Erweiterung bestehender Anwendungen um kollaborative Funktionen mit Hilfe des Collaborative Editing Framework for XML (CEFX)

Ansgar Gerlicher
FH Stuttgart, Hochschule der Medien
Studiengang Medieninformatik
The London Institute, London College of Printing
gerlicher@hdm-stuttgart.de

Zusammenfassung: Das interdisziplinäre Forschungsgebiet des Computer Supported Collaborative Work (CSCW) befasst sich mit Gruppenarbeit und Zusammenarbeit, und den die Gruppenarbeit unterstützenden Informations- und Kommunikationstechnologien. Ein Teilbereich des CSCW ist das sogenannte „Real-Time Collaborative Editing". Dabei wird untersucht, wie Systeme aufgebaut sein müssen, die es mehreren Personen gleichzeitig erlauben, in Echtzeit auf einem Dokument zu arbeiten ohne dabei Inkonsistenzen zu riskieren. Die zur Zeit existierenden CSCW Editoren aktueller Forschungsarbeiten sind spezialisiert auf ein oder wenige Datenformate. Diese Forschungsarbeit hingegen beschäftigt sich mit der Entwicklung eines Frameworks (CEFX), mit dem es ermöglicht werden soll, quasi beliebige XML Dokument-Typen in Echtzeit gemeinschaftlich und über das Internet, also von beliebigen Standorten aus, zu editieren. Damit soll ein universeller Ansatz für die Lösung der Probleme des „Real-Time Collaborative Editing" geschaffen werden.

1 Einleitung

Die Erstellung größerer Dokumente wie zum Beispiel Kataloge, Handbücher, Software-Quellcode etc. erfordert oft Teamarbeit. Arbeiteten früher in einer Organisation mehrere Personen an einem größeren Dokument, so geschah dies meist nach dem einfachen Prinzip des „Turn-Taking". Dabei wurde das

Dokument von einer Person nach getaner Arbeit an die nächste weitergereicht. Auf diese Art und Weise war es möglich das Dokument konsistent zu halten und, da es nur eine jeweils gültige Dokumentversion gab, musste keine Synchronisierung oder Versionierung von Dokumenten stattfinden. Diese einfache Methode war allerdings sehr Zeitaufwändig und umständlich. Trotzdem wird heute oft noch so verfahren, gerade bei der Erstellung kleinerer Dokumente im Team, bei denen die Installation entsprechender Software zur Unterstützung der Teamarbeit zu aufwändig erscheint.

Um die Teamarbeit zu beschleunigen und zu vereinfachen und dennoch die Konsistenz der Dokumente zu wahren, werden heutzutage besonders im Bereich der Softwareentwicklung häufig Programme zur Versionierung und Synchronisierung von Dokumenten (in diesem Fall meist Quellcode) eingesetzt. Beispiele dafür sind Microsoft Visual SourceSafe [MSVSS] oder Concurrent Versioning System [CVS]. Diese Programme können im Prinzip für die Versionierung und Synchronisierung jedes Dokumenttyps eingesetzt werden. Für die Synchronisierung der Dokumentversionen werden dabei optimistische bzw. pessimistische Sperrverfahren eingesetzt. Beim pessimistischen Sperren ist paralleles Arbeiten in Echtzeit auf ein und derselben Datenquelle (dem Dokument) nicht möglich, da hier nur ein Anwender zu einer gegeben Zeit die Sperre besitzt. Erst nach der Freigabe des Dokuments durch diesen kann ein anderer darauf zugreifen. Beim optimistischen Verfahren ist paralleles Arbeiten zwar möglich, aber nur auf lokalen Kopien des Dokuments, wobei bei deren Zusammenführung zu einer neuen Dokumentversion Konflikte auftreten können. Paralleles Arbeiten direkt auf der Datenquelle ist bei keinem der beiden Ansätze möglich.

Ein Trend ist das kollaborative Arbeiten an verteilten Dokumenten im World Wide Web. Hierzu gibt es seit einiger Zeit Systeme, die Dokumente im Internet nicht nur lesbar, sondern auch editierbar machen. Hierzu zählen Technologien wie WikiWikis [WIKI] oder WebDAV [WDAV]. Bei diesen Systemen kommen dieselben oder ähnliche Verfahren zum Einsatz, wie bei den oben genannten.

Damit ist das parallele Arbeiten mehrerer Personen in Echtzeit auf der selben Datenquelle ebenso wenig möglich, da die Granularität der eingesetzten Sperrverfahren zu grob ist. Es ist meist nur möglich komplette Dokumentinstanzen zu sperren. Die hierarchische Struktur eines Dokumentes wird dabei nicht ausgenutzt.

In der Forschung existieren kollaborative Editoren [SUN1] für spezielle proprietäre Datenformate, die paralleles Arbeiten in Echtzeit mehrerer Personen über das Internet an einem Dokument unterstützen [SUN2]. Dazu zählen Systeme wie REDUCE [REDUCE], CoWord [CWORD] oder SAMS

[SAMS]. Diese Systeme ermöglichen allerdings nur das Editieren eines bestimmten Dokumenttyps und verwenden meist ein proprietäres Datenformat. Eine Lösung, die kollaboratives Editieren verschiedenster Dokumenttypen in Echtzeit über das Internet unterstützt, existiert noch nicht.

Viele der neuen und bestehenden Anwendungen für die Erstellung und Verwaltung von Dokumenten setzen auf XML als standardisiertes Datenformat. Dazu zählen z.B. verschiedene Office Anwendungen, Editoren für Vectorgrafik und Multimedia (SVG [SVG], SMIL [SMIL]) und Editoren für virtuelle Welten (X3D [X3D]). Dieser Trend, XML als standardisiertes Datenformat einzusetzen bringt viele Vorteile und kann für verteiltes Arbeiten genutzt werden.

