Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Diplomarbeit

ShaDoW

A Collaborative Shared Document Writer

Martin Meier

Keno Albrecht
Prof. Dr. Roger Wattenhofer

Distributed Computing Group
Institute for Pervasive Computing
ETH Zürich

24. Apr. - 23. Aug. 2003

Zusammenfassung

Der vorliegende Bericht beschreibt *ShaDoW*, ein kollaboratives Textbearbeitungssystem. ShaDoW erlaubt das gleichzeitige Editieren von Textdokumenten durch mehrere Benutzer über das Internet. Die Dokumente werden in partitionierten Objekten gespeichert. Ein partitioniertes Objekt ist ein Aggregat von einfachen Objekten. Die Verwendung von adaptiver Partitionierung in Kombination mit optimistischem Locking erlaubt das uneingeschränkte parallele Editieren der Dokumente. Somit werden keine aufwändigen Synchronisations- und Konsistenzprotokolle benötigt. Die Zusammenarbeit wird vom System durch die Anzeige aller verbundener Benutzer und die farbliche Codierung der Eingaben unterstützt.

ShaDoW baut auf dem Client/Peer-System *Clippee* auf und benötigt somit keine zentrale Instanz (Server). Das System erlaubt maximale Flexibilität beim Verwalten von Clustern. Ein Cluster existiert, solange mindestens ein beliebiger Peer existiert, der Mitglied des Clusters ist. Es kann sogar der Peer, welcher den Cluster gestartet hat, diesen verlassen, ohne den verbleibenden Cluster zu stören. In einem Cluster können beliebig viele Dokumente, die solange wie der Cluster existieren, in einer dateisystemartigen Struktur verwaltet werden.

Abstract

This paper introduces *ShaDoW*, a collaborative text editing system. ShaDoW uses partitioned objects to store documents globally. Partitioned objects combined with an optimistic use of locks allow the users to edit documents completely unconstrained. Adaptive partitioning of the objects make complex consistency and synchronization protocols needless. ShaDoW is built on top of the client/peer system *Clippee* and does not need a central instance (server). Thus clusters are managed very flexibly. A cluster exists as long as at least one peer exists that has been part of the cluster. Even the peer that has started the cluster can leave without disturbing the remaining peers. A cluster stores several documents in a simple file system. The stored documents live as long as the cluster.

Vorwort

Ich habe von Oktober 1998 bis August 2003 das Informatikstudium an der Eidgenössischen Technischen Hochschule Zürich absolviert. Mit vorliegendem Bericht schliesse ich meine Diplomarbeit bei der Gruppe für "Distributed Computing" von Prof. Dr. Roger Wattenhofer ab. Mit dieser Arbeit ist mein Studium erfolgreich beendet, und ich werde gut ausgebildet ins Praxisleben entlassen.

Danksagung

An dieser Stelle möchte ich allen danken, die mir mein Studium ermöglicht haben. Zuerst meinen Eltern, die mich über 20 Jahre meiner Erziehung und Ausbildung begleitet und unterstützt haben. Prof. Dr. Roger Wattenhofer danke ich für die Möglichkeit, in seiner Gruppe eine spannende und lehrreiche Diplomarbeit zu absolviert. Während der Diplomarbeit bin ich hauptsächlich von meinem Betreuer Keno Albrecht umfänglich unterstützt worden. Er hat mich mit vielen nützlichen Tips und Anregungen versorgt, die ich in meine Arbeit einfliessen lassen konnte.

Ein spezieller Dank gilt meinen Beta-Testern Keno Albrecht, Pieric Ferrari, Nicole Stucki und Pascal Stucki. Fürs Korrekturlesen konnte ich Keno Albrecht, Peter Meier, Patrick Neukomm und Pascal Stucki gewinnen.

Last but not least danke ich meiner Freundin Nicole Stucki für die entgegengebrachte Geduld während meiner Arbeit und die moralische Unterstützung.

