Effizientes Änderungsmanagement von XML-Dokumenten

Sebastian Rönnau

sebastian.roennau@unibw.de

Abstract: XML-Dokumente bilden die Grundlage zahlreicher Informationsarchitekturen. Insbesondere in parallelen Bearbeitungsprozessen kommt der Fähigkeit, Änderungen zwischen Versionen erkennen und zusammenführen zu können, eine tragende Rolle zu. Bisherige Ansätze hierzu benötigen in der Regel einen entsprechende Applikation zum Bearbeiten der Dokumente. Im Bereich der Softwareentwicklung ist es hingegen üblich, verschiedene Versionen mit *Diff* und *Patch* zu verglichen bzw. zusammenzuführen. Entsprechende XML-basierte Tools existieren zwar, sind aber nicht in der Lage, parallel bearbeitete Versionen zusammenzuführen (*Merge*).

In dieser Arbeit stelle ich ein umfassendes Merge-fähiges Framework zum Vergleichen und Zusammenführen von XML-Dokumenten vor. Es basiert auf einem kontextorientierten Delta-Modell, auf dem effiziente Diff- und Patch-/Merge-Algorithmen aufbauen. Sowohl die Effizienz als auch die Qualität sind empirisch belegt.

1 Einleitung

XML hat sich als universelle Meta-Sprache im Bereich der Dokumentenbeschreibung etabliert. XML-basierte Formate wie ODF, OfficeOpenXML, XHTML, SVG u.v.m. dienen als de-facto Standard der Darstellung zahlreicher verschiedener Dokumentenarten.

Insbesondere im professionellen Umfeld werden Dokumente gemeinsam erstellt und bearbeitet. Hürden technischer und organisatorischer Natur verhindern oft den Einsatz kollaborativer Editoren. Daher ist es gängige Praxis, Dokumente per E-Mail o.ä. auszutauschen. Das Zusammenfügen verschiedener Dokumentenversionen wird in diesem Kontext in der Regel manuell durchgeführt, was sowohl zeitintensiv als auch fehleranfällig ist.

Während sich im Bereich des Document Engineering die Tool-Unterstützung zum Vergleichen von Dokumentenversionen noch in den Kinderschuhen befindet, stellt das automatisierte Vergleichen und Zusammenführen von Dateiversionen eine zentrale Säule des Software Engineering dar. Als bekannteste Tools seien *Diff* und *Patch* genannt. Diff vergleicht zwei Dokumentenversionen und speichert die Änderungen in einem *Delta*. Patch wendet wiederum dieses Delta auf die zu ändernde Datei an, um die Änderungen auch dort umzusetzen. Wird ein Delta auf eine bereits anderweitig geänderte Version angewendet, spricht man von einem *Merge*. Hier wird eine neue Dokumentenversion erstellt, die die Änderungen sowohl aus der vorliegenden Version als auch aus dem Delta enthält.

Ein Merge ist nicht trivial, wie Abbildung 1 zeigt. Zwei Personen ändern unterschiedliche Zellen einer Tabelle. Wenn nun die Änderungen anhand ihrer absoluten Adressierung dargestellt werden („Ändere Zelle 1,3 in 10:15“), wird beim Merge von Version A_2 mit $\delta_{A \to A_1}$

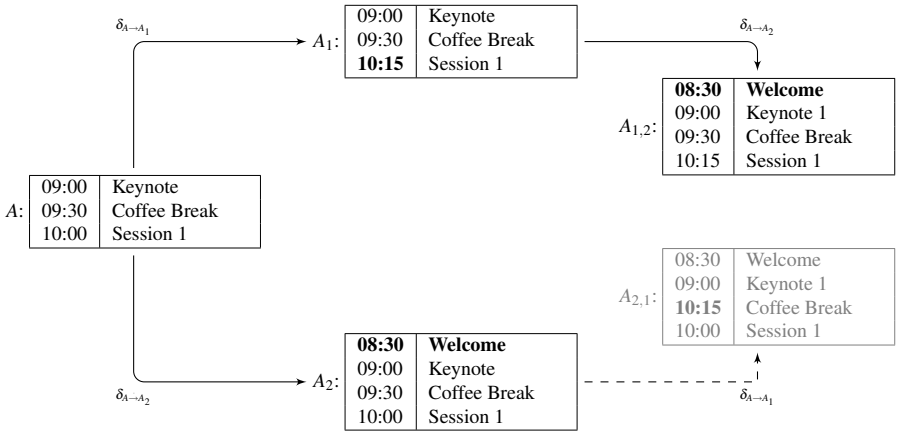


Abbildung 1: Beim Merge führt eine absolute Adressierung leicht zu unerwünschten Ergebnissen.

die falsche Zelle geändert. Um dies zu verhindern, muss die Änderungsoperation im Delta mit weiteren Daten versehen werden, um ein zuverlässiges Merge zu ermöglichen.

