Kooperatives XML-Authoring: Integration von Workgroups in Workspace-Szenarien

Francis Gropengießer
francis.gropengiesser@tu-ilmenau.de

Zusammenfassung

Kooperatives XML-Authoring, wie es u.a. in Sounddesign-Prozessen erfolgt, findet in unterschiedlichen Systemumgebungen statt. Workgroups stellen dabei eng gekoppelte, Workspace-Szenarien hingegen lose gekoppelte Systemumgebungen dar. Für das Workgroup-Szenario haben wir in vergangenen Arbeiten bereits eine Lösung vorgeschlagen und implementiert, die einen hohen Grad an Kooperation ermöglicht. In dieser Arbeit stellen wir einen ersten Ansatz vor, wie bestehende Workgroups in Workspace-Szenarien integriert werden können. Hauptbestandteil dieses Integrationskonzepts ist ein maßgeschneidertes Versionierungssystem, welches die Synchronisierung konkurrierender Updates ermöglicht. Dieses arbeitet nicht, wie z.B. Subversion, auf der textuellen Repräsentation der XML-Daten, sondern betrachtet die Daten als Baum. Deltas zwischen verschiedenen Versionen werden aus den innerhalb einzelner Workgroups gespeicherten Transaktions-Recovery-Informationen gewonnen. Es ist daher kein aufwendiger Diff-Algorithmus notwendig. Weiterhin ermöglicht uns die Wiederverwendung der Recovery-Informationen auch Verschiebeoperationen zu unterstützen. Für die serverseitige Versionsverwaltung untersuchen wir einige ausgewählte Verfahren auf ihre Einsatztauglichkeit im Workspace-Szenario. Dabei handelt es sich um die Versionsverwaltung mit „Completed Deltas“, ein referenzbasiertes Versionsmodell sowie ein Versionsstempel-Verfahren.

1 Einleitung

Kooperatives XML-Authoring ist in vielen Medienproduktionsprozessen eine zentrale Aufgabenstellung. Ein Beispiel ist die Postproduktion von Filmton mit dem räumlichen Soundsystem IOSONO (*http://www.iosono-sound.com/*). Dieses System ermöglicht die Positionierung virtueller Soundquellen im dreidimensionalen Raum, wodurch ein realistischer Raumklangeindruck entsteht. Die bei den Arbeiten der Autoren entstehenden Metainformationen werden in XML-Dateien (Szenengraphen) abgelegt. Ziel unserer Bemühungen ist es, ein kooperatives Bearbeiten gleicher Szenengraphen durch mehrere Autoren zu ermöglichen. Dabei müssen jedoch zwei unterschiedliche Systemumgebungen unterschieden werden, in denen das XML-Authoring abläuft:

- Das *Workgroup-Szenario* beschreibt eine eng gekoppelte Systemumgebung. Hier arbeitet ein kleines Autorenteam gemeinsam an einer Soundszene. Dabei ist die räumliche Trennung der Autoren relativ gering. Es kann davon ausgegangen werden, dass sie sich in einem Gebäude befinden. In [GS08, GHS09] haben wir eine Lösung für das Workgroup-Szenario vorgeschlagen und implementiert. Diese basiert auf einem Client-Server-Modell, in dem alle Autoren (Clients) permanent mit dem zentralen Datenbestand (Server) verbunden sind. Die Arbeitsschritte der Autoren werden in maßgeschneiderten Transaktionen gekapselt und über ein Sperrprotokoll, welches ihre Semantik ausnutzt, synchronisiert. Die Systemlösung berücksichtigt die besonderen Anforderungen kooperativer Designprozesse und ermöglicht daher eine unmittelbare Propagierung von Änderungen, einen multi-direktionalen Informationsfluss sowie das Verwerfen einzelner Arbeitsschritte.
- Im Gegensatz zum Workgroup-Szenario beschreibt das *Workspace-Szenario* eine lose gekoppelte Systemumgebung. Hier arbeiten mehrere Entwicklerteams, z.B. verschiedene Tonstudios, u.U. weltweit verteilt an der Postproduktion des Tons eines kompletten Films.