Inhaltsverzeichnis

Zusammenfassung **i**

Vorwort **iii**

1 Einführung **1**
1.1 Aufgabenstellung 2
1.2 Aufbau dieses Dokumentes 2

2 Verwandte Arbeiten **3**
2.1 Partitionierte Objekte 3
2.2 Kollaborative Systeme 3

3 Eigenschaften partitionierter Objekte **7**
3.1 Atomare Objekte 7
3.2 Teilobjekte 8
3.3 Partitionierte Objekte 8
3.4 Verwaltung der Teilobjekte 8
3.5 Partitionierte Objekte kombiniert mit unvollständiger Replikation 9

4 Modellierung partitionierter Objekte **11**
4.1 Anforderungen an partitionierte Objekte 12
4.2 Datenstruktur 12
4.2.1 Verkettete Liste 12
4.3 Operationen auf der Datenstruktur 13
4.3.1 Insert 13
4.3.2 Remove 13
4.3.3 Split 13
4.3.4 Merge 16
4.3.5 Import 17
4.3.6 Export 18
4.4 Skip-Liste 18
4.5 Verwaltung des partitionierten Objektes 18
4.6 Verwaltung für ein kollaboratives Textbearbeitungssystem . . 18

5 Architektur **21**
5.1 Das Modell . . . 21
5.1.1 Update des Modells . . . 22
5.2 Die Ansicht . . . 23
5.3 Der Controller . . . 23
5.4 Verwaltung der verketteten Liste . . . 27
5.4.1 Übersetzen von Offsets . . . 27
5.4.2 Verwaltung des schreibbaren Knotens . . . 29
5.5 Collaboration Awareness . . . 29
5.6 Dokumentenordner . . . 30

6 Fazit **31**
6.1 Beurteilung des Modells und des Systems . . . 31
6.2 Rückblick auf die Arbeit . . . 31

7 Ausblick **33**
7.1 Uneingeschränktes Editieren . . . 33
7.2 Zugriffsrechte auf dem Dokument . . . 34
7.3 Persistente Datenhaltung . . . 34
7.4 Hilfsmittel für die Zusammenarbeit . . . 34
7.5 Ausbau zu einer Client-Anwendung . . . 35
7.6 Performanz . . . 35
7.7 Editierfunktionen (Skiplist) . . . 36
7.8 Automatisches Verbinden von Peers . . . 36

Literaturverzeichnis **38**

A Die Benutzerschnittstelle **39**
A.1 Startbildschirm . . . 39
A.2 Arbeitsansicht . . . 40
A.3 Erstellen eines neuen Dokumentes . . . 43
A.4 Exportieren eines Dokumentes . . . 43
A.5 Keyboard Shortcuts . . . 45

B Konfiguration **47**

C Aufruf und Debugging **55**
C.1 Starten der Anwendung . . . 55
C.2 Konsoleneingaben . . . 55

Kapitel 1

Einführung

Viele Dokumente werden heutzutage in Zusammenarbeit mehrerer Personen erstellt. Häufig sind die beteiligten Personen geographisch oder zeitlich voneinander getrennt. Da zur Erstellung der Dokumente oft der Computer zum Einsatz kommt, liegt die Idee nahe, das gemeinschaftliche Arbeiten durch den Computer zu unterstützen. Durch die zunehmende Globalisierung und Interdisziplinarität von Arbeiten erregt das Thema des computerunterstützten gemeinschaftlichen Schreibens gesteigerte Aufmerksamkeit. Deshalb stellt die Unterstützung gemeinschaftlicher Dokumentbearbeitung einen wesentlichen Bestandteil der Computer Supported Collaborative Work Forschung (*CSCW*) [15] dar.

In der Entwicklung von verteilten Systemen ersetzt in vielen Bereichen das Peer-to-Peer-Paradigma das althergebrachte Client/Server-Paradigma. Weitere Entwicklungen zeigen, dass es durchaus Sinn macht, die beiden Paradigmen zu einem neuen zu kombinieren, genannt *Client/Peer*. Die neueren Paradigmen bringen auch CSCW-Systemen Vorteile, vor allem punkto Unabhängigkeit und Skalierbarkeit.

Die vorliegende Arbeit baut auf dem bestehenden Client/Peer-System *Clippee* [1] auf, das in Abbildung 1.1 dargestellt ist; Clients sind mit "C" und Peers mit "P" gekennzeichnet. Clippee unterstützt Operationen zum

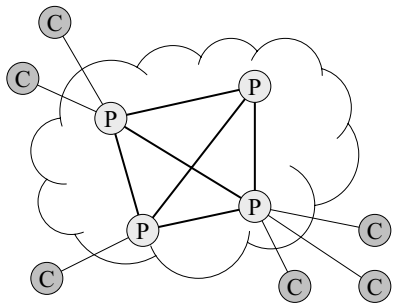

Abbildung 1.1: Illustration von Clippee

Lesen, Schreiben und Erzeugen von *kleinen* Objekten. Zum Zeitpunkt dieser Arbeit unterstützt Clippee ausschliesslich vollständige Replikation, so dass jedes Objekt auf jedem Peer repliziert wird.