1.1 Forschungsstand

Das Vergleichen von XML-Dokumentenversionen stellt einen zustandsbasierten Ansatz dar und ist eine Anwendung des Tree-to-Tree Editing Problems [Sel77]. Eine optimale Lösung ist mit quadratischer Zeitkomplexität zu berechnen. Verschiedene Lösungsansätze erreichen nahezu optimale Lösungen bei höherer Effizienz. Ein Ansatz für den zustandsbasierten Merge von XML-Dokumenten wurde kürzlich vorgestellt [TM10]. Da hier die Änderungen anhand von Schlüsseln identifiziert werden, lässt sich dieser Ansatz nur in Umgebungen verwenden, die die Schlüssel unverändert übernehmen.

Kollaborative Editoren basieren in der Regel auf dem Ansatz der operationellen Transformation. Hier wird jede Änderung umgehend an alle Beteiligten weitergegeben [SE98]. Dieser Ansatz erreicht allerdings Komplexitätsgrenzen, wenn die Verbindung über einen längeren Zeitraum unterbrochen ist. Des weiteren ist die permanente Nachvollziehbarkeit aller Aktivitäten im professionellen Umfeld häufig unerwünscht.

Um das Problem der Änderungsadressierung zu umgehen, können die Änderungen direkt im Dokument als Annotation eingepflegt werden [ARPMMG07]. Dieser Ansatz wird auch von der Änderungsverfolgung in Office verfolgt. Die Annotation führt allerdings schnell zu sehr umfangreichen Dokumenten. Außerdem bedingt das Zusammenführen verschiedener Evolutionsstränge eines Dokuments wiederum einen zustandsbasierten Lösungsansatz.

1.2 Gliederung

Das vorgestellte Framework zum Diff, Patch und Merge von XML-Dokumentversionen basiert auf einer Analyse der Eigenschaften und Änderungsmuster von XML-Dokumenten (Abschnitt 2). Den Kern bildet ein Delta-Modell, das Änderungen mit Hilfe des syntaktischen Kontexts der einzelnen Operationen identifiziert (Abschnitt 3). Darauf aufbauend entwickle ich einen Diff- (Abschnitt 4) und einen Merge-fähigen Patch-Algorithmus (Abschnitt 5), deren Effizienz und Zuverlässigkeit in Abschnitt 6 evaluiert werden.

2 Dokumentenanalyse

Voraussetzung für das Design des Delta-Modells als auch des Diff-Algorithmus sind Erkenntnisse über das Wesen und die Evolution von XML-Dokumenten.

2.1 Eigenschaften von XML-Bäumen

Um möglichst allgemein gültige Erkenntnisse zu gewinnen, habe ich Tabellen, Texte und Web-Seiten jeglicher Größe aus öffentlichen Quellen untersucht. Die Ordnung der Knoten ist hier bei allen Dokumenten wesentlich. Eine Veränderung der Reihenfolge der Knoten führt zu unterschiedlichen Dokumenten. Diese Feststellung ist wichtig, da XML bisweilen auch als reines Datenspeicherungsformat verwendet wird, in dem die nebeneinander liegenden Knoten als Menge interpretiert werden.

Generell lässt sich sagen, dass XML-Dokumente breit und flach sind. Kein Office-Dokument umfasst mehr als 15 Ebenen, was in Anbetracht von zum Teil mehr als 100.000 Knoten eine sehr flache Hierarchie ist. Auch Web-Seiten sind breit und flach, wenn auch nicht in gleichem Ausmaß. Des weiteren weist jede Knotenebene eine erhebliche Anzahl identischer Knoten auf. Dies betrifft vor allem die inneren Knoten des Baumes, die vorwiegend der Strukturierung und dem Layout dienen. Selbst auf Ebene der Blätter sind im Schnitt noch über 40% der Knoten identisch.