Diese Forschungsarbeit beschäftigt sich mit der Entwicklung eines generischen Frameworks, mit dessen Hilfe Standardanwendungen, welche XML als Datenformat verwenden, mit der Fähigkeit zum kollaborativen Editieren von Dokumenten in Echtzeit und über das Internet erweitert werden können. Das Collaborative Editing Framework for XML (CEFX) verwendet dabei die besonderen Eigenschaften des XML Datenmodels für die Synchronisierung und Versionierung der Dokumente. Folgende Hauptfunktionen sollen in dieser Arbeit umgesetzt werden:

- Standortunabhängiges kollaboratives und synchrones Arbeiten an Dokumenten in Echtzeit unter Verwendung des Internets als Datenkanal
- Unterstützung aller aktuellen und zukünftigen XML Dokumenttypen
- Einfache Integration in bestehende Editoren und Tools und deren Erweiterung mit kollaborativen Funktionen
- Verbesserung der Usability durch sogenannte „Awareness"-Mechanismen
- Semantische Erweiterung der Konsistenzerhaltung durch Kontextprüfung auf Basis von XML Schema

2 Nebenläufigkeitskontrollverfahren in kollaborativen Editoren

Nebenläufigkeitskontrolle ist ein häufig eingesetztes Verfahren um Inkonsistenzen in Datenbanksystemen vorzubeugen. Frei nach der Definition von Bernstein et al. [31]versteht man unter Nebenläufigkeitskontrolle die Koordination der Aktionen verschiedener Prozesse, die gleichzeitig auf gemeinsame

Daten zugreifen und sich damit potentiell behindern. Nebenläufigkeitskontrollmechanismen werden verwendet um die Konsistenz eines gemeinsam genutzen Dokumentes zu bewahren. Dabei gibt es prinzipiell zwei unterschiedliche Verfahrensarten: Verfahren zur Konfliktvorbeugung und Verfahren zur Konfliktauflösung.

Zu den Verfahren der Konfliktvorbeugung zählen unter anderem die, in der Datenbankwelt weit verbreiteten Sperrverfahren zum Beispiel das pessimistischen Sperrverfahren. Die einfachste Form der pessimistischen Sperrverfahren ist das exklusive Sperren (exclusive locking) von Datensätzen. Da dieses Verfahren bei vielen parallelen Transaktionen zu einer schlechten Systemleistung führt, wurde das shared lock (read lock) bzw. das SX-Verfahren (shared-exclusive lock) eingeführt. Um die auftretende Deadlock[1] Problematik zu Lösen, wurden weitere pessimistische Sperrverfahren entwickelt, wie zum Beispiel das Zwei-Phasen-Sperrverfahren. Solche pessimistischen Sperrverfahren werden sehr häufig in Datenbanksystemen und in Versionsverwaltungssystemen, wie zum Beispiel Microsoft Visual Sourcesafe [MSVSS] eingesetzt.

Ein Verfahren der Konfliktauflösung ist im Gegensatz dazu das optimistische Sperrverfahren. Dieses basiert auf der Annahme, dass kollidierende Transaktionen relativ selten auftreten und ein präventives Sperren von Datensätzen unnötig hohen Aufwand und Leistungseinbußen nach sich ziehen würde. Im Gegensatz zu den pessimistischen Sperrverfahren bleibt hierbei der Ablauf einer Transaktion bis zu ihrem Abschluss unberührt. Damit ist ein quasi paralleles Arbeiten an den selben Datensätzen möglich. Erst am Ende einer Transaktion wird festgestellt, ob ein Konflikt mit einer anderen Transaktion aufgetreten ist. Verschiedene Client/Server Systeme, wie zum Beispiel das Concurrent Versioning System [CVS], verwenden dieses oder ähnliche optimistische Verfahren zur Datensynchronisation. Beim CVS System werden Änderungen an den Datensätzen auf Client Seite zunächst ohne Sperrung (locking) durchgeführt. Sobald die Änderungen abgeschlossen (und „committed“) wurden, werden die Datensätze auf dem Server gesperrt. Danach wird in der Validierungsphase geprüft, ob Konflikte mit anderen Transaktionen vorliegen. Mithilfe von Zeitstempeln wird geprüft, ob ein Datensatz zwischenzeitlich geändert wurde. Falls die Validierung fehlschlägt, wird die Transaktion zurückgefahren und wiederholt, oder der Client wird über den Konflikt in Kenntnis gesetzt. Der Vorteil dieses Verfahrens ist die kurze Sperrzeit während der Validierungsphase. Ein Nachteil ist die relativ hohe Wahrscheinlichkeit von Validierungsfehlern, so dass dieses Verfahren hauptsächlich in Bereichen eingesetzt wird, in denen es nur wenige „gleichzeitige“ Benutzer gibt. Diese Einschränkung ist allerdings wiederum sehr von der Sperrgranularität abhängig. Mithilfe einer entsprechend feinen Sperrgranularität wäre die

[1] Zur Begriffserklärung Deadlock siehe Anhang.

Wahrscheinlichkeit für Konflikte bzw. Validierungsfehler entsprechend geringer. Die Sperrgranularität bestimmt die kleinste Einheit eines Sperrvorganges. Systeme wie [MSVSS] und [CVS] aber auch andere existierende Systeme zur Unterstützung der Arbeit im Team wie zum Beispiel WikiWikis [WIKI] oder WebDAV [WDAV] verwenden nur eine Sperrgranularität auf Dokumentebene. Das heißt es wird immer das komplette Dokument gesperrt, auch wenn nur ein kleiner Teil bearbeitet wird.

Pessimistische Sperrverfahren scheinen nicht für Echtzeiteditoren geeignet, da durch die Sperrung auf Dokumentebene ein quasi paralleles Arbeiten nicht möglich ist. Auch optimistische Sperrverfahren scheinen ungeeignet, da bei zu grober Sperrgranularität und zu vielen „gleichzeitigen" Benutzern die Häufigkeit von Validierungsfehlern und der Aufwand der Zusammenführung asynchroner Dokumentversionen zu groß wird bzw. nicht mehr automatisiert erfolgen kann.

Ein weiterer wichtiger Grund, warum Sperrverfahren für Echtzeit-Editoren eher ungeeignet sind, ist der relativ große Zeitaufwand, den die Sperrung an sich beansprucht und die damit enstehende Verzögerung auf Seite des Client. Diese Verzögerung würde in einem Editor stark den Arbeitsfluss stören und ist daher nicht akzeptabel.

