Effiziente Codierungen für SOAP-Nachrichten

Christian Werner

Institut für Telematik
Universität zu Lübeck
werner@itm.uni-luebeck.de

Abstract: Web Services konnten sich in den letzten Jahren als universelle Middleware-Technologie in vielen Bereichen der Informatik etablieren. Durch den konsequenten Einsatz von Web-Standards – zu nennen ist hier vor allem XML – ermöglichen Web Services die Realisierung von verteilten Anwendungen auch in stark heterogenen Umgebungen. Allerdings bringt der durchgängige Einsatz von XML auch einen erheblichen Nachteil mit sich: Die Darstellung aller Daten als Text verursacht im Vergleich zu älteren Middleware-Technologien mit binärer Datenrepräsentation (z. B. CORBA oder Java RMI) ein deutlich höheres Datenaufkommen. Zwar spielt dieser Nachteil bei modernen, drahtgebundenen Netzwerken kaum noch eine Rolle, denn hier reicht die zur Verfügung stehende Datenrate in aller Regel aus; in Anwendungsfeldern aber, in denen die Kommunikation über eine Funkschnittstelle erfolgt, konnte sich die Web-Service-Technologie bislang nicht durchsetzen. Um Web Services auch hier effizient implementieren zu können, hat der Autor im Rahmen seiner Dissertation verschiedene Konzepte zur Reduzierung des Datenaufkommens bei der Web-Service-Kommunikation untersucht. Im folgenden Beitrag fasst er die wichtigsten Ergebnisse seiner Dissertation zusammen.

1 Einleitung

Obwohl XML ursprünglich gar nicht als Basis für Middleware-Anwendungen konzipiert wurde, sondern als universelles Datenaustauschformat für das World-Wide-Web, hat sich die Web-Service-Technologie in den letzten Jahren zu einem der wichtigsten Anwendungsbereiche für XML entwickelt.

Allerdings zeigt die durchgängige Verwendung von XML auch einen Nachteil, der je nach Anwendungsbereich mehr oder weniger gravierend ist: Das Verhältnis zwischen Payload und Overhead ist bei XML-basierter Kommunikation deutlich ungünstiger als bei der binären Übertragung.

Wie bereits oben angedeutet, führt diese größere Datenmenge in vielen Anwendungsbereichen zu keinen nennenswerten Nachteilen. Denn in Unternehmensnetzwerken – und diese sind heute zweifelsohne ein Hauptanwendungsbereich für XML-Protokolle wie SOAP – ist die Infrastruktur in aller Regel ausreichend dimensioniert, so dass der zusätzliche Overhead hier kaum ins Gewicht fällt. Bei drahtlos vernetzten Geräten indessen ist eine effi-

ziente Nutzung der zur Verfügung stehenden Datenrate[1] oftmals ein wichtiges Kriterium. Durch die physikalischen Eigenschaften der Luftschnittstelle arbeiten drahtlose Kommunikationstechniken in aller Regel langsamer als drahtgebundene. Kommunizieren viele Geräte im selben Frequenz- und Kommunikationsbereich, kommt hinzu, dass sich diese Geräte die ohnehin knappe Datenrate teilen müssen.

Im Rahmen seiner Dissertation [Wer07] hat der Autor verschiedene Verfahren untersucht, um dem hohen Datenaufkommen bei der Web-Service-Kommunikation zu begegnen.

Im ersten Teil seiner Dissertation präsentiert der Autor zwei neuartige Ansätze zur Datenkompression, die auf die besonderen Erfordernisse von Web Services zugeschnitten sind. Das erste Verfahren basiert auf dem Ansatz der Differenzcodierung. Hier wird die Differenz zwischen dem zu übertragenden SOAP-Dokument und einem so genannten Skelettdatensatz berechnet, der zuvor aus der WSDL-Beschreibung des Web Services generiert wurde. Das zweite Verfahren beruht auf der Erzeugung eines Kellerautomaten aus einer XML-Schema-Beschreibung, welcher dann zur Kompression der SOAP-Dokumente verwendet wird.