Jedes Team arbeitet vorwiegend entkoppelt von allen anderen Teams an einem Teil des gesamten Szenendatenbestands. Es kann nicht davon ausgegangen werden, dass alle Autoren permanent mit einer zentralen Datenbasis verbunden sind. Die Synchronisierung konkurrierender Änderungen unterschiedlicher Teams an gleichen Szenendaten erfordert daher spezielle, entkoppelte Techniken.

Die besondere Herausforderung bei der Realisierung des Workspace-Szenarios liegt in der fehlenden permanenten Verbindung aller Teams mit einer zentralen Datenbasis. Diese Einschränkung verhindert die unmittelbare Propagierung von Änderungen, wodurch es nicht möglich ist, dass alle Teams immer auf dem aktuellen Stand des Projekts sind. Somit lässt sich das Kooperationsprinzip nicht vollständig realisieren. Um zumindest eine teilweise Umsetzung des Prinzips zu erzielen, haben wir uns für den Einsatz eines Versionierungssystems entschieden. Dieses bietet gegenüber traditionellen Transaktionsmodellen den Vorteil, dass es auf Serialisierbarkeit als Korrektheitskriterium verzichtet. Dieses Kriterium verhindert multi-direktionale Informationsflüsse und widerspricht somit gänzlich dem kooperativen Grundgedanken.

In dieser Arbeit stellen wir einen ersten Ansatz für die Realisierung eines geeigneten Versionierungssystems vor. Dieses arbeitet nicht, wie z.B. Subversion (*http://subversion.tigris.org/*), auf der textuellen Repräsentation der XML-Daten, sondern betrachtet die Daten als Baum. Im nächsten Abschnitt stellen wir dazu zunächst das Integrationskonzept vor und gehen dabei konkret auf die Ermittlung der Deltas aus den Transaktions-Recovery-Informationen einzelner Workgroups ein. In Abschnitt 3 werden anschließend vier Versionsverwaltungssysteme vorgestellt und hinsichtlich ihrer Einsatztauglichkeit im Workspace-Szenario bewertet. Im letzten Abschnitt geben wir eine kurze Zusammenfassung und einen Ausblick auf zukünftige Arbeiten.

2 Integrationskonzept

Ein mögliches Konzept zur Integration von Workgroups in Workspace-Umgebungen ist in Abbildung 1 dargestellt. Es beruht auf einer Client-Server-Architektur mit einem globalen Server und den lokalen Servern der einzelnen Workgroups als Clients. Der globale Server verwaltet dabei den gesamten Szenendatenbestand, wohingegen jeder Client lediglich Teile dieses Bestands besitzt. Wie bereits in der Einleitung erwähnt, besteht keine permanente Verbindung zwischen den Clients und dem globalen Server. Der typische Arbeitsablauf in dieser Workspace-Umgebung sieht wie folgt aus:

1. Jedes Entwicklerteam (Workgroup) lädt sich einen Teil des gesamten Szenendatenbestands vom globalen Server auf den lokalen Server. Danach wird die Verbindung zum globalen Server geschlossen.
2. Jedes Team arbeitet entkoppelt von allen anderen Teams am lokalen Szenendatenbestand. Diese Arbeit erfolgt kooperativ mit den in [GS08] beschriebenen Modellen und Methoden.
3. Hat sich das Team auf einen finalen Zustand des lokalen Szenendatenbestands geeinigt, wird eine Verbindung zum globalen Server aufgebaut und die finale Version hochgeladen.

Punkt 3 wirft dabei zwei entscheidende Fragen auf:

1. Wie werden die Änderungen eines Teams erkannt?
2. Wie lässt sich das Mergen der Änderungen verschiedener Teams gestalten? Dabei ist besonders die Behandlung von Konfliktfällen interessant.