Für kollaborative Systeme, welche quasi Echtzeitanforderungen besitzen, wurden daher andere Synchronisationsverfahren entwickelt, wie zum Beispiel das Verfahren der Operational Transformation.

3 Operational Transformation

Operational Transformation (OT) ist ein Verfahren zur Konsistenzerhaltung durch Konfliktauflösung. Der größte Vorteil der Operational Transformation gegenüber Sperrverfahren ist, dass eine Operation sofort, das heißt ohne Verzögerung, durchgeführt wird. Der Benutzer muss also, im Gegensatz zu Sperrverfahren, nicht warten, bis eine Sperre auf dem zu bearbeitenden Datensatz durchgeführt wurde. Bei der Operational Transformation werden Operationen direkt auf einer lokalen Kopie des Dokuments durchgeführt und danach an die anderen Clients verteilt und dort erneut durchgeführt. Eine Operation kann dabei, wenn sie von einem Client empfangen wurde, vor ihrer Durchführung zunächst transformiert werden. Die Transformation hat dabei das Ziel, die Intention der Benutzer beizubehalten und die auf allen Clients vorliegenden Dokumentkopien zu konvergieren [8]. Es gibt verschiedenste Vorschläge für Operational Transformation Algorithmen für Dokumente, die auf einem linearen Datenformat basieren. Einige davon sind unter anderem

dOPT [6], adOPTed [16], GOT [9], GOTO [7], SOCT2 [17]. Im Gegensatz zu linearen Datenformaten gibt es zur Zeit nur einen Algorithmus für Operational Transformation, welcher auf einem hierarchischen Datenformat basiert: Der treeOPT [8] Algorithmus.

Die meisten dieser Algorithmen arbeiten nach dem selben Prinzip der Konsistenzerhaltung. Hier soll ein kurzer Überblick darüber gegeben werden. Der Operational Transformation Algorithmus wurde entwickelt um die Probleme der Divergenz, Intentionsverletzung und der Kausalitätsverletzung[2] zu beheben. Ein kollaboratives Editiersystem wird dabei als konsistent bezeichnet, wenn es immer den Erhalt von Konvergenz, Intention und Kausalität gewährleistet.

Zur Bewahrung der Kausalität wird ein Zeitstempel-Verfahren auf Basis der „Vector-Logical Clock" [15, 17] verwendet. Dies erlaubt es sicherzustellen, dass eine Operation A, die nach der Kausalordnung[3] „vor" einer Operation B liegt (A < B), auch vor dieser ausgeführt wird, unabhängig davon in welcher Reihenfolge die Operationen jeweils eintreffen.

Um die Konvergenz und die Intention einer Operation zu erhalten wird eine totale Ordnungsrelation [7, 8, 30, 36] zwischen den Operationen definiert. Die totale Ordnungsrelation definiert, welche Operationen in welcher Reihenfolge auf der jeweiligen lokalen Kopie eines Dokuments – welche im jeweiligen Client („Site") in einem kollaborativen Editier-System vorliegt – ausgeführt wird. Zusätzlich werden alle ausgeführten Operationen der jeweiligen „Site" in einem History-Buffer gespeichert. Basierend auf der totalen Ordnungsrelation und dem History-Buffer wird ein Undo/Do/Redo Schema definiert. Dabei werden beim Eintreffen einer neuen Operation zunächst alle Operationen im History-Buffer, welche aufgrund der totalen Ordnungsrelation abhängig von der neuen Operation sind (also ihr nachstehen), rückgängig gemacht (Undo). Damit wird der ursprüngliche Dokumentzustand wieder hergestellt. Danach wird die neue Operation ausgeführt (Do) und danach wieder alle Operationen, die zuvor rückgängig gemacht wurden (Redo). Zusätzlich wird jede Operation vor ihrer Ausführung transformiert um auf die Änderungen zu reagieren, die durch die anderen Operationen hervorgerufen wurden. Um dies zu verdeutlichen zeigt die Abbildung 4 ein Szenario eines Editiervorganges ohne Transformational Operation. Dabei arbeiten zwei Benutzer an einem gemeinsamen Dokument, welches den Text „efect" enthält. Der Text kann durch die Operation Ins(p,c) modifiziert werden. Durch diese Operation wird ein Buchstabe c an Position p im Text eingefügt. Es wird angenommen, das die Position des ersten Buchstaben im Text 1 (nicht 0) ist. Die Benutzer generieren die Operationen O1 = Ins(2,f) und O2 = Ins(6,s).

[2] Zur Begriffserklärung Divergenz, Intentionsverletzung, Kausalitätsverletzung siehe Anhang.