Die vorliegende Arbeit beschreibt die Architektur und die Implementation von *ShaDoW*, eines kollaborativen verteilten Texteditors, der Benutzern das gleichzeitige, verteilte Bearbeiten eines gemeinsamen Textdokumentes erlaubt. Einen wesentlichen Bestandteil dieser Arbeit stellt die Integration von *partitionierten Objekten* in Clippee dar. Partitionierte Objekte bilden die Grundlage, um verteilte Dokumente zu realisieren, siehe Abschnitt 3.3.

1.1 Aufgabenstellung

Aufbauend auf dem Client/Peer-System Clippee sollte ein verteiltes Objekt realisiert werden. Mehrere Benutzer sollen das Objekt gleichzeitig lesen und an beliebigen Stellen verändern können. Gefordert war ein theoretisches Modell sowie dessen praktische Umsetzung in einer Demoapplikation, zum Beispiel ein verteiltes Textdokument oder eine verteilte Highscore-Liste für ein Internetspiel.

1.2 Aufbau dieses Dokumentes

Der restliche Teil des Berichtes ist folgendermassen strukturiert. In Kapitel 2 werden verschiedene bereits existierende Systeme mit ähnlichem Zweck diskutiert sowie Strukturen zur Speicherung von partitionierten Objekten vorgestellt. Die verwendete Struktur wird in Kapitel 3 charakterisiert und in Kapitel 4 modelliert. Kapitel 5 erklärt die Implementation des entwickelten Systems, welches in Kapitel 6 rückblickend beurteilt wird. Ebenfalls in Kapitel 6 wird ein Rückblick auf die gesamte Diplomarbeit gegeben. Kapitel 7 beschreibt Ideen, die während der Arbeit entstanden, aus zeitlichen Gründen aber nicht verwirklicht werden konnten.

Kapitel 2

Verwandte Arbeiten

2.1 Partitionierte Objekte

Im Folgenden werden zwei Ansätze diskutiert, wie partitionierte Objekte realisiert werden können, vergleiche Abschnitt 3.3. Partitionierte Objekte werden benötigt, damit Dokumente gespeichert werden können und eine maximale Parallelität bei der Bearbeitung erreicht wird, vergleiche Kapitel 3 und 4.

Fragmented Objects (FO) [11] stellen einen Ansatz dar, wie sich partitionierte Objekte realisieren lassen. Sie erweitern das Objektkonzept hinsichtlich einer verteilten Umgebung. Die abstrakte Sicht eines FO ist ein einzelnes, gemeinsames Objekt, dessen Verteilung dem Anwender verborgen bleibt. In der konkreten Sicht, kontrolliert der Entwickler die Verteilung und Partitionierung der Daten und Methoden. [10] zeigt, wie ein Naming Service als FO strukturiert werden kann.

Einen alternativen Ansatz zu fragmentierten Objekten bieten *Distributed Shared Objects* (DSO), welche im Zusammenhang mit der Globe Architektur [14] eingeführt werden. DSO erlauben, den Zustand des Objektes mittels *lokaler Objekte* physikalisch zu verteilen. Ein lokales Objekt existiert in genau einem Adressraum und kommuniziert mit anderen lokalen Objekten, um ein verteiltes Objekt zu bilden. Zustand und Operationen sind vollständig vom Objekt gekapselt, so dass alle Implementationsaspekte inklusive Kommunikationsprotokollen, Replikationsstrategien und die Verteilung des Zustandes Teil des Objektes sind und hinter seinem Interface verborgen bleiben.

2.2 Kollaborative Systeme

Im Folgenden werden exemplarisch drei Systeme besprochen, mit denen gemeinsam ein Textdokument bearbeitet werden kann und die als Referenzen für die entwickelte Applikation gesehen werden können. Eine ausführliche

Übersicht von kollaborativen Textbearbeitungssystemen inklusive deren Kategorisierung ist in der Masterarbeit von Zafer [16] zu finden.

NetEdit Zafer beschreibt in seiner Arbeit den kollaborativen Texteditor NetEdit, welchen er geschrieben hat. NetEdit erlaubt das gleichzeitige Arbeiten von mehreren Benutzern an einem Dokument. Das Editieren ist dabei vollkommen uneingeschränkt, und Benutzer können Zeichen an beliebigen Stellen einfügen und löschen. Effektiv können zwei oder mehr Benutzer an derselben Stelle Einfüge- sowie Löschoperationen ausführen. NetEdit verwaltet Dateien und Sessions mittels eines zentralisierten Systems. Beim Bearbeiten von Dateien kommen Synchronisations- und Konsistenzprotokolle zum Einsatz. Die Zusammenarbeit wird von NetEdit durch Radarviews und Telepointers unterstützt, vergleiche Abschnitt 7.4.