Im zweiten Teil untersucht der Autor das Datenaufkommen bei der Web-Service-Kommunikation in den einzelnen Protokollschichten und stellt ein neuartiges, besonders effizientes Anwendungsprotokoll vor, welches auf dem *User Datagramm Protocol (UDP)* [Pos80] aufsetzt.

Im Folgenden möchte der Autor keinen vollständigen Überblick über die in seiner Dissertation vorgestellten Ergebnisse liefern, denn dieser müsste wegen der gegebenen Textlängenlimitierung recht oberflächlich ausfallen. Stattdessen wird er nur einen ausgewählten Teilbereich seiner Arbeit vorstellen: die SOAP-Kompression mittels Kellerautomaten.

Im folgenden Abschnitt 2 gibt der Autor zunächst einen kurzen Abriss über bisher bekannte XML-Kompressionsverfahren und geht kurz auf ihre spezifischen Vor- und Nachteile ein. In Abschnitt 3 stellt er dann die SOAP-Kompression mittels Kellerautomaten vor und demonstriert in Abschnitt 4 ihre Leistungsfähigkeit anhand von Vergleichsmessungen. In Abschnitt 5 fasst er die Ergebnisse schließlich zusammen und zeigt mögliche Ansatzpunkte für weitere wissenschaftliche Untersuchungen auf diesem Themengebiet.

2 Stand der Forschung

Mit der fortschreitenden Verbreitung von XML wurde auch die Notwendigkeit einer kompakteren, binären Repräsentation von XML-Dokumenten zunehmend evident. Es gibt bereits eine Vielzahl von Arbeiten zu diesem Thema.

Eine erste Möglichkeit, um die Größe von XML-Nachrichten zu reduzieren, ist der Einsatz von *Universalkompressoren*. Hierunter sind solche Ansätze zu verstehen, die nicht nur für

[1] Der Begriff *Datenrate* bezeichnet das Datenvolumen pro Zeiteinheit, gemessen in Bytes pro Sekunde. Alternativ ist in der Informatik auch der Begriff *Bandbreite* gebräuchlich. Dieser wird vom Autor allerdings nicht verwendet, da er in der Physik und Nachrichtentechnik bereits mit einer anderen Bedeutung belegt ist (Breite eines Frequenzbandes).

XML-Daten funktionieren, sondern auch für beliebige Eingabedaten. Hier wird der Eingabedatensatz als eine Folge von Bytes aufgefasst, die mit Hilfe eines Kompressionsalgorithmus umcodiert wird. Ein besonders gängiges Verfahren ist dabei die gzip-Kompression. Andere bekannte Beispiele für Universalkompressoren sind bzip2, ZIP, RAR, ARJ usw., einen sehr guten und umfassenden Überblick bietet [Say00].

Leider führt die Verwendung solcher Universalkompressoren – neben einer nur mäßigen Effektivität der Kompression – auch zu ungünstigen Nebeneffekten bei der XML-Verarbeitung: Das XML-Dokument wird als Ganzes (bzw. in Blöcken fester Größe) codiert, und zwar ohne Berücksichtigung der durch das XML-Markup gegebenen Strukturinformation. Dies hat zur Folge, dass ein komprimiertes Dokument nicht geparst oder geändert werden kann, ohne das Dokument als Ganzes zu dekomprimieren. Im Anschluss an die durchgeführten Verarbeitungsschritte muss es dann erneut komprimiert werden.

Um diesen Nachteilen zu begegnen, stellt LIEFKE in [LS00] den Kompressor *XMill* vor. Dieser separiert zunächst das Markup eines Dokuments von den Zeichendaten[2] und komprimiert dann beides separat. Neben der XMill-Kompression wurden inzwischen weitere XML-Kompressoren entwickelt. Besonders bekannt ist xmlppm [Che01], weil die hier zugrunde liegenden Algorithmen sehr effektiv arbeiten. Solche Kompressoren, die sämtliche XML-Sprachen verarbeiten können, bezeichnen wir als *generische* XML-Kompressoren.