[3] Zur Begriffserklärung Kausalordnung siehe Anhang

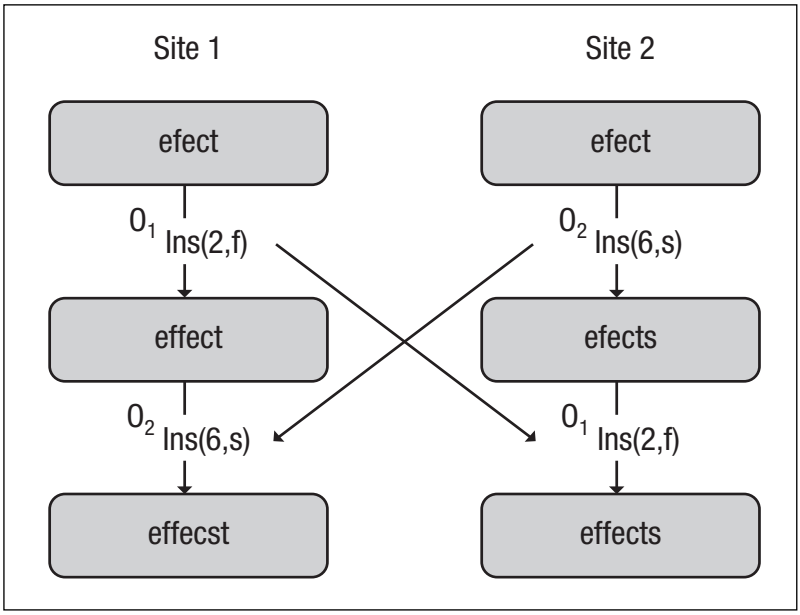

Abbildung 4: Fehlerhafte Integration von Operationen

Wird die Operation O1 an „Site" 2 empfangen und ausgeführt, so führt dies zu dem erwarteten Ergebnis „effects". Im Fall von „Site" 1 sieht dies anders aus. Dort wird nicht beachtet, dass die Operation O1 bereits zuvor ausgeführt wurde und damit die Textlänge verändert wurde. Das Resultat ist eine Divergenz der beiden lokalen Dokumente. Um ein korrektes Ergebnis zu erhalten, muss die Operation O2 unter Einbeziehung von O1 zuerst transformiert werden, bevor sie ausgeführt werden kann. Wird nun zum Beispiel die Operation O2 auf „Site" 1 in Ins(7,s) transformiert, so wird eine Konvergenz der Dokumente erzielt.

4 Synchronisation der XML Dokumente

Operational Transformation (OT) ist ein häufig eingesetztes Verfahren zur Synchronisation verteilter Dokumente in Echtzeit. Die meisten Algorithmen für Operational Transformation basieren allerdings auf einem linearen Datenformat. XML hingegen kann als hierarchisches Datenformat betrachtet werden. Bei der Operational Transformation kann ein hierarchisches Datenformat von Vorteil sein. Ein Problem der OT ist der oben erwähnte History-Buffer.

Dieser kann bei entsprechender Anzahl an Benutzern rapide wachsen. Damit kann sich die Ausführung von neuen Operationen entsprechend stark verzögern, da für jede neue Operation die Operationen im History-Buffer geprüft werden müssen. Der History-Buffer bei Operational Transformation Algorithmen für lineare Datenformate bezieht sich immer auf alle Operationen im gesamten Dokument. Dies hängt mit der Eigenschaft linearer Datenformate zusammen: Eine Operation, wie zum Beispiel das Einfügen eines Buchstabens, hat immer Auswirkungen auf alle nachfolgenden Dokumentteile.

Unterteilt man ein Dokument nun hierarchisch in verschiedene Einheiten, so wirkt sich eine Änderung am Dokument nicht automatisch auf alle nachfolgenden Dokumentteile aus, sondern nur auf die Bereiche, die sich auf derselben Hierarchieebene befinden. Ein Beispiel soll diesen Zusammenhang erläutern:

Ein Textdokument wird in fünf Hierarchieebenen gegliedert: Gesamtdokument, Paragraphen, Sätze, Wörter und Buchstaben. Wird nun ein Buchstabe in einem Wort eingefügt, so hat dies eine Positionsänderung der Buchstaben dieses Wortes zur Folge, die nach dem neu eingefügten Buchstaben stehen. Das Einfügen eines Buchstabens hat allerdings keinen Einfluss auf die Position der Nachfolgenden Wörter, Sätze oder Paragraphen, da es sich um ein hierarchisches Dokument handelt.

Dies hat für die Operational Transformation eine wichtige Bedeutung. Der History-Buffer muss nun nicht mehr für das Gesamtdokument gehalten werden, sondern für jede Hierarchieebene muss nun ein entsprechender Buffer angelegt werden. Dies scheint zwar Mehraufwand zu bedeuten, gleichzeitig bleiben aber die einzelnen History-Buffer viel kleiner und führen damit zu einem Vorteil in der Prüfung neuer Operationen und der Berechnung einer Operationstransformation. Die Prüfung einer neuen Operation muss gegen weniger Operationen im History-Buffer durchgeführt werden und auch die Berechnung der Transformation gestaltet sich einfacher.

Zur Zeit existiert nur ein Algorithmus für Operational Transformation, der auf einem hierarchischen Datenformat basiert [8]. Das dabei verwendete Datenformat ist proprietär und auf eine bestimmte Hierarchietiefe fixiert. Ziel dieses Forschungsprojektes ist es unter anderem, einen Algorithmus für Operational Transformation zu entwickeln, der eine variable und beliebige Hierarchietiefe erlaubt und als Datenformat den XML Standard unterstützt.

Operational Transformation ist sehr gut für die Synchronisation von strukturellen Operationen wie Einfügen und Löschen von Daten geeignet, besitzt aber Schwächen bei der Synchronisation von inhaltlichen Operationen, wie zum Beispiel das Ändern eines Attributs. Hier kann die Operational Transformation keine Konflikte wie die Intentionsverletzung verhindern. Für

diese Problematik muss ein Algorithmus für Operational Transformation durch zusätzliche Verfahren zur Konsistenzerhaltung ergänzt werden. Des Weiteren kann durch bestimmte so genannte „Awareness" Mechanismen die Aufmerksamkeit des Benutzers auf eventuell auftretende Probleme während des Editier-Prozesses gelenkt werden, um so vorab Inkonsistenzen und Intentionsverletzungen zu vermeiden. Das CEFX wird daher verschiedene „Awareness" Mechanismen zur Verfügung stellen.

XML bietet durch seine Eigenschaften, wie zum Beispiel eine in XML Schema vorliegende Grammatik weitere Möglichkeiten zur Unterstützung der Synchronisation. Ziel dieser Forschungsarbeit ist es unter anderem einen Algorithmus zu entwickeln, welcher diese Eigenschaften von XML Dokumenten nutzt um eine Verbessung der Konsistenzerhaltung durch Kontextprüfung auf Basis von XML Schema zu erreichen.

5 Systemarchitektur des CEFX

Viele kollaborative Echtzeit-Editoren verwenden die replizierte Architektur. Bei dieser Systemarchitektur gibt es im Gegensatz zu den meisten Client/Server Systemen, welche eine zentralisierte Architektur verwenden keine zentrale Datenquelle oder Server. Jeder Client besitzt dabei eine Kopie des Server Prozesses und der gemeinschaftlich genutzten Datenquellen. Der Vorteil einer replizierten Architektur besteht in den schnellen Antwortzeiten bei optimistischer Ausführung von Operationen. Wird eine lokale Operation generiert, so wird diese sofort ausgeführt und das Ergebnis wird damit sofort sichtbar. Im Gegensatz zur zentralisierten Architektur ist dabei kein Aufbau einer Datenverbindung zum Server oder gar eine sofortige Validierung der Operation (Stichwort: Sperrung) notwendig, was zu einem deutlichen Geschwindigkeitsvorteil führt. Die lokal ausgeführten Operationen werden danach an alle weiteren Clients übertragen und dort ausgeführt. Dabei kann es zu den schon besprochenen Inkonsistenzen kommen.