Im Gegensatz zu den Synchronisations- und Konsistenzprotokollen setzt ShaDoW optimistisches Locking ein. Ein weiterer wichtiger Unterschied zu NetEdit ist die Client/Peer-Architektur, die ShaDoW sehr flexibel macht.

Hydra Eine Gruppe von Studenten der Technischen Universität München (The Coding Monkeys) haben den kollaborativen Editor "Hydra" [5] entwickelt. Hydra läuft auf Mac OS X und findet andere Benutzer spontan mittels Apples "Rendezvous" Technologie [2]. Hydra ermöglicht mehreren Benutzern das uneingeschränkte parallele Bearbeiten eines Dokumentes. Dabei wird eine replizierte Architektur mit Nebenläufigkeitskontrolle eingesetzt. Diese wird durch den Einsatz der "Operation Transformation" Technologie [13] erreicht, bei der spät ankommende Operationen transformiert werden, so dass die Session korrekt fortgesetzt werden kann (optimistic concurrency control). Als Kollaborationshilfsmittel setzt Hydra farbliche Codierung der Eingaben der einzelnen Benutzer ein.

Der Hauptunterschied des entwickelten Systems gegenüber Hydra besteht in der Verwendung eines optimistischen Locking-Algorithmus anstelle der Technologie "Operation Transformation".

NetMeeting [6] NetMeeting ist ein "Collaboration Transparency System" von Microsoft Corporation. In einem "Collaboration Transparency System" [4] arbeiten mehrere Benutzer mittels eines Single-User-Systems zusammen. NetMeeting unterstützt nur stark gekoppelte Kollaboration zwischen den Beteiligten: Wird NetMeeting verwendet, um ein Word-Dokument zu bearbeiten, kann nur ein Benutzer gleichzeitig das Dokument editieren. Das Lockgranulat ist das gesamte Dokument. Da das System eng gekoppelte Kollaboration ermöglicht, sind Kollaborationshilfsmittel wie Telepointers und Radarviews unnütz. Telepointers und Radarviews werden in Abschnitt 7.4 diskutiert.

NetMeeting ist nicht wegen der Qualität als kollaboratives System erwähnenswert, sondern vielmehr weil es als erstes System kommerziell verfügbar war. Somit werden vielen Benutzern die Möglichkeiten einer kollaborativen Umgebung näher gebracht.

Kapitel 3

Eigenschaften partitionierter Objekte

Das verwendete Client/Peer-System Clippee unterstützt Lese-, Erzeugungs- und Schreiboperationen auf atomaren Objekten. Ein wesentlicher Bestandteil dieser Arbeit bestand darin, die Verwaltung partitionierter Objekte in Clippee zu ermöglichen.

Ein kollaboratives Textbearbeitungssystem muss die Möglichkeit bieten, beliebig lange Dokumente effizient zu bearbeiten. Aus diesem Grund ist es zwingend notwendig, im zugrundeliegenden System partitionierte Objekte verwalten zu können.

3.1 Atomare Objekte

Um in den nächsten Abschnitten Teilobjekte und partitionierte Objekte beschreiben zu können, muss zuerst festgelegt werden, was unter *atomaren Objekten* verstanden wird. Die folgenden zwei Punkte charakterisieren ein atomares Objekt.

1. *Vollständiges Update:* Die Veränderung eines atomaren Objektes führt zum Senden des veränderten, atomaren Objektes an alle Instanzen, die ein Replikat des betroffenen atomaren Objektes halten. Die Replikate werden mit den empfangenen atomaren Objekten überschrieben.

2. *Exklusiver Zugriff:* Der Schreibzugriff auf ein atomares Objekt ist exklusiv für genau einen Benutzer. Greift ein Benutzer schreibend auf ein atomares Objekt zu, wird dieses als ganzes gelockt. Während ein Benutzer die Schreibsperre auf einem atomaren Objekt hält, kann kein zweiter schreibend auf dieses Objekt zugreifen.

Ein *atomares Objekt* sei definiert als ein Objekt, das die beiden Punkte der obigen Charakterisierung erfüllt.