In dieser Arbeit gehen wir lediglich auf die erste Frage genauer ein. Die Ermittlung von Änderungen (des Deltas) zwischen zwei Versionen eines XML-Dokumentes erfordert typischerweise die Anwendung aufwendiger Diff-Algorithmen, wie sie z.B. in [CRGMW96, CAM01] beschrieben sind. Einen besonders kritischen Fall stellt dabei das Erkennen von Move-Operationen dar [CAHB00]. In unserem System kann jedoch auf einen derartigen Diff-Algorithmus verzichtet werden. Dies ist möglich, da wir die Transaktions-Recovery-Informationen, die in den

einzelnen Workgroups in Form von Logs vorliegen, wiederverwenden. Diese beinhalten alle Änderungen in Form einer Historie von Änderungsoperationen. Derzeit werden in unserem System die Änderungsoperationen *Insert*, *Edit*, *Move* und *Delete* unterstützt. Voraussetzung für die Funktionsweise dieses Verfahrens ist, dass alle Datenobjekte (Knoten und Kanten) global eindeutig identifiziert werden können. Dies ist durch das in [GS08] beschriebene Datenmodell, in dem IDs zur eindeutigen Identifikation von Knoten und Kanten verwendet werden, sichergestellt. Ein Beispiel für ein derartiges Log ist in Abbildung 2 dargestellt. Für jeden Knoten

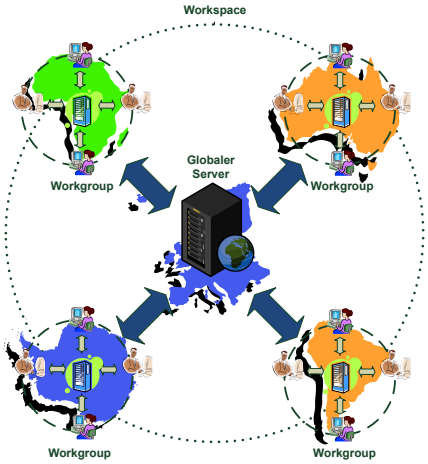

Abbildung 1: Workspace-Szenario

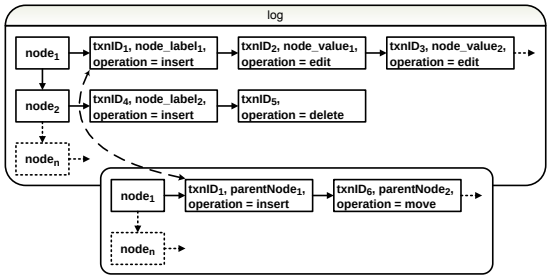

Abbildung 2: Beispiel für ein Log

werden in diesem Log folgende Informationen festgehalten:

- Wurde ein Knoten neu eingefügt, wird ein entsprechender Eintrag im Log erstellt.
- Für jede Editieroperation, die den Text eines Elements oder den Attributwert eines Attributes ändert, wird ein neuer Eintrag mit dem neuen Text oder Attributwert angelegt. Dabei wird angenommen, dass der alte Text oder Attributwert vollständig ersetzt wird.
- Das Löschen eines Knotens führt zu einem Eintrag.
- Die Kanten werden in Form von Vater-Kind-Beziehungen repräsentiert. So wird beim Einfügen eines Knotens ein zusätzlicher Eintrag geschrieben, der die Vater-ID beinhaltet. Jede Verschiebeoperation führt zu einem Eintrag im Log, der den neuen Vater des verschobenen Knotens enthält.

Wurden die Arbeiten innerhalb der Workgroup abgeschlossen, kann die finale Version der Szenendaten zum globalen Server übertragen werden. Dabei ist es ausreichend, den letzten Eintrag aus dem Log für jeden Knoten zu übertragen. Der Text oder Attributwert ergibt sich aus dem letzten Editiereintrag. Die Kanten ergeben sich aus den aktuellen Vater-Kind-Beziehungen. Lediglich bei neu eingefügten und danach verschobenen und editierten Knoten muss eine zusätzliche Operation $Insert(node\_id, parent\_id, node\_label, node\_value)$ generiert und übertragen werden. Diese Knoten existieren noch nicht auf dem globalen Server, und somit haben einfache Verschiebe- und Editieroperationen dort keinen Bezug. Wurde ein Knoten innerhalb der Arbeiten einer Workgroup eingefügt und wieder gelöscht, ist keine Übermittlung eines Eintrags für diesen Knoten erforderlich.

