verwendet.
Bei der Berechnung wird ermittelt, wie viel zusätzlicher Speicherplatz für die Listen der Sequenzen benötigt wird und wie viel Speicherplatz dadurch wegfällt, dass nur die Anzahl für Bausteine codiert werden muss, die nicht in ihrer minimalen Anzahl auftreten. Der Speicherplatz für die Sequenzlisten ergibt sich aus der Anzahl aller Bausteine, die in abzubildenden Sequenzen auftreten. Die Einsparungen im Rahmen der Häufigkeitscodierung der Bausteine berechnen sich als Summe des nötigen Speicherplatzes für die Anzahlen der Bausteine, die nicht häufiger als ihre minimal erlaubte Anzahl auftreten.
Bei der automatenbasierten Modellierung werden sich wiederholende Bausteine als Folge der entsprechenden Zustandsübergänge abgebildet und damit nicht gesondert behandelt. Im Rahmen dieser Wiederholungen treten Zustandsübergänge auf, deren Quelle und Ziel der gleiche Zustand sind. Folgen dieser Übergänge können im Sinn einer Lauflängencodierung alternativ abgebildet werden. Dazu wird das Symbol, das den Zustandsübergang identifiziert, gespeichert und im Anschluss daran die Häufigkeit, wie oft dieser Zustandsübergang in Folge zu wählen ist. Bei dieser Art der Häufigkeitsabbildung wird für jeden gewählten Zustandsübergang, dessen

Quelle und Ziel identisch ist, immer die Anzahl seiner Wiederholungen gespeichert. Ob die Abbildung der Zustandsübergänge auf die normale oder die alternative Weise erfolgt, wird pro Kompressionsvorgang individuell entschieden. Die entsprechende Berechnung kann nur für die statische Codierung durchgeführt werden. Für die dynamische und gemischte Strukturcodierung ist der Nutzen von dem eingesetzten Universalkompressor abhängig, weshalb die Berechnung für den statischen Fall hier als Heuristik verwendet wird.
Zur Ermittlung, ob die alternative Häufigkeitsabbildung bei der automatenbasierten Modellierung von Vorteil ist, wird berechnet, wie viel zusätzlicher Speicherplatz die Codierung der zusätzlichen Anzahlen benötigt und wie viel Speicherplatz dadurch eingespart wird, dass die Symbole für die alternativ abgebildeten Zustandsübergänge nur einmal codiert werden müssen.
Die Art der Häufigkeitsabbildung wird bei der baumbasierten Modellierung mit dynamischer Strukturcodierung und der automatenbasierten Modellierung mit gemischter und dynamischer Codierung nur heuristisch bestimmt. Aus diesem Grund kann die alternative Häufigkeitsabbildung in Verbindung mit nachgelagerten Universalkompressoren von Nachteil sein, obwohl sie im statischen Fall von Vorteil ist. Um diesen Effekt untersuchen zu können, kann die **alternative Häufigkeitsabbildung** über den dritten Parameter des Strukturcodierungsbausteins aktiviert werden. Die Aktivierung bedeutet, dass nach den oben beschriebenen Berechnungen individuell entschieden wird, wie Häufigkeiten abgebildet werden. Eine Deaktivierung führt dazu, dass die alternative Häufigkeitsabbildung nicht verwendet wird.

4.3.5 Kompression der Nutzdaten

Die Nutzdatenkompression besteht aus zwei Funktionsbausteinen, deren Verhalten jeweils über zwei Parameter gesteuert werden kann. Die Gestaltung der Bausteine und deren Konfigurationsmöglichkeiten wird im Folgenden erläutert.

Organisation der Nutzdaten

Die verschiedenen in Abschnitt 3.2.3.4 beschriebenen Vorgehensweisen **Nutzdaten in Container zu verteilen**, werden im Folgenden bewertet und auf dieser Basis werden eine Reihe von untersuchungsrelevanten Strategien erstellt. Der Ansatz, Struktur- und Nutzdaten nicht zu trennen und mit der gleichen Technik zu komprimieren, ist nicht mit dem der Variantencodierung der Struktur kombinierbar und ist deshalb hier nicht relevant.

Nach welcher Regel Nutzdaten idealerweise in Container aufzuteilen sind, kann nicht pauschal entschieden werden, da es signifikant von der Beschaffenheit der Daten abhängt. Eine weitere Zerlegung der Daten ist solange von Vorteil, bis ähnliche Nutzdaten in verschiedenen Containern zerlegt werden und die Kompression pro Container sich dadurch verschlechtert. Aufgrund der geringen Größe von Geschäftsdokumenten wird es hier als nicht sinnvoll angesehen, eine feinere Unterteilung als die nach Namen zu berücksichtigen.

Die Bandbreite der Aufteilung der Nutzdaten reicht somit von einem Container für alle Nutzdaten, bis hin zur Aufteilung der Nutzdaten in Container anhand des Namens des datentragenden Bausteins. Um den in diesem Bereich liegenden, individuellen Zerlegungskonzepten Rechnung zu tragen, die aufgrund ihrer Vielfälligkeit hier nicht vollständig berücksichtigt werden können, wird das Typenkonzept von XSD verwendet, um weitere Zerlegungsstufen zu realisieren.

Abbildung 4.10 zeigt die Hierarchie der Zerlegungs- bzw. Aggregationsstufen, die aus der Verwertung von Typinformationen resultiert. Die Werte der Elemente bzw. Attribute (in der Abbildung grün dargestellt) können in erster Instanz entsprechend ihres direkten bzw. speziellsten Typs zusammengefasst werden (unterste gelbe Reihe). Diese Einteilung ist parallel zur Gruppierung nach Namen (blau) zu sehen, da zwei Elemente mit gleichem Namen unterschiedliche Typen haben können und Elemente mit unterschiedlichem Namen gleiche Typen.

Im Gegensatz dazu besitzt jeder Typ einen eindeutigen Obertyp. Da jeder Wert eines Elements oder Attributs ein einfacher Datentyp ist, kann er über eine Kette von Generalisierungen auf einen der in XSD eingebauten primitiven Typen, wie sie

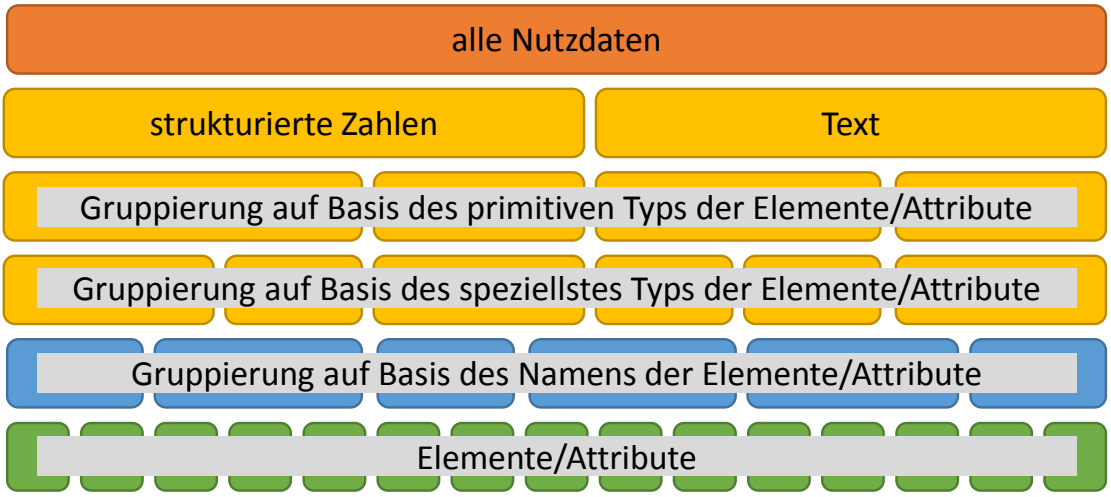

Abbildung 4.10: Hierarchische Gruppierung der Nutzdaten (Eigene Darstellung)

in Abbildung 4.11 dargestellt sind, zurückgeführt werden. Anhand dieser primitiven Typen können die Gruppen der speziellsten Typen weiter gebündelt werden (mittlere gelbe Reihe).

Bei den primitiven Typen können des Weiteren zwei Gruppen unterschieden werden (oberste gelbe Reihe). Eine Gruppe beinhaltet Typen, die strukturierte Zahlen darstellen. Dazu gehören beispielsweise ganze Zahlen, Dezimalzahlen sowie Datums- und Zeitangabe. Alle anderen Typen werden in einer zweiten Gruppe vereint und verallgemeinert als Text weiterverarbeitet. Diese Zusammenfassung als textuelle Daten ist im Allgemeinen etwas ungenau, aber im vorliegenden Umfeld ausreichend, da beispielsweise Daten vom primitiven Typ `base64Binary`, in dem mitunter Bilder codiert werden, nicht zu erwarten sind. Letztlich werden alle Nutzdatenwerte in einem Container zusammengefasst (orange Ebene).

Neben der Aufteilung der Nutzdaten in Container wird bei manchen bestehenden Verfahren auch die **Reihenfolge der Nutzdaten** verändert.[29] Die meisten Verfahren legen die Nutzdaten in Dokumentenreihenfolge in den Containern ab. Die Alternative dazu ist die Sortierung nach dem Pfad der Daten im XML-Dokument. Dieser Ansatz wird von bestehenden Verfahren nur dann verwendet, wenn die Nutzdaten nicht in mehrere Container zerlegt werden. Aufgrund der zusätzlichen Ag-

[29] Siehe dazu Abschnitt 3.2.3.4.

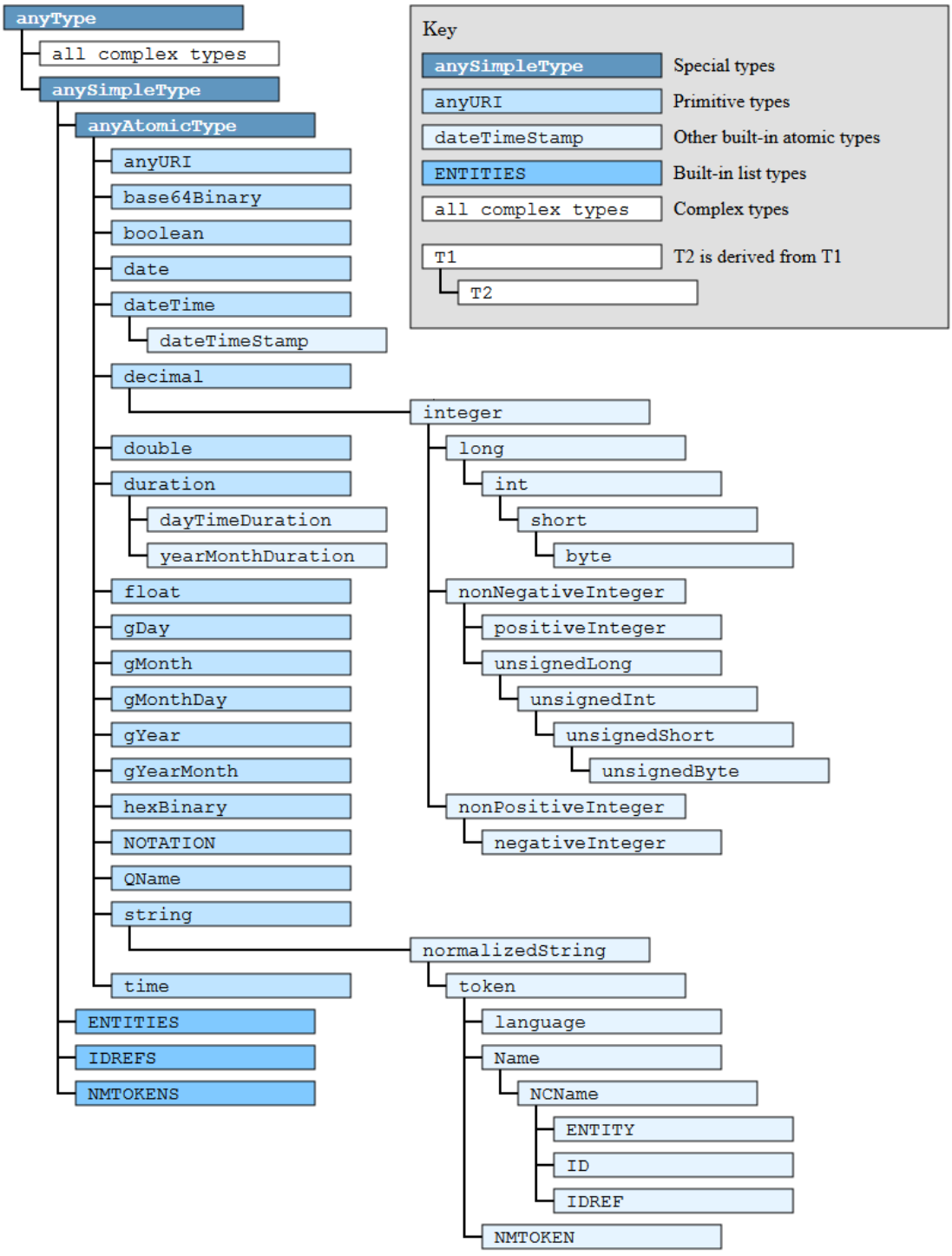

Abbildung 4.11: Hierarchie der vordefinierten Datentypen bei XSD (Quelle: Peterson u. a.(2012))

gregationsstufen kann es aber sinnvoll sein, die Daten in jeder Stufe zu sortieren. Zur Bildung einer Reihenfolge wird daher ein Schlüssel verwendet, der die gesamte Aggregationshierarchie abbildet. Damit werden in jeder Stufe Daten mit ähnlichen Typen gruppiert.

Codierung der Nutzdaten

Das parametrisierte Verfahren ist so gestaltet, dass die erstellten Nutzdatencontainer mit **beliebigen Universalkompressoren** verarbeitet werden können. Kompressionsverfahren, die an eine spezielle Strukturverarbeitung geknüpft oder verfahrensspezifisch gestaltet sind, werden hier nicht berücksichtigt, da keine Indizien dafür gefunden werden konnten, dass ihre Leistungsfähigkeit den erheblichen Aufwand ihrer Integration rechtfertigt.

Bei *XSDS* werden Zahlenwerte getrennt verarbeitet und über bytebasierte Codes variabler Länge statisch codiert.[30] Dieser Ansatz wird hier als optionales Konzept, das über einen Parameter aktiviert werden kann, berücksichtigt. Die in Abschnitt 4.2.5 angesprochene Ersetzung von Werten aus einer Werteliste durch deren Positionsnummern wird als festes Konzept übernommen, da in jedem Fall zu erwarten ist, dass sich dieser Ansatz positiv auswirkt. Da Nutzdaten im Zuge dessen durch Zahlen ersetzt werden, ändert sich ihr Datentyp, weshalb dieser Schritt vor der Containeraufteilung positioniert wird.

Zusammenfassung

Für die beiden Funktionsbausteine der Nutzdatenkompression ergeben sich je zwei Parameter. Im Rahmen der Organisation der Nutzdaten kann über den Parameter **Sortierung** die oben beschriebene, **spezielle Sortierung** der Nutzdaten aktiviert werden. Wird die Sortierung nicht aktiviert, werden die Nutzdaten in **Dokumentenreihenfolge** belassen. Der zweite Parameter steuert die **Containerstrategie** und kann folgende Ausprägungen annehmen:

[30] Vgl. Böttcher/Hartel/Messinger (2011), S. 457.

1 - alle Nutzdaten in einen Container

2 - Unterteilung in Text und strukturierte Zahlen

3 - Unterteilung nach dem primitiven Datentyp

4 - Unterteilung nach dem speziellsten Datentyp

5 - Unterteilung nach dem Namen des datentragenden Bausteins

Bei der Codierung der Nutzdaten kann über den Parameter **gemischte Codierung** aktiviert werden, dass **Zahlenwerte separat statisch codiert** und nicht an **nachgelagerte Universalkompressoren** weitergegeben werden. Der letzte Parameter steuert welcher **Universalkompressor** als **Backendkompressor** für die Verarbeitung der Nutzdatencontainer verwendet wird.

4.4 Prototypische Implementierung

Zur Konzeption von *PSXC* wurde ein objektorientiertes Design gewählt und in Java unter Verwendung der Java Plattform, Standard Edition (Java SE) in der Version 8 als Prototyp implementiert.

4.4.1 Die Verarbeitungsprozesse und deren Komponenten

Im Folgenden werden zunächst der Kompressions- und anschließend der Dekompressionsprozess jeweils aus konzeptioneller Implementierungssicht beschrieben.

4.4.1.1 Der Kompressionsprozess

Abbildung 4.12 stellt den Kompressionsprozess schematisch dar und dient als Grundlage für die weiteren Ausführungen.

Einlesen des XML-Dokuments

Im ersten Schritt des Prozesses ist das XML-Dokument einzulesen, wofür die Java API for XML Processing (JAXP) verwendet wird, die Teil der verwendeten Java Plattform ist. Intern stammt deren Implementierung aus der Apache Xerces2-J

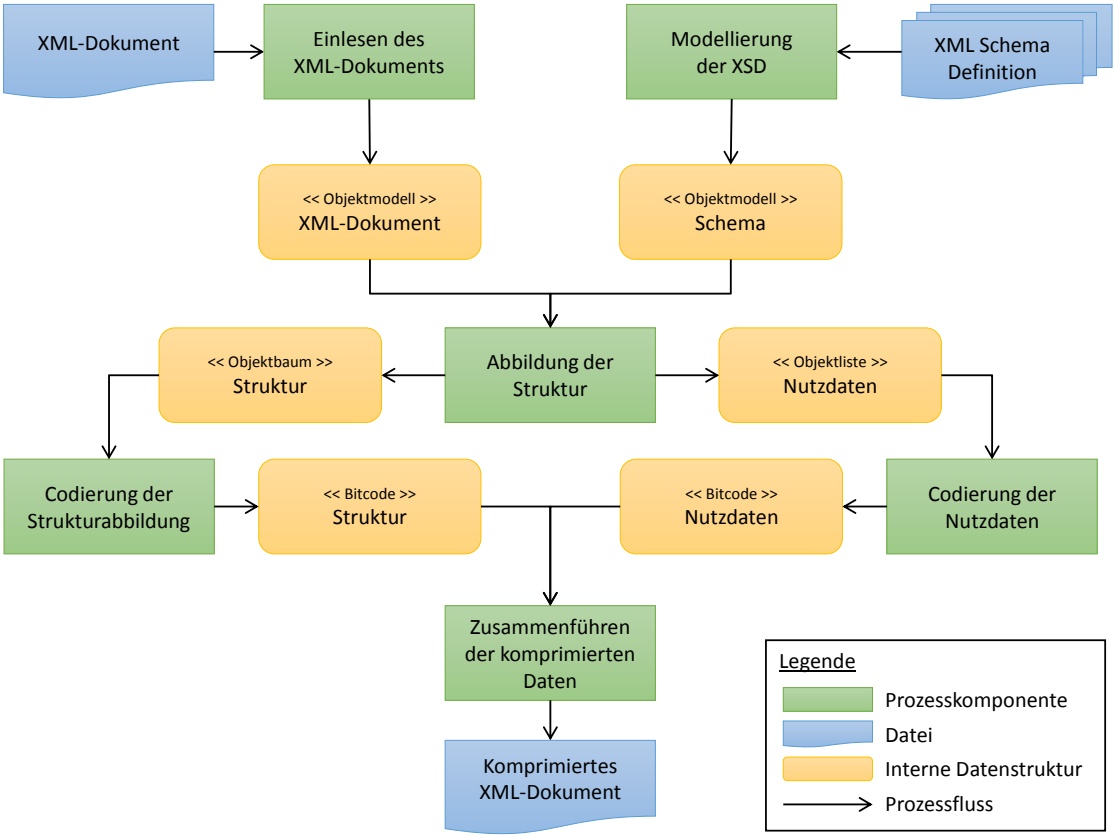

Abbildung 4.12: Prozess der Kompression eines XML-Dokuments (Eigene Darstellung)

Bibliothek.[31] Sie bietet vier grundlegende Schnittstellen, um ein XML-Dokument anzusprechen. Den hier vorliegenden Bedarf erfüllt am besten der Zugriff über das Document Object Model (DOM)[32]. Im Gegensatz zu den anderen Schnittstellen kann diese direkt ohne zusätzliche Verarbeitungslogik verwendet werden. Als Nachteil ist der höhere Speicherverbrauch zu nennen, der jedoch aufgrund der geringen Größe der XML-basierten Geschäftsdokumente zu vernachlässigen ist. Im Zuge der internen Abbildung des XML-Dokuments wird auch die zu verwendende XML-Schema-Definition (XSD) bestimmt.

Modellierung der XML-Schema-Definition

Entsprechend der Ausführungen in Abschnitt 4.3.1 ist es nötig, eine Schemainstanz sowohl als Baum als auch automatenbasiert abbilden zu können. Die verwendete

[31] Details zur Apache Xerces2-J Bibliothek finden sich unter `http://xerces.apache.org/xerces2-j`.
[32] Details zum DOM finden sich unter `https://www.w3.org/DOM`.

Java Plattform stellt als Teil der integrierten Apache Xerces2-J Bibliothek bereits im Standard Funktionen zur Verarbeitung einer XML-Schema-Definition (XSD) zur Verfügung. Mit dieser ist es möglich, eine XSD direkt im Sinne eines Baumes zu verarbeiten. Im Gegensatz dazu fehlt die Erzeugung von Automaten für die Inhaltsmodelle der XML-Elemente.
Aus diesem Grund wird die Bibliothek der Model Development Tools (MDT) des Eclipse Modeling Projekts eingesetzt. Die MDT XSD Bibliothek modelliert eine XSD sowohl im Sinne eines Baumes als auch als Sammlung von deterministischen endlichen Automaten (DEAs) für die einzelnen Inhaltsmodelle der definierten Elemente.[33] Da die MDT XSD Bibliothek immer beide Ansätze parallel bereitstellt, ist die Einstellung des Parameters, der die Modellierungsart der Schemainstanz steuert, an dieser Stelle ohne Auswirkung.

Abbildung der Struktur

Nachdem das XML-Dokument und die zugehörige XSD eingelesen wurden, kann die Abbildung der Struktur durchgeführt werden. Die Struktur wird als Baum von Objekten modelliert, von denen jedes für ein XML-Element steht. In die Objekte werden einerseits die bei der Abbildung erzeugten Daten und andererseits die Metadaten der XSD, die zur binären Codierung dieser Daten nötig sind, gespeichert. Per Parametereinstellung wird in dieser Komponente geregelt, welche der beiden Arten der Schemamodellierung zur Abbildung verwendet wird.
Bei der automatenbasierten Abbildung wird bei jedem Zustandsübergang die identifizierende Nummer des Übergangs und die Anzahl der möglichen Alternativen gespeichert. Bei der baumbasierten Abbildung unterscheiden sich die in den Objekten des Strukturbaums gespeicherten Daten pro Knotenart der Schemamodellierung:

- Häufigkeitsknoten:
  Hier wird neben der konkret auftretenden Anzahl auch die minimal und maximal erlaubte Häufigkeit des Bausteins gespeichert.

[33] Details zur MDT XSD Bibliothek finden sich unter `http://projects.eclipse.org/projects/modeling.mdt.xsd`.

- Sequenzknoten:

  Hier wird eine Folge von Einsen und Nullen gespeichert, die für jeden Baustein in der Sequenz, dessen Häufigkeit variabel ist, kennzeichnet, ob dieser häufiger als in seiner minimal erlaubten Anzahl auftritt. Diese Folge wird nur dann in das komprimierte Dokument gespeichert, wenn die alternative Häufigkeitsabbildung verwendet wird.

- Auswahlknoten:

  Bei diesem Knoten wird die identifizierende Nummer der vorliegenden Alternative und die Gesamtanzahl der Alternativen gespeichert. Letztere ist nötig, um die identifizierende Nummer bei der statischen Codierung der Struktur korrekt codieren zu können.

- Knoten für Elementpermutation:

  Die Vorgehensweise bei diesem Knoten unterscheidet sich je nach Häufigkeitsdefinition für diesen Knoten, für die in diesem Fall nur verpflichtend oder optional erlaubt ist. Im zweiten Fall können beliebig viele Elemente der Gruppe im XML-Dokument auftreten. Im ersten Fall sind hingegen alle verpflichtend. Für den zweiten Fall ist daher zunächst die Anzahl der vorhandenen Elemente zu speichern. Anschließend werden die identifizierenden Nummern der einzelnen Elemente wie bei einer mehrfachen Abbildung eines Auswahlknotens gespeichert. Abschließend wird die für die binäre Codierung der identifizierenden Nummern bei der statischen Codierung nötige Anzahl der Elemente der Permutationsgruppe in den Knoten gespeichert.