3.2 Teilobjekte

Teilobjekte sind bezüglich Eigenschaften den atomaren Objekten ganz ähnlich. Die beiden Punkte "vollständiges Update" und "exklusiver Zugriff" gelten auch für diese Art von Objekten. Der Unterschied zu einem atomaren Objekt liegt darin, dass ein *Teilobjekt* Teil eines partitionierten Objektes ist. Ein Teilobjekt hängt mit anderen Teilobjekten zusammen, um ein partitioniertes Objekt zu bilden; zum Beispiel verbindet eine verkettete Liste Teilobjekte zu einem partitionierten Objekt.

Ein *Teilobjekt* sei definiert als ein atomares Objekt, das im Zusammenhang mit anderen Teilobjekten ein partitioniertes Objekt bildet.

3.3 Partitionierte Objekte

Der Wunsch nach der separaten Behandlung von *partitionierten Objekten* liegt nahe. Erstens ist es wünschenswert, dass bei grossen Datenmengen nur der bearbeitete Teil eines Objektes versendet wird und nicht das ganze Objekt, was zu unnötigem Datenaufkommen im Netz führen würde. Zweitens ist die parallele Bearbeitung von Objekten ungemein wichtig, wenn man zum Beispiel an eine Anwendung wie Document Sharing denkt. Die folgenden Punkte charakterisieren ein partitioniertes Objekt.

1. *Aggregat von Teilobjekten:* Das Objekt besteht aus mehreren Teilobjekten, die in einem bestimmten Zusammenhang stehen.

2. *Selektives Update:* Die Veränderung eines Objektes führt zum Versenden eines Teiles des Objektes. Es werden nur die betroffenen Teilobjekte des partitionierten Objektes aktualisiert.

3. *Paralleler Zugriff:* Mehrere Benutzer können das Objekt an verschiedenen Stellen bearbeiten. Für einen Benutzer werden nur die benötigten Teilobjekte aus dem ganzen gelockt.

4. *Fragmentierbarkeit der Daten:* Damit ein partitioniertes Objekt erzeugt werden kann, müssen die Daten des Objektes in Fragmente aufgeteilt werden können, die in Teilobjekten abgelegt werden.

Ein *partitioniertes Objekt* sei definiert als ein Objekt, das die vier Punkte der obigen Charakterisierung erfüllt.

3.4 Verwaltung der Teilobjekte

Um mit partitionierten Objekten arbeiten zu können, müssen die Teilobjekte in einer Komponente verwaltet werden. Um ein Objekt an einer vorgegebenen Stelle zu verändern, muss das Teilobjekt bestimmt werden, welches das

Datenfragment beinhaltet. Zum Beispiel sollen in einem Terminkalender einzelne Termine bearbeitet werden. Um den Termin am 23.08.2003 um 17:00 Uhr bearbeiten zu können, muss zuerst das Teilobjekt bestimmt werden, welches diesen Termin speichert.

3.5 Partitionierte Objekte kombiniert mit unvollständiger Replikation

Besonders sinnvoll ist der Einsatz von partitionierten Objekten, wenn dazu unvollständige Replikation verwendet wird, bei der nicht jedes Teilobjekt auf jedem Peer repliziert ist. In diesem Fall kommt die Eigenschaft von partitionierten Objekten zum Tragen, dass nicht alle Teilobjekte zur Bearbeitung verfügbar sein müssen. Es wird nur auf einer Teilmenge aller Teilobjekte eines Objektes gearbeitet. Dies hat den Vorteil, dass nicht alle Teilobjekte auf dem bearbeitenden Peer repliziert werden müssen, womit die Netzlast verkleinert und der Speicherbedarf auf einem Peer verringert wird.

Kapitel 4

Modellierung partitionierter Objekte

Im folgenden Kapitel wird ein Modell entwickelt, mit dem *partitionierte Objekte*, wie sie in Kapitel 3 charakterisiert wurden, gehandhabt werden können.

Die Abschnitte in diesem Kapitel sind spezifisch für ein kollaboratives Textbearbeitungssystem dargestellt. Viele weitere Anwendungen können auf einem ähnlichen Modell für partitionierte Objekte aufgesetzt werden, das mit wenig Aufwand angepasst werden kann. Die folgende Aufzählung ist bei weitem nicht vollständig, trotzdem soll sie eine Vorstellung davon geben, in welchem Kontext ein ähnliches Modell für Anwendungen auf einem Peer-to-Peer-System eingesetzt werden kann.