3 Versionsverwaltung

Die Versionsverwaltung von XML-Dokumenten ist ein altbekanntes Problem. Viele verschiedene Verfahren, wie z.B. [CTZ00, CTZZ01], wurden bereits entwickelt und kommen auch zum Einsatz. In dieser Arbeit untersuchen wir vier ausgewählte Verfahren, welche die Versionsverwaltung auf logischer Ebene, also auf Dokument-Ebene, betrachten. Diese Verfahren werden

hinsichtlich ihrer Einsatztauglichkeit im Workspace-Szenario bewertet und verglichen. Ein Versionsverwaltungssystem ist als tauglich für unseren speziellen Anwendungsfall anzusehen, wenn es mindestens die folgenden drei Kriterien erfüllt:

- Es sollten alle im letzten Abschnitt aufgeführten Änderungsoperationen unterstützt werden. Insbesondere sollte eine Verschiebeoperation nicht als Lösch- und Einfügeoperation betrachtet werden, sondern als „Umhängen“ eines Teilbaums unter einen anderen Vaterknoten.
- Der Aufwand zur Ermittlung einer bestimmten Version sollte möglichst gering sein. Dies garantiert kurze Antwortzeiten.
- Es sollte „Branched-Versioning“ unterstützt werden. Dies ist in Designprozessen besonders wichtig, da viele verschiedene kreative Ideen archiviert werden müssen.

Das Versionsverwaltungssystem des Web-Warehouses im Xyleme-Projekt [MACM01] setzt das Prinzip der „Completed Deltas“ ein. Diese ermöglichen es, in der Versionshistorie sowohl vorwärts als auch rückwärts zu gehen. Die Deltas werden als XML-Dokumente gespeichert. Dadurch ist es möglich, Änderungen direkt mit Hilfe von XPath/XQuery anzufragen. Das System unterstützt alle Änderungsoperationen, insbesondere wird eine Verschiebeoperation auch als „Umhängen“ eines Teilbaums betrachtet. Im System wird immer die aktuelle Version der Daten sowie alle „Completed Deltas“ gespeichert. Somit kann die aktuelle Version sofort angefragt werden. Ältere Zustände können mit Hilfe der Deltas berechnet werden. Dies führt jedoch zu hohen Berechnungskosten. Die Verwaltung von Branches ist möglich.

In [WL02] wird ein weiteres Versionsverwaltungssystem auf Basis von „Completed Deltas“ beschrieben. Das Grundprinzip ist ähnlich dem des eben beschriebenen Web-Warehouses im Xyleme-Projekt. Auch hier wird die aktuelle Version eines Dokuments und alle „Completed Deltas“ gespeichert. Allerdings werden zusätzlich Zwischenversionen des Dokuments materialisiert, um den Rekonstruktionsaufwand für eine Version zu verringern. Der Materialisierungsalgorithmus basiert auf dem Verhältnis der Anzahl notwendiger Operationen zur Rekonstruktion einer Version und dem Speicherbedarf für das Dokument.

Das „Reference-Based Version Model“(RBVM) [CTZ01] arbeitet mit Objektreferenzen. Teilbäume vorhergehender Versionen, die in der aktuellen Version unverändert sind, werden referenziert und nicht kopiert. Somit werden Redundanzen verringert, und es wird Speicherplatz gespart. Ein Vorteil ist, dass Verschiebeoperationen als „Umhängen“ betrachtet werden. Ein Nachteil dieses Systems ist, dass bei der Rekonstruktion einer Version eine Rückverfolgung von Objektreferenzen stattfinden muss, was zusätzlichen Arbeitsaufwand bedeutet und die Rekonstruktion einer Version verlangsamt. Die Verwaltung von Branches ist möglich.

Ein weiteres, vielversprechendes Modell zur Versionsverwaltung ist V-XML [RMG06]. Es ermöglicht die Repräsentation verschiedener Versionen eines XML-Dokumentes in einem einzigen Dokument. Dazu werden alle Elemente im Dokument mit Versionsstempeln versehen. Diese geben an, für welche Versionen das betroffene Element Gültigkeit besitzt. Vorteil dieses Modells ist, dass sehr leicht eine bestimmte Version eines Dokumentes angefragt werden kann. Dabei kann auf XQuery zurückgegriffen werden. Auch das „Branched-Versioning“ wird unterstützt. Ein Nachteil des Systems ist, dass Verschiebeoperationen als Löschen und Einfügen betrachtet werden.