2.2 Evolution von XML-Dokumenten

Bei der Frage der Evolution von XML-Dokumenten stütze ich mich auf Untersuchungen zum Änderungsaufkommen von Web-Seiten [FMNW03]. Hier wurde erkannt, dass Änderungen in Dokumenten vorwiegend räumlich zusammenhängend auftreten. Des weiteren ist die Anzahl der Änderungen im Vergleich zum Gesamtumfang des Dokumentes in der Regel eher klein.

Neben der Anzahl der Änderungen ist auch ihre Ausprägung interessant. Bereits einfache Änderungen im Applikationskontext eines Dokumentes können zu weitreichenden

Änderungen der XML-Repräsentation führen; selbst kleine Änderungen haben oft das Einfügen bzw. Löschen kompletter Teilbäume bzw. Wälder zur Folge.

3 Ein kontext-orientiertes Delta-Modell

Grundlage der zustandsbasierten Änderungsverfolgung von Dokumenten ist die Darstellung der Änderungen in einem Delta. Hierzu stelle ich ein Delta-Modell vor, dessen zentraler Bestandteil die Einbeziehung des syntaktischen Kontextes einer Operation darstellt.

3.1 Eigenschaften

Ein Delta besteht aus einer Menge von Operationen, die Änderungen an einem Dokument definieren. Hierbei lassen sich folgende Operationsarten unterscheiden: Einfügung, Löschung, Änderung und Verschiebung. Für jede Operation wird der Ort der Änderung und ihr Umfang festgehalten.

Das Delta-Modell definiert Einfügungen, Löschungen und Verschiebungen auf Teilbäume bzw. Wälder; Änderungen adressieren einzelne Knoten im Baum. Für jede Operation wird sowohl der vorherige als auch der neue Zustand des zu ändernden Objekts gespeichert. Dadurch wird die Invertierbarkeit jeder Operation gewährleistet, die das Wiederherstellen vorheriger Versionen gewährleistet. Sequentielle Deltas können zur kompakteren Darstellung größerer Änderungsumfänge zusammengeführt werden.

3.2 Darstellung des syntaktischen Kontextes

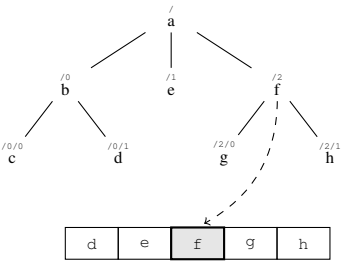
Die Definition des syntaktischen Kontextes orientiert sich an der Darstellung von Deltas im Bereich der zeilenbasierten Textdateien. Hier werden die vorhergehenden und nachfolgenden Zeilen einer Änderung gespeichert, wie Abbildung 2(a) zeigt. Diese Darstellung ist noch heute üblich und wird sowohl von klassischen Versionskontrollsystemen wie CVS und Subversion, aber auch von modernen verteilten Systeme wie Git, Arch und Bazaar auf Grund ihrer Zuverlässigkeit und Einfachheit genutzt.

Die Darstellung des syntaktischen Kontextes im Bereich von XML-Dokumenten ist hingegen nicht trivial. Die Baumstruktur bietet sowohl Eltern-Kind-, als auch Geschwisterbeziehungen als mögliche Ordnung. Im Hinblick auf die sehr flache Baumstruktur ist die Eltern-Kind-Beziehung nicht zielführend. Auch die Verwendung von Geschwisterknoten ist unzweckmäßig, da viele Blätter über keine Geschwister verfügen. Daher betrachte ich die Knoten in Pre-Order-Traversierung, um so eine Quasi-Linearität zu gewährleisten.

Für jede Operation im Delta wird ein *Context Fingerprint* erstellt. Dieser enthält den den Knoten vor und nach der Operation in Pre-Order-Traversierung. Bei Löschoperationen sind die zu löschende Knoten nicht Teil des Context Fingerprints.Abbildung 2(b) skizziert

```
@@ -3,6 +3,7 @@
finem di dederint, Leuconoe, nec Babylonios
temptaris numeros. Ut melius quicquid erit
pati!
Seu pluris hiemes seu tribuit Iuppiter
ultimam,
+
quae nunc oppositis debilitat pumicibus mare
Tyrrhenum, sapias, vina liques et spatio
brevi
spem longam reseces. Dum loquimur, fugerit
invida
```

(a) Kontext zum Einfügen einer Leerzeile



(b) Context Fingerprint von `f`

Abbildung 2: Bei Textdateien wird der syntaktische Kontext durch die vorhergehenden und nachfolgenden Zeilen dargestellt, beim Context Fingerprint durch die umliegenden Knoten in Pre-Order.

einen Context Fingerprint. Hierbei enthält der mittlere Wert den zu ändernden Knotens oder Teilbaums[1]; dies ermöglicht dem Patch-Algorithmus, Merge-Konflikte zu erkennen.