Ein weiterer Ansatz zur Erzeugung kompakter Binärcodierungen für XML-Dokumente ist *WAP Binary XML (WBXML)*. Diese binäre Repräsentation wurde bereits 1999 vom W3C standardisiert [Wor99] und dient der effizienten Codierung von *Wireless Markup Language (WML)* Dokumenten [Wir01]. WBXML arbeitet äußerst effektiv. Vergleichsmessungen [WBF05] haben ergeben, dass dieser Kompressor bei typischen Eingabedaten die Kompressionsleistung von generischen XML-Kompressoren wie xmlppm um ein Mehrfaches übertrifft. Dieser Vorteil kommt dadurch zu Stande, dass bei WBXML statistisch optimierte Codierungsregeln für die zu erwartenden Tag- und Attributnamen bereits fest im Kompressor integriert sind. Dieses „Vorabwissen“ über die zu verarbeitende XML-Sprache wirkt sich vor allem bei kurzen Datensätzen äußerst positiv auf das Kompressionsergebnis aus.

Allerdings ist es prinzipbedingt nicht möglich, SOAP-Nachrichten mit Hilfe von Verfahren wie WBXML zu komprimieren. Dies liegt an den besonderen Eigenschaften von SOAP: Anders als bei Sprachen wie WML, die nur eine sehr begrenzte Anzahl möglicher Tag- und Attributnamen erlauben, sind bei SOAP beliebige anwendungsspezifische XML-Daten innerhalb des Bodys möglich. SOAP ist damit eine besonders „dynamische“ XML-Sprache, für die es folglich kaum sinnvoll ist, feste Codetabellen vorzugeben.

Bei WBXML handelt es sich also um ein hochspezialisiertes Verfahren, das nur für ausgewählte XML-Sprachen geeignet ist. Die Grammatiken, welche diese Sprachen beschreiben, spiegeln sich in den Codetabellen wider, die im Kompressor statisch integriert sind. Darum bezeichnen wir diese Verfahren als *statisch-grammatikspezifisch*.

Die große Effektivität der statisch-grammatikspezifischen Kompressoren kommt also dadurch zu Stande, dass bereits vor dem Kompressionsvorgang bekannt ist, welche Attribute

[2] Unter dem Begriff *Zeichendaten* (engl.: character data) versteht man die Teile eines XML-Dokuments, die nicht Markup sind [Wor98].

und Tags in der zu verarbeitenden XML-Sprache vorkommen können. Bei WBXML ist diese Information fest im Kompressor integriert. Diese statische Integration der Codetabellen hat jedoch den Nachteil, dass der Anwendungsbereich dieser Kompressoren auf wenige ausgewählte XML-Sprachen beschränkt ist.

Diesem Nachteil begegnen *dynamisch-grammatikspezifische* Kompressoren, indem sie die Codierungsregeln nicht fest in den Kompressor integrieren, sondern stattdessen dynamisch (d. h. zur Laufzeit) aus einer XML-Grammatik berechnen. Eine solche Beschreibung liegt bei vielen praktischen Anwendungen als DTD- [Wor98] oder XML-Schema-Datei [Wor04] vor. Auch bei Web Services existiert eine solche Grammatikbeschreibung – typischerweise in Form einer XML-Schema-Grammatik, welche in der WSDL-Datei des Web Services eingebettet ist. Voraussetzung für alle dynamisch-grammatikspezifischen Verfahren ist, dass die Grammatikbeschreibung sowohl bei Kompression als auch bei der Dekompression bekannt ist. Ein besonders moderner Kompressor dieser Kategorie ist *Xebu* [KTL05]. Er kombiniert mehrere Kompressionstechniken miteinander, um eine möglichst hohe Effektivität zu erreichen.

3 XML-Kompression mittels Kellerautomaten

Leider eignen sich die bisher verfügbaren XML-Kompressionsverfahren nur bedingt für einen Einsatz auf Geräten mit beschränkten Speicher- und CPU-Ressourcen. Der Autor hat daher ein besonders ressourcenschonendes Kompressionsverfahren entwickelt, welches zugleich äußerst effektiv arbeitet.