Eine weitere Aufgabe, die im Rahmen der Strukturabbildung durchgeführt wird, ist die Ermittlung redundanter Strukturteile. Redundante Teilstrukturen werden unabhängig von der Einstellung für die Kürzung redundanter Strukturteile nur einmal modelliert und bei wiederholtem Auftreten als Objektreferenz auf die bereits bestehenden Strukturteile gespeichert. Die Entscheidung, ob bei der binären Codierung der Strukturabbildung die Objektreferenz oder die vollständige Abbildung der verknüpften Teilstruktur gespeichert wird, wird in der Komponente zur Co-

dierung der Strukturabbildung getroffen. Die Einstellung des Parameters, der die Kürzung redundanter Strukturteile steuert, wirkt sich nur dort aus.
Im Zuge der Strukturabbildung werden auch die Nutzdaten extrahiert. Dabei wird für jedes Attribut und jedes Element mit Nutzdaten ein Objekt angelegt, das in eine Liste eingereiht wird. In diesem Objekt werden, neben den Nutzdaten, auch der Name des umgebenden Bausteins und die Typinformationen der Nutzdaten aus der XSD gespeichert. Diese zusätzlichen Metadaten sind in der Komponente, die die binäre Codierung der Nutzdaten durchführt, notwendig.

Struktur- und Nutzdatencodierung

Die in der Strukturabbildungskomponente erzeugten Daten werden sowohl bei der Codierung der Strukturabbildung als auch bei der Kompression der Nutzdaten verwendet. Die Strukturcodierung nutzt dazu die Daten aus den Objekten des Strukturbaums. Die Nutzdatenkompression bezieht alle notwendigen Daten aus den Objekten der Nutzdatenliste. Das Verhalten beider Komponenten kann über Parameter gesteuert werden. Für die Beschreibung der Parameter wird für die Strukturverarbeitungskomponente auf Abschnitt 4.3.4 und für die Nutzdatenkompression auf Abschnitt 4.3.5 verwiesen.

Zusammenführen der komprimierten Daten

In der letzten Komponente werden die von den beiden Codierungskomponenten erzeugten binären Daten verknüpft und um einen Header aus Metadaten ergänzt. Dieser enthält die für die Dekompression notwendigen Informationen, wie einen Schlüssel für die zu verwendende XSD und die Konfiguration der Verfahrensparameter. Letztere besteht einerseits aus der externen Konfiguration und andererseits aus den von *PSXC* dynamisch bestimmten Einstellungen für die Art der Häufigkeitsabbildung sowie die Variante der Zeigerabbildung.

4.4.1.2 Der Dekompressionsprozess

Der in Abbildung 4.13 dargestellte Prozess zur Decodierung beginnt mit dem Einlesen der komprimierten Daten. In diesen ist auch die bei der Kompression benutzte Konfiguration sowie einen Schlüssel für die zu verwendende XSD gespeichert. Die Modellierung der XSD erfolgt mit der Komponente, die auch im Rahmen des Kompressionsprozesses verwendet wird.

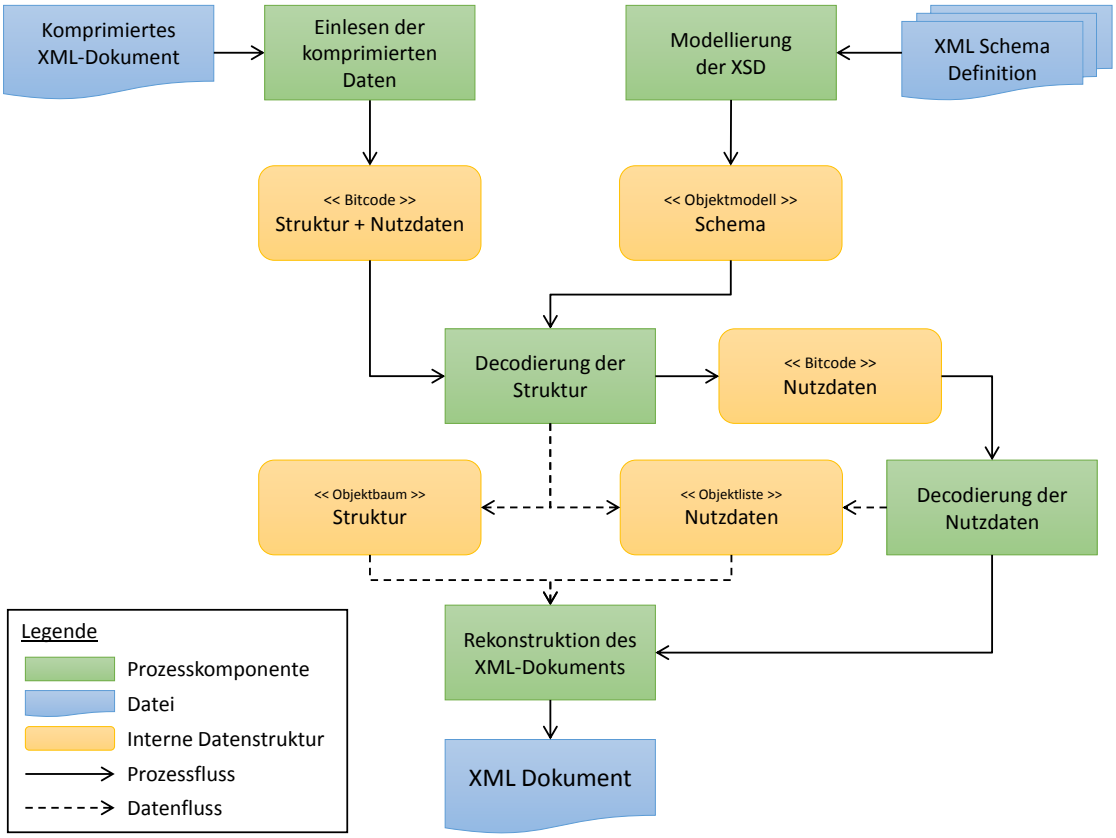

Abbildung 4.13: Prozess der Dekompression eines XML-Dokuments (Eigene Darstellung)

Die Komponente zur Decodierung der Struktur extrahiert aus den komprimierten Daten den für die Struktur relevanten Teil und gibt den verbleibenden Rest an die Komponente zur Decodierung der Nutzdaten weiter. Das Resultat der Strukturdecodierung besteht aus den beiden Datenstrukturen, die auch zur Codierung aufgebaut werden. Einerseits ist dies der Baum aus Strukturobjekten und andererseits die Liste aus Nutzdatenobjekten.

Der Strukturbaum wird identisch zu dem im Kompressionsprozess aufgebaut, um die bei der Abbildung erzeugten Zeiger durch die jeweiligen Teilbäume ersetzen zu können. Bei der Verwendung des Strukturbaums zur Rekonstruktion des XML-Dokuments werden zum einen die Namen der Elemente in den Knotenobjekten des Baumes gespeichert. Zum anderen werden für Elemente mit Nutzdaten leere Nutzdatenobjekte erstellt und in die Nutzdatenliste eingereiht. Da jedes Objekt des Strukturbaums einen Verweis auf das zugehörige Nutzdatenobjekt besitzt, kann anhand des Strukturbaums das XML-Dokument vollständig rekonstruiert werden. Bei der Decodierung der Nutzdaten wird die Liste der noch leeren Nutzdatenobjekte entsprechend der Reihenfolge bei der Kompression angeordnet. Die Sortierung ergibt sich aus den Struktur- und Typinformationen und der jeweiligen Konfiguration. Auf diese Weise können die dekomprimierten Daten der Reihe nach in die Nutzdatenobjekte gespeichert werden. Ist dieser Schritt erledigt, durchläuft die letzte Komponente den Strukturbaum und erstellt das dekomprimierte XML-Dokument. Zum Aufbau des XML-Dokuments sowie zu dessen Serialisierung wird JAXP verwendet.

4.4.2 Die objektorientierte Architektur

Zur Realisierung der im vorherigen Abschnitt beschriebenen Komponenten und Prozesse wurde eine objektorientierte Architektur entwickelt. Das Klassendiagramm in Abbildung 4.14 stellt das Resultat davon dar. Um das Diagramm übersichtlich zu halten, ist es auf die zentralen Bestandteile beschränkt. Aus diesem Grund enthält es nur für Erklärungen notwendige Funktionen und Klassenattribute. Im Speziellen wurde auf Standardkonstruktoren und Zugriffsfunktionen verzichtet.

Konzeption als Bibliothek

Der Prototyp ist als Bibliothek konzipiert, für dessen Nutzung die Klassen `PSXC` und `Settings` relevant sind. Zur Kompression und Dekompression eines XML-Dokuments stellt die `PSXC`-Klasse die Funktionen `encode` und `decode` bereit. Der `encode`-Funktion ist das zu komprimierende Dokument als `String` zu übergeben.

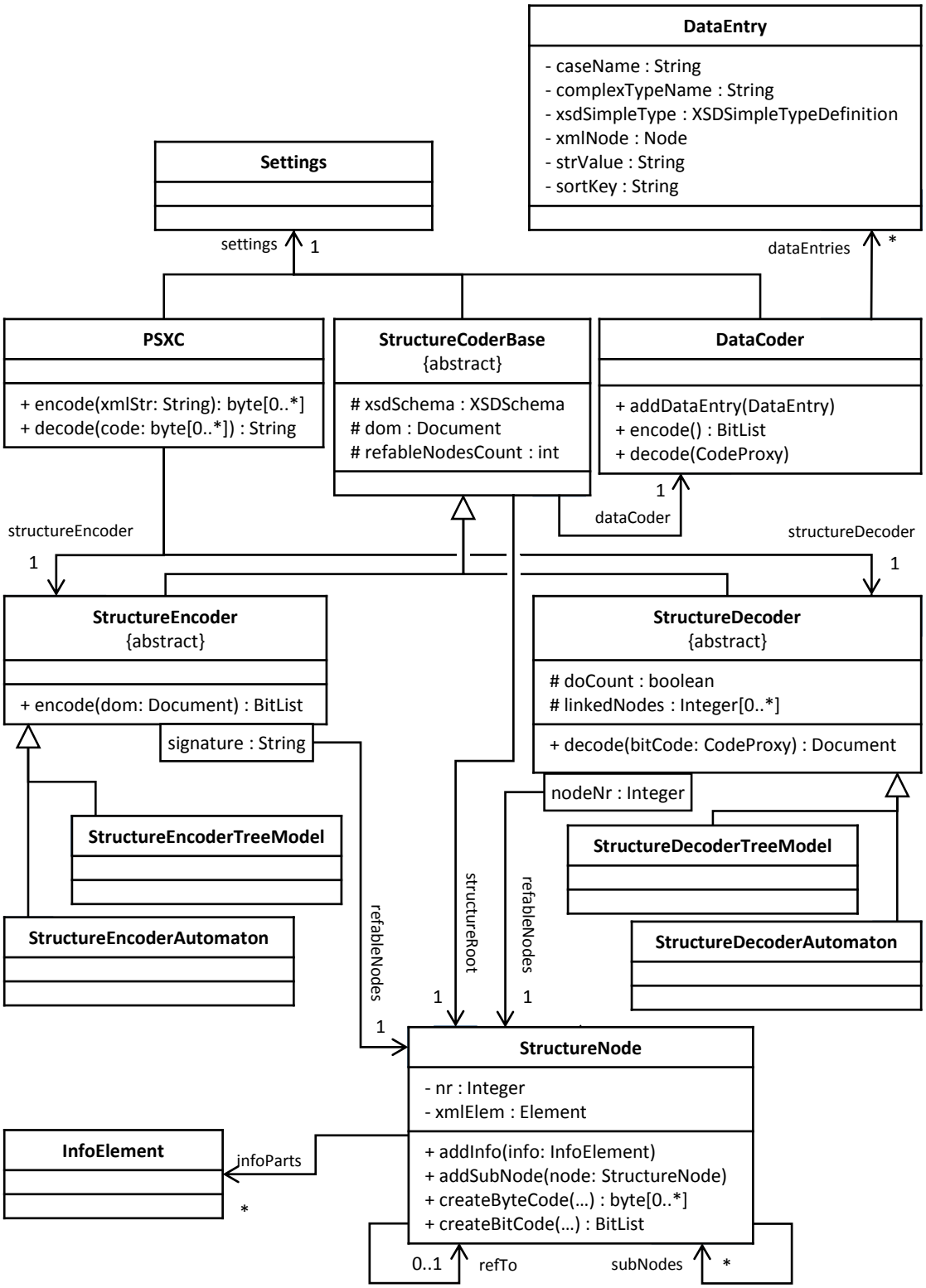

Abbildung 4.14: Klassendiagramm des Prototyps (Eigene Darstellung)

Die komprimierten Daten liefert sie als `byte`-Array zurück. Die Dekompression dieses `byte`-Arrays in einen `String` erledigt die `decode`-Funktion.
Die Konfiguration des *PSXC*-Verfahrens erfolgt über die `Settings`-Klasse. In ihr können per Zugriffsfunktionen alle in Abschnitt 4.3 beschriebenen Parameter konfiguriert werden. Dies ist nur bei der Kompression nötig, da bei der Dekompression alle Einstellungen aus dem Header der komprimierten Daten in die `Settings`-Klasse geladen werden.

Der grundlegende Aufbau

Die abstrakte `StructureCoderBase`-Klasse fungiert als Basis für die Verarbeitung der Struktur und beinhaltet als solche diejenigen Klassenattribute, die sowohl zur Kompression als auch zur Dekompression von den abgeleiteten Klassen benötigt werden. Das `xsdSchema`-Attribut enthält das Objektmodell der XSD und das `dom`-Attribut speichert das XML-Dokument als DOM. Letzteres wird bei der Kompression verarbeitet und bei der Dekompression aufgebaut. Die Assoziation zur `StructureNode`-Klasse, die durch das `structureRoot`-Attribut realisiert wird, bildet die Verknüpfung zur Abbildung der Struktur des XML-Dokuments als Objektbaum aus `StructureNode`-Objekten, die jeweils ein XML-Element repräsentieren. Dieser Objektbaum wird sowohl bei der Kompression als auch bei der Dekompression aufgebaut. Das `refableNodesCount`-Attribut wird im Rahmen der Kürzung redundanter Strukturteile verwendet, um die referenzierbaren Elemente zu nummerieren. Es erhalten nur die `StructureNode`-Objekte eine Nummer (im Attribut `nr`), die für XML-Elemente stehen, die weitere XML-Elemente beinhalten. Die `DataCoder`-Klasse realisiert sowohl die Kompression als auch die Dekompression der Nutzdaten, weshalb ihre Instanziierung auf Ebene der `StructureCoderBase`-Klasse in dem `dataCoder`-Attribut gespeichert wird.
Die Klassen `StructureEncoder` und `StructureDecoder` sind von der `StructureCoderBase`-Klasse abgeleitet und beinhalten die von der Art der Schemamodellierung unabhängigen Funktionen für die Kompression bzw. Dekompression der Struktur. Beides sind abstrakte Klassen, von denen jeweils zwei weitere Klassen

– je eine pro Schemamodellierungsart – abgeleitet werden. Diese enthalten die Funktionen zur Abbildung und Rekonstruktion der Struktur, die sich pro Modellierungsart grundlegend unterscheiden. Die Strukturabbildung bei baumbasierter Modellierung stellt die `StructureEncoderTreeModel`-Klasse und bei automatenbasierter Modellierung die `StructureEncoderAutomaton`-Klasse bereit. Bei der Dekompression werden analog dazu die Klassen `StructureDecoderTreeModel` und `StructureDecoderAutomaton` verwendet.

Die zentrale Steuerung

Die `PSXC`-Klasse bildet die zentrale Steuerung der Anwendung und delegiert die Kernaufgaben an andere Klassen. Für die Kompression der Struktur wird je nach Modellierungsart entweder ein `StructureEncoderTreeModel`-Objekt oder ein `StructureEncoderAutomaton`-Objekt angelegt und in der Klassenvariablen `structureEncoder` gespeichert. Analog dazu wird für die Dekompression ein `StructureDecoderTreeModel`-Objekt oder ein `StructureDecoderAutomaton`-Objekt in das `structureDecoder`-Attribut abgelegt. Zudem wird für die Nutzdatenverarbeitung ein `DataCoder`-Objekt erstellt und in der jeweiligen Instanz der `StructureEncoder`- und `StructureDecoder`-Klasse gespeichert.

Wird die `encode`-Funktion der `PSXC`-Klasse aufgerufen, nutzt diese die DOM-Schnittstelle der JAXP, um das zu komprimierende XML-Dokument als Objektstruktur abzubilden. Das Ergebnis ist eine Instanz der `Document`-Klasse des `org.w3c.dom`-Pakets, die zur Kompression der Struktur an die `encode`-Funktion der jeweiligen `StructureEncoder`-Instanz übergeben wird. Vor diesem Schritt wird die zu verwendende XSD mit Hilfe der XSD MDT Bibliothek in ein `XSDSchema`-Objekt eingelesen und in das `StructureEncoder`-Objekt gespeichert. Nach der Strukturverarbeitung wird die `encode`-Funktion des `DataCoder`-Objekts aufgerufen, um die Nutzdaten zu komprimieren.

Die `encode`-Funktionen des `StructureEncoder`- und `DataCoder`-Objekts geben jeweils ein `BitList`-Objekt zurück, die die binäre Codierung der Struktur und der Nutzdaten enthalten. Diese werden im letzten Schritt der `encode`-Funktion der

PSXC-Klasse zunächst miteinander verknüpft und um einen Header erweitert, bevor sie als byte-Array von der Funktion zurückgegeben werden. Der Header beinhaltet zum einen die Konfiguration des in der Klassenvariable settings gespeicherten Settings-Objekts und zum anderen einen Schlüssel für die zu verwendende XSD. Im ersten Schritt der Dekompression liest die decode-Funktion der PSXC-Klasse die Konfiguration aus dem Header der komprimierten Daten und speichert diese im Settings-Objekt. Anschließend wird die zu verwendende XSD aus den Headerdaten ermittelt und über die XSD MDT Bibliothek in ein XSDSchema-Objekt eingelesen. Dieses wird analog zur Kompression im StructureDecoder-Objekt gespeichert.

Im nächsten Schritt werden die komprimierten Daten an die decode-Funktion des StructureDecoder-Objekts übergeben, um die Struktur zu rekonstruieren. Die dazu verwendete CodeProxy-Klasse kapselt die Decodierung der binären Daten in Zahlenwerte, wie sie bei der Rekonstruktion der Struktur aus der Schemamodellierung benötigt werden. Dementsprechend ist das Verhalten der Funktionen in der CodeProxy-Klasse von der Einstellung für den Parameter, der die Art der Strukturcodierung steuert, abhängig.

Bei der Erstellung der Struktur aus den komprimierten Daten wird eine Liste von Nutzdatenobjekten angelegt. Diese sind mit den zugehörigen Objekten der XML-Bausteine verknüpft und werden durch den Aufruf der decode-Funktion des DataCoder-Objekts mit den dekomprimierten Nutzdaten gefüllt. Durch die Verknüpfung der Nutzdatenobjekte mit den Objekten der XML-Bausteine können die Nutzdaten direkt in die XML-Bausteine eingetragen werden. Dadurch ergibt sich nach Abarbeitung der decode-Funktion des DataCoder-Objekts eine vollständige DOM-Instanz des dekomprimierten XML-Dokuments. Diese wird im letzten Schritt der decode-Funktion der PSXC-Klasse mit Hilfe von JAXP serialisiert und als String zurückgegeben.

Strukturabbildung

Im Rahmen der Abbildung der Struktur eines XML-Dokuments wird für jedes XML-Element ein `StructureNode`-Objekt erstellt und mit diesen die Struktur des Dokuments nachgebildet. Der Aufbau einer baumartigen Struktur wird dadurch ermöglicht, dass einem `StructureNode`-Objekt über die reflexive Assoziation `subNodes` mehrere `StructureNode`-Objekte, im Sinne von Kindknoten, zugeordnet werden können. Das Einfügen neuer Kindknoten erfolgt über die `addSubNode`-Funktion. Jedes Knotenobjekt enthält im `xmlElem`-Attribut, das vom Typ `org.w3c.dom.Element` ist, eine Referenz auf das XML-Element, das es repräsentiert.
Über die `addInfo`-Funktion kann einem `StructureNode`-Objekt ein `InfoElement`-Objekt übergeben werden, das der Liste des `infoParts`-Attributs hinzugefügt wird. Ein `InfoElement`-Objekt wird bei einem einzelnen Schritt der Strukturabbildung erzeugt und nimmt die dabei erstellten Daten auf. Ein einzelner Schritt ist ein Zustandsübergang bei der automatenbasierten Modellierung oder die Verarbeitung eines Knotens bei der baumbasierten Modellierung. In einem `StructureNode`-Objekt werden alle `InfoElement`-Objekte abgelegt, die bei der Abbildung des Inhaltsmodells des zugehörigen XML-Elements entstehen. Folglich sind dies die Daten, die angeben, welche direkten Kindelemente das XML-Element des `StructureNode`-Objekts besitzt.
Die `StructureEncoderTreeModel`-Klasse stellt Funktionen bereit, mit denen die Objektstrukturen in den beiden Attributen `xsdSchema` und `dom` im Sinne eines Visitor-Patterns verarbeitet werden. Ausgehend von einem XML-Element werden einerseits dessen Kinder aus dem DOM ermittelt und andererseits dessen Inhaltsmodell als Teil des XSD-Objektmodells geladen. Anschließend wird ermittelt, wie sich aus dem Objektbaum des Inhaltsmodells die konkret vorhandenen Kindelemente ergeben. Die bei Sequenz-, Auswahl- und Häufigkeitsobjekten ermittelten Daten werden in `InfoElement`-Objekten gespeichert und in das `StructureNode`-Objekt des zugehörigen XML-Elements abgelegt.
Die `StructureEncoderAutomaton`-Klasse nutzt die von der XSD MDT Bibliothek im Objektmodell des `xsdSchema`-Attributs bereitgestellte Modellierung eines In-

haltsmodells als deterministischen endlichen Automaten (DEA). Dieser wird auf die Liste von Kindelementen angewendet. Für jeden nicht alternativlosen Zustandsübergang wird ein `InfoElement`-Objekt angelegt, in dem die Nummer des Zustandsübergangs und die Anzahl der Alternativen gespeichert wird.
In beiden Fällen folgt nach der Abarbeitung eines Elements die Verarbeitung der Kindelemente in der Reihenfolge ihres Auftretens. Dazu werden rekursiv die soeben beschriebenen Prozesse für die einzelnen Kindelemente gestartet.

Behandlung redundanter Strukturteile

Wie in Abschnitt 4.3.3 beschrieben, werden bei *PSXC* redundante Strukturteile ermittelt und durch Zeiger ersetzt. Auf objektorientierter Ebene werden die Zeiger über die reflexive Assoziation des `refTo`-Attributs der `StructureNode`-Klasse abgebildet.
Die Ermittlung redundanter Strukturteile erfolgt bei beiden Arten der Schemamodellierung identisch. Dazu stellt die `StructureEncoder`-Klasse in dem `refableNodes`-Attribut eine Datenstruktur bereit, die `StructureNode`-Objekte in Relation zu ihrer Signatur verwaltet. Damit ist es möglich, redundante Strukturteile zu erkennen und über das `refTo`-Attribut die entsprechenden `StructureNode`-Objekte zu verknüpfen.
In der binären Codierung wird ein Zeiger als Anzahl von referenzierbaren Elementen zwischen Quell- und Zielelement gespeichert. Diese ergibt sich aus der Differenz der `nr`-Attribute der entsprechenden `StructureNode`-Objekte. Bei der Rekonstruktion von Strukturteilen, die durch Zeiger ersetzt wurden, ist es folglich nötig, auf ein `StructureNode`-Objekt anhand seiner Nummer zugreifen zu können. Zu diesem Zweck stellt des `refableNodes`-Attribut der `StructureDecoder`-Klasse ein assoziatives Datenfeld bereit, in das die im Rahmen der Rekonstruktion eines XML-Dokuments erstellten referenzierbaren `StructureNode`-Objekte mit ihrer Nummer als Schlüssel abgelegt werden.
Die von der `StructureDecoder`-Klasse bereitgestellten Variablen dienen zur Rekonstruktion von Teilstrukturen, die durch Zeiger ersetzt wurden. Die Liste von

Zahlen im `linkedNodes`-Attribut wird verwendet, wenn die verknüpften Knoten als Liste in den komprimierten Daten gespeichert wurden. Die Zahlen stellen die Nummern der `StructureNode`-Objekte dar, für die Zeiger gespeichert wurden.
Im Gegensatz zur Strukturabbildung werden bei der Rekonstruktion Zeiger aufgelöst und redundante Strukturteile mehrfach erstellt, um das gesamte XML-Dokument aufzubauen. Da auch bei der Rekonstruktion die `StructureNode`-Objekte nummeriert werden müssen, ist es wichtig, dass hierbei redundante Strukturteile übersprungen werden. Zu diesem Zweck wird das `doCount`-Attribut verwendet. Immer wenn eine durch einen Zeiger ersetzte Teilstruktur expandiert wird, wird dieses Attribut auf `false` gesetzt, wodurch während des Aufbaus der redundanten Teilstruktur die Nummerierung der Knoten unterbrochen wird.