4 Diff-Algorithmus

Dem Diff-Algorithmus liegt die aus der Dokumentenanalyse entwickelte Annahme zu Grunde, dass die Blätter des Baumes den Inhalt des Dokumentes enthalten, während die Knoten dem Markup und der Struktur dienen. Weiterhin ist der Algorithmus auf einen geringen Umfang der Änderungen im Vergleich zum Gesamtdokument hin optimiert.

4.1 Idee des Algorithmus

Vereinfacht dargestellt, durchläuft der Diff-Algorithmus drei Phasen:

1. Berechnen der größten gemeinsamen Sequenz an Blättern. Diese Phase stützt sich auf das wohlerforschte Feld der Berechnung der Longest Common Subsequence (LCS) ab. Durch den Vergleich der Blätter konzentriert sich der Algorithmus bereits zu Beginn auf die Änderungen am Inhalt der Dokuments.

2. Vergleichen der Ahnen der gemeinsamen Blätter. Dieser Schritt dient dem Aufspüren von Knoten- bzw. Attributänderungen.

3. Untersuchen der nicht-gemeinsamen Blätter und deren Ahnen. Erst in diesem Schritt werden die Änderungen an der XML-Struktur des Dokuments untersucht.

[1] Tatsächlich werden jeweils die Hash-Werte der Knoten bzw. Teilbäume gespeichert.

Sowohl der zweite als auch der dritte Schritt sind mit Hilfe der dynamischen Programmierung umgesetzt. Diese verhindert, dass bereits besuchte Elternknoten mehrfach untersucht werden. Hierdurch wird die Laufzeit des Algorithmus erheblich reduziert.

4.2 Komplexitätsbetrachtung

Die Komplexität des Diff-Algorithmus hängt vom gewählten LCS-Algorithmus im ersten Schritt ab. Ich verwende den etablierten Algorithmus von Myers [Mye86], der mit einer Zeitkomplexität von $O(ND)$ insbesondere für Dokumente mit relativ wenig Änderungen geeignet ist[2]. Auf dieser Grundlage lässt sich die Komplexität des Diff-Algorithmus mit $O(leaves(A) \times D + ancestors(A) + D)$ abschätzen[3]. Augenscheinlich sind die Anzahl der Blätter und die Anzahl der Änderungen die maßgeblichen Faktoren.

Neben der Zeitkomplexität ist die Speicherkomplexität nicht außer Acht zu lassen. Zahlreiche LCS-Algorithmen weisen eine quadratische Komplexität auf, da sie eine Matrix zur Berechnung der LCS verwenden. Im Hinblick auf die Größe mancher XML-Dokumente ist dies ungeeignet. Mit $O(A)$ erreicht mein Diff-Algorithmus eine lineare Komplexität.

5 Merge von Dokumentenversionen

Patch und Merge von Dokumenten unterscheiden sich lediglich dadurch, dass bei einem Merge die Operationen nicht auf die Dokumentversion angewendet werden, für das das Delta berechnet wurde.

5.1 Arten des Merge

Wird eine Operation in der im Delta beschriebenen Form an entsprechender Stelle angewendet, liegt ein Patch vor. Sollte das zu patchende Dokument hingegen in der Zwischenzeit verändert worden sein, können folgende Merge-Fälle eintreten:

1. Die Position der Operation hat sich verschoben, wie im einleitenden Beispiel skizziert. Hier wird die richtige Position anhand des Context Fingerprints identifiziert.

2. Ein Knoten in der Umgebung der Operation hat sich geändert, weshalb der Context Fingerprint nur in Teilen der gefundenen Umgebung entspricht. Hier ist zu definieren, ob, und wenn ja, in wie weit eine teilweise Entsprechung akzeptabel ist.