Existierende Arbeiten zum Thema XML-Kompression, insbesondere Xebu, haben bereits gezeigt, dass sich Automatenstrukturen zur Repräsentation möglicher XML-Dokumentstrukturen einsetzen lassen [KTL05]. Allerdings verwenden die Autoren hier ausschließlich deterministische endliche Automaten (DEA). Diese sind jedoch nicht mächtig genug, um die Grammatik, die durch ein XML-Schema-Dokument beschrieben wird, vollständig zu repräsentieren. Dies liegt daran, dass die Sprachen, die durch XML-Grammatiken wie DTDs oder XML-Schema-Dokumente ausgedrückt werden können, nicht Teilmenge der regulären Sprachen sind. Ein DEA ist lediglich in der Lage, mögliche Sequenzen der *direkten* Kindelemente eines Elements zu beschreiben; dagegen ist eine Beschreibung, die auch die Kindelemente der Kindelemente (usw.) mit einschließt, mit einem einzigen DEA im Allgemeinen nicht zu realisieren. SEGOUFIN und VIANU behandeln dieses Problem detailliert in [SV02].

Das vom Autor entwickelte Datenkompressionsverfahren basiert dagegen auf der Verarbeitung von XML mittels *deterministischer Kellerautomaten (DKA)*. Obwohl das Parsen von XML mit DKA zumindest als theoretisches Konzept schon seit längerem bekannt ist [SV02], wurde dieser Automatentyp bisher noch nicht als Grundlage für XML-Datenkompressionsverfahren herangezogen.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="a" type="A"/>
 <xsd:complexType name="A">
  <xsd:choice>
   <xsd:element name="b" minOccurs="0">
    <xsd:complexType>
     <xsd:choice>
      <xsd:element ref="a" minOccurs="2" maxOccurs="2"/>
      <xsd:element name="c" type="xsd:int"/>
     </xsd:choice>
    </xsd:complexType>
   </xsd:element>
  </xsd:choice>
 </xsd:complexType>
</xsd:schema>
```

Abbildung 1: XML-Grammatik in XML-Schema-Darstellung

3.1 Erzeugung des Kellerautomaten

Vor dem eigentlichen Kompressionsvorgang wird aus der vorliegenden XML-Grammatik (ausgedrückt als XML-Schema-Datei) ein DKA konstruiert, der in der Lage ist, sämtliche XML-Dokumente aus der durch die Grammatik gegebenen XML-Sprache zu parsen.

Betrachten wir zunächst ein Beispiel: Abbildung 1 zeigt eine XML-Schema-Beschreibung und Abbildung 2 den resultierenden DKA.

Die Zustände des DKA ergeben sich wie folgt: Für jeden komplexen Typ der Grammatik werden im DKA zwei Zustände erzeugt, ein öffnender und ein schließender. Ein öffnender Zustand wird nach der Verarbeitung eines öffnenden Tags eingenommen und ein schließender nach der Verarbeitung des schließenden Tags. Für jeden einfachen Datentyp (z. B. `xsd:int`, `xsd:string`) wird hingegen nur ein Zustand erzeugt – einfache Datentypen beschreiben Zeichendaten und folglich werden keine öffnenden oder schließenden Tags verarbeitet. Weiterhin gibt es im DKA einen dedizierten Startzustand.