Binäre Codierung der Strukturabbildung

Die `StructureNode`-Klasse ermöglicht nicht nur die Strukturabbildung, sondern stellt mit den Funktionen `createByteCode` und `createBitCode` auch die Funktionalität zu deren binären Codierung bereit. Die aufgrund ihrer Länge hier nicht aufgeführten Listen der Funktionsparameter beider Funktionen dienen der Anpassung der Funktionalität an die Parametereinstellungen von *PSXC* für die Strukturcodierung. Die `createByteCode`-Funktion liefert eine byteorientierte Repräsentation der Teilstruktur unterhalb des Knotens als `byte`-Array und erfüllt damit eine Doppelfunktion. Zum einen wird das `byte`-Array als Signatur für den Strukturteil unterhalb des Knotens verwendet, die im Rahmen der Ermittlung redundanter Strukturteile notwendig ist. Zum anderen liefert es für die dynamische und gemischte Codierung eine Strukturdarstellung, die für die Verarbeitung durch Universalkompressoren vorgesehen ist.

Nutzdatenverarbeitung

Die Kompression und Dekompression der Nutzdaten wird von der `DataCoder`-Klasse bereitgestellt. Die Nutzdaten werden bei der Strukturabbildung in `DataEntry`-Objekte gepackt und über die `addDataEntry`-Funktion an das `DataCoder`-Objekt

der `StructureEncoder`-Instanz übergeben. Im `DataCoder`-Objekt werden sie in die durch das `dataEntries`-Attribut bereitgestellte Liste für `DataEntry`-Objekte eingereiht.

Die `DataEntry`-Klasse wird sowohl bei der Kompression als auch bei der Dekompression verwendet. Bei der Abbildung sowie bei der Rekonstruktion der Struktur wird für jedes XML-Element mit Nutzdaten und jedes Attribut ein `DataEntry`-Objekt angelegt. Im `xmlNode`-Attribut wird jeweils die Referenz auf den zugehörigen XML-Baustein (Objekt eines Elements oder Attributs) gespeichert und zusätzlich dessen Name in das `caseName`-Attribut abgelegt.

Ist der Typ des XML-Bausteins ein einfacher Datentyp, wird er in dem Klassenattribut `xsdSimpleType` gespeichert. Andernfalls wird der Name des komplexen Typs in das Attribut `complexTypeName` und der einfache Typ innerhalb des komplexen Typs in das `xsdSimpleType`-Attribut abgelegt. Das `sortKey`-Attribut wird aus den soeben genannten Typinformationen und dem Namen des Bausteins aus dem `caseName`-Attribut generiert. Dieser Schlüssel wird sowohl zur Sortierung als auch bei der Zerlegung der Nutzdaten in Container verwendet.

Die bisher beschriebenen Attribute der `DataCoder`-Klasse werden bei der Kompression und Dekompression auf die gleiche Weise mit Informationen aus der XSD gefüllt. Das `strValue`-Attribut hingegen wird bei der Kompression mit der textuellen Repräsentation der Nutzdaten aus dem zu verarbeitenden XML-Dokument und bei der Dekompression aus den dekomprimierten Nutzdaten gefüllt.

5 Fallstudie zur optischen Codierung von Rechnungsdokumenten

In diesem Kapitel wird das in Kapitel 4 entwickelte parametrisierte Verfahren *PSXC* evaluiert. Dazu wird die prototypische Implementierung im Rahmen einer Fallstudie einer dynamischen Analyse unterzogen. Im Zuge dessen wird die optische Codierung aller Ausgangsrechnungen eines mittelständischen IT-Dienstleisters über einen Zeitraum von 18 Monaten untersucht.
Im ersten Teil der Fallstudie wird analysiert, welche Konzepte der schemabasierten XML-Kompression die höchste Leistung erzielen, indem das parametrisierte Verfahren in allen möglichen Konfigurationen auf die Echtdaten angewendet wird. Die Ergebnisse daraus werden im zweiten Teil in einen Vergleich mit bestehenden Verfahren zur XML-Kompression eingebunden und dadurch gezeigt, welche Leistungssteigerung die beste Konfiguration von *PSXC* ermöglicht. Die Erkenntnisse des Vergleichs werden anschließend genutzt, um darzulegen, wie sich die höhere Kompressionsleistung von *PSXC* auf die optische Codierung der XML-basierten Rechnungsdokumente auswirkt.

5.1 Grundlagen zur Fallstudie

Im Folgenden werden die in der Fallstudie verwendeten Daten und die technische Umgebung beschrieben. Da Universalkompressoren sowohl als Backendkompressoren bei speziellen XML-Kompressoren als auch als eigenständige Kompressoren im Rahmen der Fallstudie zum Einsatz kommen, wird als Nächstes ein Überblick über die verwendeten Kompressoren und deren Konfiguration gegeben. Im letzten Teil der Grundlagen wird erklärt, welche Daten erhoben und wie diese verdichtet

werden. Zudem wird erläutert, wie die aggregierten Werte in der nachfolgenden Fallstudie ausgewertet werden.

5.1.1 Echtdaten als Basis

Die Fallstudie basiert durchgehend auf Echtdaten von 1.757 Ausgangsrechnungen, die von einem mittelständischen IT-Dienstleistungsunternehmen zur Verfügung gestellt wurden. Da Rechnungsdaten sehr sensibel behandelt werden, war es nicht möglich, Daten weiterer Unternehmen zu erhalten. Besonders aufgrund dieser Sensibilität der Daten ist hervorzuheben, dass es sich um gänzlich unverfälschte Echtdaten handelt, wodurch der Einfluss künstlicher Zufallsprozesse durchweg entfällt. Des Weiteren bilden die 1.757 Dokumente keine zufällig erhobene Stichprobe, sondern ein vollständiges Abbild der Realität der Rechnungsdaten eines Unternehmens aus einem Zeitraum von 18 Monaten.
Zur Speicherung der Rechnungsdaten wurde das ZUGFeRD-Format gewählt, da mit der optischen Codierung von Geschäftsdokumenten die Übertragung der Ziele des ZUGFeRD-Konzepts auf papierbasierte Dokumente verfolgt wird.[1] Um eine automatisierte Weiterverarbeitung zu erreichen, ist von den drei möglichen Profilen des ZUGFeRD-Formats das *COMFORT*-Profil zu verwenden.[2] Die Rechnungsdaten wurden in einem speziellen Format der Firma SAP zur Verfügung gestellt, weshalb eine eigene Software zur Konvertierung der erhaltenen Daten in das ZUGFeRD-Format entwickelt und implementiert wurde.
Zur Charakterisierung der Daten stellt Abbildung 5.1 die Anzahl der Rechnungen und deren durchschnittliche Größe in Abhängigkeit von der Anzahl der Rechnungspositionen dar. Dabei wird deutlich, dass es sich vorwiegend um Rechnungen mit wenigen Positionen handelt. So haben 81,7 % der Rechnungen nur eine bis vier Positionen und der Anteil der Rechnungen mit bis zu zehn Positionen liegt sogar bei 96,3 %. Rechnungen mit 20 und mehr Positionen bilden mit 0,5 % hingegen ei-

[1] Vgl. Bergmann u. a.(2014), S. 12.
[2] Vgl. Bergmann u. a.(2014), S. 22 f.

ne Ausnahme. Dementsprechend haben 98,7 % der Rechnungen nur eine Seite und lediglich 1,3 % zwei.

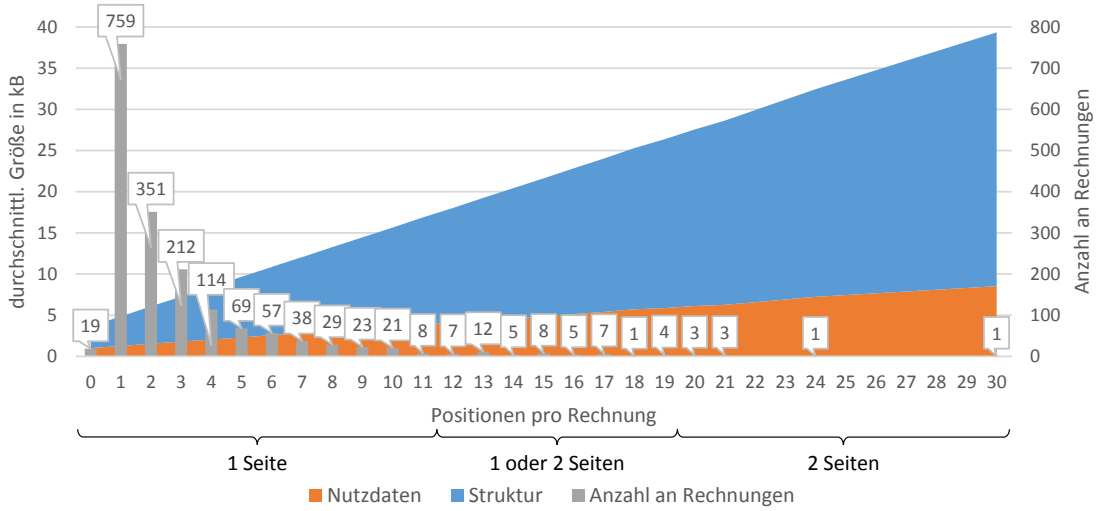

Abbildung 5.1: Beschaffenheit der Echtdaten (Eigene Darstellung)

Die durchschnittliche Größe der Dokumente wird in Abbildung 5.1 als Summe aus Struktur- und Nutzdaten veranschaulicht. Die XML-Dokumente wurden ohne jegliche formatierende Leerzeichen und Zeilenumbrüche erzeugt. Auf diese Weise wird einerseits sichergestellt, dass unterschiedliche Arten der Informationserhaltung (siehe Abschnitt 3.2.2.2) den späteren Vergleich der Verfahren nicht beeinflussen und andererseits die Größe des Strukturteils nicht künstlich erhöht wird. Der Anteil der Nutzdaten an der gesamten Größe des jeweiligen XML-Dokuments liegt zwischen 27,3 % bei der kleinsten und 21,7 % bei der größten Rechnung.

5.1.2 Technischer Rahmen

Als Basis der Versuchsumgebung fungierte eine Maschine mit einem Intel Core i7-930 und 12 GB Arbeitsspeicher. Als Betriebssystem wurde primär Microsoft Windows 10 in 64-bit verwendet. Da zwei spezielle XML-Kompressoren nicht auf Windows compiliert werden konnten, kam zusätzlich Ubuntu 14.04 in 64-bit zum Einsatz.

Zum Compilieren der in C und C++ vorliegenden Quellcodes bestehender Kom-

pressionsverfahren wurde in Windows MinGW[3], eine Portierung der GNU-Entwicklerwerkzeuge für die Windows-Plattform, mit den Compilerversionen 4.9.3-1 verwendet. In Linux wurden die GNU Compiler für C und C++ jeweils in der Version 4.8.4 eingesetzt.
In der Fallstudie werden eine Reihe von Java-basierten Programmen verwendet. Zu diesen gehören neben dem Prototyp von *PSXC* auch das Programm zur Konvertierung der Rechnungsdaten und das in Abschnitt 5.1.4 beschriebene Framework zur Erhebung der Messwerte. Zudem liegt ein XML-spezifischer Kompressor als Java-Quelltext vor. Zum Compilieren und Ausführen dieser Programme wurde Java 8 Update 51 von Oracle in der 64-bit Version benutzt.[4]

5.1.3 Verwendete Universalkompressoren

Im Rahmen der Fallstudie werden acht verschiedene Universalkompressoren betrachtet. Diese werden einerseits als Backendkompressoren für *PSXC* und andererseits als eigenständige Verfahren zur Kompression von XML-Dokumenten im zweiten Teil der Fallstudie verwendet. Die eingesetzten Verfahren sind in Tabelle 5.1 inklusive der Bezugsquellen für die konkret verwendete Implementierung zusammengestellt. Es wurden alle Universalkompressoren berücksichtigt, die in der Literatur zum Leistungsvergleich der in den Abschnitten 3.4 und 3.5 betrachteten Verfahren herangezogen wurden.
Die Leistungsfähigkeit von PPM- und PAQ-Verfahren kann über Parameter gesteuert werden, wobei eine bessere Kompressionsrate mit einer höheren Laufzeit einher geht. Bei PPM-Verfahren kann die Anzahl der als Kontext zu berücksichtigenden Zeichen (*order*) und die Größe des zu verwendenden Arbeitsspeichers (*memory*) in MB spezifiziert werden. Bei PAQ-Verfahren wird die Größe des zu benutzenden Arbeitsspeichers durch die Angabe einer Leistungsstufe (*level*) definiert.
Um passende Parametereinstellungen zu ermitteln, wurde eine unterstützende Untersuchung durchgeführt. Für diese wurden aus der Datenbasis zufällige Dokumen-

[3] Verfügbar unter http://www.mingw.org.
[4] Verfügbar unter http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Kompressor	Verwendete Implementierung
gzip	Integriert in Java SE 8 Update 51
LZMA	http://7-zip.org/a/lzma1514.7z
PPMd	http://www.compression.ru/ds/ppmdi2.rar
PPMVC	http://pskibinski.pl/research/PPMVC/PPMVC12.zip
LPAQ	http://mattmahoney.net/dc/lpaq6.zip
PAQ8	http://mattmahoney.net/dc/paq8l.zip
bzip2	http://code.google.com/p/jbzip2/downloads/detail?name=jbzip2-0.9.1.jar
Sequitur	https://github.com/craignm/sequitur

Tabelle 5.1: In der Fallstudie berücksichtigte Universalkompressoren

te in der Form zusammengestellt, dass aus jeder Gruppe mit gleicher Anzahl an Positionen ein Dokument enthalten ist. Dieser Satz an Dokumenten dient als Basis für die Bestimmung der Parameterwerte, die zur eigenständigen Kompression der Echtdaten im zweiten Teil der Fallstudie verwendet werden. Um die Parameterwerte für den Einsatz der Universalkompressoren als Backendkompressoren im parametrisierten Verfahren zu ermitteln, wurden die Nutzdaten der zufällig ausgewählten Dokumente verwendet.

Die soeben beschriebenen Datensätze wurden in der Untersuchung mit verschiedenen Parametereinstellungen der Universalkompressoren verarbeitet. Für die Parameter *level* und *order* wurden alle einstellbaren Werte untersucht. Bei der Größe des zu verwendenden Arbeitsspeichers der beiden PPM-Verfahren, die bis zu einer Obergrenze von 256 MB frei wählbar ist, wurden die drei Werte 10, 32 und 256 MB berücksichtigt.

Zur Auswertung wurde pro Verfahren und Konfiguration die durchschnittlich erzeugte Kompressionsrate berechnet. Bei den PPM-Verfahren zeigte sich, dass die Verwendung von mehr als 10 MB Arbeitsspeicher die Kompressionsrate nicht verbessert. Die Länge des Kontextes und die Leistungsstufe wurden so ausgewählt, dass sie einen ausgewogenen Kompromiss zwischen Laufzeit und Kompressionsrate

ergeben. Dabei wurde die Konfiguration gewählt, bei der der Anstieg der Leistung zur nächst besseren Konfiguration geringer ist als der Anstieg der Laufzeit. Die gewählten Werte sind in Tabelle 5.2 zusammengestellt. Bei *PAQ8* wurden für den Einsatz als eigenständiger Kompressor zwei Parameterwerte berücksichtigt, da sich hier zwei ausgewogene Kompromisse zwischen Kompressionsrate und Laufzeit ergaben.

Einsatzzweck	Kontextlänge		Leistungsstufe	
	PPMd	PPMVC	LPAQ	PAQ8
im parametrisierten Verfahren	9	8	2	4
als eigenständige Kompressoren	15	16	9	3 u. 4

Tabelle 5.2: Einstellungen konfigurierbarer Universalkompressoren

5.1.4 Ermittlung und Verdichtung von Messwerten

Im Folgenden wird beschrieben, wie in den ersten beiden Teilen der Fallstudie die 1.757 Rechnungsdokumente genutzt werden, um zunächst die unterschiedlichen Konfigurationen von *PSXC* und anschließend unterschiedliche Verfahren zur Kompression von XML-Dokumenten zu vergleichen.

Messwerte

Bei jedem Kompressionsvorgang werden die Kompressionsrate, die komprimierte Größe des Dokuments und die Laufzeit der Verarbeitung erhoben und gespeichert. Die Kompressionsrate ist in dieser Arbeit, entsprechend der *compression ratio* bei Salomon (2007a), wie folgt definiert:[5]

$$\text{Kompressionsrate} = \frac{\text{komprimierte Größe}}{\text{ursprüngliche Größe}}$$

Bei dem parametrisierten Verfahren ist die Kompression der Struktur unabhängig von der Kompression der Nutzdaten, weshalb die beste Kombination von Konzep-

[5] Vgl. Salomon (2007a), S. 10.

ten in beiden Bereichen getrennt analysiert werden kann. Aus diesem Grund wurde das Verfahren so implementiert, dass die Verarbeitung der Struktur bzw. die der Nutzdaten wahlweise übersprungen werden kann. Da somit entweder nur Nutzdaten oder nur Strukturdaten komprimiert werden, basiert die Kompressionsrate nicht auf der Größe des gesamten XML-Dokuments, sondern lediglich auf der des Struktur- bzw. Nutzdatenteils.

Automatisierte Datenerhebung

Im Rahmen der Fallstudie sind 571.025 Kompressionsvorgänge durchzuführen, die sich daraus ergeben, dass die 1.757 Dokumente zum einen mit verschiedenen Kompressionsverfahren und zum anderen mit unterschiedlichen Konfigurationen dieser Verfahren verarbeitet werden. Im Folgenden wird der Begriff **Kompressionskonfiguration** dazu verwendet, um auf ein Verfahren in einer konkreten Konfiguration Bezug zu nehmen.

Beim Vergleich der verschiedenen Konzeptkombinationen der schemabasierten XML-Kompression mithilfe von *PSXC* werden 136 Konfigurationen im Bereich der Strukturverarbeitung und 160 im Bereich der Nutzdatenverarbeitung untersucht. Im Gesamtvergleich der Verfahren zur XML-Kompression werden 26 Verfahren in teilweise unterschiedlichen Konfigurationen berücksichtigt, sodass dort die Echtdaten in 29 verschiedenen Kompressionskonfigurationen verarbeitet werden.

Aufgrund der hohen Anzahl an Messvorgängen wurde ein eigenes Framework zur Automatisierung der Erhebung sowie Speicherung der Messwerte in einer Datenbank entwickelt und in Java implementiert. Mit diesem ist es möglich, eine Liste von Kompressionskonfigurationen zu erstellen und auf jedes der 1.757 Dokumente automatisch anwenden zu lassen. Die Erstellung der 296 verschiedenen Konfigurationen für *PSXC* vereinfacht das Framework, indem nur die pro Parameter zu untersuchenden Parameterwerte zu spezifizieren sind und die entsprechenden Kombinationen daraus automatisch gebildet werden.

Die Einbindung von Kompressionsverfahren in den Automatisierungsprozess, die nicht Java-basiert sind, erfolgt über eine Klasse, die das jeweilige Verfahren kapselt.

Diese Klasse steuert die Verfahren über Kommandozeilenparameter und verarbeitet die Rückgaben der jeweiligen Kompressionsprogramme. Die zu verarbeitenden XML-Dokumente können in dem Framework dynamisch geladen werden, indem ein Dateipfad für den beinhaltenden Ordner und ein Muster für die zu berücksichtigenden Dateien angegeben werden.
Das Framework speichert bei jedem Kompressionsvorgang den Namen des Verfahrens, die Konfiguration, den Namen des komprimierten Dokuments, die Kompressionsrate, die komprimierte Größe und die Laufzeit in eine Datenbank. Wird *PSXC* in dem Modus ausgeführt, in dem nur die Struktur- oder Nutzdaten verarbeitet werden, so wird die Berechnung der Kompressionsrate entsprechend angepasst.

Verdichtete Messwerte und deren Bedeutung

Eine Kompressionskonfiguration wird untersucht, indem jede der 1.757 Rechnungen komprimiert wird und die dabei erhobenen Messwerte zu einem Datensatz verdichtet werden. Dabei werden die folgenden Daten ermittelt:

- minimale Kompressionsrate
- maximale Kompressionsrate
- durchschnittliche Kompressionsrate
- Standardabweichung der Kompressionsrate
- minimale Größe der komprimierten Dokumente
- maximale Größe der komprimierten Dokumente
- durchschnittliche Größe der komprimierten Dokumente
- Standardabweichung der Größe der komprimierten Dokumente
- minimale Laufzeit
- maximale Laufzeit
- durchschnittliche Laufzeit
- Standardabweichung der Laufzeit

Die Leistung von Kompressionsverfahren wird meist anhand der Kompressionsrate gemessen. Der Vergleich der Verfahren auf einer identischen Datenbasis – wie dies in dieser Arbeit der Fall ist – ermöglicht es, die Größe der komprimierten Daten direkt zu betrachten. In den meisten Szenarien, in denen ein Kompressor auszuwählen ist, zeigt sich die durchschnittliche Kompressionsrate als Maß hilfreicher als die durchschnittliche Größe der komprimierten Daten. Im vorliegenden Szenario ist dies anders, da die komprimierten XML-Dokumente letztlich als optische Codes gespeichert werden und die resultierende Größe entscheidend ist. Somit ist die Größe der komprimierten Dokumente ein geeigneteres Maß, wie im Folgenden dargelegt wird.

Wenngleich die durchschnittliche Kompressionsrate als Indikator für die Durchschnittsgröße der resultierenden optischen Codes verwendet werden kann, ermöglichen die beiden Extremwerte der Kompressionsrate aufgrund der unterschiedlichen Dokumentengrößen keine entsprechenden Rückschlüsse. Die Erklärung für dieses Phänomen liefert ein kurzes Beispiel: Es werden drei Dokumente A, B und C betrachtet. A ist wesentlich kleiner als C und B liegt zwischen den beiden. Des Weiteren tritt bei A die schlechteste Kompressionsrate und bei C die beste auf. Falls nun B eine wesentlich schlechtere Kompressionsrate als C aufweist, aber die Größe von B nahe an der von C liegt, kann B komprimiert deutlich größer als C im komprimierten Zustand sein. In diesem Fall erzeugt das Dokument mit der schlechtesten Kompressionsrate nicht den größten optischen Code. Damit ist gezeigt, dass die Kompressionsrate keinen Rückschluss auf die Größe des größten optischen Codes ermöglicht.

Im praktischen Einsatz optisch codierter Geschäftsdokumente wird das Layout eines Dokumententyps mit hoher Wahrscheinlichkeit einen statischen Platzhalter für den optischen Code vorsehen, wodurch dieser in der Größe beschränkt ist. Folglich muss jedes Dokument dieses Typs so komprimiert werden können, dass der entsprechende Platz ausreichend ist. In dieser Hinsicht ist die Maximalgröße der komprimierten Dokumente ein wichtiges Maß.

5.2 Leistungsvergleich der Konzeptkombinationen

Nachfolgend wird die Leistungsfähigkeit verschiedener Konzepte der schemabasierten XML-Kompression verglichen. Zu diesem Zweck werden die Rechnungsdokumente mit den verschiedenen Einstellungen des im Kapitel 4 entwickelten, parametrisierten Verfahrens *PSXC* verarbeitet.[6] Die dabei berücksichtigten Universalkompressoren sind in Abschnitt 5.1.3 aufgelistet. Entsprechend der getrennten Verarbeitung von Struktur- und Nutzdaten bei *PSXC* werden auch bei dieser Untersuchung die Konzepte zur Struktur- und Nutzdatenkompression getrennt betrachtet.
Da sich die Kürzung redundanter Strukturteile und die alternative Häufigkeitsabbildung bei der nicht-statischen Codierung negativ auswirken können, wird zunächst untersucht, bei welchen Universalkompressoren sich Nachteile ergeben. Auf Basis dieser Erkenntnisse werden die verschiedenen Konfigurationen im Strukturbereich analysiert. Im Anschluss daran folgt ein Vergleich der verschiedenen Konzeptkombinationen zur Nutzdatenkompression.