In Tabelle 1 ist die Bewertung der vier Systeme noch einmal kurz zusammengefasst. Es kann festgehalten werden, dass lediglich das System der „Completed Deltas“ mit dem zusätzlichen Materialisierungsalgorithmus alle Anforderungen vollständig erfüllt. Dieses System ist daher für den Einsatz im Workspace-Szenario geeignet.

4 Zusammenfassung und Ausblick

In dieser Arbeit haben wir einen ersten Ansatz vorgestellt, wie bestehende Workgroups in ein Workspace-Szenario integriert werden können. Hauptbestandteil dieses Integrationskonzepts ist

System	Move	Versionsrekonstruktionskosten	„Branched-Versioning“
Web-Warehouse	unterstützt	hoch	unterstützt
Compl. Deltas + Mat.-Algo.	unterstützt	gering	unterstützt
RBVM	unterstützt	hoch	unterstützt
V-XML	nicht unterstützt	gering	unterstützt

Tabelle 1: Vergleich der Systeme

ein maßgeschneidertes Versionierungssystem. Eine Besonderheit dieses Systems ist, dass auf aufwendige Diff-Algorithmen verzichtet werden kann, da Deltas aus den Transaktions-Recovery-Informationen einzelner Workgroups gewonnen werden können. Für die serverseitige Verwaltung der Szenendaten und der Änderungen haben wir vier Systeme kurz vorgestellt und bewertet. Diese Bewertung hat ergeben, dass das System der „Completed Deltas“ mit dem dazugehörigen Materialisierungsalgorithmus alle Anforderungen erfüllt und daher für den Einsatz im Workspace-Szenario geeignet ist. In zukünftigen Arbeiten werden wir uns folgenden drei Problemen zuwenden: Zunächst werden wir uns mit dem Merge-Prozess beschäftigen. Hierbei geht es primär um die Definition von Konfliktfällen. Ein weiteres interessantes Gebiet ist die Anfrageverarbeitung auf versionierten XML-Dokumenten. Des Weiteren werden wir versuchen, den gleichzeitigen Betrieb von Workgroup und Workspace auf dem globalen Server zu realisieren. Das bedeutet, dass stark verteilte Entwicklerteams mit Hilfe von Versionierung und einzelne Autoren mit Hilfe kooperativer Transaktionen auf dem zentralen Szenendatenbestand des globalen Servers gleichzeitig arbeiten können.

Literatur

[CAHB00] Grégory Cobéna, Talel Abdessalem, Yassine Hinnach, and Domaine De Voluceau Rocquencourt Bp. A comparative study for xml change detection. Technical report, 2000.

[CAM01] Gregory Cobena, Serge Abiteboul, and Amelie Marian. Detecting changes in xml documents. In *ICDE*, pages 41–52, 2001.

[CRGMW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *SIGMOD '96*, pages 493–504, 1996.

[CTZ00] Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo. Version management of xml documents. In *WebDB (Informal Proceedings)*, pages 75–80, 2000.

[CTZ01] Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo. Efficient management of multiversion documents by object referencing. In *VLDB '01*, pages 291–300, 2001.

[CTZZ01] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, and Donghui Zhang. Storing and querying multiversion xml documents using durable node numbers. In *Proc. of WISE*, pages 270–279, 2001.

[GHS09] Francis Gropengießer, Katja Hose, and Kai-Uwe Sattler. Ein kooperativer XML-Editor für Workgroups. In *BTW*, 2009.

[GS08] Francis Gropengießer and Kai-Uwe Sattler. An Extended Cooperative Transaction Model for XML. In *PIKM'08 icw CIKM'08*, pages 41–48, 2008.

[MACM01] Amélie Marian, Serge Abiteboul, Gregory Cobena, and Laurent Mignet. Change-centric management of versions in an xml warehouse. In *VLDB '01*, pages 581–590, 2001.

[RMG06] Luis Arévalo Rosado, Antonio Polo Márquez, and Juan María Fernández González. Representing versions in xml documents using versionstamp. In *ER (Workshops)*, pages 257–267, 2006.

[WL02] Raymond K. Wong and Nicole Lam. Managing and querying multi-version xml data with update logging. In *DocEng '02*, pages 74–81, 2002.