Dann werden die Zustandsübergänge erzeugt: Jeder Zustandsübergang im DKA ist mit einem 3-Tupel ($read$ / pop / $push$) als Zustandsübergangsmarkierung versehen. Hierbei bezeichnet $read$ das Zeichen, das der Automat als Nächstes aus dem Eingabedokument liest, pop das Symbol, welches an oberster Position im Kellerspeicher steht (dieses wird bei Ausführung der Transition aus dem Keller entfernt), und $push$ die Werte, die in den Keller geschrieben werden. Die Werte im Ausdruck $push$ werden von rechts nach links abgearbeitet und in dieser Reihenfolge in den Keller geschrieben.

Wie man in der Abbildung 2 sieht, wird bei jeder Transition, die zu einem öffnenden Zustand führt, ein zusätzliches Symbol auf den Stack geschrieben. Bei jeder Transition, die zu einem schließenden Zustand führt, wird dagegen ein Symbol vom Stack entfernt. Auf diese Weise werden über den Stack die einzelnen Schachtelungsebenen im zu verarbeitenden XML-Dokument dargestellt. Beim Erreichen des Dokument-Endes (in der Abbildung dargestellt durch das Zeichen $\#$) ist bei einem gültigen Eingabedokument der Stack leer, und der DKA terminiert.

Den genauen Algorithmus zur Erzeugung der Transitionen findet der Leser in [Wer07].

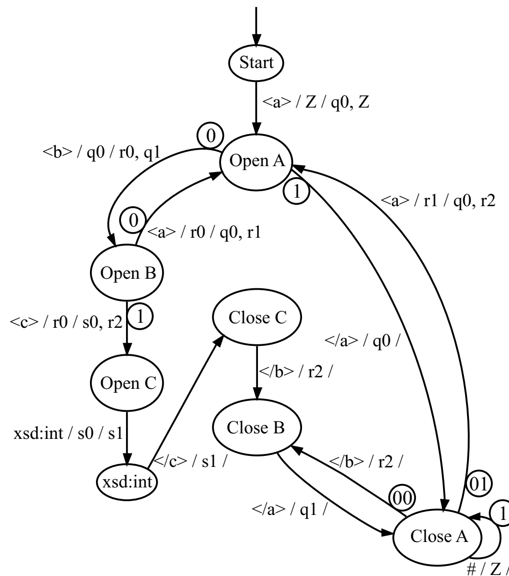


Abbildung 2: Kellerautomat, ergänzt um binäre Codewörter zur Codierung von Zustandsübergängen

3.2 Ablauf von Kompression und Dekompression

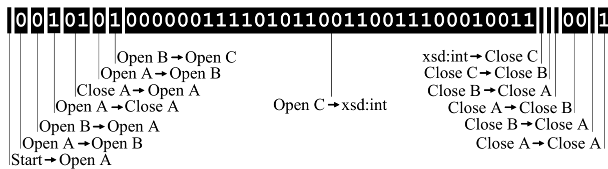
Die zentrale Idee des vom Autor entwickelten Kompressionsverfahrens besteht nun darin, einen solchen Parserautomaten auch für die Datenkompression zu verwenden: Der DKA wird hierzu sowohl auf Sender- als auch auf Empfängerseite aus der WSDL-Beschreibung (bzw. der darin enthaltenen XML-Schema-Grammatik) des Web-Services konstruiert, so dass beide Seiten über identische Kopien verfügen. Der Sender verarbeitet das zu übertragende Dokument mit seinem Automaten. Eine Datenkompression wird nun dadurch erreicht, dass lediglich der Pfad durch den DKA codiert wird. Anhand der Codewortfolge kann der Empfänger den Pfad durch den DKA nachvollziehen und so das XML-Dokument rekonstruieren. Wie wir im Folgenden sehen, ist die Codewortfolge, die den Pfad durch den Automaten repräsentiert, deutlich kompakter als die Textdarstellung des Dokuments.

Zur Codierung der Zustandsübergänge im DKA wird jeder Zustandsübergang mittels eines binären Codeworts codiert. Um ein optimales Kompressionsergebnis zu erreichen, muss die Länge dieser Bitsequenz – nach den Regeln der Informationstheorie – möglichst genau dem Entropiewert [Sha48] dieses Zustandsübergangs entsprechen. Alle möglichen Folgezustände werden dabei als gleich wahrscheinlich angenommen, denn in der Grammatikbeschreibung sind keine Angaben über die zu erwartenden Häufigkeiten enthalten.

Zur technischen Umsetzung dieser Idee kommt das Huffman-Verfahren [Say00] zum Einsatz. Für jeden Zustandsübergang wird so ein möglichst kurzes, eindeutig decodierbares Codewort erzeugt. Gibt es nur einen möglichen Folgezustand, braucht dieser nicht mit codiert zu werden. Abbildung 2 zeigt den Parser-DKA mit den so erzeugten Codewörtern (eingekreist dargestellt).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<a>
  <b>
    <a>
    </a>
    <a>
      <b>
        <c>64382739</c>
      </b>
    </a>
  </b>
</a>
```