5.2.1 Strukturverarbeitungskonzepte mit Einfluss auf nachgelagerte Universalkompressoren

In diesem Abschnitt wird für die nicht-statische Codierung untersucht, inwiefern die alternative Häufigkeitsabbildung sowie die Kürzung redundanter Strukturteile in Verbindung mit nachgelagerten Universalkompressoren die Kompression der Struktur negativ beeinflusst. Des Weiteren wird der positive Effekt der beiden Konzepte bei der statischen Codierung der Strukturabbildung verdeutlicht.
Um den Einfluss der beiden Konzepte auf nachgelagerte Universalkompressoren zu bestimmen, werden beide einzeln analysiert, indem für alle Konfigurationsvarianten die Messwerte bei aktiviertem Parameter von denen des deaktivierten Parameters abgezogen werden.[7] Negative Zahlen bedeuten somit, dass durch die Aktivierung des Konzepts der jeweilige Wert kleiner wird. Dies stellt eine Verbesserung dar, da

[6] Die einzelnen Parameter sind in Kapitel 4.3 beschrieben.
[7] Die aggregierten Werte aller Parameterkonstellationen, aus denen die Differenzen berechnet wurden, befinden sich im Anhang A auf Seite VIII.

bei der Kompressionsrate, der komprimierten Größe, der Laufzeit und der Standardabweichung kleinere Werte von Vorteil sind.
Beim automatenbasierten Ansatz sind alle Differenzen bezüglich der alternativen Häufigkeitsabbildung gleich null. Dies bedeutet, dass sie von *PSXC* in keinem Fall als Vorteil eingestuft und daher nie verwendet wurde. Im Gegensatz dazu wirkt sie sich bei der baumbasierten Modellierung nicht einheitlich aus und hängt davon ab, ob redundante Strukturteile gekürzt werden.

5.2.1.1 Einfluss der Kürzung redundanter Strukturteile

Die Kürzung redundanter Strukturteile beeinflusst den Nutzen der alternativen Häufigkeitsabbildung und muss daher als Erstes betrachtet werden.

Einfluss bei automatenbasierter Modellierung

Die Auswirkungen der Kürzung redundanter Strukturteile bei der automatenbasierten Modellierung sind in Tabelle 5.3 zusammengefasst. Die alternative Häufigkeitsabbildung wird, wie eingangs angemerkt, bei der automatenbasierten Modellierung in keinem Fall angewendet, weswegen sie keinen Einfluss auf die Auswirkung der Kürzung redundanter Strukturteile hat und somit hier unberücksichtigt bleiben kann.
Wie die Tabelle 5.3 zeigt, führt die Verarbeitung der Strukturredundanz nur bei der statischen Codierung zur Verbesserung aller Messwerte. Im Gegensatz dazu ergeben sich bei der dynamischen Codierung mit allen Universalkompressoren Nachteile. Bei der gemischten Codierung verringert sie zwar die maximale Größe der komprimierten Dokumente, jedoch nur in einem vernachlässigbaren Umfang. Zusammenfassend ist festzuhalten, dass der Einsatz von Universalkompressoren bessere Ergebnisse liefert, wenn die Redundanz noch in der Struktur vorhanden ist und nicht vorverarbeitet wurde.

Einfluss bei baumbasierter Modellierung

Die Ergebnisse der Analyse zur Kürzung redundanter Strukturteile im Fall der baumbasierten Modellierung sind in Tabelle 5.4 dargestellt. Da eine Abhängigkeit

Konfiguration		Differenz Kompressionsrate [%]			Differenz kompr. Größe [Byte]		
Codierungs-art	Backend-kompressor	minimal	durch-schnittl.	maximal	durch-schnittl.	maximal	Std.Abw.
statisch	—	-0,64	-0,15	-0,06	-13,5	-234	-24,5
dynamisch	bzip2	0,13	0,11	0,00	6,7	37	5,6
dynamisch	gzip	0,18	0,06	0,00	4,8	58	7,0
dynamisch	LPAQ	0,12	0,04	0,00	3,6	37	5,1
dynamisch	LZMA	0,20	0,07	0,00	5,6	61	6,9
dynamisch	PAQ8	0,09	0,07	0,00	4,9	28	5,0
dynamisch	PPMd	0,17	0,05	0,00	4,5	54	6,1
dynamisch	PPMVC	0,15	0,06	0,00	4,4	47	5,7
dynamisch	Sequitur	0,20	0,12	0,00	8,9	64	9,4
gemischt	bzip2	0,00	0,07	0,00	4,1	-1	2,0
gemischt	gzip	0,01	0,03	0,00	1,8	1	1,6
gemischt	LPAQ	-0,02	0,02	0,00	1,5	-6	1,3
gemischt	LZMA	-0,01	0,03	0,00	2,0	-5	1,4
gemischt	PAQ8	0,00	0,03	0,00	2,2	-2	2,1
gemischt	PPMd	0,00	0,02	0,00	1,0	-2	0,9
gemischt	PPMVC	-0,02	0,01	0,00	1,0	-6	0,6
gemischt	Sequitur	0,00	0,07	0,00	4,1	-2	2,5

Tabelle 5.3: Auswirkungen der Kürzung redundanter Strukturteile bei automatenbasierter Modellierung

von der gewählten Parametereinstellung für die alternative Häufigkeitsabbildung nicht ausgeschlossen werden kann, wurde auch diese bei den Untersuchungen variiert. Dabei steigert die Entfernung der Redundanz in der Struktur bei statischer Codierung der Strukturabbildung die Kompressionsleistung unabhängig davon, welche Art der Häufigkeitsabbildung verwendet wird.

Im Fall der gemischten Codierung führt die Entfernung der Strukturredundanz ebenso zu Leistungssteigerungen, unabhängig von der Art der Häufigkeitsabbildung. Verbesserungen ergeben sich sowohl bei der durchschnittlichen Kompressionsrate als auch in der durchschnittlichen und maximalen Kompressionsgröße sowie in deren Schwankungsbreite, die durch die Standardabweichung gegeben ist. Die Aktivierung der alternativen Häufigkeitsabbildung verstärkt den Effekt.

Bei der dynamischen Codierung wirkt sich die alternative Häufigkeitsabbildung ebenfalls nicht darauf aus, ob die Verarbeitung der Strukturredundanz von Vor-

Konfiguration			Differenz Kompressionsrate [%]			Differenz kompr. Größe [Byte]		
Codierungs-art	Backend-kompressor	Alternative Häufigkeits-abbildung	minimal	durch-schnittl.	maximal	durch-schnittl.	maximal	Std.Abw.
statisch	—	ja	-0,68	-0,22	0,00	-18,7	-310	-32,8
statisch	—	nein	-0,83	-0,31	-0,08	-27,9	-448	-48,0
dynamisch	bzip2	ja	0,12	0,02	0,00	1,7	37	3,0
dynamisch	gzip	ja	0,13	0,07	0,00	4,6	41	5,4
dynamisch	LPAQ	ja	-0,08	-0,05	0,00	-4,6	-26	-6,9
dynamisch	LZMA	ja	0,13	0,06	0,00	4,7	42	5,9
dynamisch	PAQ8	ja	0,10	0,08	0,00	6,4	29	6,9
dynamisch	PPMd	ja	-0,19	-0,05	0,00	-4,2	-60	-7,0
dynamisch	PPMVC	ja	0,09	-0,01	0,00	0,0	28	1,9
dynamisch	Sequitur	ja	0,18	0,17	0,05	11,2	54	10,8
dynamisch	bzip2	nein	0,12	0,05	0,00	3,3	33	3,5
dynamisch	gzip	nein	0,14	0,09	0,00	5,8	44	6,5
dynamisch	LPAQ	nein	-0,13	-0,06	0,00	-5,1	-34	-7,1
dynamisch	LZMA	nein	0,17	0,07	0,00	5,2	52	6,3
dynamisch	PAQ8	nein	0,10	0,09	0,00	6,2	31	6,6
dynamisch	PPMd	nein	-0,07	-0,01	0,00	-1,3	-23	-2,0
dynamisch	PPMVC	nein	0,10	0,01	0,00	1,0	31	2,1
dynamisch	Sequitur	nein	0,16	0,09	0,00	5,9	52	6,5
gemischt	bzip2	ja	-0,77	-0,13	0,00	-12,5	-244	-24,0
gemischt	gzip	ja	-0,80	-0,17	0,00	-15,2	-254	-25,9
gemischt	LPAQ	ja	-0,80	-0,14	0,00	-13,2	-255	-25,2
gemischt	LZMA	ja	-0,80	-0,17	0,00	-14,5	-252	-25,4
gemischt	PAQ8	ja	-0,77	-0,11	0,00	-10,9	-243	-22,6
gemischt	PPMd	ja	-0,79	-0,16	0,00	-14,5	-251	-25,9
gemischt	PPMVC	ja	-0,80	-0,16	0,00	-14,7	-253	-25,9
gemischt	Sequitur	ja	-0,82	-0,14	0,00	-13,1	-258	-26,0
gemischt	bzip2	nein	-0,25	-0,03	0,00	-3,4	-80	-7,1
gemischt	gzip	nein	-0,31	-0,04	0,00	-4,5	-91	-8,7
gemischt	LPAQ	nein	-0,32	-0,05	0,00	-5,0	-90	-8,9
gemischt	LZMA	nein	-0,30	-0,05	0,00	-4,4	-93	-8,3
gemischt	PAQ8	nein	-0,28	-0,01	0,00	-2,0	-87	-5,9
gemischt	PPMd	nein	-0,43	-0,07	0,00	-7,2	-117	-12,5
gemischt	PPMVC	nein	-0,33	-0,08	0,00	-7,4	-93	-11,4
gemischt	Sequitur	nein	-0,31	0,00	0,00	-2,1	-95	-7,3

Tabelle 5.4: Auswirkungen der Kürzung redundanter Strukurteile bei baumbasierter Modellierung

teil ist. Hingegen führt die Wahl des Universalkompressors zu unterschiedlichen Effekten. Die meisten Kompressoren erzeugen kompaktere Codierungen, wenn die Struktur nicht durch Zeiger ersetzt wurde. Ausnahmen dazu sind *LPAQ* und *PPMd*. Bei diesen Verfahren führt die Vorverarbeitung der Struktur unabhängig von der Einstellung für die alternative Häufigkeitsabbildung zu einer höheren Kompressionsleistung.

Zusammenfassung

Die Untersuchung zeigt, dass die Art der Häufigkeitsabbildung keinen Einfluss darauf hat, ob die Kürzung redundanter Strukturteile von Vorteil ist. Die Eignung der Verarbeitung der Strukturredundanz in den verschiedenen Parameterkonstellationen ist in Tabelle 5.5 zusammengefasst. Bei der statischen Codierung ist die Kürzung der Redundanz in der Struktur immer von Vorteil. Für die gemischte Codierung trifft dies nur auf die automatenbasierte, nicht aber auf die baumbasierte Modellierung zu. Im Bereich der dynamischen Codierung wirkt sich die Kürzung redundanter Strukturteile im Allgemeinen nachteilig und nur in Verbindung mit den Kompressoren *LPAQ* und *PPMd* positiv aus.

Codierungsart	Modellierungsart	
	automatenbasiert	baumbasiert
statisch	Vorteil	Vorteil
dynamisch	Nachteil	Allg. Nachteil, Vorteil nur bei LPAQ u. PPMd
gemischt	Nachteil	Vorteil

Tabelle 5.5: Effekt der Kürzung redundanter Strukturteile

5.2.1.2 Einfluss der alternativen Häufigkeitsabbildung

Auf Basis der bisherigen Erkenntnisse zum Einfluss der Kürzung redundanter Strukturteile auf nachgelagerte Universalkompressoren kann der Nutzen der alternativen Häufigkeitsabbildung bestimmt werden. Wie eingangs beschrieben, hat die alternative Häufigkeitsabbildung bei der automatenbasierten Modellierung keine Auswirkung, weswegen sie in der Zusammenfassung der Ergebnisse in Tabelle 5.7 als neutral gekennzeichnet wird.

Konfiguration			Differenz Kompressionsrate [%]			Differenz kompr. Größe [Byte]		
Codierungsart	Backendkompressor	Strukturredundanz verarbeiten	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.
statisch	—	ja	-0,03	-0,21	-0,50	-9,4	-9	0,3
dynamisch	bzip2	nein	0,01	-0,02	-0,04	-0,6	1	0,5
dynamisch	gzip	nein	0,01	0,09	0,11	4,2	2	0,1
dynamisch	LPAQ	ja	-0,01	-0,02	-0,04	-1,3	-10	-0,4
dynamisch	LZMA	nein	0,02	0,08	0,07	3,1	4	-1,0
dynamisch	PAQ8	nein	0,00	0,06	0,04	2,5	1	-0,3
dynamisch	PPMd	ja	-0,01	0,01	0,07	0,5	-2	-0,5
dynamisch	PPMVC	nein	0,00	0,03	0,04	1,0	1	-0,2
dynamisch	Sequitur	nein	-0,01	0,04	0,11	1,2	-1	-1,8
gemischt	bzip2	ja	-0,07	-0,13	-0,04	-6,7	-22	-2,8
gemischt	gzip	ja	-0,04	-0,35	-0,09	-16,3	-21	0,2
gemischt	LPAQ	ja	0,00	-0,03	-0,08	-1,0	-11	0,0
gemischt	LZMA	ja	-0,04	-0,27	-0,13	-12,3	-16	0,3
gemischt	PAQ8	ja	-0,02	-0,19	-0,22	-8,8	-7	0,3
gemischt	PPMd	ja	-0,01	-0,13	0,00	-5,9	-23	-0,6
gemischt	PPMVC	ja	-0,02	-0,12	-0,14	-5,5	-18	-0,2
gemischt	Sequitur	ja	-0,04	-0,17	-0,07	-8,4	-15	-3,1

Tabelle 5.6: Auswirkungen der alternativen Häufigkeitsabbildung bei baumbasierter Modellierung

Der Einfluss der alternativen Häufigkeitsabbildung ist bei der baumbasierten Modellierung nicht einheitlich, wie der Tabelle 5.6 zu entnehmen ist. Bei der statischen und gemischten Codierung ist sie von Vorteil. Hier verbessert sie die Durchschnittssowie Maximalwerte und senkt die Streuung. Im Bereich der dynamischen Codierung ist sie hingegen für die meisten Kompressoren von geringfügigem Nachteil. Lediglich bei der Verwendung von *LPAQ* und *bzip2* ergeben sich leichte Leistungs-

steigerungen. Es ist festzuhalten, dass die Unterschiede bei der dynamischen Codierung im Gesamtkontext keine Auswirkungen haben.

Codierungsart	Modellierungsart	
	automatenbasiert	baumbasiert
statisch	neutral	Vorteil
dynamisch	neutral	Allg. Nachteil, Vorteil nur bei *LPAQ* u. *bzip2*
gemischt	neutral	Vorteil

Tabelle 5.7: Effekt der alternativen Häufigkeitsabbildung

5.2.2 Varianten der Strukturkompression

Im nächsten Schritt der Untersuchung werden die aggregierten Messwerte verwendet, um die Leistungsfähigkeit der verschiedenen Konzepte zur schemabasierten Strukturkompression zu vergleichen. Die aggregierten Messwerte der 136 verschiedenen Konfigurationsmöglichkeiten sind im Anhang A tabellarisch dargestellt. Zur besseren Übersicht wurden dabei für die Laufzeit nur Durchschnittswerte angegeben.

Tabelle 5.8 stellt eine reduzierte Version der gesamten Ergebnisse dar, in der nur die Daten enthalten sind, die bezüglich der obigen Erkenntnisse über den Einfluss der alternativen Häufigkeitsabbildung und der Kürzung redundanter Strukturteile als vorteilhaft identifiziert wurden. Bei der automatenbasierten Modellierung wurde die alternative Häufigkeitsabbildung trotz aktiviertem Parameter in keinem Fall angewendet, weswegen die Parametereinstellung unerheblich ist und in der entsprechenden Spalte der Tabelle „j/n“ (für ja/nein) angegeben wird.

Die Werte in Tabelle 5.8 zeigen, dass die statische Codierung mit alternativer Häufigkeitsabbildung und Reduzierung der Strukturredundanz, unabhängig von der gewählten Art der Modellierung, den anderen Konfigurationsvarianten überlegen ist.

Konfiguration					durchschnittl. Kompressionsrate [%]	komprimierte Größe [Byte]			Laufzeit [ms]		
Modellierungsart	Codierungsart	Backendkompressor	Alt. Häufigkeitsabbildung	Strukturredundanz verarbeiten		durchschnittl.	maximal	Std.Abw.	durchschnittl.	maximal	Std.Abw.
Automat	statisch	—	j/n	ja	0,78	38,7	75	6,0	29,7	141	12,2
Baum	statisch	—	ja	ja	0,85	41,8	76	6,1	20,6	47	7,9
Baum	gemischt	PPMd	ja	ja	0,94	46,7	90	8,4	55,5	907	24,0
Baum	gemischt	LPAQ	ja	ja	0,97	48,4	87	8,6	207,4	843	20,7
Baum	dynamisch	LPAQ	ja	ja	1,05	51,9	88	9,0	206,9	641	21,2
Automat	gemischt	LPAQ	j/n	nein	1,15	55,6	102	6,6	217,5	422	18,1
Baum	dynamisch	PPMd	nein	ja	1,20	59,4	113	10,2	54,8	2391	57,5
Baum	gemischt	Sequitur	ja	ja	1,21	59,7	101	8,7	496,3	813	61,8
Baum	gemischt	LZMA	ja	ja	1,25	61,6	104	9,1	37,5	125	12,1
Automat	dynamisch	LPAQ	j/n	nein	1,29	61,7	72	4,1	216,6	406	18,7
Automat	gemischt	PPMd	j/n	nein	1,27	61,9	111	7,1	63,8	203	15,2
Baum	gemischt	PAQ8	ja	ja	1,25	62,2	111	11,5	108,3	375	18,6
Baum	gemischt	gzip	ja	ja	1,32	64,5	107	9,0	22,1	78	8,6
Baum	gemischt	PPMVC	ja	ja	1,39	67,7	110	8,4	59,8	234	13,1
Automat	dynamisch	PPMd	j/n	nein	1,44	68,5	76	3,3	64,0	2766	66,0
Automat	gemischt	Sequitur	j/n	nein	1,43	69,9	124	8,9	524,4	844	76,8
Baum	dynamisch	PAQ8	nein	nein	1,48	70,6	79	3,8	135,4	750	36,0
Baum	dynamisch	Sequitur	nein	nein	1,56	75,8	97	8,8	465,9	765	56,7
Baum	dynamisch	PPMVC	nein	nein	1,59	77,1	97	7,4	58,8	359	15,3
Automat	dynamisch	Sequitur	j/n	nein	1,66	79,5	95	5,9	511,3	812	67,1
Automat	gemischt	gzip	j/n	nein	1,65	80,0	131	7,6	28,7	109	11,2
Baum	gemischt	bzip2	ja	ja	1,64	80,1	129	10,2	24,2	94	9,5
Baum	dynamisch	LZMA	nein	nein	1,69	80,8	96	4,2	35,8	406	14,7
Automat	gemischt	PPMVC	j/n	nein	1,74	83,5	135	7,0	69,6	359	16,7
Automat	gemischt	LZMA	j/n	nein	1,77	84,9	134	6,4	44,3	328	16,3
Baum	dynamisch	gzip	nein	nein	1,79	85,8	104	5,4	20,8	141	8,6
Automat	gemischt	PAQ8	j/n	nein	1,84	88,0	134	6,9	128,2	843	27,8
Automat	dynamisch	PPMVC	j/n	nein	1,88	89,5	100	3,5	68,6	234	15,2
Automat	dynamisch	gzip	j/n	nein	1,92	91,6	104	4,9	28,1	78	10,2
Automat	gemischt	bzip2	j/n	nein	1,93	93,3	153	9,8	31,3	94	11,5
Automat	dynamisch	PAQ8	j/n	nein	2,01	95,2	100	3,1	133,0	266	23,9
Baum	dynamisch	bzip2	ja	nein	1,95	95,9	137	13,3	22,8	63	8,5
Automat	dynamisch	LZMA	j/n	nein	2,11	99,6	108	2,9	43,2	125	13,5
Automat	dynamisch	bzip2	j/n	nein	2,12	102,7	141	9,7	30,7	235	12,6

Tabelle 5.8: Aggregierte Messwerte relevanter Konfigurationsvarianten der Strukturkompression

Die beste durchschnittliche Kompressionsleistung liefert dabei der automatenbasierte Ansatz. Im Hinblick auf die maximale Größe der komprimierten Dokumente liefert er mit vernachlässigbaren drei Byte Rückstand das zweitbeste Ergebnis. Die beste Leistung in diesem Bereich erzeugt die Kombination aus automatenbasierter Modellierung und dynamischer Codierung durch *LPAQ*. Diese Konfiguration ist jedoch durchschnittlich über siebenmal langsamer als die statische Codierung der automatenbasierten Abbildung und liegt zudem in der durchschnittlichen Kompressionsleistung um 59 % zurück.

5.2.3 Varianten der Nutzdatenkompression

Bei der Analyse der verschiedenen Möglichkeiten zur Nutzdatenkompression werden die erhobenen Messwerte pro Konfigurationsvariante verdichtet. Die Tabelle der resultierenden 160 Datensätze, aufsteigend nach dem Durchschnitt der komprimierten Größe sortiert, findet sich im Anhang unter Abschnitt B. Zur besseren Übersicht wurden die Informationen über die Laufzeit auch hier auf den Durchschnittswert beschränkt. Die Beschreibung der in diesem Abschnitt betrachteten Parameter findet sich unter Punkt 4.3.5.

Containerstrategie

Der Vergleich der aggregierten Daten – die in der Tabelle im Anhang B aufgelistet sind – zeigt, dass Containerstrategie 1 und damit die Kompression der gesamten Nutzdaten in einem Container im Hinblick auf die durchschnittliche Kompressionsleistung und die durchschnittliche Laufzeit in allen Parameterkonfigurationen den anderen Containerstrategien überlegen ist. Der Vorteil ist auf die geringe Größe der Nutzdaten zurückzuführen. So kann durch die Gruppierung gleichartiger Daten nur dann eine Verbesserung erzielt werden, wenn die Nutzdaten eine gewisse Größe aufweisen, die hier offensichtlich nicht erreicht wird.
Der Vorteil, die gesamten Nutzdaten in einem Block zu komprimieren, bestätigt sich auch durch den Vergleich der maximalen Größen der komprimierten Dokumente, wenngleich nicht ohne Ausnahme. Für die Universalkompressoren *bzip2*, *PPMd*

Konfiguration			durchschnittl. Kompressionsrate [%]	komprimierte Größe [Byte]			Laufzeit [ms]		
Sortierung	Backend-kompressor	gemischte Codierung		durchschnittl.	maximal	Std.Abw.	durchschnittl.	maximal	Std.Abw.
ja	LPAQ	nein	22,50	373,8	609	41,8	197,1	703	19,1
ja	LPAQ	ja	22,54	374,5	609	42,0	196,9	766	25,3
nein	LPAQ	nein	22,85	380,1	634	43,7	198,6	422	16,8
nein	LPAQ	ja	22,90	380,8	634	43,8	196,1	406	15,1
ja	PAQ8	nein	24,21	400,3	600	37,1	181,5	469	41,4
ja	PAQ8	ja	24,25	400,9	599	37,0	181,4	688	44,5
ja	PPMd	nein	24,09	401,4	683	49,2	38,7	1203	29,7
ja	PPMd	ja	24,12	401,8	684	49,2	37,8	172	10,6
nein	PPMd	nein	24,16	402,8	691	50,7	38,4	250	11,4
nein	PPMd	ja	24,16	403,0	692	50,6	38,0	828	21,6
nein	PAQ8	nein	24,55	405,8	607	37,3	177,0	453	44,3
nein	PAQ8	ja	24,58	406,2	607	37,3	186,6	1563	58,1
ja	PPMVC	nein	25,42	423,2	714	50,5	41,7	375	15,3
ja	PPMVC	ja	25,45	423,7	715	50,5	41,4	688	22,1
nein	PPMVC	nein	25,57	426,0	721	51,4	41,7	391	13,6
nein	PPMVC	ja	25,62	426,7	723	51,4	40,3	375	13,0
ja	gzip	nein	25,93	431,4	746	51,1	0,7	16	3,2
ja	gzip	ja	26,00	432,6	748	51,2	0,8	16	3,4
ja	LZMA	nein	26,23	435,5	719	48,4	13,7	219	11,4
ja	LZMA	ja	26,25	435,9	720	48,3	13,6	219	9,6
nein	gzip	nein	26,61	443,9	776	56,2	0,8	16	3,5
nein	gzip	ja	26,64	444,4	777	56,1	0,7	32	3,2
nein	LZMA	nein	26,97	448,7	755	52,5	17,2	125	7,7
nein	LZMA	ja	26,99	448,9	755	52,5	13,6	187	9,6
nein	Sequitur	nein	27,95	467,0	833	61,8	446,7	1500	78,6
nein	Sequitur	ja	27,97	467,3	834	61,8	415,8	750	71,3
ja	Sequitur	nein	28,33	475,1	896	68,7	419,1	719	75,9
ja	Sequitur	ja	28,35	475,3	897	68,7	433,6	735	77,0
ja	bzip2	ja	34,34	574,8	989	78,2	3,0	16	6,2
ja	bzip2	nein	34,34	574,9	986	78,5	2,7	16	5,9
nein	bzip2	nein	34,40	576,6	1018	81,7	3,3	47	6,5
nein	bzip2	ja	34,43	577,0	1003	81,4	3,0	16	6,2

Tabelle 5.9: Aggregierte Messwerte relevanter Konfigurationsvarianten der Nutzdatenkompression

und *Sequitur* erzielt die Containerstrategie 2 (Zerlegung in strukturierte Zahlenwerte und textuelle Daten) geringere Maximalwerte als Strategie 1, unabhängig davon, ob die Sortierung oder die gemischte Codierung aktiviert ist. Jedoch beträgt der Vorsprung nur wenige Byte und ist daher vernachlässigbar gering.
Entsprechend der Erkenntnis bezüglich der Containerstrategien stellt Tabelle 5.9 die aggregierten Daten beschränkt auf die Containerstrategie 1 dar. Sie ist aufsteigend nach der durchschnittlichen Kompressionsgröße sortiert.