Die Systemarchitektur, auf der das CEFX basieren wird, ist eine Mischung aus der zentralisierten und der replizierten Systemarchitektur. Bei dieser so genannten Hybrid-Architektur gibt es einen zentralen Server, der immer eine aktuelle Dokumentversion besitzt. Die verschiedenen Clients besitzen jeweils lokale Kopien des gemeinsam genutzten Dokuments. Wie auch bei der replizierten Architektur, werden Operationen auf dem Dokument sofort lokal ausgeführt und danach an die anderen Clients übertragen. Dort werden die Operationen in dem entsprechenden History-Buffer gespeichert. Zusätzlich werden die Operationen auch an den Server übertragen. Da

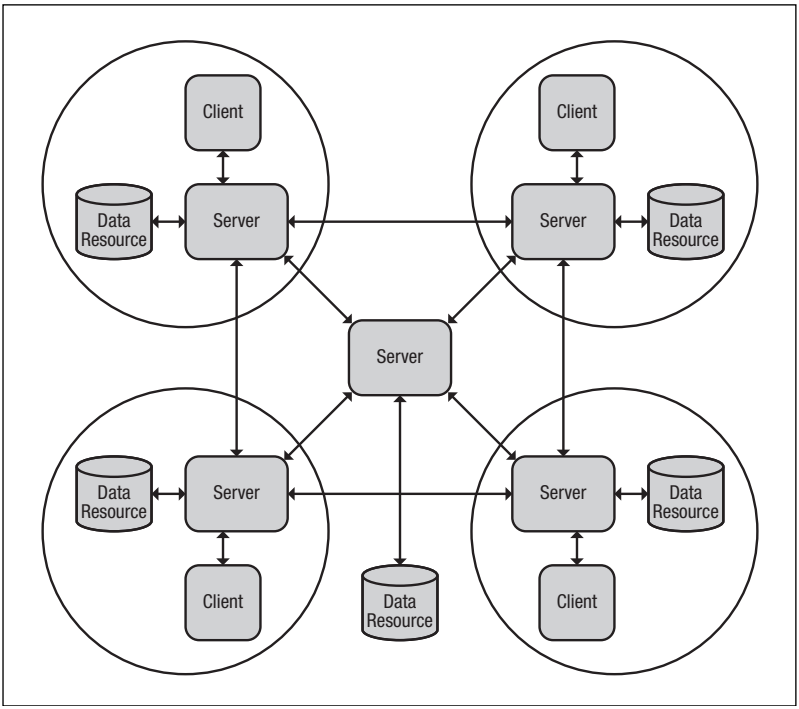

Abbildung 5: Hybride Systemarchitektur des CEFYX

das System auf XML als Datenformat basiert, müssen Operationen, die ein Client empfängt, nur bei Bedarf sofort ausgeführt werden: nämlich nur dann, wenn ein Benutzer den Bereich eines Dokumentes betrachtet, auf dem ein anderer gerade Operationen durchführt. Auf dem Server werden hingegen alle Operationen sofort ausgeführt. Somit liegt auf dem Server immer eine aktuelle Version des Dokuments vor. Dies ist neben den schnellen Antwortzeiten beim Editieren ein weiterer Vorteil der hybriden Systemarchitektur. Eine Kopie des aktuellen Dokuments kann dadurch zum Beispiel immer abgerufen werden, sobald ein neuer Benutzer am kollaborativen Editier-Prozess teilnehmen will. Auch kann das Dokument auf dem Server für die Synchronisation genutzt werden. Arbeiten mehrere Personen zum Beispiel an einem sehr großen Dokument, so ist die Wahrscheinlichkeit, dass diese sich eventuell gegenseitig stören relativ gering. Betrachtet nun ein Benutzer einen Teil des Dokuments das von einem anderen Benutzer intensiv bearbeitet wurde, so muss zuvor die lokale Kopie des Dokuments synchronisiert werden. Dies kann nun entweder durch Ausführen der Operationen im lokalen History-

Buffer geschehen, oder durch Abrufen der benötigten Teile der aktuellen Dokumentversion vom zentralen Server. Enthält der lokale History-Buffer zum Beispiel sehr viele Operationen, so ist es eventuell schneller, die aktuellen Teile vom Server nachzuladen, als die entsprechenden Operationen zu prüfen, zu transformieren und auszuführen.

Die beim CEFX verwendete Architektur macht erst durch die hierarchische Struktur des zu Grunde liegenden Datenformats (XML) Sinn. Bei Systemen, die auf einem linearen Datenformat basieren, würde diese Architektur eher hinderlich wirken. Dort hat sich die replizierte Systemarchitektur etabliert. Dieser neue Ansatz verspricht eine effiziente Möglichkeit zur Synchronisation verteilter Dokumente zu werden und macht damit ein Editieren in Echtzeit möglich.

6 Ausblick

Der Einsatz von XML in freien und kommerziellen Produkten aus verschiedensten Bereichen nimmt ständig zu. Eines von vielen Beispielen ist das OpenOffice Produkt von Sun Microsystems, welches als Speicherformat für Dokumente XML einsetzt. Auch Microsoft kann sich vor der XML-Welle nicht verschließen und bietet in der neuesten MS Office Version nun auch die Möglichkeit an, XML Dokumente zu importieren.