(a) Codierung als Text (119 Bytes)



(b) Komprimierte Binärdarstellung (42 Bits ≈ 6 Bytes)

Abbildung 3: Beispieldokument vor und nach der Kompression

Neben dem Pfad durch den DKA – dieser repräsentiert schließlich nur das Markup des codierten Dokuments – müssen auch die Zeichendaten codiert werden: Für einige ausgewählte XML-Schema-Datentypen sind im Kompressions-DKA optimierte Codierungsregeln abgelegt. Diese erzeugen besonders kompakte Bitcodes fester Länge, z. B. 32 Bits für einen `xsd:int` Wert, ein einzelnes Bit für einen `xsd:boolean` Wert usw. Bei anderen Datentypen – also solchen, bei denen die Codierung als Bitfolge fester Länge nicht sinnvoll ist (z. B. bei `xsd:string`) – wird zeichenorientiert gearbeitet; sämtliche Zeichen werden in UTF-8-Codierung direkt in den komprimierten Bitstrom geschrieben und mit einer reservierten Stop-Byte-Sequenz terminiert. Statt der direkten (d. h. unkomprimierten) UTF-8-Codierung kann aber auch ein komprimierendes Codierungsverfahren wie z.B. Huffman oder PPM [Say00] verwendet werden.

In beiden Fällen wird der binär codierte Wert direkt an die aktuelle Position im Ausgabedatenstrom des Kompressors geschrieben. Dieses Vorgehen ist besonders vorteilhaft, denn auf diese Weise kann auf die Pufferung von Daten bei der Kompression oder Dekompression vollständig verzichtet werden. Dies ist ein klarer Vorteil gegenüber container-basierten Kompressionsverfahren wie etwa XMill.

Abbildung 3 zeigt ein Beispiel für ein Instanzdokument unserer Beispielgrammatik und die zugehörige Binärdarstellung. Wie hier deutlich zu erkennen ist, ist der binäre Datenstrom deutlich kompakter als die Textdarstellung.

Anhand dieser Abbildung lässt sich auch der Dekompressionsprozess anschaulich nachvollziehen. Der Empfänger verwendet den Bitstrom zur Steuerung seiner Kopie des DKA. Wegen der so genannten Präfixfreiheit [Say00] der verwendeten Huffman-Codierung kann ein Empfänger stets den Beginn und das Ende eines Codeworts erkennen. Bei jeder aus-

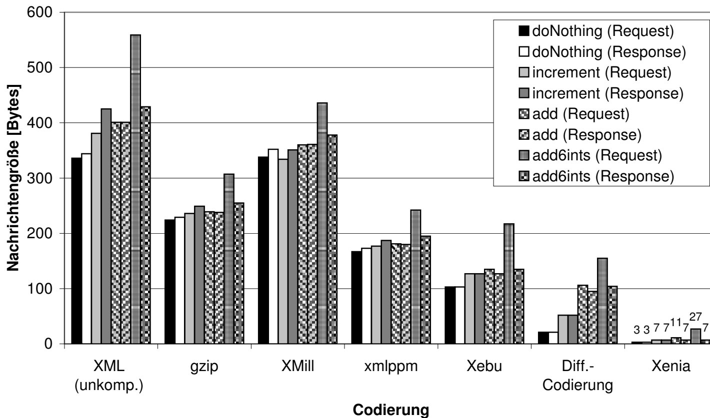


Abbildung 4: Kompressionsleistung des Xenia-Kompressors im Vergleich zu verwandten Arbeiten

geführten Transition schreibt er den korrespondierenden $read$ Wert in die Ausgabe und rekonstruiert so die Tag-Sequenz aus dem unkomprimierten Dokument. Auch die Zeichendaten können stets eindeutig rekonstruiert werden: In Abhängigkeit vom vorliegenden Datentyp wird entweder eine feste Anzahl von Bits gelesen (in unserem Beispiel wären dies 32 Bits für den `xsd:int` Wert) oder aber es werden solange Bytes von der Eingabe gelesen, bis die reservierte Stop-Byte-Sequenz im Bitstrom auftritt.