3. Die durch die Operation angesprochenen Knoten haben sich geändert. Dies stellt einen Konflikt dar, der sich nicht automatisch lösen lässt.

[2] N steht hierbei für die Größe der Sequenz und D für die Anzahl der Änderungen.
[3] A stellt hier die Anzahl der Knoten dar, während D für die Anzahl der Änderungen steht.

Tool	Zeitkomplexitätsklasse
XCC Diff	$O(\text{leaves}(A) \times D + \text{ancestors}(A) + D)$
diffxml	$O(\text{leaves}(A) \times D + D^2)$
faxma	$O(A)/O(A^2)$ (Durchschnitt / ungünstigstenfalls)
jXyDiff	$O(A \times \log A)$
Microsoft XML Diff	$O(A_1 \times A_2 \times \min(\text{depth}(A_1), \text{leaves}(A_1)) \times \min(\text{depth}(A_2), \text{leaves}(A_2)))$

Tabelle 1: Übersicht der verglichenen XML-Diff-Tools.

Generell können die einzelnen Fälle auch kombiniert auftreten. Insbesondere die Kombination der letzten beiden Fälle lässt ein (sinnvolles) automatisches Merge nicht zu.

5.2 Idee des Algorithmus

Die Anwendung einer Operation beginnt mit der Erstellung einer Suchumgebung um die im Delta gespeicherten Position. Für jeden möglichen Knoten in der Suchumgebung auf der selben Baumebene wird der Context Fingerprint angelegt und mit dem gefunden Muster verglichen. Die Suchumgebung verhindert, dass im Falle einer nicht anwendbaren Operation das gesamte Dokument durchsucht wird.

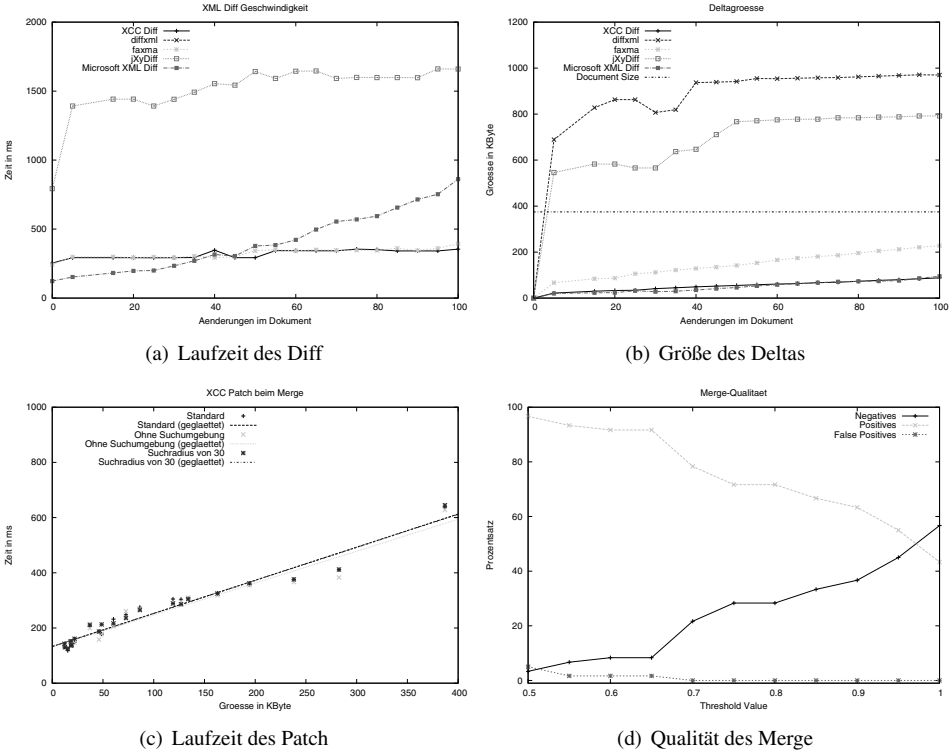
Ist der Context Fingerprint an keiner Position vollständig anwendbar, wird überprüft, welche Einzelteile des Context Fingerprints passen. Eine Gewichtungsfunktion bewertet das Ergebnis. Ein *Threshold Value* legt fest, bis zu welcher Bewertung eine Operation angewendet werden soll. Abschließend wird die Operation mit Hilfe ihres gespeicherten Inhalts auf einen Konflikt hin überprüft.