Das „Collaborative Editing Framework for XML“ (CEFX) soll eine einfache Möglichkeit schaffen beliebige Editoren, welche auf dem Datenformat XML basieren, mit kollaborativen Fähigkeiten auszustatten. Dazu wird eine Programmierschnittstelle angeboten, die eine einfache Einbindung in bestehende Editoren ermöglichen soll. Das so genannte Application Programming Interface (API) des CEFX soll dabei neben den normalen Funktionalitäten wie Einfügen, Löschen, Verschieben und Ändern von Elementen im vorliegenden XML Dokument auch „Awareness“ Mechanismen bereitstellen. Dabei sollen bestimmte Metainformationen zum Beispiel über den aktuellen Fokus eines Benutzers im Dokument über „Call-Back“ Funktionen abgerufen und im Editor entsprechend dargestellt werden können. Eine zusätzliche Validierungsfunktion der Benutzereingaben soll auf Basis von XML Schema eine quasi semantische Vorprüfung der durchgeführten Dokumentänderungen erlauben. Damit können inhaltliche Fehler im Voraus vermieden werden. Da das Framework eine generelle Unterstützung von XML als Datenformat vorsieht, soll praktische jede Form von XML Anwendung damit unterstützt werden können.

7 Anhang

Einige Begriffserläuterungen:

Divergenz

Die Abbildung 1 zeigt ein Szenario, bei dem drei Personen auf einem gemeinsamen Dokument in Echtzeit arbeiten. Dazu wird vor dem Editier-Vorgang vom Dokument jeweils eine lokale Kopie gemacht, auf der die Teilnehmer dann Änderungen durchführen. Die Abbildung 1 zeigt dazu den zeitlichen Ablauf der Operationen O1 bis O4, wie sie bei den verschiedenen Teilnehmern (Site 1-3) durchgeführt werden. Dabei werden die Operationen in jeweils unterschiedlichen Reihenfolgen ausgeführt. Die Reihenfolge der Operationen „Site 1" ist O1,O2,O4,O3, „Site 2" O2,O1,O3,O4 und „Site 3" O2,O4,O3,O1.

Sind die Operationen O1,O2,O3,O4 nicht kommutativ, so ist das endgültige Ergebnis der Operationen eventuell bei jedem Teilnehmer unterschiedlich. Eine Divergenz zwischen den verschiedenen lokalen Dokumenten liegt vor. Betrachten wir dazu die Baumstruktur eines einfachen XML Dokuments. Zu Beginn sind die beiden Bäume identisch.

Nun werden wie im obigen Szenario gezeigt verschiedene Operationen auf den Bäumen durchgeführt. Operation O1 erzeugt auf „Site" 1 einen Kindknoten A unterhalb Knoten 2 an erster Position. Operation 2 erzeugt auf „Site" 2 einen neuen Kindknoten B auch unterhalb Knoten 2 an Position 1. O1 und O2 werden quasi zeitgleich auf dem lokalen Dokument ausgeführt. Danach werden sie jeweils an den anderen Teilnehmer übertragen. Durch die Ver-zögerung in der Ausführung der Operationen, ist die Reihenfolge in welcher diese ausgeführt werden auf den jeweiligen lokalen Dokumenten unterschiedlich. Auf dem Dokument „Site 1" wird O1 vor O2 ausgeführt. Auf dem Dokument „Site 2" geschieht dies in umgekehrter Reihenfolge. Die beiden Dokumente sind divergent.

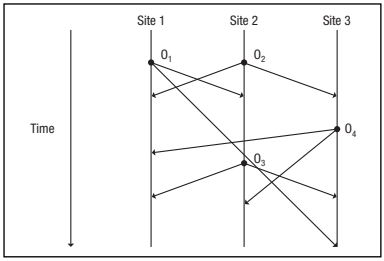

Abbildung 1:

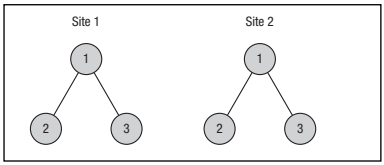

Abbildung 2.1: Beispiel Divergenz: zwei identische Bäume

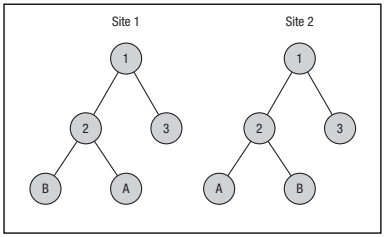

Abbildung 2.1: Beispiel Divergenz: zwei divergente Bäume

Intentionsverletzung

Eine Intentionsverletzung tritt dann auf, wenn zwei Operationen parallel zum Beispiel ein Attribut eines Objektes auf zwei unterschiedliche Werte abändern. Nehmen wir zum Beispiel einen kollaborativen Grafikeditor in dem die Operationen O1 und O2 parallel von zwei Benutzern ausgeführt werden. Operation O1 ändert dabei die Farbe eines Objektes innerhalb einer Grafik auf grün. Zur selben Zeit führt der zweite Benutzer Operation O2 auf dem selben Objekt in der selben Grafik aus, wobei O2 die Farbe des Objekts auf rot ändert. Nun werden die Operationen jeweils zum anderen Benutzer übertragen. Das Ergebnis ist, dass auf beiden Seiten das Objekt den falschen Farbwert besitzt. Es ist in diesem Fall nicht möglich den Konflikt ohne weiteres automatisch zu lösen, solange das betroffene Attribut nicht gleichzeitig mehrere Werte zulässt. In diesem Fall ist es nicht möglich einen konsistenten Eindruck dessen, was die Intention des jeweiligen Benutzers war, herzustellen. Eine Intentionsverletzung liegt vor.

Kausalitätsverletzung

Wie im Szenario in Abbildung 1 gezeigt, wird die Operation O3 erst nach der Ankunft von O1 bei „Site 2“ generiert. Wurde O3 bei „Site 2“ aufgrund von O1 generiert dann ist O3 von O1 abhängig. Zwischen O1 und O3 besteht eine so genannte kausale Abhängigkeit der Ereignisse. Auf „Site 3“ wird nun im obigen Szenario die Operation O3 vor der Operation O1 ausgeführt. Ist O3 aber kausal von O1 abhängig, so entsteht bei „Site 3“ eine Kausalitätsverletzung, da sich O3 auf einen zu diesem Zeitpunkt noch nicht vorhandenen Kontext bezieht. Eine Kausalitätsverletzung kann zu einem Zustand führen, in dem eine Operation nicht durchgeführt werden kann, da die Operation, von der sie abhängt noch nicht ausgeführt wurde.