4 Kompressionsergebnisse

Im Folgenden zeigt der Autor die Kompressionsleistung seiner Implementierung – genannt Xenia – anhand eines Beispiels:

Der Web Service „Taschenrechner“ implementiert vier einfache RPC-Operationen, wie sie z. B. auch für Sensornetzanwendungen typisch wären. Dieser Web Service ist dadurch gekennzeichnet, dass die ausgetauschten Nachrichten durchgängig recht klein sind (zwischen 336 und 559 Bytes) und zu einem Großteil aus Markup bestehen. Die in der WSDL-Beschreibung enthaltene XML-Grammatik ist zudem sehr restriktiv, d. h. sie gibt die Tag-Reihenfolge in den Nachrichten weitestgehend fest vor.

Abbildung 4 zeigt eine grafische Darstellung der Kompressionsergebnisse verschiedener Kompressoren für die Request- und Response-Nachrichten dieses Web Services.

In der Abbildung ist klar zu erkennen, dass Xenia mit Abstand die besten Kompressionsergebnisse liefert. Dies war auch zu erwarten, denn die hier ausgetauschten SOAP-Nachrichten bestehen nahezu vollständig aus Markup. Zudem schreibt das aus der WSDL-Datei erzeugte XML-Schema-Dokument sehr restriktiv vor, wie die einzelnen Tags geschachtelt sein können, und somit ergeben sich im Kompressions-DKA nur sehr wenige Zustände mit mehr als einem Folgezustand. Dies bewirkt, dass nur sehr wenige Bits für die Codierung des Markups erforderlich sind.

Xenia arbeitet bei allen Nachrichten dieser Messreihe deutlich effektiver als der Universalkompressor gzip. Auch die beiden generischen Kompressoren XMill und xmlppm sowie der dynamisch-grammatikspezifische Kompressor Xebu werden von Xenia klar übertroffen. Auch der Ansatz „Differenzcodierung“ (dieser wurde ebenfalls vom Autor entwickelt [Wer07]) schneidet schlechter ab als Xenia.

Es ist anzumerken, dass der Vorteil der Xenia-Kompression weniger deutlich ausfällt, falls das Schema-Dokument nicht so restriktiv ist wie in diesem Beispiel. Wie in [Wer07] jedoch gezeigt wird, funktioniert auch in solchen Fällen die Xenia-Kompression im Vergleich zu anderen Verfahren äußerst vielversprechend.

5 Zusammenfassung und Ausblick

Im Rahmen dieses Beitrags hat der Autor ein ausgewähltes Thema seiner Doktorarbeit dargestellt: Es wurde ein neuartiger, dynamisch-grammatikspezifischer XML-Kompressor entwickelt. Anders als bisherige Ansätze kombiniert dieser nicht mehrere Kompressionsstrategien miteinander; vielmehr basiert er auf dem geschlossenen und einfach strukturierten Modell eines deterministischen Kellerautomaten (DKA).

Vergleichende Messungen anhand typischer SOAP-Nachrichten zeigten, dass der Ansatz des Autors hinsichtlich der erreichten Kompressionsrate den existierenden Verfahren deutlich überlegen ist. Die Vorteile sind bei restriktiven Schema-Beschreibungen am größten.