Sortierung

Die spezielle Sortierung der Daten, wie sie in Abschnitt 4.3.5 beschrieben ist, erzeugt gegenüber der Dokumentenreihenfolge in nahezu jeder Konfiguration eine Verbesserung (siehe Tabelle 5.9). Dies gilt gleichwohl für die durchschnittliche und maximale Kompressionsleistung als auch für deren Streuung. Die einzige Ausnahme bildet die Verwendung des Universalkompressors *Sequitur*, bei dem die Dokumentenreihenfolge in jedem Fall bessere Werte liefert.

Gemischte Codierung

Die Werte in Tabelle 5.9 zeigen, dass die Verwendung der gemischten Codierung die Kompressionsleistung kaum beeinflusst. Die Unterschiede liegen sowohl bei den durchschnittlichen als auch bei den maximalen Kompressionsgrößen bei höchstens 3 Byte. Im Durchschnitt verschlechtert sich die Leistung beim Einsatz der gemischten Codierung in fast allen Konfigurationsvarianten.

Backendkompressor

Entsprechend der bisherigen Erkenntnisse berücksichtigt der Vergleich der verschiedenen Universalkompressoren zur Nutzdatenverarbeitung nur diejenigen Konfigurationen, bei denen die Nutzdaten sortiert werden und die gemischte Codierung deaktiviert ist.
Die im Durchschnitt beste Kompressionsleistung erzeugt der Universalkompressor *LPAQ*. Im Hinblick auf die maximale Größe liegt er 1,5 % hinter *PAQ8*, der aber im

Durchschnitt 7,1 % größere Dokumente erzeugt. Die beste Laufzeit hat mit durchschnittlich 0,69 ms und maximal 16 ms *gzip*. Die komprimierten Dokumente sind damit im Durchschnitt um 15,4 % und in Bezug auf die maximale Größe um 22,5 % größer. Die höhere Kompressionsleistung bei *LPAQ* und *PAQ8* ist mit einer höheren Laufzeit verbunden, die durchschnittlich bei 197,1 ms bzw. 181,5 ms liegt.
Die Beurteilung, welche Laufzeit akzeptabel ist, wird in vorliegender Arbeit nicht näher betrachtet und bedarf somit weiterführender Forschung. Als Kompromiss zwischen *gzip* und *LPAQ* bietet sich *PPMd* an. Es ist mit durchschnittlich 38,7 ms mehr als fünfmal schneller als *LPAQ* und liegt beim Durchschnitt der komprimierten Größe um 7,4 % sowie bei der maximalen Größe um 12,2 % zurück.

Zusammenfassung

Die Messergebnisse verdeutlichen, dass es unabhängig von anderen Parametereinstellungen am besten ist, die gesamten Nutzdaten in einem Container zu komprimieren und damit Containerstrategie 1 zu verwenden. Zur Verarbeitung des Containers sind drei Kompressoren sinnvoll: *LPAQ* aufgrund der besten Kompressionsleistung, *gzip* aufgrund der schnellsten Kompression und *PPMd* als ausgewogener Kompromiss dazwischen. Bei allen drei Kompressoren ist es von Vorteil, die Nutzdaten zu sortieren und keine gemischte Codierung zu verwenden.

5.2.4 Ergebnis des Vergleichs der Konzepte

Im Rahmen der Analyse dieses Abschnitts konnten für sechs Parameter optimale Einstellungen bestimmt werden. Bei der Modellierungsart und den zu verwendenden Nutzdatenkompressoren konkurrieren Laufzeit und Kompressionsleistung, weshalb drei verschiedene Parameterkonstellationen sinnvoll sind. Für die höchste Kompressionsleistung ist die automatenbasierte Modellierung und der Nutzdatenkompressor *LPAQ* zu verwenden. Die schnellste Kompression liefert die baumbasierte Modellierung mit *gzip* als Nutzdatenkompressor. Einen Kompromiss aus Laufzeit und Kompressionsleistung bildet die baumbasierte Modellierung in Ver-

bindung mit *PPMd* zur Kompression der Nutzdaten. Einen gesamten Überblick über die ermittelten Konfigurationen enthält Tabelle 5.10.

Parameter	Einstellungen optimiert für		
	Leistung	Laufzeit	Kompromiss
Modellierungsart	Automat	Baum	Baum
Codierungsart	statisch	statisch	statisch
Alt. Häufigkeitsabbildung	ja	ja	ja
Strukturredundanz kürzen	ja	ja	ja
Nutzdatenkompressor	LPAQ	gzip	PPMd
Containerstrategie	1	1	1
Sortierung	ja	ja	ja
Gemischte Codierung	nein	nein	nein

Tabelle 5.10: Parameterwerte von PSXC für optimale Konzeptkombinationen

5.3 Leistungsvergleich ausgewählter Kompressionsverfahren

In diesem zweiten Teil der Fallstudie werden die Echtdaten verwendet, um die Leistungsfähigkeit verschiedener Verfahren zur XML-Kompression zu vergleichen. Indem die im vorherigen Kapitel ermittelten besten drei Konfiguration von *PSXC* in die Untersuchung mit einfließen, kann analysiert werden, wie gut die Kompressionsleistung der bestehenden Verfahren ist und ob diese übertroffen werden kann.

5.3.1 Überblick über die untersuchten Verfahren

In den Vergleich gehen acht schemaneutrale und neun schemabasierte XML-Kompressoren sowie die acht in Abschnitt 5.1.3 genannten Universalkompressoren ein. Die folgenden Ausführungen thematisieren die Implementierungen und Konfigurationen der XML-spezifischen Verfahren.

5.3.1.1 Schemaneutrale XML-Kompressoren

In Kapitel 3 wurden acht schemaneutrale Verfahren identifiziert, die in dieser Fallstudie zu berücksichtigen sind (siehe Abschnitt 3.6.1). Die Verfahren *XMill*, *XWRT*, *XBzip*, *XMLPPM*, *SCMPPM* und *Exalt* liegen in implementierter Form vor, während für *XSeq* und *DAG+BSBC* keine öffentlich zugängliche Implementierung gefunden wurde und diese daher nach den jeweiligen Verfahrensbeschreibungen aus Lin u. a.(2005) bzw. Böttcher/Hartel/Heinzemann (2009) in Java implementiert wurden.

Die verwendeten Implementierungen der XML-Kompressionsverfahren sowie deren Bezugsquellen sind in Tabelle 5.11 aufgelistet. Die Kompressoren, deren Leistungsfähigkeit über Parameter beeinflusst werden kann, werden so konfiguriert, dass ihre Kompressionsleistung maximal ist. Dementsprechend wird *XWRT* mit dem Parameter „`-l14`“, *XMLPPM* und *SCMPPM* mit „`-l -9`“ und *XMill* mit „`-P -9`“ ausgeführt.

Kompressor	Bezugsquelle
XWRT v3.2	https://github.com/inikep/XWRT
XBzip v1.0	http://pages.di.unipi.it/ferragina/Libraries/xbwt-demo.zip
XMLPPM v0.98.3	https://sourceforge.net/projects/xmlppm
SCMPPM v0.93.3	http://www.infor.uva.es/ jadiego/files/scmppm-0.93.3.zip
XMill v0.9.1	http://homes.cs.washington.edu/ suciu/XMILL/xmill-0.9.1.tar.gz
Exalt v0.1.0	https://sourceforge.net/projects/exalt

Tabelle 5.11: Bezugsquellen der untersuchten XML-Kompressoren

Alle sechs implementierten Verfahren sind unter den in Tabelle 5.11 angegebenen Bezugsquellen als Quellcode in C bzw. C++ verfügbar. Die Verfahren *XMill* und *XWRT* liegen zudem direkt als ausführbare Programme für Windows vor. *XMLPPM* und *SCMPPM* konnten unter Windows kompiliert werden, wohingegen sich *XBzip* und *Exalt* nur unter Linux kompilieren ließen. Da der in *XBzip* eingesetzte Universalkompressor *PPMd* Speicherzugriffsfehler erzeugte, wurde er durch den Universalkompressor *LPAQ* ersetzt.

5.3.1.2 Schemabasierte XML-Kompressoren

Aus dem Bereich der schemabasierten Verfahren wurden in Kapitel 3 fünf Verfahren ausgewählt, die quantitativ zu vergleichen sind (siehe Abschnitt 3.6.2). Ferner wurde dort der Bedarf für das in Kapitel 4 beschriebene parametrisierte Verfahren identifiziert. Dieses Verfahren erfüllt in diesem Teil der Fallstudie zwei Aufgaben. Zum einen geht es mit den zuvor bestimmten Parametern in den Gesamtvergleich ein. Zum anderen dient es zur Simulation der nicht in implementierter Form vorliegenden schemabasierten Verfahren.

EXI

EXI ist das einzige in implementierter Form vorliegende schemabasierte Verfahren. Von den verschiedenen Realisierungen des Standardformats[8] wird die Java-basierte Implementierung des EXIficient Projekts, das als Open Source Projekt von der Siemens AG initiiert wurde, in der Version 0.9.2 verwendet.[9]
Wenngleich das Verfahren auch schemaneutral verwendet werden kann, wird es hier nur in seiner vollständig schemabasierten Form untersucht. Standardmäßig kommt bei *EXI* der Universalkompressor *gzip* als Backend zum Einsatz. Da *EXI* mit beliebigen Universalkompressoren kombinierbar ist, wird neben der Standardvariante auch *LPAQ* als nachgelagerter Kompressor betrachtet.

Simulierte Verfahren

Die nicht in implementierter Form vorliegenden schemabasierten Verfahren werden durch das parametrisierte Verfahren simuliert. Neben den vier, in Abschnitt 3.6.2 als untersuchungsrelevant identifizierten Verfahren werden zusätzlich vier weitere Verfahren in der Fallstudie berücksichtigt, obwohl diese als nicht leistungsstark genug klassifiziert wurden. Diese Entscheidung wird damit begründet, dass die Verfahren durch das parametrisierte Verfahren simulierbar sind und auf diese Weise das qualitative Ausschlussverfahren auf quantitativer Ebene bekräftigt werden kann.

[8] Eine Liste möglicher Implementierungen findet sich unter https://www.w3.org/XML/EXI.
[9] Die Projektseite findet sich unter http://exificient.github.io/java und der Quellcode kann von https://github.com/EXIficient/exificient heruntergeladen werden.

Die Simulation der zu untersuchenden Verfahren erfolgt mit verschiedenen Konfigurationen von *PSXC*. Ein Teil der Konfigurationen ist für alle Verfahren identisch. Da die Nutzdaten bei allen Verfahren in der Dokumentenreihenfolge in die Datencontainer gelegt werden, ist die Sortierung immer deaktiviert. Gleiches gilt für die alternative Häufigkeitsabbildung, da dies ein Konzept ist, das speziell im Rahmen von *PXSC* entwickelt wurde.
Die Reduzierung der Strukturredundanz ist nur bei *KST+DAG* aktiviert. Zur Abbildung der Zeiger wird dort die Variante mit Knotenmarkierungen verwendet, weshalb die dynamische Ermittlung der kompakteren Zeigerabbildung deaktiviert ist.[10] Die gemischte Codierung der Nutzdaten ist nur bei *XSDS* zu finden[11] und daher bei allen anderen Verfahren deaktiviert.

Verfahren	Modellierung	Strukturcodierung	Nutzdatencontainer	Nutzdatenkompressor
rngzip	Automat	statisch	ein Container	LPAQ
DPDT-L	Automat	dynamisch mit PPMd (order: 9)	ein Container	PPMd (order: 9)
KST+DAG	Baum	statisch	ein Container	LPAQ
XAUST	Automat	dynamisch mit PPMd (order: 4)	pro Name	PPMd (order: 4)
XSDS	Baum	statisch	pro Name	LPAQ
XCQ	Baum	statisch Alternativen bytebasiert Anzahlen als Zählcode	pro Pfad	LPAQ
SCA	Baum	statisch Anzahlen als Beta-Code	ein Container	LPAQ

Tabelle 5.12: Simulierte Verfahren und die entsprechenden Konfigurationen von *PSXC*

Die weiteren Konfigurationen von `PSXC` zur Simulation der verschiedenen Verfahren ist in Tabelle 5.12 zusammengefasst.[12] Die Simulation von zwei Verfahren, die

[10] Siehe dazu Abschnitt 4.2.3.
[11] Siehe dazu Abschnitt 4.3.5.
[12] Zur Beschreibung der Bedeutung der Parameter siehe Abschnitt 4.3.

eigentlich nicht leistungsstark genug sind, erfordert die geringfügige Erweiterung von *PSXC* um Konzepte, die aufgrund unzureichender Leistungsfähigkeit nicht berücksichtigt worden sind. So werden bei *XCQ* die Codewörter zur Codierung von Elementalternativen mit führenden Nullen auf volle Bytes aufgefüllt und Anzahlen als Zählcode abgebildet.[13] Ferner werden dort die Nutzdaten nach Pfaden im Dokument aufgeteilt. Bei *SCA* werden Anzahlen als Blockcode abgebildet.[14] Dementsprechend wurde die statische Codierung der Strukturabbildung um die speziellen Codierungen erweitert und eine Containerstrategie für die Aufteilung nach Pfaden hinzugefügt.

Mit Ausnahme der PPM-basierten Verfahren wird bei allen Verfahren *LPAQ* als Nutzdatenkompressor verwendet, da dieser in Abschnitt 5.2.3 als leistungsstärkste Wahl identifiziert wurde. Die Verfahren *DPDT-L* und *XAUST* sind von ihrer Konzeption an die PPM-basierten Nutzdatenkompressoren gebunden. Bei *XAUST* ist für die Kompression der Struktur- und Nutzdaten ein PPM-Verfahren vorgesehen, das hier mit *PPMd* simuliert wird.

DPDT-L basiert wie auch *XMLPPM* auf dem Kompressionsverfahren *PPMD+*[15], für das keine Implementierung verfügbar ist. Da bei der Implementierung von *XMLPPM* der Kompressor *PPMd* verwendet wird, wird dieser auch hier herangezogen. Die Simulation von *DPDT-L* ist insofern eingeschränkt, da die Nutzdatenkompression grundsätzlich die Struktur als Kontext berücksichtigen müsste, was bei *PSXC* aber nicht vorgesehen ist.

Die bei *DTDPPM* verwendete Trennung der Nutzdaten in Attribut- und Elementwerte kann in *PSXC* nicht abgebildet werden. Somit ist es nicht möglich, das Verfahren zu simulieren. Auf konzeptioneller Ebene ist es jedoch ähnlich zu *DPDT-L* und der Unterschied bezüglich der Strategie zur Kompression der Nutzdaten ist sehr gering.

[13] Für diesen und den nächsten Satz vgl. Ng u. a.(2006), S. 429-431.
[14] Vgl. Levene/Wood (2002), S. 64.
[15] Vgl. Harrusi/Averbuch/Yehudai (2006b), S. 404 u. 409.

5.3.2 Ergebnis des Vergleichs der Kompressionsverfahren

Bei dem Vergleich wurden die zu untersuchenden Kompressionskonfigurationen auf die Echtdaten angewendet und die Messwerte pro Konfiguration verdichtet. Die resultierenden Ergebnisse sind in Tabelle 5.13, aufsteigend nach der maximalen Kompressionsgröße sortiert, dargestellt. In der Spalte *Gruppe* ist angegeben, ob ein Verfahren ein Universalkompressor (U) oder ein schemaneutraler (X) bzw. schemabasierter (SX) XML-Kompressor ist. Bei Letzteren wird das Gruppenkürzel um ein hochgestelltes „S“ erweitert, wenn es sich um ein simuliertes Verfahren handelt. Wurde ein Verfahren in unterschiedlichen Varianten untersucht, ist die jeweilige Konfiguration in eckigen Klammern dem Namen des Verfahrens beigefügt.

PSXC in der leistungsstärksten Konfiguration erzeugt sowohl im Durchschnitt als auch im Maximum die beste Kompressionsleistung. Der Vorsprung auf die Simulation von *KST+DAG* beträgt 6,0 % bei der durchschnittlichen und 7,0 % bei der maximalen Größe der komprimierten Dokumente. Das leistungsstärkste, in implementierter Form vorliegende Verfahren ist *EXIficient*. Die komprimierten Dokumente sind bei diesem im Durchschnitt 14,7 % (61 Byte) größer als bei *PSXC* in der leistungsstärksten Konfiguration. Im Hinblick auf die maximale Größe liegt es um 107 Byte und damit 17,2 % zurück. Welche Auswirkungen dieser Rückstand auf die Größe der optischen Codierung letztlich hat, wird in Abschnitt 5.4 betrachtet. Der Vergleich bestätigt zudem die in Abschnitt 3.3.2.3 begründete Vermutung, dass schemabasierte Verfahren zur Kompression von Geschäftsdokumenten besser geeignet sind als schemaneutrale Verfahren. Nahezu alle schemabasierten Verfahren sind leistungsstärker als das schemaneutrale Verfahren mit der höchsten Leistung. Die Nutzung des Schemas führt in der Fallstudie dazu, dass die Größe der komprimierten Dokumente nahezu halbiert werden kann.

Ferner zeigen die aggregierten Messwerte in Tabelle 5.13, dass das Ziel XML-spezifischer Kompressionsverfahren, höhere Kompressionsraten als Universalkompressoren zu erzeugen[16], nur durch Nutzung des Schemas erreicht wird. Die Leis-

[16] Vgl. beispielsweise Sakr (2009a) S. 50.

Gruppe	Verfahren	Kompressionsrate [%]			komprimierte Größe [Byte]			Laufzeit [ms]		
		min.	durchschnittl.	max.	durchschnittl.	max.	Std-Abw.	durchschnittl.	max.	Std-Abw.
SX	PSXC [A LPAQ]	1,57	6,38	8,91	416	676	48	180	390	13
SX^S	KST+DAG	1,61	6,78	9,28	441	715	47	181	422	11
SX	PSXC [T PPMd]	1,81	6,83	9,44	446	756	55	36	297	11
SX	EXIficient [LPAQ]	1,88	7,32	9,95	477	783	55	186	266	19
SX^S	DPDT-L	1,92	7,17	9,87	468	795	57	62	203	11
SX	PSXC [T gzip]	1,95	7,30	10,09	476	819	57	21	62	9
SX	EXIficient [gzip]	2,21	8,28	11,48	539	890	59	18	438	16
SX^S	rngzip	2,22	6,66	9,04	441	909	75	179	406	11
SX^S	SCA	2,29	6,60	8,88	438	929	78	178	297	9
SX^S	XAUST	2,62	11,65	16,39	752	1087	61	753	1079	42
U	PAQ8 [-4]	3,14	17,41	25,60	1111	1329	44	880	5321	483
X	XBzip	3,38	17,51	25,52	1122	1413	61	108	349	30
X	XWRT	3,48	18,14	26,62	1161	1480	58	214	1141	54
U	LPAQ	3,55	18,24	26,30	1169	1480	63	261	969	71
U	PAQ8 [-3]	3,54	18,25	26,62	1168	1481	59	220	922	92
SX^S	XSDS	3,71	13,23	18,28	861	1495	96	5392	5610	86
X	XMLPPM	3,67	19,44	28,41	1241	1511	51	41	172	11
SX^S	XCQ	3,86	15,98	21,12	1032	1555	84	7857	8890	183
U	PPMVC	3,93	19,48	27,98	1250	1636	73	41	203	17
X	XMill	4,36	22,67	32,45	1449	1790	66	29	453	18
U	PPMd	4,55	19,31	27,37	1247	1833	99	40	174	18
U	LZMA	4,65	22,42	31,86	1436	1875	76	21	156	12
U	Sequitur	4,77	23,72	34,00	1518	1947	77	476	1097	159
U	gzip	5,19	22,90	32,00	1473	2092	100	0	16	2
X	Exalt	5,95	25,04	34,62	1624	2400	149	11	59	2
U	bzip2	5,96	26,74	38,22	1728	2404	139	4	47	7
X	XSeq	8,08	45,67	61,86	2904	3269	90	19101	20282	1286
X	DAG+BSBC	8,43	48,29	65,65	3071	3438	96	95	266	24
X	SCMPPM	9,11	49,77	65,20	3178	3679	133	53	359	14

Tabelle 5.13: Vergleich ausgewählter Kompressoren

tungsfähigkeit der schemaneutralen Verfahren liegt bei der Kompression von Geschäftsdokumenten in etwa auf dem Niveau von Universalkompressoren.
Im schemaneutralen Bereich dominieren Lösungen, die PAQ-Verfahren einsetzen. Dies gilt sowohl im universalen Bereich mit *PAQ8* oder *LPAQ* als auch im XML-spezifischen Bereich mit *XBzip* und *XWRT*, deren untersuchte Versionen *LPAQ* als Backendkompressor verwenden. Die Leistung von *PAQ8* in Leistungsstufe 4 ist jedoch kritisch zu betrachten, da die Laufzeit mit durchschnittlich 880 ms und maximal 5.321 ms verhältnismäßig hoch ist und für den praktischen Einsatz mit sehr großer Wahrscheinlichkeit ein Problem darstellt. Zudem ist im Vergleich zu den anderen Verfahren der Unterschied der Kompressionsleistung zu gering, als dass damit die höhere Laufzeit zu rechtfertigen ist.
Von den Universalkompressoren erzeugt *LPAQ* die beste Kompressionsleistung mit vertretbarer Laufzeit von durchschnittlich 261 ms und maximal 969 ms. Der Vorsprung der beiden besten schemaneutralen XML-Kompressoren *XBzip* und *XWRT* auf *LPAQ* ist vernachlässigbar gering. Da beide Verfahren *LPAQ* als Backend verwenden, zeigt sich, dass die XML-spezifische Vorverarbeitung bei Geschäftsdokumenten kaum Vorteile liefert.

5.4 Auswirkung von Kompressionsleistungsunterschieden auf die optische Codierung

Die Wahl des Kompressionsverfahrens, das zur Abbildung von Geschäftsdokumenten in optischen Codes verwendet wird, ist vom Szenario des praktischen Einsatzes abhängig. In geschlossenen Systemen oder bei der Konzeption proprietärer Software kann das Verfahren frei gewählt werden. Anders verhält es sich, wenn optisch codierte Geschäftsdokumente ganzheitlich zur Unterstützung eines unternehmensübergreifenden Prozesses dienen sollen, bei dem die Geschäftspartner häufig wechseln – ähnlich wie dies das ZUGFeRD-Format im Bereich der elektronischen Rechnung beabsichtigt.[17] In diesem Fall hemmt proprietäre Software durch die Bindung an einen Softwarehersteller die nötige Verbreitung in den Unternehmen.

[17] Vgl. Bergmann u. a.(2014), S. 12-20.

Es ist zu erwarten, dass Unternehmen die optische Codierung von Geschäftsdokumenten zur Unterstützung eines unternehmensübergreifenden Geschäftsprozesses schneller adaptieren, wenn diese durchgängig auf etablierten oder sogar standardisierten Bausteinen basiert. Ein neues Kompressionsverfahren einzuführen und seine Verbreitung voranzutreiben ist nur dann sinnvoll, wenn dieses die Leistungsfähigkeit der bestehenden Verfahren so stark übertrifft, dass dadurch der Prozess verbessert werden kann. Der folgende Vergleich soll deshalb zeigen, wie stark sich die Größe des optischen Codes verringert, wenn *PSXC* in der leistungsstärksten Konfiguration anstelle von etablierten Kompressionsverfahren verwendet wird.