Die Abbildung 3 zeigt, wie auf „Site 2“ die Operationen O1 und O3 in der richtigen Reihenfolge ausgeführt werden. Operation O1 fügt dabei einen Knoten A unterhalb Knoten 1 ein. Operation 3 fügt danach einen Knoten B unterhalb Knoten A ein. Dies führt bei „Site 3“ zu einem Zustand in dem Operation O3 nicht durchgeführt werden kann, da O1 noch nicht durchgeführt wurde.

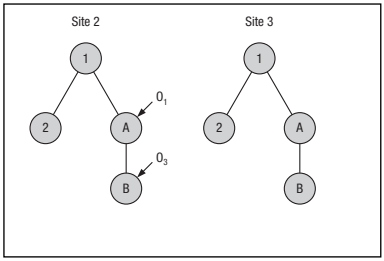

Abbildung 3: Kausalitätsverletzung

Kausalordnung

Die Kausalordnung ist eine Halbordnung, die über die Relation der kausalen Abhängigkeit über einer Menge von Ereignissen definiert wird: Ein Ereignis A ist eine Ursache von Ereignis B ($A < B$ bzw. A liegt vor B) oder umgekehrt ($A > B$), oder die Ereignisse beeinflussen sich gegenseitig nicht ($A \parallel B$), das heißt, sie sind kausal unabhängig oder nebenläufig. Die Kausalität wird zu dem von den meisten Theoretikern als transitiv betrachtet: Wenn Ereignis A eine Ursache von B ist, und B ist eine Ursache von C, dann ist A auch eine Ursache von C (wenn $A < B$ und $B < C$ ist, dann ist auch $A < C$). Andere wenden dagegen ein, dass zumindest unsere gewöhnliche Urteilspraxis bezüglich der Kausalität nicht transitiv ist, da wir bei der Suche nach der Ursache eines Ereignisses stets nach dem unmittelbar verursachenden Ereignis forschen.

Deadlock

Ein Deadlock (auch Verklemmung genannt) ist in der Informatik ein Zustand von Prozessen, bei dem mindestens zwei Prozesse untereinander auf Betriebsmittel warten, die dem jeweils anderen Prozess zugeteilt sind. Beispielsweise kann einem Prozess p1 der Bildschirm zugeteilt worden sein. Gleichzeitig benötigt p1 allerdings den Drucker. Auf der Gegenseite ist der Drucker dem Prozess p2 zugeteilt, der wiederum den Bildschirm fordert. Ein Beispiel für eine Verklemmung aus dem realen Leben ist eine Straßenkreuzung, an der von allen vier Seiten ein Auto gekommen ist und nun (die Regel rechts vor links vorausgesetzt) darauf wartet, dass das Auto rechts von ihm fährt. Nach Coffman et al. (1971) müssen vier notwendige Kriterien für einen Deadlock zutreffen:

1. Die Betriebsmittel werden ausschließlich durch die Prozesse freigegeben (No Preemption).
2. Die Prozesse fordern Betriebsmittel an, besitzen aber zugleich den Zugriff auf andere (Hold and Wait).
3. Der Zugriff auf die Betriebsmittel ist exklusiv (Mutual Exclusion).
4. Nicht weniger als zwei Prozesse warten in einem geschlossenen System (Circular Wait).

Deadlocks können bei Systemen eintreten, die fähig sind mehrere Prozesse parallel ablaufen zu lassen (Multitasksysteme) und bei denen die Reihenfolge der Betriebsmittelvergabe nicht festgelegt ist.

8 Referenzen

[CVS] *Concurrent Versions System. A open standard for version control.* http://ccvs.cvshome.org

[CWORD] C. Sun, Y. Yang, Y. Zhang, and D. Chen: *A consistency model and supporting schemes in real-time cooperative editing systems,* Proc. of the 19th Australian Computer Science Conference, Melbourne, S. 582-591, Jan. 1996.

[MSVSS] Microsoft Corporation. *Visual SourceSafe* http://msdn.microsoft.com/vstudio/previous/ssafe/

[REDUCE] D. Chen. *REDUCE – REal-time Distributed Unconstrained Collaborative Editing System.* Research Project at the Griffith University, Brisbane. 2001.

[SUN1] C. Sun, Y. Yang, Y. Zhang, and D. Chen: *A consistency model and supporting schemes in real-time cooperative editing systems,* Proc. of the 19th Australian Computer Science Conference, Melbourne, S. 582-591, Jan. 1996

[SUN2] C. Sun and D. Chen. *Consistency maintenance in real-time collaborative graphics editing systems.* ACM Transactions on Computer-Human Interaction. 2002.

[SAMS] H. Skaf, P. Molli. *SAMS – Synchronous Asynchronous Multisynchronous Editor.* Forschungsarbeit an der Université Henri Poincaré, Nancy1.

[SMIL] *SMIL – Synchronized Multimedia Integration Language.* www.w3.org/AudioVideo/

[SVG] *SVG – Scalabale Vector Graphics 1.0 Specification,* 2001: www.w3.org/TR/SVG/

[WDAV] *WebDAV. Web Distributed Authoring and Versioning.* Arbeitsgruppe der IETF. www.ics.uci.edu/~ejw/authoring/

[WIKI] Im World Wide Web verfügbare Seitensammlungen, die von den Benutzern nicht nur gelesen, sondern auch online geändert werden können. http://de.wikipedia.org/wiki/Wiki

[X3D] *X3D – EXtensible 3D.* Draft specification committed to ISO/IEC JTC1/SC24 for registration, December 2002: www.web3d.org/x3d.html

9 Literaturverzeichnis

Conference Proceedings:

[1] C. Sun and D. Chen. *Consistency maintenance in real-time collaborative graphics editing systems.* ACM Transactions on Computer-Human Interaction. 2002.

[2] D. Chen. *REDUCE – Real-time Distributed Unconstrained Collaborative Editing System.* Research Project at the Griffith University, Brisbane. 2001.

[3] A. H. Davis, C. Sun, and J. Lu. *Collaborative Editing of XML Documents. An Operational Transformation Approach.* International Conference on Supporting Group Work. ACM Proceedings. 2001.