Der hier vorgestellte Ansatz hebt sich – neben den sehr guten Kompressionsleistungen – auch in einem zweiten wesentlichen Punkt von anderen Arbeiten ab: Die Vorgänge der Kompression (auf Senderseite) bzw. der Dekompression (auf Empfängerseite) beschränken sich im Wesentlichen auf den Durchlauf eines Kellerautomaten. Die Struktur dieses Kompressors ist damit so überschaubar, dass eine Implementierung selbst auf sehr kleinen, ressourcenbeschränkten Geräten praktikabel wird. Da der zur Kompression bzw. Dekompression verwendete Automat gleichzeitig auch als Parser fungiert, sind hierfür keine zusätzlichen Verarbeitungsschritte erforderlich. Dies ist ein erheblicher Vorteil bei einer Implementierung auf Geräten mit knappen Speicher- und CPU-Ressourcen.

Die konsequente Weiterentwicklung dieses Gedankens ist die Erzeugung spezialisierter Hardwarestrukturen: Der erzeugte DKA lässt sich offenbar auch in einer Hardwarebeschreibungssprache wie VHDL oder Verilog formulieren. Mit Hilfe entsprechender Synthesewerkzeuge ließen sich daraus dann spezialisierte Hardwarestrukturen zum Parsen und Komprimieren von XML erzeugen.

Zusammenfassend ist festzustellen, dass Kellerautomaten zur Kompression von XML-Dokumenten besonders vorteilhaft sind – und zwar nicht nur mit Blick auf praktische Anwendungen, sondern auch konzeptionell: Über die Anzahl möglicher Folgezustände beim Automatendurchlauf lässt sich eine direkte Beziehung zum SHANNON’schen Entropiebegriff herstellen. Damit ist dieses Modell ganz bestimmt auch für weitere theoretische Forschungsarbeiten interessant.

Literatur

[Che01] James Cheney. Compressing XML with Multiplexed Hierarchical PPM Models. In *Data Compression Conference*, Seiten 163–173, Snowbird, Utah, USA, 2001.

[KTL05] Jaakko Kangasharju, Sasu Tarkoma und Tancred Lindholm. Xebu: A Binary Format with Schema-Based Optimizations for XML Data. In *Proceedings of the International Conference on Web Information Systems Engeneering*, Seiten 528–535, New York City, New York, USA, November 2005.

[LS00] Hartmut Liefke und Dan Suciu. XMill: an efficient compressor for XML data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Seiten 153–164, Dallas, Texas, USA, 2000.

[Pos80] Jon Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[Say00] Khalid Sayood. *Introduction to data compression (2nd ed.).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 20:379–423, 623–656, Juli und Oktober 1948.

[SV02] Luc Segoufin und Victor Vianu. Validating Streaming XML Documents. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Seiten 53–64, Madison, Wisconsin, USA, 2002.

[WBF05] Christian Werner, Carsten Buschmann und Stefan Fischer. WSDL-Driven SOAP Compression. *International Journal of Web Services Research*, 2(1), 2005.

[Wer07] Christian Werner. *Optimierte Protokolle für Web Services mit begrenzten Datenraten*. Logos, Berlin, 2007.

[Wir01] Wireless Application Protocol Forum. Wireless Markup Language 2.0, September 2001.

[Wor98] World Wide Web Consortium (W3C). Recommendation: Extensible Markup Language (XML) 1.0, Februar 1998.

[Wor99] World Wide Web Consortium (W3C). Member Submission: WAP Binary XML Content Format, Juni 1999.

[Wor04] World Wide Web Consortium (W3C). Recommendation: XML Schema Part 1 – Structures, Second Edition, Oktober 2004.

Christian Werner ist seit April 2007 Juniorprofessor an der Universität zu Lübeck und leitet dort die Arbeitsgruppe Verteilte Systeme. Er hat im August 2006 seine Promotion in Informatik abgeschlossen. Zuvor war er am Forschungszentrum L3S in Hannover sowie an der Technischen Universität Braunschweig als wissenschaftlicher Mitarbeiter tätig. Seine Forschungsarbeit konzentriert sich derzeit auf Technologien für den Aufbau von serviceorientierten Architekturen, insbesondere Web Services. Sein besonderes Interesse gilt dabei Verfahren zur Effizienzsteigerung, so dass die Web-Service-Technologie künftig auch auf Kleinstgeräten mit sehr geringen Energie-, Speicher- und CPU-Ressourcen eingesetzt werden kann.