5.4.1 Bestandteile des Vergleichs

In Abschnitt 2.3 wurde dargelegt, dass für die optische Codierung von Geschäftsdokumenten drei verschiedene 2D-Codes relevant sind. Da ein 2D-Code nicht mit jedem zusätzlich gespeicherten Byte größer wird, sondern in einer Größenstufe einen bestimmten Kapazitätsbereich abdeckt, wirkt sich letztlich nicht jeder Leistungsunterschied zwischen Kompressionsverfahren aus. Zudem führt nicht jeder Anstieg der Größe des 2D-Codes zu einer Einschränkung des Prozesses.
Aus diesem Grund beleuchtet dieser dritte Teil der Fallstudie die resultierenden Größen der drei 2D-Codes beim Einsatz verschiedener Kompressionsverfahren. Im Zentrum des Vergleichs stehen *EXI* und *PSXC*. Während ersteres das beste etablierte Verfahren und zudem noch standardisiert ist, bildet *PSXC* den Maßstab für die maximal mögliche Kompressionsleistung. Beide werden in zwei verschiedenen Konfigurationen untersucht, indem zum einen die Kompressionsleistung maximiert und *LPAQ* als Backendkompressor verwendet wird. Zum anderen wird mit dem Einsatz von *gzip* die Kompressionsgeschwindigkeit berücksichtigt.
Um den Unterschied zur Verwendung von etablierten Verfahren aus dem schemaneutralen sowie universellen Bereich zu zeigen, werden zudem *XWRT*[18] und *LPAQ*[19] berücksichtigt.

[18] Wenngleich *XBzip* geringfügig besser ist, wird dies nicht als etabliert angesehen, da die Implementierung mehr eine Demonstration des Verfahrens ist und im Gegensatz zu der von *XWRT* nicht aktiv gepflegt wird.
[19] *LPAQ* dient als Stellvertreter für Programme wie Peazip (http://www.peazip.org), die PAQ-basierte Kompression beinhalten.

Bei dem Vergleich werden für alle drei relevanten Codes – Aztec, Data Matrix und QR Code – sowie für die ausgewählten Kompressionsverfahren jeweils die mittlere und maximale Kantenlänge ermittelt. Als Datengrundlage fungieren die im Rahmen des vorherigen Abschnitts 5.3 erzeugten Werte für die durchschnittliche und maximale Größe der komprimierten Dokumente aus der Tabelle 5.13. Als Fehlertoleranzstufen der 2D-Codes werden die empfohlenen Einstellungen, wie sie in Abschnitt 2.3.2 dargelegt sind, verwendet. Die Zellengröße hat mit 0,33 mm einen üblichen Wert für den Druck mit mittlerer Informationsdichte.[20] Ob eine kleinere Zellengröße möglich oder eine größere nötig ist, ist nicht Teil dieser Untersuchung und bietet einen Ansatzpunkt für weitere Forschung.

5.4.2 Die Auswertung

Verfahren	Aztec		Data Matrix		QR	
	mittel	max.	mittel	max.	mittel	max.
PSXC [A LPAQ]	2,50	3,03	2,73	3,27	2,97	3,63
EXI [LPAQ]	2,63	3,37	3,00	3,53	3,10	3,90
PSXC [T gzip]	2,63	3,37	3,00	4,07	3,10	3,90
EXI [gzip]	2,77	3,50	3,00	4,07	3,23	4,03
XWRT	3,90	4,50	4,47	4,87	4,57	5,10
LPAQ	3,90	4,50	4,47	4,87	4,57	5,10

Tabelle 5.14: Kantenlängen in Zentimeter von Aztec, Data Matrix und QR Code für unterschiedliche Kompressionsverfahren

Die ermittelten Werte für die Größe der 2D-Codes bei den jeweils verwendeten Kompressionsverfahren sind in Tabelle 5.14 und Abbildung 5.2 dargestellt. Der Leistungsrückstand der schemabasierten Kompressoren und Universalkompressoren wirkt sich auch in der Größe der optischen Codierung deutlich aus. Die Kanten der 2D-Codes sind hier zwischen 40 % und 60 % länger als bei *PSXC* in der leis-

[20] Vgl. Lenk (2005a), S. 170.

tungsstärksten Konfiguration. Zudem verdeutlicht Abbildung 5.2 den Vorteil des Aztec Codes im Hinblick auf den Platzbedarf.

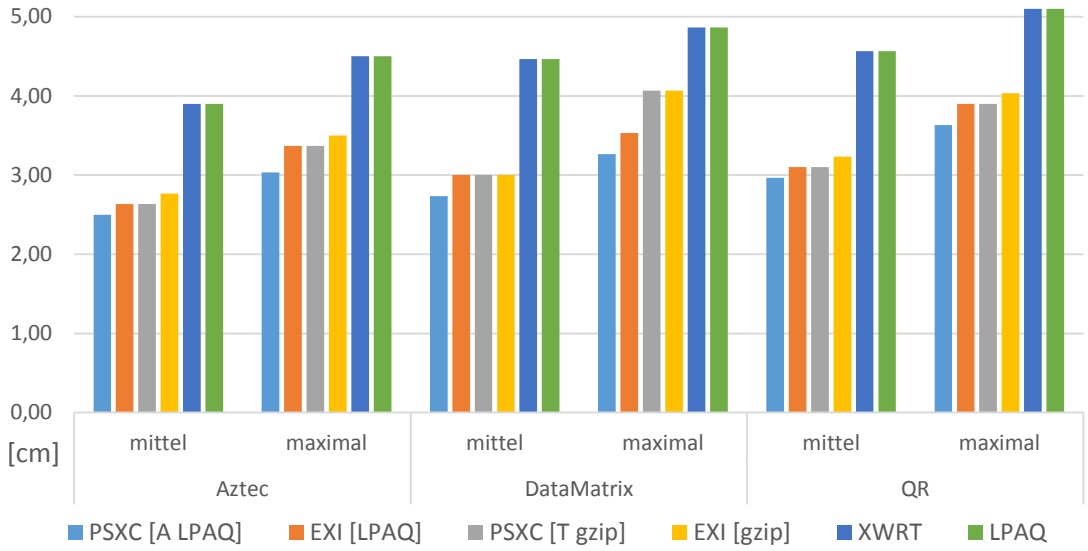

Abbildung 5.2: Kantenlängen in Zentimeter von Aztec, Data Matrix und QR Code für unterschiedliche Kompressionsverfahren (Eigene Darstellung)

Der Vergleich zeigt, dass die Verwendung von *EXI* im Vergleich zu der von *PSXC* nur zu geringfügig größeren Codes führt. Bei der schnellen Kompression mit *gzip* als Backend und Aztec Code zur optischen Codierung steigt die mittlere Kantenlänge um 5,0 % sowie die maximale Kantenlänge um 4,0 % an. Für QR Codes liegt der jeweilige Zuwachs bei 4,3 % bzw. 3,4 %. Bei Data Matrix Codes sind die Codegrößen identisch. Beim Einsatz von *LPAQ* als Backend ist der Rückstand von *EXI* etwas größer. Hier fällt die maximale Kantenlänge je nach verwendeter optischer Codierung um 7,3 % bei QR, 8,2 % bei Data Matrix und 11,0 % bei Aztec Codes größer aus. Die Unterschiede der weniger wichtigeren mittleren Kantenlänge betragen minimal 5,3 % bei QR und maximal 10 % bei Data Matrix Codes.

Wird statt der Kantenlänge die von den Codes belegte Fläche verglichen, wirken sich die Unterschiede prozentual größer aus. Dazu sei lediglich die Verwendung des leistungsstärksten Backendkompressors *LPAQ* betrachtet. In Bezug auf die maximale Fläche als wichtigstes Kriterium tritt der größte Unterschied bei der optischen Codierung durch den Aztec Code auf. Hier ist der Code bei Verwendung von *EXI*

um 23,2 % größer als bei *PSXC*. Mit 17,0 % und 15,2 % fallen die Unterschiede beim Einsatz von Data Matrix bzw. QR Codes etwas geringer aus.

5.4.3 Zusammenfassung

Zusammenfassend kann festgehalten werden, dass *PSXC* gegenüber *EXI* Einsparungen von bis zu 18,8 % in der Codegröße ermöglicht.[21] Ob diese für den jeweiligen Geschäftsprozess nötig sind, muss eine eigene Untersuchung klären. An dieser Stelle wird davon ausgegangen, dass die Leistung von *EXI* ausreichend ist und bei *PSXC* der Nachteil überwiegt, dass seine Verwendung die Adaption in der Praxis hemmt. In jedem Fall ist es nicht sinnvoll, andere Kompressionsverfahren zu verwenden oder ein noch nicht implementiertes Verfahren, wie beispielsweise *KST+DAG*, zu implementieren.

[21] Der Einsatz von *PSXC* verringert die maximal benötigte Codefläche bei der optischen Codierung durch den Aztec Code um 18,8 % im Vergleich zur Verarbeitung durch *EXI*, wenn jeweils *LPAQ* als Backendkompressor zum Einsatz kommt. Mit dem Data Matrix Code liegt die Verbesserung entsprechend bei 14,5 % und mit QR Code bei 13,2 %.

6 Fazit und Implikationen für weitere Untersuchungen

In der vorliegenden Arbeit wurde die in Kapitel 1.2 gestellte Forschungsfrage, inwiefern die Leistung bestehender Kompressionsverfahren für die optische Codierung XML-basierter Geschäftsdokumente verbessert werden kann, in einem mehrstufigen Analyseprozess detailliert untersucht. Dabei wurde die Erkenntnis gewonnen, dass die Kompressionsleistung durch eine neuartige Kombination bestehender Konzepte zur schemabasierten XML-Kompression gesteigert werden kann.

Im Zuge der Untersuchung wurde in Kapitel 3.2 ein Klassifikationssystem für bestehende Verfahren zur Kompression von XML-Dokumenten erarbeitet. Die Heterogenität der Verfahren erlaubt keine monohierarchische Klassifizierung, weshalb ein facettiertes Klassifikationssystem entwickelt wurde, welches die Einordnung relevanter Verfahren ermöglicht. Auf Basis der Klassifikation wurden relevante Verfahrensklassen argumentativ-deduktiv abgeleitet, indem der Einfluss der identifizierten Merkmale auf die zu erwartende Leistungsfähigkeit analysiert wurde.

Die als relevant eingestuften Verfahren wurden in Kapitel 3 in ihre technischen Bestandteile zerlegt, um daraus sinnvolle Konzepte zu extrahieren. Aufbauend darauf wurde ein parametrisiertes Verfahren entwickelt, mit dem die Leistungsfähigkeit neuer Kombinationen der technischen Konzepte untersucht werden kann. Dieses parametrisierte Verfahren wurde prototypisch realisiert.

Der in Kapitel 4 entwickelte Prototyp diente in Kapitel 5 zum fallstudienbasierten Vergleich der Kompressionsleistungen verschiedener Rekombinationen der Konzepte zur schemabasierten XML-Kompression auf Basis von über 500.000 Kompressionsinstanzen. Die leistungsstärksten Rekombinationen entlang der Dimensionen Kompressionsrate und Kompressionsgeschwindigkeit wurden daraufhin mit

den vielversprechendsten der bestehenden Verfahren[1] verglichen. Der Vergleich, in dem über 50.000 Kompressionsinstanzen erhoben wurden, zeigte, dass die Leistung der bestehenden Verfahren durch die identifizierten Rekombinationen übertroffen werden konnte. Im Zuge dessen wurde auch untersucht, wie hoch der Einfluss der Kompressionsleistungsunterschiede auf die Größe der resultierenden optischen Codierung ist. Diese soll – wie in der Motivation in Kapitel 1.1 ausgeführt – im Praxiseinsatz die automatisierte Weiterverarbeitung von Geschäftsdokumenten ermöglichen, indem Probleme von Medienbrüchen umgangen werden.

Diese Arbeit leistet einen Beitrag zur formulierten Forschungsfrage, indem sie erstmalig einen strukturierten Überblick über das breite Spektrum bestehender XML-Kompressionsverfahren bietet und damit die konzeptionelle Basis für weitere anwendungsorientierte Forschung in diesem Bereich legt. Das entstandene Klassifikationsschema ermöglicht die mehrdimensionale Einordnung auch zukünftig entstehender Verfahren. Die gesamten technischen Details schemabasierter Kompressionsverfahren wurden erstmalig konzeptionalisiert. Die darauf aufbauende, umfangreiche quantitative Untersuchung von Rekombinationen der extrahierten Konzepte ergibt wichtige Anhaltspunkte für die Übertragung in andere Anwendungsgebiete und bietet Raum für weitere technische und konzeptionelle Forschungsansätze.

In Bezug auf die Praxis ermöglicht das in dieser Arbeit entwickelte parametrisierte Verfahren die Bewertung bestehender Kompressionsverfahren im entsprechenden Anwendungskontext. Durch diesen konkreten Bewertungsrahmen werden Entscheidungen zur Auswahl von Kompressionsverfahren dahingehend unterstützt, ob im jeweiligen Anwendungsfall auf bestehende Verfahren zurückgegriffen werden kann oder Eigenentwicklungen sinnvoll sind – hat somit Potential zur Unterstützung klassischer betriebswirtschaftlicher 'Make-or-Buy'-Entscheidungen.

Außerdem werden im Rahmen von Grundlagenforschung Kompressionsverfahren in Hinblick auf deren Einsatz im Rahmen der optischen Codierung XML-basierter Geschäftsdokumente untersucht. Dies legt die konzeptionelle Basis zur Verbesserung von Geschäftsprozessen durch die Vermeidung von Medienbrüchen. Neben dem in

[1] Siehe dazu Abschnitte 3.4 und 3.5.

der Motivation dargelegten Szenario der Eingangsrechnungsverarbeitung ist beispielsweise auch der Einsatz der optischen Codierung auf Lieferscheinen eine vielversprechende Anwendung. Hier ermöglicht der Code eine unmittelbare Verarbeitung der Lieferung ohne manuelle Erfassung, auch bei fehlendem Lieferavis. Damit legt die Arbeit den Grundstein für weitere anwendungsorientierte Forschungsarbeiten, die beispielsweise branchen- und szenariobasiert untersuchen können, welche XML-basierten Datenformate in der optischen Codierung zu verwenden sind.

Anhang

A Aggregierte Messwerte für Konfigurationen der Strukturkompression

Konfiguration					Kompressionsrate [%]			kompr. Größe [Byte]			
Modellierungsart	Codierungsart	Backendkompressor	Alternative Häufigkeitsabbildung	Strukturredundanz verarbeiten	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
Automat	stat.	—	nein	ja	0,24	0,78	0,92	38,7	75	6,0	29,7
Automat	stat.	—	ja	ja	0,24	0,78	0,92	38,7	75	6,0	29,7
Baum	stat.	—	ja	ja	0,24	0,85	1,28	41,8	76	6,1	20,6
Baum	mix	PPMd	ja	ja	0,29	0,94	1,25	46,7	90	8,4	55,5
Baum	mix	LPAQ	ja	ja	0,28	0,97	1,17	48,4	87	8,6	207,4
Baum	mix	LPAQ	nein	ja	0,28	1,00	1,25	49,4	98	8,7	210,9
Baum	stat.	—	nein	ja	0,27	1,06	1,61	51,2	85	5,9	20,8
Baum	dyn.	LPAQ	ja	ja	0,28	1,05	1,25	51,9	88	9,0	206,9
Automat	stat.	—	nein	nein	0,88	0,93	0,98	52,1	309	30,5	27,6
Automat	stat.	—	ja	nein	0,88	0,93	0,98	52,1	309	30,5	27,6
Baum	mix	PPMd	nein	ja	0,30	1,07	1,25	52,6	113	9,0	54,9
Baum	dyn.	LPAQ	nein	ja	0,29	1,07	1,29	53,2	98	9,4	206,3
Baum	mix	LPAQ	nein	nein	0,60	1,05	1,25	54,4	188	17,5	207,5
Automat	mix	LPAQ	nein	nein	0,32	1,15	1,40	55,6	102	6,6	217,5
Automat	mix	LPAQ	ja	nein	0,32	1,15	1,40	55,6	102	6,6	217,5
Baum	dyn.	LPAQ	ja	nein	0,36	1,10	1,25	56,4	114	15,9	207,0
Automat	mix	LPAQ	nein	ja	0,30	1,17	1,40	57,1	96	7,8	219,3
Automat	mix	LPAQ	ja	ja	0,30	1,17	1,40	57,1	96	7,8	219,3
Baum	dyn.	LPAQ	nein	nein	0,42	1,13	1,29	58,3	132	16,6	209,1
Baum	dyn.	PAQ9	ja	nein	0,20	1,24	1,69	59,0	64	3,2	76,3
Baum	dyn.	PPMd	nein	ja	0,36	1,20	1,40	59,4	113	10,2	54,8
Baum	mix	Sequitur	ja	ja	0,32	1,21	1,55	59,7	101	8,7	496,3
Baum	mix	PPMd	nein	nein	0,73	1,14	1,25	59,8	230	21,5	54,5
Baum	dyn.	PPMd	ja	ja	0,35	1,21	1,47	59,9	111	9,7	54,8
Baum	dyn.	PAQ9	nein	nein	0,21	1,26	1,73	60,2	66	3,4	76,6
Baum	stat.	—	ja	nein	0,92	1,07	1,22	60,5	386	38,9	19,3
Baum	dyn.	PPMd	nein	nein	0,43	1,21	1,40	60,6	136	12,2	53,8
Baum	mix	PPMd	ja	nein	1,08	1,10	1,25	61,2	341	34,2	55,0
Baum	mix	LPAQ	ja	nein	1,08	1,11	1,17	61,6	342	33,8	206,5
Baum	mix	LZMA	ja	ja	0,33	1,25	1,82	61,6	104	9,1	37,5
Automat	dyn.	LPAQ	ja	nein	0,23	1,29	1,66	61,7	72	4,1	216,6
Automat	dyn.	LPAQ	nein	nein	0,23	1,29	1,66	61,7	72	4,1	216,6
Automat	mix	PPMd	nein	nein	0,35	1,27	1,55	61,9	111	7,1	63,8
Automat	mix	PPMd	ja	nein	0,35	1,27	1,55	61,9	111	7,1	63,8
Baum	mix	PAQ8	ja	ja	0,35	1,25	1,66	62,2	111	11,5	108,3

Konfiguration					Kompressionsrate [%]			kompr. Größe [Byte]			
Modellierungsart	Codierungsart	Backendkompressor	Alternative Häufigkeitsabbildung	Strukturredundanz verarbeiten	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
Baum	mix	PAQ9	ja	ja	0,31	1,27	1,62	62,5	97	8,6	78,1
Automat	mix	PPMd	nein	ja	0,35	1,29	1,55	62,8	109	8,0	64,8
Automat	mix	PPMd	ja	ja	0,35	1,29	1,55	62,8	109	8,0	64,8
Baum	mix	PPMs	ja	ja	0,34	1,30	1,73	63,7	107	8,3	54,8
Baum	mix	PAQ9	nein	ja	0,32	1,30	1,66	64,1	107	9,3	78,2
Baum	dyn.	PPMd	ja	nein	0,54	1,26	1,47	64,1	171	16,6	53,7
Baum	mix	gzip	ja	ja	0,34	1,32	2,01	64,5	107	9,0	22,1
Baum	dyn.	PAQ9	ja	ja	0,34	1,32	1,69	65,3	107	10,1	76,2
Automat	dyn.	LPAQ	nein	ja	0,35	1,33	1,66	65,3	109	9,2	219,6
Automat	dyn.	LPAQ	ja	ja	0,35	1,33	1,66	65,3	109	9,2	219,6
Baum	dyn.	PAQ9	nein	ja	0,34	1,34	1,73	65,9	108	9,7	76,6
Baum	mix	PAQ9	nein	nein	0,62	1,32	1,66	66,5	196	16,2	79,7
Baum	mix	PPMVC	ja	ja	0,35	1,39	1,88	67,7	110	8,4	59,8
Baum	mix	Sequitur	nein	ja	0,36	1,38	1,62	68,1	116	11,7	489,8
Automat	dyn.	PPMd	ja	nein	0,24	1,44	1,84	68,5	76	3,3	64,0
Automat	dyn.	PPMd	nein	nein	0,24	1,44	1,84	68,5	76	3,3	64,0
Baum	mix	PPMs	nein	ja	0,36	1,42	1,88	69,4	140	9,1	53,5
Automat	mix	PAQ9	ja	nein	0,37	1,44	1,88	69,5	117	7,2	86,6
Automat	mix	PAQ9	nein	nein	0,37	1,44	1,88	69,5	117	7,2	86,6
Automat	mix	Sequitur	ja	nein	0,39	1,43	1,77	69,9	124	8,9	524,4
Automat	mix	Sequitur	nein	nein	0,39	1,43	1,77	69,9	124	8,9	524,4
Baum	mix	Sequitur	nein	nein	0,67	1,38	1,62	70,2	211	19,0	477,3
Baum	dyn.	PAQ8	nein	nein	0,25	1,48	2,06	70,6	79	3,8	135,4
Baum	mix	PAQ8	nein	ja	0,37	1,44	1,88	71,0	118	11,2	115,0
Automat	mix	PAQ9	nein	ja	0,35	1,47	1,88	71,6	109	8,7	88,0
Automat	mix	PAQ9	ja	ja	0,35	1,47	1,88	71,6	109	8,7	88,0
Automat	dyn.	PAQ9	ja	nein	0,25	1,53	2,06	72,7	79	3,9	86,4
Automat	dyn.	PAQ9	nein	nein	0,25	1,53	2,06	72,7	79	3,9	86,4
Baum	mix	Sequitur	ja	nein	1,14	1,35	1,55	72,8	359	34,7	491,1
Baum	mix	PAQ8	nein	nein	0,65	1,45	1,88	72,9	205	17,0	122,0
Automat	dyn.	PPMd	ja	ja	0,41	1,49	1,84	72,9	130	9,4	63,9
Automat	dyn.	PPMd	nein	ja	0,41	1,49	1,84	72,9	130	9,4	63,9
Baum	dyn.	PAQ8	ja	nein	0,25	1,54	2,10	73,1	80	3,5	136,0
Baum	mix	PAQ8	ja	nein	1,12	1,36	1,66	73,2	354	34,0	108,0
Baum	mix	PPMVC	nein	ja	0,37	1,51	2,02	73,2	128	8,5	60,6
Baum	dyn.	PPMs	ja	ja	0,38	1,50	1,99	73,4	119	9,0	53,8
Automat	mix	Sequitur	nein	ja	0,39	1,50	1,77	74,0	122	11,4	530,8
Automat	mix	Sequitur	ja	ja	0,39	1,50	1,77	74,0	122	11,4	530,8
Baum	mix	LZMA	nein	ja	0,37	1,52	1,95	74,0	120	8,9	38,9
Baum	dyn.	PPMs	nein	ja	0,39	1,52	2,02	74,7	152	10,8	52,7
Baum	dyn.	Sequitur	nein	nein	0,31	1,56	1,91	75,8	97	8,8	465,9
Baum	mix	PAQ9	ja	nein	1,13	1,42	1,62	76,0	357	34,1	77,5
Baum	mix	LZMA	ja	nein	1,13	1,42	1,82	76,2	356	34,5	36,0
Automat	dyn.	PAQ9	ja	ja	0,38	1,58	2,06	76,7	120	9,3	88,1
Automat	dyn.	PAQ9	nein	ja	0,38	1,58	2,06	76,7	120	9,3	88,1
Baum	dyn.	PAQ8	nein	ja	0,35	1,57	2,06	76,9	110	10,4	124,0
Baum	dyn.	Sequitur	ja	nein	0,30	1,60	2,02	77,0	96	7,0	481,6
Baum	dyn.	PPMVC	nein	nein	0,31	1,59	2,13	77,1	97	7,4	58,8
Baum	dyn.	PPMVC	ja	nein	0,31	1,62	2,17	78,0	98	7,2	59,1
Baum	dyn.	PPMVC	nein	ja	0,41	1,60	2,13	78,1	128	9,5	61,4
Baum	dyn.	PPMVC	ja	ja	0,40	1,61	2,17	78,1	126	9,1	59,5
Baum	mix	PPMs	ja	nein	1,13	1,47	1,73	78,3	358	34,3	53,7
Baum	mix	LZMA	nein	nein	0,67	1,57	1,95	78,4	213	17,1	36,8