5.3 Komplexitätsbetrachtung

Die Laufzeit des Merge-Algorithmus lässt sich mit $O(A \times D)$ abschätzen. Die Speicherkomplexität ist hingegen linear mit $O(A)$.

6 Evaluation

Die Nutzbarkeit eines Diff- und Merge-Algorithmus hängt neben der theoretisch nachgewiesenen Komplexität wesentlich von der Effizienz der verfügbaren Implementierung ab. Weiterhin muss das Merge sinnvolle Ergebnisse garantieren.



(a) Laufzeit des Diff

(b) Größe des Deltas

(c) Laufzeit des Patch

(d) Qualität des Merge

Abbildung 3: Sowohl Zeit- und Speichereffizienz als auch Merge-Zuverlässigkeit sind sehr gut.

6.1 Effizienz des Diff

Die Effizienz der Implementierung wird im Vergleich zu besten bekannten XML-Diff-Implementierungen untersucht. Als Testszenario dient eine ODF-Tabelle mit 5-100 Änderungen, die jeweils mit dem Ursprungsdokument verglichen wird.

Während manche der Vergleichskandidaten speziell für XML entwickelt wurden (XyDiff [CAM02]; faxma [LKT06]), basieren die anderen auf allgemeinen Baumvergleichsalgorithmen (diffxml [CRGMW96]; Microsoft XML Diff, basierend auf [ZS89]). Die verglichenen Tools sind in Tabelle 1 aufgeführt. Der Komplexitätsklasse nach zu urteilen, sollten Microsoft XML Diff, faxma und jXyDiff von der Anzahl der Änderungen unbeeinflusst bleiben. Dennoch zeigt das Ergebnis der Untersuchung in Abbildung 3(a), dass lediglich faxma und XCC Diff eine nahezu konstante Laufzeit aufweisen[4].

Deltas sollten in der Regel deutlich kleiner als das Ursprungsdokument sein. Abbildung 3(b) stellt die Größe der im Diff-Test generierten Deltas dar. Da die Deltas größer als das eigentliche Dokument werden, erweisen sich diffxml und jXyDiff als wenig zweckmäßig.

[4]Die Laufzeit von diffxml ist auf der Grafik nicht aufgeführt, da sie stetig über 35 Sekunden betragen hat.

6.2 Merge-Effizienz und -Qualität

Die Effizienz des Merge untersuche ich anhand der Fragestellung, wie sich die Laufzeit für die gleiche Anzahl Operationen bei unterschiedliche Dokumentengröße entwickelt. Des weiteren untersuche ich den Einfluss der Umgebungssuche nach der besten Lösung sowie der Konflikterkennung. Da mir bislang ist kein vergleichbarer Merge-fähiger Patch-Algorithmus für XML-Dokumente bekannt ist, kann die Evaluation vom XCC Patch nicht vergleichend erfolgen. Abbildung 3(c) zeigt, dass die Laufzeit linear mit der Dokumentgröße skaliert, was der Komplexitätsabschätzung entspricht. Der Einfluss der Konflikterkennung sowie die Größe der Suchumgebung beeinflussen die Laufzeit nur marginal.

Um die Qualität des Merge zu messen, wurden über 1.000 Operationen auf ein Dokument angewandt. Hierbei wurde vorher festgelegt, ob, und wenn ja, an welcher Stelle eine Operation durchzuführen ist. Im Anschluss wurde das Ergebnis des Algorithmus mit dem erwarteten Ergebnis verglichen. Abbildung 3(d) zeigt das Ergebnis in Abhängigkeit zum Threshold Value (siehe Abschnitt 5.2). Ist der Threshold Value zu hoch, werden akzeptable Operationen abgelehnt (*False Negatives*); ist er zu niedrig, werden womöglich Operationen irrtümlich angewandt (*False Postives*). Letzteres gilt es auf jeden Fall zu verhindern. Ab einem Threshold Value über 0,7 treten in der Evaluation keine False Positives mehr auf; ein noch höherer Threshold Value führt hingegen zu einem starken Anstieg der False Negatives. Daher ist 0,7 als Standardwert gesetzt.