[4] Peter Naur. *Revised Report on the Algorithmic Language ALGOL 60.* Communications of the ACM, Vol. 3 No. 5, S. 299-314. May 1960.

[5] R. Galli and Y. Luo. MU3D: *A Causal Consistency Protocol for a Collaborative VRML Editor.* ACM Proceedings. Web3D-VRML: Fifth symposium on Virtual Reality Modeling Language. 2000.

[6] C.A. Ellis and S.J. Gibbs. *Concurrency Control in Groupware Systems.* ACM Proceedings. SIGMOD International Conference on Management of Data. 1989.

[7] C. Sun and C. Ellis. *Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements.* ACM Proceedings. ACM conference on Computer supported cooperative work, Seattle, Washington. 1998.

[8] C. Ignat and M. Norrie. *Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems.* ACM. Fourth International Workshop on Collaborative Editing Systems, New Orleans, Louisiana. 2002.

[9] C. Sun, X. Jia, and et al. *Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems.* ACM. Transactions on Computer-Human Interaction. 1998.

[10] P. A. Franaszek, J. T. Robinson, and A. Thomasian. *Concurrecy Control for High Contention Environments.* ACM. Transactions on Database Systems. 1992.

[11] D. J. Rosenkrantz, R. Stearns, and P. Lewis. *System Level Concurrency Control for Distributed Database Systems.* ACM. Transactions on Database Systems. 1978.
[12] H. T. Kung and J. T. Robinson. *On Optimistic Methods for Concurrency Control.* ACM. Transactions on Database Systems. 1981.
[13] D. Gawlick. *Processing Hot Spots in High Performance Systems.* IEEE Springer. CompCon Conference. 1985.
[14] C. Ignat and M. Norrie. *Customizable Collaborative Editor Relying on treeOPT Algorithm.* Kluwer Academic Publishers. Eighth European Conference on Computer Supported Cooperative Work. 2003.
[15] C. J. Fidge. *Timestamps in message-passing systems that preserve the partial ordering.* University of Queensland, Australia. 11th Australian Computer Science Conference. 1988.
[16] M. Ressel, D. Nitsche-Ruhland, and et al. *An integrating, transformation-oriented approach to concurrency control and undo in group editors.* ACM. Conference on Computer Supported Cooperative Work. Nov. 1996.
[17] M. Suleiman, M. Cart, and et al. *Serialization of Concurrent Operations in a Distributed Collaborative Environment.* ACM. International Conference on Supporting Group Work. 1997.
[18] P. Dourish. *Extending Awareness Beyond Synchronous Collaboration.* Position paper. CHI 97 Workshop on Awareness in Collaborative Systems. 1997. S. 31.
[19] C. Gutwin, M. Roseman, and S. Greenberg. *A Usability Study of Awareness Widgets in a shared Workspace Groupware System.* ACM. Conference on Computer Human Interaction '96 Conference Companion. 1996.
[20] R. M. Baecker, D. Nastos, I. R. Posner, and K. L. Mawby. *The User-Centred Iterative Design of Collaborative Writing Software.* Conference on Computer Human Interaction '93. 1993.
[21] Abdessamad Imine, Pascal Molli, Gerald Oster, and Michael Rusinowitch. *Proving Correctness of Transformation Functions in Real-Time Groupware.* Kluwer Academic Publishers. ECSCW 2003: Eighth European Conference on Computer Supported Cooperative Work. Sept. 2003.

Books

[22] Brian W. Kernigham and Dennis M. Ritchie. *The C Programming Language,* Second Edition. Prentice Hall Inc. 1988.
[23] Th. Michel. *XML Kompakt. Eine praktische Einführung.* Page 36. Carls Hanser Munich. 1999.
[24] F. Arciniegas. *XML Developer's Guide.* Page 49. Franzis. Poing. 2001.
[25] F. Van der Vlist. *XML Schema.* O'Reilly. 2002.
[26] P. Walmsley. *Definitive XML Schema.* Prentice Hall PTR London. 2002.
[27] R. Wyke and A. Watt. *XML Schema Essentials.* Wiley New York. 2002.
[28] C.J. Date. *An Introduction To Database Systems.* Addison-Wesley. 2000.
[29] P. Bernstein, N. Goodman, and V. Hadzilacos. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley. 1987.
[30] L. Lamport. *Time, clocks and the ordering of events in a distributed system.* Computer Associates. 1978.
[31] P. Bernstein, A. V. Hadzilacos, and et al. *Concurrency control and recovery in database systems.* Addison-Wesley. 1987.
[32] T. Haerder. *Datenbanksysteme: Konzepte und Techniken der Implementierung.* 2. Edition. Springer. 2001.
[33] P. A. Bernstein. *Newcomer: Transaction Processing.* Morgan Kaufmann. 1997.

Articles

[34] P. Peinl. *Synchronisation in zentralisierten Datenbanksystemen.* Informatik-Fachberichte. 161. Springer. 1987.
[35] P. Butterworth, A. Otis, and J. Stein. *The GemStone Object Database Management System.* Communications of the ACM. S. 64-77. ACM. 1991.
[36] M. Raynal, M. Singhai, and. *Logical Time: Capturing Causality in Distributed Systems.* IEEE Computer. S. 49-56. IEEE. 1996.

Theses

[37] D. Chen. *Consistency Maintenance in Collaborative Graphics Editing Systems.* Dissertation at the Griffith University Brisbane. 2001.

[38] J. C. Lauwers. *Collaboration transparency in desktop teleconferencing environments.* Ph.D Thesis. Stanford, CA. 1990.

[39] Csaszar Lorant Zeno. *Real-Time Collaborative Graphical Editor.* Swiss Federal Institute of Technology Zurich. Institute of Information Systems, Global Information Systems Group. June 2003.

RFCs

[40] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945. May 1996.

Links

[41] Microsoft Corporation. 2000. www.microsoft.com/windows/netmeeting/

[42] *The History of HTML.* www.w3.org/MarkUp/historicalW3C

[43] *Concurrent Versions System. A open standard for version control.* http://ccvs.cvshome.org