Konfiguration					Kompressionsrate [%]			kompr. Größe [Byte]			
Modellierungsart	Codierungsart	Backendkompressor	Alternative Häufigkeitsabbildung	Strukturredundanz verarbeiten	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
Baum	mix	PPMs	nein	nein	0,87	1,53	1,88	78,8	276	24,8	54,1
Baum	stat.	—	nein	nein	1,10	1,37	1,69	79,1	533	53,9	18,4
Automat	mix	PPMs	nein	nein	0,41	1,65	2,21	79,4	128	6,8	63,1
Automat	mix	PPMs	ja	nein	0,41	1,65	2,21	79,4	128	6,8	63,1
Baum	dyn.	PAQ8	ja	ja	0,35	1,62	2,10	79,5	109	10,4	126,4
Automat	dyn.	Sequitur	ja	nein	0,30	1,66	2,13	79,5	95	5,9	511,3
Automat	dyn.	Sequitur	nein	nein	0,30	1,66	2,13	79,5	95	5,9	511,3
Baum	mix	gzip	ja	nein	1,14	1,49	2,01	79,7	361	34,8	20,8
Automat	mix	gzip	nein	nein	0,41	1,65	2,28	80,0	131	7,6	28,7
Automat	mix	gzip	ja	nein	0,41	1,65	2,28	80,0	131	7,6	28,7
Baum	mix	bzip2	ja	ja	0,41	1,64	2,21	80,1	129	10,2	24,2
Automat	mix	PPMs	nein	ja	0,40	1,67	2,21	80,5	125	7,6	64,6
Automat	mix	PPMs	ja	ja	0,40	1,67	2,21	80,5	125	7,6	64,6
Baum	mix	PPMVC	nein	nein	0,70	1,59	2,02	80,6	221	19,9	61,1
Baum	dyn.	LZMA	nein	nein	0,30	1,69	2,32	80,8	96	4,2	35,8
Baum	mix	gzip	nein	ja	0,38	1,67	2,10	80,9	128	8,8	22,4
Baum	dyn.	Sequitur	nein	ja	0,47	1,65	1,91	81,7	149	15,2	479,5
Automat	mix	gzip	nein	ja	0,42	1,68	2,28	81,9	132	9,2	30,9
Automat	mix	gzip	ja	ja	0,42	1,68	2,28	81,9	132	9,2	30,9
Baum	mix	PPMVC	ja	nein	1,15	1,55	1,88	82,4	363	34,3	61,3
Automat	mix	PPMVC	ja	nein	0,43	1,74	2,36	83,5	135	7,0	69,6
Automat	mix	PPMVC	nein	nein	0,43	1,74	2,36	83,5	135	7,0	69,6
Baum	dyn.	LZMA	ja	nein	0,32	1,77	2,39	83,9	100	3,2	35,4
Baum	dyn.	PPMs	nein	nein	0,92	1,63	2,02	84,3	290	27,1	52,8
Automat	mix	PPMVC	nein	ja	0,41	1,75	2,36	84,5	129	7,6	70,7
Automat	mix	PPMVC	ja	ja	0,41	1,75	2,36	84,5	129	7,6	70,7
Baum	dyn.	PPMs	ja	nein	0,98	1,63	1,99	84,8	308	28,9	52,6
Automat	mix	LZMA	ja	nein	0,42	1,77	2,28	84,9	134	6,4	44,3
Automat	mix	LZMA	nein	nein	0,42	1,77	2,28	84,9	134	6,4	44,3
Baum	mix	gzip	nein	nein	0,69	1,71	2,10	85,4	219	17,5	21,2
Automat	dyn.	PPMs	nein	nein	0,30	1,81	2,43	85,8	96	3,7	62,6
Automat	dyn.	PPMs	ja	nein	0,30	1,81	2,43	85,8	96	3,7	62,6
Baum	dyn.	gzip	nein	nein	0,33	1,79	2,47	85,8	104	5,4	20,8
Baum	dyn.	LZMA	nein	ja	0,47	1,76	2,32	85,9	148	10,5	36,9
Baum	mix	bzip2	nein	ja	0,48	1,77	2,25	86,8	151	12,9	24,7
Automat	mix	LZMA	nein	ja	0,41	1,80	2,28	86,9	129	7,8	47,7
Automat	mix	LZMA	ja	ja	0,41	1,80	2,28	86,9	129	7,8	47,7
Automat	mix	PAQ8	ja	nein	0,42	1,84	2,39	88,0	134	6,9	128,2
Automat	mix	PAQ8	nein	nein	0,42	1,84	2,39	88,0	134	6,9	128,2
Baum	dyn.	Sequitur	ja	ja	0,48	1,77	2,07	88,2	150	17,8	487,1
Automat	dyn.	Sequitur	ja	ja	0,50	1,78	2,13	88,4	159	15,3	517,2
Automat	dyn.	Sequitur	nein	ja	0,50	1,78	2,13	88,4	159	15,3	517,2
Baum	dyn.	LZMA	ja	ja	0,45	1,83	2,39	88,6	142	9,1	36,8
Automat	dyn.	PPMVC	nein	nein	0,32	1,88	2,58	89,5	100	3,5	68,6
Automat	dyn.	PPMVC	ja	nein	0,32	1,88	2,58	89,5	100	3,5	68,6
Automat	dyn.	PPMs	ja	ja	0,46	1,86	2,43	89,9	144	9,2	63,4
Automat	dyn.	PPMs	nein	ja	0,46	1,86	2,43	89,9	144	9,2	63,4
Baum	dyn.	gzip	ja	nein	0,34	1,88	2,58	90,1	106	5,5	20,4
Baum	mix	bzip2	nein	nein	0,73	1,80	2,25	90,2	231	20,1	23,9
Automat	mix	PAQ8	ja	ja	0,42	1,87	2,39	90,3	132	8,9	125,1
Automat	mix	PAQ8	nein	ja	0,42	1,87	2,39	90,3	132	8,9	125,1
Automat	dyn.	gzip	nein	nein	0,33	1,92	2,65	91,6	104	4,9	28,1
Automat	dyn.	gzip	ja	nein	0,33	1,92	2,65	91,6	104	4,9	28,1

Konfiguration					Kompressionsrate [%]			kompr. Größe [Byte]			
Modellierungsart	Codierungsart	Backendkompressor	Alternative Häufigkeitsabbildung	Strukturredundanz verarbeiten	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
Baum	dyn.	gzip	nein	ja	0,47	1,88	2,47	91,6	148	11,9	22,0
Baum	mix	bzip2	ja	nein	1,18	1,77	2,21	92,6	373	34,2	23,3
Automat	mix	bzip2	ja	nein	0,48	1,93	2,69	93,3	153	9,8	31,3
Automat	mix	bzip2	nein	nein	0,48	1,93	2,69	93,3	153	9,8	31,3
Automat	dyn.	PPMVC	ja	ja	0,47	1,94	2,58	93,9	147	9,2	71,0
Automat	dyn.	PPMVC	nein	ja	0,47	1,94	2,58	93,9	147	9,2	71,0
Baum	dyn.	gzip	ja	ja	0,47	1,95	2,58	94,6	147	10,9	22,2
Automat	dyn.	PAQ8	ja	nein	0,32	2,01	2,76	95,2	100	3,1	133,0
Automat	dyn.	PAQ8	nein	nein	0,32	2,01	2,76	95,2	100	3,1	133,0
Baum	dyn.	bzip2	ja	nein	0,43	1,95	2,50	95,9	137	13,3	22,8
Baum	dyn.	bzip2	nein	nein	0,42	1,97	2,54	96,4	136	12,8	22,9
Automat	dyn.	gzip	nein	ja	0,51	1,98	2,65	96,4	162	11,9	30,2
Automat	dyn.	gzip	ja	ja	0,51	1,98	2,65	96,4	162	11,9	30,2
Automat	mix	bzip2	nein	ja	0,48	2,00	2,69	97,4	152	11,9	33,8
Automat	mix	bzip2	ja	ja	0,48	2,00	2,69	97,4	152	11,9	33,8
Baum	dyn.	bzip2	ja	ja	0,55	1,97	2,50	97,5	174	16,3	24,4
Automat	dyn.	LZMA	nein	nein	0,34	2,11	2,87	99,6	108	2,9	43,2
Automat	dyn.	LZMA	ja	nein	0,34	2,11	2,87	99,6	108	2,9	43,2
Baum	dyn.	bzip2	nein	ja	0,54	2,02	2,54	99,7	169	16,3	24,4
Automat	dyn.	PAQ8	ja	ja	0,41	2,08	2,76	100,1	128	8,2	130,5
Automat	dyn.	PAQ8	nein	ja	0,41	2,08	2,76	100,1	128	8,2	130,5
Automat	dyn.	bzip2	ja	nein	0,43	2,12	2,72	102,7	141	9,7	30,7
Automat	dyn.	bzip2	nein	nein	0,43	2,12	2,72	102,7	141	9,7	30,7
Automat	dyn.	LZMA	ja	ja	0,54	2,18	2,87	105,2	169	9,9	46,0
Automat	dyn.	LZMA	nein	ja	0,54	2,18	2,87	105,2	169	9,9	46,0
Automat	dyn.	bzip2	ja	ja	0,56	2,23	2,72	109,3	178	15,3	33,4
Automat	dyn.	bzip2	nein	ja	0,56	2,23	2,72	109,3	178	15,3	33,4

B Aggregierte Messwerte für Konfigurationen der Nutzdatenkompression

Konfiguration				Kompressionsrate [%]			kompr. Größe [Byte]			durchschnittl. Laufzeit [ms]
Containerstrategie	Sortierung	Backendkompressor	gemischte Codierung	minimal	durchschnittl.	maximal	durchschnittl.	maximal	Std.Abw.	
1	ja	LPAQ	nein	6,33	22,50	29,97	373,8	609	41,8	197,1
1	ja	LPAQ	ja	6,35	22,54	29,97	374,5	609	42,0	196,9
1	nein	LPAQ	nein	6,41	22,85	30,07	380,1	634	43,7	198,6
1	nein	LPAQ	ja	6,41	22,90	30,17	380,8	634	43,8	196,1
2	ja	LPAQ	nein	6,46	23,27	31,15	386,5	619	42,2	389,9
2	ja	LPAQ	ja	6,47	23,28	31,15	386,6	619	42,3	380,0
3	ja	LPAQ	ja	6,47	23,28	31,15	386,6	619	42,3	377,3
2	nein	LPAQ	nein	6,58	23,70	31,54	393,7	646	43,4	388,0
2	nein	LPAQ	ja	6,59	23,72	31,54	393,9	646	43,4	381,0
3	nein	LPAQ	ja	6,59	23,72	31,54	393,9	646	43,4	379,7
3	ja	LPAQ	nein	6,58	23,90	32,13	396,6	629	42,3	570,9
1	ja	PAQ8	nein	6,15	24,21	33,01	400,3	600	37,1	181,5
1	ja	PAQ8	ja	6,15	24,25	33,01	400,9	599	37,0	181,4
1	ja	PPMd	nein	7,45	24,09	31,83	401,4	683	49,2	38,7
1	ja	PPMd	ja	7,46	24,12	31,93	401,8	684	49,2	37,8
1	nein	PPMd	nein	7,57	24,16	31,73	402,8	691	50,7	38,4
1	nein	PPMd	ja	7,57	24,16	31,83	403,0	692	50,6	38,0
3	nein	LPAQ	nein	6,71	24,34	32,52	403,9	656	43,4	571,0
2	ja	PPMd	nein	7,36	24,40	32,81	405,8	671	46,9	73,3
1	nein	PAQ8	nein	6,15	24,55	33,20	405,8	607	37,3	177,0
2	ja	PPMd	ja	7,37	24,42	32,81	406,1	672	46,9	66,8
3	ja	PPMd	ja	7,37	24,42	32,81	406,1	672	46,9	66,9
1	nein	PAQ8	ja	6,16	24,58	33,30	406,2	607	37,3	186,6
2	nein	PPMd	nein	7,49	24,54	32,71	408,3	684	48,3	73,2
2	nein	PPMd	ja	7,49	24,56	32,71	408,7	684	48,3	67,1
3	nein	PPMd	ja	7,49	24,56	32,71	408,7	684	48,3	65,6
3	ja	PPMd	nein	7,43	24,73	33,30	411,1	677	46,9	100,3
3	nein	PPMd	nein	7,54	24,87	33,20	413,7	689	48,3	102,3
1	ja	PPMVC	nein	7,85	25,42	33,99	423,2	714	50,5	41,7
1	ja	PPMVC	ja	7,85	25,45	34,08	423,7	715	50,5	41,4
1	nein	PPMVC	nein	7,78	25,57	33,89	426,0	721	51,4	41,7
1	nein	PPMVC	ja	7,78	25,62	33,99	426,7	723	51,4	40,3
1	ja	gzip	nein	8,12	25,93	34,18	431,4	746	51,1	0,7
1	ja	gzip	ja	8,12	26,00	34,38	432,6	748	51,2	0,8
1	ja	LZMA	nein	7,74	26,23	35,16	435,5	719	48,4	13,7
1	ja	LZMA	ja	7,75	26,25	35,26	435,9	720	48,3	13,6
2	ja	PAQ8	nein	6,73	26,77	36,63	443,0	654	41,5	273,7
2	ja	PAQ8	ja	6,73	26,79	36,73	443,2	654	41,5	260,0
3	ja	PAQ8	ja	6,73	26,79	36,73	443,2	654	41,5	259,9
1	nein	gzip	nein	8,47	26,61	34,48	443,9	776	56,2	0,8
1	nein	gzip	ja	8,47	26,64	34,48	444,4	777	56,1	0,7
2	nein	PAQ8	ja	6,82	27,04	37,02	447,3	666	42,2	265,9
3	nein	PAQ8	ja	6,82	27,04	37,02	447,3	666	42,2	261,9
2	nein	PAQ8	nein	6,82	27,04	37,12	447,4	666	42,1	276,0
1	nein	LZMA	nein	8,16	26,97	35,26	448,7	755	52,5	17,2
1	nein	LZMA	ja	8,16	26,99	35,26	448,9	755	52,5	13,6
2	ja	PPMVC	nein	8,04	27,10	36,92	449,9	726	48,6	80,4
2	ja	PPMVC	ja	8,04	27,12	36,92	450,1	727	48,6	80,2

Konfiguration				Kompressionsrate [%]			kompr. Größe [Byte]			
Container-strategie	Sortierung	Backend-kompressor	gemischte Codierung	minimal	durch-schnittl.	maximal	durch-schnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
3	ja	PPMVC	ja	8,04	27,12	36,92	450,1	727	48,6	71,7
2	nein	PPMVC	nein	8,05	27,31	37,02	453,4	738	49,4	81,3
2	nein	PPMVC	ja	8,05	27,33	37,12	453,7	738	49,4	79,1
3	nein	PPMVC	ja	8,05	27,33	37,12	453,7	738	49,4	73,4
2	ja	gzip	nein	8,33	27,74	37,12	460,8	759	50,8	0,6
2	ja	gzip	ja	8,33	27,80	37,32	461,9	760	50,8	0,9
3	ja	gzip	ja	8,33	27,80	37,32	461,9	760	50,8	0,7
2	ja	LZMA	nein	8,17	27,90	37,81	463,1	756	50,5	26,3
3	ja	PAQ8	nein	6,96	28,02	38,69	463,2	674	41,5	348,5
2	ja	LZMA	ja	8,17	27,91	37,90	463,2	756	50,5	25,7
3	ja	LZMA	ja	8,17	27,91	37,90	463,2	756	50,5	25,0
1	nein	Sequitur	nein	9,54	27,95	36,34	467,0	833	61,8	446,7
3	nein	PAQ8	nein	7,05	28,28	38,98	467,3	686	42,2	357,3
1	nein	Sequitur	ja	9,55	27,97	36,34	467,3	834	61,8	415,8
2	nein	gzip	nein	8,84	28,22	37,61	469,1	788	53,5	0,7
2	nein	gzip	ja	8,85	28,27	37,71	470,0	790	53,6	0,9
3	nein	gzip	ja	8,85	28,27	37,71	470,0	790	53,6	0,7
2	nein	LZMA	nein	8,52	28,40	37,90	471,7	773	53,0	34,1
2	nein	LZMA	ja	8,52	28,41	37,90	471,9	773	53,0	26,1
3	nein	LZMA	ja	8,52	28,41	37,90	471,9	773	53,0	26,1
1	ja	Sequitur	nein	10,26	28,33	36,14	475,1	896	68,7	419,1
1	ja	Sequitur	ja	10,27	28,35	36,14	475,3	897	68,7	433,6
3	ja	PPMVC	nein	8,33	28,73	39,47	476,1	753	48,6	113,6
3	nein	PPMVC	nein	8,34	28,94	39,67	479,7	764	49,4	113,8
2	nein	Sequitur	nein	9,39	28,90	37,71	481,7	820	59,6	924,8
2	nein	Sequitur	ja	9,39	28,93	37,71	482,2	820	59,6	833,8
3	nein	Sequitur	ja	9,39	28,93	37,71	482,2	820	59,6	833,6
3	ja	LZMA	nein	8,39	29,08	39,76	482,2	775	50,5	39,0
3	ja	gzip	nein	8,57	29,10	39,37	482,9	781	50,8	0,7
2	ja	Sequitur	nein	9,96	28,91	37,51	483,0	870	63,4	836,2
2	ja	Sequitur	ja	9,97	28,93	37,51	483,4	871	63,5	869,0
3	ja	Sequitur	ja	9,97	28,93	37,51	483,4	871	63,5	870,7
3	nein	LZMA	nein	8,73	29,58	39,76	490,9	792	53,0	49,3
3	nein	gzip	nein	9,09	29,57	39,76	491,0	811	53,6	0,6
3	nein	Sequitur	nein	9,56	29,86	39,18	497,2	835	59,6	1403,0
3	ja	Sequitur	nein	10,14	29,86	38,98	498,4	886	63,5	1255,1
4	nein	PPMd	ja	8,57	31,19	40,25	516,2	791	48,9	477,6
4	ja	PPMd	ja	8,60	31,32	40,84	518,1	793	48,2	476,9
4	nein	PPMd	nein	8,63	31,50	40,74	521,2	796	48,9	540,2
4	ja	PPMd	nein	8,65	31,63	41,33	523,1	798	48,2	526,8
4	ja	LPAQ	ja	8,15	32,81	43,00	540,8	773	43,0	2788,4
4	nein	LPAQ	ja	8,04	32,83	43,00	541,4	780	43,6	2794,4
4	ja	LPAQ	nein	8,26	33,43	43,98	550,8	783	43,0	3005,5
4	nein	LPAQ	nein	8,15	33,45	43,98	551,4	790	43,6	3026,9
1	ja	bzip2	ja	10,86	34,34	48,87	574,8	989	78,2	3,0
1	ja	bzip2	nein	10,93	34,34	48,09	574,9	986	78,5	2,7
1	nein	bzip2	nein	11,65	34,40	46,62	576,6	1018	81,7	3,3
1	nein	bzip2	ja	11,40	34,43	48,38	577,0	1003	81,4	3,0
2	ja	bzip2	ja	11,16	36,08	52,11	600,6	975	71,5	5,2
3	ja	bzip2	ja	11,16	36,08	52,11	600,6	975	71,5	4,8
2	ja	bzip2	nein	11,15	36,08	52,11	600,8	974	71,6	5,0
2	nein	bzip2	ja	11,56	36,32	52,79	604,8	1010	72,6	4,9
3	nein	bzip2	ja	11,56	36,32	52,79	604,8	1010	72,6	5,2
2	nein	bzip2	nein	11,56	36,32	52,40	604,8	1010	72,8	4,9

Konfiguration				Kompressionsrate [%]			kompr. Größe [Byte]			
Container-strategie	Sortierung	Backend-kompressor	gemischte Codierung	minimal	durch-schnittl.	maximal	durch-schnittl.	maximal	Std.Abw.	durchschnittl. Laufzeit [ms]
3	ja	bzip2	nein	11,58	38,37	55,73	637,6	1012	71,5	6,9
3	nein	bzip2	nein	11,98	38,61	56,42	641,8	1047	72,6	6,9
5	ja	PPMd	ja	10,65	41,34	54,65	681,0	967	52,6	1048,2
5	nein	PPMd	ja	10,65	41,35	57,20	681,2	1060	53,3	1053,8
5	ja	PPMd	nein	10,70	41,65	55,14	686,0	972	52,5	1079,4
5	nein	PPMd	nein	10,70	41,66	55,14	686,2	1132	53,6	1136,5
4	nein	Sequitur	ja	12,27	42,70	55,44	705,9	1072	63,0	6270,9
4	ja	Sequitur	ja	12,28	42,79	55,44	707,3	1073	62,9	6623,5
4	nein	Sequitur	nein	12,44	43,63	56,90	720,9	1087	63,0	7353,7
4	ja	Sequitur	nein	12,45	43,72	56,90	722,3	1088	62,9	6634,9
4	nein	LZMA	ja	11,41	46,14	60,92	758,9	1062	53,7	186,4
4	ja	LZMA	ja	11,60	46,27	61,12	760,9	1068	53,9	183,2
5	nein	LPAQ	ja	10,86	46,92	62,59	769,3	1153	46,0	6134,6
5	ja	LPAQ	ja	10,86	46,93	85,85	769,4	1122	45,8	6124,2
4	nein	LZMA	nein	11,63	47,32	62,78	777,9	1081	53,7	244,0
5	ja	LPAQ	nein	10,98	47,53	63,57	779,1	1016	45,1	6339,4
5	nein	LPAQ	nein	10,98	47,53	63,57	779,1	1016	45,1	6400,7
4	ja	LZMA	nein	11,81	47,45	62,98	779,9	1087	53,9	200,6
4	nein	PAQ8	ja	10,69	47,72	64,05	783,0	1026	47,1	1357,2
4	ja	PAQ8	ja	10,73	47,79	63,57	784,1	1173	47,7	1352,4
4	nein	gzip	ja	11,74	47,93	63,47	787,9	1088	53,3	1,1
4	ja	gzip	ja	11,88	48,11	63,66	790,6	1089	52,8	1,2
4	nein	PAQ8	nein	10,92	48,96	66,01	803,0	1046	47,1	1483,9
4	ja	PAQ8	nein	10,95	49,03	65,52	804,1	1137	47,5	1463,6
4	nein	gzip	nein	11,98	49,23	65,52	808,9	1109	53,3	1,1
4	ja	gzip	nein	12,12	49,41	65,72	811,6	1110	52,8	1,3
4	ja	PPMVC	ja	12,21	50,79	67,38	833,0	1117	49,8	555,7
4	nein	PPMVC	ja	12,20	50,81	67,38	833,7	1120	51,1	558,0
4	ja	PPMVC	nein	12,51	52,41	76,07	859,2	1170	50,3	598,8
4	nein	PPMVC	nein	12,50	52,42	69,93	859,7	1146	51,1	615,8
5	ja	Sequitur	ja	15,76	63,23	82,76	1037,9	1389	64,9	15139,6
5	nein	Sequitur	ja	15,76	63,23	82,76	1037,9	1389	64,9	14030,7
5	ja	Sequitur	nein	15,93	64,16	84,23	1052,9	1404	64,9	14073,6
5	nein	Sequitur	nein	15,93	64,16	84,23	1052,9	1404	64,9	14781,2
4	nein	bzip2	ja	16,91	70,84	93,58	1162,6	1533	71,7	33,1
5	ja	LZMA	ja	16,47	71,09	94,71	1163,5	1491	60,5	408,8
5	nein	LZMA	ja	16,47	71,09	94,71	1163,5	1491	60,5	416,5
4	ja	bzip2	ja	16,69	71,03	96,29	1165,4	1554	71,1	32,2
5	ja	LZMA	nein	16,69	72,26	96,57	1182,5	1510	60,5	431,1
5	nein	LZMA	nein	16,69	72,26	96,57	1182,5	1510	60,5	433,7
4	nein	bzip2	nein	17,33	73,13	97,13	1199,6	1570	71,7	34,3
4	ja	bzip2	nein	17,11	73,32	99,81	1202,4	1591	71,1	34,0
5	ja	gzip	ja	17,27	75,62	100,10	1237,4	1560	62,8	2,0
5	nein	gzip	ja	17,27	75,62	100,10	1237,4	1560	62,8	1,9
5	ja	gzip	nein	17,51	76,92	102,15	1258,4	1581	62,8	1,8
5	nein	gzip	nein	17,51	76,92	102,15	1258,4	1581	62,8	1,9
5	ja	PAQ8	ja	16,98	77,03	102,64	1259,2	1549	59,4	2872,2
5	nein	PAQ8	ja	16,98	77,03	102,64	1259,2	1549	59,5	2872,1
5	nein	PAQ8	nein	17,20	78,26	104,60	1279,2	1569	59,4	3033,7
5	ja	PAQ8	nein	17,20	78,27	104,60	1279,3	1569	59,6	2949,5
5	ja	PPMVC	ja	18,61	84,19	112,34	1374,0	1666	56,4	1229,3
5	nein	PPMVC	ja	18,61	84,19	112,34	1374,0	1666	56,4	1238,6
5	ja	PPMVC	nein	18,91	85,80	114,89	1400,0	1692	56,4	1287,0
5	nein	PPMVC	nein	18,91	85,82	114,89	1400,3	1992	58,1	1294,3

Konfiguration				Kompressionsrate [%]			kompr. Größe [Byte]			durchschnittl. Laufzeit [ms]
Container-strategie	Sortierung	Backend-kompressor	gemischte Codierung	minimal	durch-schnittl.	maximal	durch-schnittl.	maximal	Std.Abw.	
5	ja	bzip2	ja	25,66	117,90	154,16	1922,5	2281	75,9	69,5
5	nein	bzip2	ja	25,66	117,90	154,16	1922,5	2281	75,9	70,7
5	ja	bzip2	nein	26,09	120,19	157,79	1959,5	2318	75,9	71,4
5	nein	bzip2	nein	26,09	120,19	157,79	1959,5	2318	75,9	72,2

Quellenverzeichnis

Literaturverzeichnis

Adiego, J./La Fuente, P. de/Navarro, G. (2005). „Combining Structural and Textual Contexts for Compressing Semistructured Databases“. In: *Sixth Mexican International Conference on Computer Science, 2005*. Hrsg. von V. Estívill-Castro. Los Alamitos, CA: IEEE Computer Society, S. 68–73.