7 Zusammenfassung und Ausblick

Diese Arbeit stellt die erste ganzheitliche Betrachtung der verschiedenen Aspekte der zustandsbasierten Änderungsverfolgung in XML-Dokumenten dar. Die Grundlage legt hierbei die Analyse ihrer Eigenschaften und Änderungsmuster. Ein daraus abgeleitetes Delta-Modell ermöglicht es, Änderungen präzise zu adressieren und mit Hilfe ihres syntaktischen Kontextes zu identifizieren. Darauf aufbauend ermöglichen ein Diff- und ein Merge-fähiger Patch-Algorithmus das Vergleichen und Zusammenführen von Dokumentenversionen. Beide Algorithmen arbeiten äußerst effizient, wie auch das Delta eine äußerst kompakte und zuverlässige Darstellungsform bietet. Im Rahmen einer Evaluation wurden sowohl die Effizienz als auch die Qualität des Ergebnisses nachgewiesen.

Das vorgestellte Framework ist vielseitig einsetzbar und bereits in Anwendungen integriert worden. Hier sei stellvertretend ein grafischer XML-Editor und ein auto-versionierendes Dateisystem genannt. Die weitere Forschung konzentriert sich zum einen auf die Frage der inkrementellen Validierung der Merge-Ergebnisse. Zum anderen untersuche ich weitere Anwendungsmöglichkeiten meines Frameworks zur Sicherstellung der strukturierten Dokumentenbearbeitung, um im Bereich des Document Engineering ähnlich hohe Qualitätsstandards wie im Software Engineering erreichen zu können.

Literatur

[ARPMMG07] Luis J. Arévalo Rosado, Antonio Polo Márquez und Jorge Martínez Gil. Managing Branch Versioning in Versioned/Temporal XML Documents. In *Proc. of the 5th int. Symp. on XML Database: Database and XMLTechnologies*, Seiten 107–121, Berlin, Heidelberg, 2007. Springer-Verlag.

[CAM02] Grégory Cobéna, Serge Abiteboul und Amélie Marian. Detecting Changes in XML Documents. In *ICDE'02: Proc. of the 18th Int. Conf. on Data Engineering*, Seiten 41–52. IEEE Computer Society, 2002.

[CRGMW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina und Jennifer Widom. Change detection in hierarchically structured information. In *SIGMOD '96: Proc. of the 1996 ACM SIGMOD Int. Conf. on the Management of Data*, Seiten 493–504, New York, NY, USA, 1996. ACM.

[FMNW03] Dennis Fetterly, Mark Manasse, Marc Najork und Janet Wiener. A large-scale study of the evolution of web pages. In *WWW'03: Proc. of the 12th Int. Conf. on the World Wide Web*, Seiten 669–678, New York, NY, USA, 2003. ACM.

[LKT06] Tancred Lindholm, Jaakko Kangasharju und Sasu Tarkoma. Fast and simple XML tree differencing by sequence alignment. In *DocEng'06: Proc. of the 6th ACM Symp. on Document Engineering*, Seiten 75–84, New York, NY, USA, 2006. ACM.

[Mye86] Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

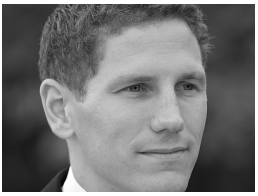
[SE98] Chengzheng Sun und Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *CSCW'98: Proc. of the 1998 ACM Conf. on Computer Supported Cooperative Work*, Seiten 59–68, New York, NY, USA, 1998. ACM.

[Sel77] Stanley M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 6(6):184–186, 1977.

[TM10] Cheng Thao und Ethan V. Munson. Using versioned tree data structure, change detection and node identity for three-way XML merging. In *Proc. of the 10th ACM symp. on Document engineering*, DocEng '10, Seiten 77–86, New York, NY, USA, 2010. ACM.

[ZS89] K. Zhang und D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

Sebastian Rönnau



Sebastian Rönnau wurde 1981 in Kiel geboren. Nach seinem Abitur trat er in die Luftwaffe ein und wurde zum Offizier ausgebildet. Er studierte Informatik an der Universität der Bundeswehr München von 2001 bis 2004. Anschließend wechselte er in das SAP-Projekt der Bundeswehr nach Bonn. Von 2007 bis 2010 war er als Wissenschaftlicher Mitarbeiter am Institut für Softwaretechnologie der Universität der Bundeswehr München tätig. Seit 2010 leitet er die Mobile Systembetriebszentrale des Führungsunterstützungsregimentes 38 in Storkow. Sebastian Rönnau wird die Bundeswehr in 2012 verlassen.