Alkhatib, R./Scholl, M. H. (2008a). „CXQU: A compact XML storage for efficient query and update processing“. In: *Third International Conference on Digital Information Management, 2008*. Piscataway, NJ: IEEE Computer Society, S. 605–612.

Alkhatib, R./Scholl, M. H. (2008b). „Efficient Compression and Querying of XML Repositories“. In: *19th International Workshop on Database and Expert Systems Application (DEXA 2008)*. Hrsg. von A. M. Tjoa. Piscataway, NJ: IEEE Computer Society, S. 365–369.

Alkhatib, R./Scholl, M. H. (2009). „Compacting XML Structures Using a Dynamic Labeling Scheme“. In: *Dataspace: the final frontier*. Hrsg. von A. P. Sexton. Bd. 5588. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 158–170.

Arion, A. u. a. (2007). „XQueC: A query-conscious compressed XML database“. In: *ACM Transactions on Internet Technology* 7.2, S. 1–35.

Augeri, C. J. u. a. (2007). „An analysis of XML compression efficiency“. In: *Proceedings of the 2007 workshop on Experimental computer science*. New York, NY: ACM, S. 1–11.

Bächle, M./Lehmann, F. R. (2010). *E-Business: Grundlagen elektronischer Geschäftsprozesse im Web 2.0*. Wirtschaftsinformatik kompakt. München: de Gruyter.

Baterip, C. (2013). „Invoice Data Capture Software - What to Know Before You Buy“. In: *Credit Control* 34.8/9, S. 41–48.

Bergmann, R. u. a. (2014). *Das ZUGFeRD-Format: Spezifikation und Umsetzungsregeln zum branchen- übergreifenden Kern-Rechnungsformat des Forums elektronische Rechnung Deutschland (FeRD): Version 1.0*. URL: `http://konik.io/ZUGFeRD-Spezifikation/Das-ZUGFeRD-Format_1p0.pdf`.

Böttcher, S./Hartel, R./Heinzemann, C. (2008). „BSBC: Towards a Succinct Data Format for XML Streams“. In: *4th International Conference on Web Information Systems and Technologies (WEBIST 2008)*. Bd. 1. Funchal, Madeira, Portugal, S. 13–21.

Böttcher, S./Hartel, R./Heinzemann, C. (2009). „Compressing XML Data Streams with DAG+BSBC“. In: *Web Information Systems and Technologies: Funchal, Madeira, Portugal, May 4-7, 2008, Revised Selected Papers*. Hrsg. von W. Aalst u. a. Bd. 18. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, S. 65–79.

Böttcher, S./Hartel, R./Messinger, C. (2009). „XML Stream Data Reduction by Shared KST Signatures“. In: *42nd Hawaii International Conference on System Sciences, 2009*. Hrsg. von R. H. Sprague. Piscataway, NJ: IEEE Computer Society, S. 1–10.

Böttcher, S./Hartel, R./Messinger, C. (2010). „Searchable Compression of Office Documents by XML Schema Subtraction“. In: *Database and XML technologies*. Hrsg. von M. L. Lee u. a. Bd. 6309. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 103–112.

Böttcher, S./Hartel, R./Messinger, C. (2011). „Using XML Schema Subtraction to Compress Electronic Payment Messages;“ in: *Enterprise Information Systems*. Hrsg. von W. van der Aalst u. a. Bd. 73. Lecture Notes in Business Information Processing. Berlin: Springer, S. 451–463.

Böttcher, S./Steinmetz, R./Klein, N. (2007). „XML index compression by DTD subtraction“. In: *Proceedings of the Ninth International Conference on Enterprise Information Systems (ICEIS 2007)*. Hrsg. von J. Cardoso. Setúbal: INSTICC, S. 86–94.

Buchanan, B. (1989). *Bibliothekarische Klassifikationstheorie*. München: Saur.

Burrows, M./Wheeler, D. J. (1994). *A block-sorting lossless data compression algorithm*. URL: `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf`.

Cannataro, M./Comito, C./Pugliese, A. (2002). „SqueezeX: synthesis and compression of XML data“. In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2002)*. Los Alamitos, CA: IEEE Computer Society, S. 326–331.

Cheney, J. (2001). „Compressing XML with multiplexed hierarchical PPM models“. In: *Proceedings of the Data Compression Conference (DCC 2001)*. Hrsg. von J. A. Storer. Los Alamitos, CA: IEEE Computer Society, S. 163–172.

Cheney, J. (2005). „An Empirical Evaluation of Simple DTD-Conscious Compression Techniques“. In: *Proceedings of the Eight International Workshop on the Web & Databases (WebDB 2005)*. Hrsg. von A. Doan u. a., S. 43–48.

Cheng, J./Ng, W. (2004). „XQzip: Querying Compressed XML Using Structural Indexing“. In: *Advances in Database Technology - EDBT 2004*. Hrsg. von E. Bertino u. a. Bd. 2992. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 219–236.

Eckstein, R./Eckstein, S. (2004). *XML und Datenmodellierung: XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*. 1. Aufl. Heidelberg: dpunkt-Verl.

Ferragina, P. u. a. (2006). „Compressing and searching XML data via two zips“. In: *Proceedings of the 15th International Conference on World Wide Web (2006)*. Hrsg. von L. Carr u. a. New York, NY: ACM, S. 751–760.

Ferragina, P. u. a. (2009). „Compressing and indexing labeled trees, with applications“. In: *Journal of the ACM* 57.1, S. 1–33.

Fink, A./Schneidereit, G./Voss, S. (2005). *Grundlagen der Wirtschaftsinformatik*. 2. Aufl. Heidelberg: Physica.

Girardot, M./Sundaresan, N. (2000). „Millau: an encoding format for efficient representation and exchange of XML over the Web“. In: *Computer Networks* 33.1-6, S. 747–765.

Hansen, H. R./Mendling, J./Neumann, G. (2015). *Wirtschaftsinformatik*. 11. Aufl. Berlin: de Gruyter.

Harrusi, S./Averbuch, A./Yehudai, A. (2006b). „XML Syntax Conscious Compression“. In: *Proceedings of the Data Compression Conference (DCC 2006)*. Hrsg. von J. A. Storer. Los Alamitos, CA: IEEE Computer Society, S. 402–411.

Heise, W./Quattrocchi, P. (1989). *Informations- und Codierungstheorie: Mathematische Grundlagen der Daten-Kompression und -Sicherung in diskreten Kommunikationssystemen*. 2. Aufl. Berlin, Heidelberg: Springer.

Herold, H./Lurz, B./Wohlrab, J. (2012). *Grundlagen der Informatik*. 2. Aufl. München: Pearson Studium.

Hevner, A. R. u. a. (2004). „Design Science in Information Systems Research“. In: *MIS Quarterly* 28.1, S. 75–105.

Hu, X./Zhang, H./Yuan, X. (2012). „A Compact XML Storage Scheme Supporting Efficient Path Querying“. In: *Web technologies and applications*. Hrsg. von Q. Z. Sheng u. a. Bd. 7235. Lecture Notes in Computer Science. Berlin: Springer, S. 711–718.

Jeuring, J./Haag, P. (2002). *Generic Programming for XML Tools*. Institute of Information und Computing Sciences, Utrecht University, Niederlande. URL: https://www.researchgate.net/profile/Johan_Jeuring/publication/2946480_Generic_Programming_for_XML_Tools/links/0fcfd5102585fef0ba000000.pdf.

Kabak, Y./Dogac, A. (2010). „A survey and analysis of electronic business document standards“. In: *ACM Computing Surveys* 42.3, S. 1–31.

Kieffer, J. C. u. a. (2000). „Universal lossless compression via multilevel pattern matching“. In: *IEEE Transactions on Information Theory* 46.4, S. 1227–1245.

Knuchel, T. u. a. (2011). „2D-Codes“. In: *WIRTSCHAFTSINFORMATIK* 53.1, S. 49–52.

Kwasnik, B. H. (1999). „The Role of Classifiction in Knowledge Representation and Discovery“. In: *Library Trends* 48.1, S. 22–47.

Laudon, K. C./Laudon, J. P./Schoder, D. (2016). *Wirtschaftsinformatik: Eine Einführung*. 3. Aufl. Hallbergmoos, Germany: Pearson Studium.

League, C./Eng, K. (2007a). „Schema-Based Compression of XML Data with Relax NG“. In: *Journal of Computers* 2.10, S. 9–17.

League, C./Eng, K. (2007b). „Type-Based Compression of XML Data“. In: *Proceedings of the Data Compression Conference (DCC 2007)*. Hrsg. von J. A. Storer. Los Alamitos, CA: IEEE Computer Society, S. 273–282.

Leighton, G. (2005). „Two new approaches for compressing XML“. Master Thesis. Wolfville, Nova Scotia, Canada: Acadia University. url: `http://webdocs.cs.ualberta.ca/~gleighto/research/MThesis.pdf`.

Leighton, G./Diamond, J./Müldner, T. (2005). *A grammar-based approach for compressing XML*. Vol. 2 No. 6, Technical Report, Acadia University. url: `http://cs-dev.acadiau.ca/research/technical_reports/files/tr-2005-004.pdf`.

Lenk, B. (2002). *2D-Codes: Matrixcodes, Stapelcodes, Composite Codes, Dotcodes*. 1. Aufl. Bd. 2. Handbuch der automatischen Identifikation. Kirchheim unter Teck: Lenk.

Lenk, B. (2003). *ID-Techniken: 1D-Codes, 2D-Codes, 3D-Codes*. 2. Aufl. Bd. 1. Handbuch der automatischen Identifikation. Kirchheim unter Teck: Lenk.

Lenk, B. (2004). *Strichcode-Praxis: Codeauswahl, Drucktechnik, Codeprüfung, Etikettierung, Lesegeräte*. 1. Aufl. Bd. 3. Handbuch der automatischen Identifikation. Kirchheim unter Teck: Lenk.

Lenk, B. (2005a). *Barcode: Das Profibuch der Lesetechnik für die Optischen Identifikation*. 1. Aufl. Kirchheim unter Teck: Lenk.

Lenk, B. (2005b). *Einführung in die Identifikation: Technologie zur Kopplung von Material- und Informationsfluss sowie Einsatzstrategien*. 1. Aufl. Kirchheim unter Teck: Lenk.

Lenk, B. (2012). *QR-Code*. 1. Aufl. Kirchheim unter Teck: Lenk.

Levene, M./Wood, P. T. (2002). „XML Structure Compression". In: *Proceedings of the Second International Workshop on Web Dynamics*. Hrsg. von M. Levene/A. Poulovassilis. CEUR Workshop Proceedings, S. 56–69.

Liefke, H./Suciu, D. (2000). „XMill: An Efficient Compressor for XML Data". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. Hrsg. von M. Dunham. New York, NY: ACM, S. 153–164.

Lin, Y. u. a. (2005). „Supporting efficient query processing on compressed XML files". In: *Proceedings of the 2005 ACM Symposium on Applied Computing*. Hrsg. von H. Haddad. New York, NY: ACM, S. 660–665.

Lohrey, M./Maneth, S./Mennicke, R. (2011). „Tree Structure Compression with RePair". In: *Proceedings of the Data Compression Conference (DCC 2011)*. Hrsg. von J. A. Storer. Piscataway, NJ: IEEE Computer Society, S. 353–362.

Mahalakshmi, B./Hanchate, R. S. (2013). „Review of Compression Techniques, Labelling Schemas & Keyword Searching In XML Documents". In: *International Journal of Computer Science Research & Technology (IJCSRT)* 2.8, S. 165–172.

Mahoney, M. V. (2005). *Adaptive weighing of context models for lossless data compression*. Florida Institute of Technology, Melbourne, FL, USA. url: `http://mattmahoney.net/dc/cs200516.pdf`.

Malaka, R./Butz, A./Hussmann, H. (2009). *Medieninformatik: Eine Einführung*. München: Pearson Studium.

Maneth, S./Mihaylov, N./Sakr, S. (2008). „XML Tree Structure Compression". In: *19th International Workshop on Database and Expert Systems Application (DEXA 2008)*. Hrsg. von A. M. Tjoa. Piscataway, NJ: IEEE Computer Society, S. 243–247.

Min, J.-K./Park, M.-J./Chung, C.-W. (2003). „XPRESS: a queriable compression for XML data“. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. Hrsg. von Y. Papakonstantinou. New York, NY: ACM, S. 122–133.

Müldner, T./Leighton, G./Diamond, J. (2005). „Using XML Compression for WWW Communication“. In: *Proceedings of the IADIS International Conference Applied Computing 2005*. Hrsg. von N. Guimarães/P. T. Isaías. Lisboa: IADIS, S. 459–467.

Müldner, T./Miziołek, J. K./Fry, C. (2012). „Online Internet Communication Using an XML Compressor“. In: *Proceedings of The Seventh International Conference on Internet and Web Applications and Services (ICIW 2012)*. Hrsg. von F. Laux/P. Lorenz. Stuttgart, Germany, S. 131–136.

Müldner, T. u. a. (2008). „SXSAQCT and XSAQCT: XML Queryable Compressors“. In: *Structure-Based Compression of Complex Massive Data*. Hrsg. von S. Böttcher u. a. Bd. 8261. Dagstuhl Seminar Proceedings. Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Müldner, T. u. a. (2009). „XSAQCT: XML Queryable Compressor“. In: *Balisage: The Markup Conference 2009*. Hrsg. von B. T. Usdin. Balisage Series on Markup Technologies. Rockville, Maryland: Mulberry Technologies, Inc.

Murata, M. u. a. (2005). „Taxonomy of XML schema languages using formal language theory“. In: *ACM Transactions on Internet Technology* 5.4, S. 660–704.

Nevill-Manning, C. G./Witten, I. H. (1997). „Linear-time, incremental hierarchy inference for compression“. In: *Proceedings of the Data Compression Conference (DCC 1997)*. Hrsg. von J. A. Storer. Los Alamitos, CA: IEEE Computer Society, S. 3–11.

Ng, W./Lam, W.-Y./Cheng, J. (2006). „Comparative Analysis of XML Compression Technologies“. In: *World Wide Web* 9.1, S. 5–33.

Ng, W. u. a. (2006). „XCQ: A queriable XML compression system“. In: *Knowledge and Information Systems* 10.4, S. 421–452.

Nomikos, M. (2002). „Zwischenbetriebliche Anwendungen“. In: *Ganzheitliches E-Business*. Hrsg. von J. Biethahn/M. Nomikos. München: Oldenbourg, S. 149–180.

Rocco, D./Caverlee, J./Liu, L. (2005). „XPack: A High-Performance WEB Document Encoding“. In: *Proceedings of the First International Conference on Web Information Systems and Technologies (WEBIST 2005)*. Hrsg. von V. Pedrosa u. a. Setúbal: INSTICC, S. 32–39.

Sakr, S. (2009a). „An Empirical Evaluation of XML Compression Tools“. In: *Database systems for advanced applications*. Hrsg. von L. Chen. Bd. 5667. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 49–63.

Sakr, S. (2009b). „XML compression techniques: A survey and comparison“. In: *Journal of Computer and System Sciences* 75.5, S. 303–322.

Salomon, D. (2007a). *Data compression: The complete reference*. 4. Aufl. New York: Springer.

Salomon, D. (2007b). *Variable-length Codes for Data Compression*. 1. Aufl. London: Springer.

Schönfeld, D./Klimant, H./Piotraschke, R. (2012). *Informations- und Kodierungstheorie*. 4. Aufl. Wiesbaden: Vieweg+Teubner.

Schulz, M. (2014). „Eingangsrechnungen ... und die Folgeprozesse“. In: *DOK* 5, S. 66–68.

Serin, E. (2003). „Design and Test of the Cross-Format Schema Protocol (XFSP) for Networked Virtual Environments“. Master Thesis. Monterey, California: NAVAL POSTGRADUATE SCHOOL. URL: `https://www.movesinstitute.org/xmsf/projects/XSBC/03Mar_Serin.pdf`.

Shannon, C. E. (1948). „A Mathematical Theory of Communication“. In: *The Bell System Technical Journal* 27.3 und 4, 379–423 und 623–656.

Shkarin, D. A. (2001). „Improving the Efficiency of the PPM Algorithm“. In: *Problems of Information Transmission* 37.3, S. 226–235.

Skibinski, P./Grabowski, S. (2004). „Variable-length contexts for PPM“. In: *Proceedings of the Data Compression Conference (DCC 2004)*. Hrsg. von J. A. Storer. Los Alamitos, CA: IEEE Computer Society, S. 409–418.

Skibinski, P./Grabowski, S./Swacha, J. (2007). „Fast Transform for Effective XML Compression“. In: *Proceedings of 9th International Conference The Experience of Designing and Applications of CAD Systems in Microelectronics (CASDM 2007)*. Piscataway, NJ: IEEE Computer Society und IEEE, S. 323–326.

Skibiński, P./Grabowski, S./Swacha, J. (2008). „Effective asymmetric XML compression“. In: *Software: Practice and Experience* 38.10, S. 1027–1047.

Skibiński, P./Swacha, J. (2007). „Combining Efficient XML Compression with Query Processing“. In: *Advances in databases and information systems*. Hrsg. von Y. Ioannidis/B. Novikov/B. Rachev. Bd. 4690. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 330–342.

Statistisches Bundesamt, Hrsg. (2015). *Nutzung von Informations- und Kommunikationstechnologien in Unternehmen*. Wiesbaden. URL: https://www.destatis.de/DE/Publikationen/Thematisch/UnternehmenHandwerk/Unternehmen/InformationstechnologieUnternehmen5529102157004.pdf?__blob=publicationFile.

Stock, W. G./Stock, M. (2008). *Wissensrepräsentation: Informationen auswerten und bereitstellen*. München: Oldenbourg.

Subramanian, H./Shankar, P. (2006). „Compressing XML Documents Using Recursive Finite State Automata“. In: *Implementation and application of automata*. Hrsg. von J. Farré/I. Litovsky/S. Schmitz. Bd. 3845. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, S. 282–293.

Sundaresan, N./Moussa, R. (2002). „Algorithms and programming models for efficient representation of XML for Internet applications“. In: *Computer Networks* 39.5, S. 681–697.

Tolani, P. M./Haritsa, J. R. (2002). „XGrind: a query-friendly XML compressor“. In: *Proceedings of 18th International Conference on Data Engineering,* hrsg. von R. Agrawal. Los Alamitos, CA: IEEE Computer Society, S. 225–234.

Toman, V. (2003). „Compression of XML data“. Master Thesis. Prague: Charles University. URL: http://exalt.sourceforge.net/thesis.pdf.

Toman, V. (2004). „Syntactical compression of XML data“. In: *Presented at the doctoral consortium of the 16th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'04)*. Riga, Lativa.

Uitz, I./Harnisch, M. (2012). „Der QR-Code – aktuelle Entwicklungen und Anwendungsbereiche“. In: *Informatik-Spektrum* 35.5, S. 339–347.

Ulrich, H. (1984). *Management*. Bd. 13. Schriftenreihe Unternehmung und Unternehmungsführung. Bern u.a.: Haupt.

Ulrich, H. (1985). „Von der Betriebswirtschaftlehre zur systemorientierten Managementlehre“. In: *Betriebswirtschaftslehre als Management- und Führungslehre*. Hrsg. von R. Wunderer. Stuttgart: Poeschel, S. 2–32.

Vlist, E. (2011). *RELAX NG*. Sebastopol: O'Reilly Media Inc.

Vonhoegen, H. (2015). *Einstieg in XML*. 8. Aufl. Bd. 3798. Rheinwerk Computing. Bonn: Rheinwerk.

WKWI/GI FB WI, Hrsg. (2011). *Profil der Wirtschaftsinformatik: Einstimmiger Beschluss der gemeinsamen Sitzung der Wissenschaftlichen Kommission Wirtschaftsinformatik (WKWI) im Verband der Hochschullehrer für Betriebswirtschaft e.V. und des Fachbereichs Wirtschaftsinformatik (FB WI) in der Gesellschaft für Informatik e.V. (GI) vom 18. Februar 2011*. Zürich.

Wang, H. u. a. (2004). „XCpaqs: compression of XML document with XPath query support“. In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2004)*. Hrsg. von P. K. Srimani. Los Alamitos, CA: IEEE Computer Society, S. 354–358.

Weimin, L. (2003). „XComp: An XML Compression Tool“. Master Thesis. Waterloo: University of Waterloo. URL: https://cs.uwaterloo.ca/~tozsu/ddbms/publications/distdb/Weimin.pdf.

Wilde, T./Hess, T. (2007). „Forschungsmethoden der Wirtschaftsinformatik“. In: *WIRTSCHAFTSINFORMATIK* 49.4, S. 280–287.

Wong, R. K./Lam, F./Shui, W. M. (2007). „Querying and maintaining a compact XML storage“. In: *Proceedings of the 16th international conference on World Wide Web*. Hrsg. von C. Williamson. New York, NY: ACM, S. 1073–1082.

Internetquellen

Bray, T. u. a., Hrsg. (2006). *Extensible Markup Language (XML) 1.1: Second Edition*. URL: http://www.w3.org/TR/xml11 (besucht am 05. 03. 2016).

Clark, J./Murata, M., Hrsg. (2001). *RELAX NG Specification: Committee Specification 3 December 2001*. URL: https://www.oasis-open.org/committees/relax-ng/spec.html (besucht am 05. 03. 2016).

Gao, S./Sperberg-McQueen, C. M./Thompson, H. S., Hrsg. (2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. URL: https://www.w3.org/TR/xmlschema11-1/ (besucht am 05. 03. 2016).

Harrusi, S./Averbuch, A./Yehudai, A. (2006a). *Compact XML grammar based compression*. URL: http://www.cs.tau.ac.il/~amir1/PS/XML-compression.pdf (besucht am 16. 03. 2016).

Kamiya, T., Hrsg. (2016). *Efficient XML Interchange Working Group: Public Page*. URL: https://www.w3.org/XML/EXI/ (besucht am 08. 05. 2016).

Mahoney, M. V. (2002). *The PAQ1 Data Compression Program*. URL: http://mattmahoney.net/dc/paq1.pdf (besucht am 16. 03. 2016).

Martin, B./Basher, J. (1999). *WAP Binary XML Content Format*. URL: https://www.w3.org/TR/wbxml/ (besucht am 16. 04. 2016).

Nair, S. S. (2007). *XML Compression Techniques: A Survey*. URL: https://people.ok.ubc.ca/rlawrenc/research/Students/SN_04_XMLCompress.pdf (besucht am 14. 03. 2016).

Peterson, D. u. a., Hrsg. (2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. URL: https://www.w3.org/TR/xmlschema11-2/ (besucht am 05. 03. 2016).

Schneider, J. u. a. (2014). *Efficient XML Interchange (EXI) Format 1.0 (Second Edition)*. URL: https://www.w3.org/TR/exi/ (besucht am 23. 05. 2016).

Seward, J. (2007). *bzip2 and libbzip2, version 1.0.5: A program and library for data compression*. URL: http://bzip.org/1.0.5/bzip2-manual-1.0.5.pdf (besucht am 04. 05. 2016).

UniTeK GmbH, Hrsg. (2007). *Der PaperMatic Standard: Spezifikation des PaperMatic-Codes (PMC)*. URL: http://www.gebev.org/Media/PaperMatic-Standard_Spec.pdf (besucht am 22. 11. 2012).