- *Highscore-Liste:* In einem Online-Spiel wird häufig eine Highscore-Liste verwendet. Die Liste ist allen Spielern gemein und muss von mehreren Spielern gleichzeitig bearbeitet werden können. Die Liste kann bei vielen Spielern beliebig gross werden, wobei ein einzelner Spieler wohl immer nur einen Ausschnitt der Liste benötigt, um zum Beispiel ähnlich gute Spieler zu finden oder sein Ranking zu erfahren.

- *Terminplaner:* Ein Terminplaner besteht aus vielen Terminen zu verschiedenen Zeiten an verschiedenen Tagen, die chronologisch geordnet werden können. Wenn sich mehrere Benutzer einen Terminkalender teilen, zum Beispiel einen Vereins- oder Firmenkalender, wollen mehrere Benutzer denselben Kalender parallel bearbeiten und Termine eintragen, verändern und löschen.

- *Reservationssystem:* Ein Reservationssystem ist einem Terminplaner ähnlich, nur dass die Termine einen oder mehrere Räume betreffen.

4.1 Anforderungen an partitionierte Objekte

Die Anforderungen sind spezifisch für ein kollaboratives Textbearbeitungssystem dargestellt. Ein partitioniertes Objekt wird in diesem Kontext verwendet, um ein Dokument gemeinsam zu verwalten.

Die Charakterisierung von partitionierten Objekten, wie sie in Kapitel 3 beschrieben ist, gibt den Rahmen zur Entwicklung einer Datenstruktur vor. Der Charakterisierung zufolge muss die gesuchte Datenstruktur einerseits Replikation (vollständig sowie auch unvollständig) unterstützen und andererseits ein Höchstmass an Parallelität zulassen. Wie während der Charakterisierung erläutert, sollen partitionierte Objekte aus mehreren Teilobjekten aufgebaut werden.

4.2 Datenstruktur

Da die Teilobjekte zusammenhängen und in vielen Fällen geordnet werden können, liegt die Verwendung einer verketteten Liste nahe. Das Problem mit einer verteilten verketteten Liste liegt darin, dass fürs Einfügen eines neuen Knotens zwei Knoten gelockt werden müssen, da bei den Nachbarn des neuen Knotens die Referenzen angepasst werden müssen. Diese Tatsache lässt sich nicht vereinbaren mit der Anforderung, dass höchste Parallelität zu gewährleisten ist.

4.2.1 Verkettete Liste

Mit der folgenden Idee lässt sich das Problem umgehen, dass beim Ändern der Anzahl Knoten in der verketteten Liste zwei Knoten gelockt werden müssen: Vor und nach jedem Knoten wird ein so genannter *Link-Point* eingeführt. Somit wird erreicht, dass für die wichtigsten Operationen auf der verketteten Liste, Split und Merge, nur die absolut notwendigen Knoten gelockt werden müssen, vergleiche Abschnitt 4.3.3 und 4.3.4. In der Abbildung 4.1 wird ein Knoten, wie er für die Datenstruktur zum Einsatz kommt, dargestellt. Im Inneren des Knotens wird hauptsächlich ein *Textblock* untergebracht, um ein Stück Text zu speichern.

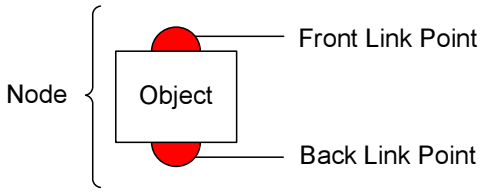

Abbildung 4.1: Die Struktur `List Node` besteht aus einem Objekt (zur Speicherung eines Textblockes) und zwei Link-Points.

Die verkettete Liste kommt zustande, indem mehrere solcher Knoten aneinander gereiht und die entsprechenden Link-Points verbunden werden.

4.3 Operationen auf der Datenstruktur

Die nachfolgend beschriebenen Operationen ermöglichen die Handhabung der verketteten Liste.

4.3.1 Insert

Diese Methode wird im Kontext der kollaborativen Textbearbeitung zum Beispiel beim Lesen aus einer Datei verwendet, siehe Abschnitt A.4. Die Operation fügt einen neuen Knoten hinter den betreffenden Knoten ein.

4.3.2 Remove

Diese Operation entfernt den betreffenden Knoten aus der verketteten Liste. Der Knoten wird nicht gelöscht, sonder in einen Pool zur späteren Weiterverwendung gelegt. Somit wird die Effizienz gesteigert, da ein neu benötigter Knoten nicht erzeugt werden muss, sondern direkt aus dem Pool bezogen werden kann.

4.3.3 Split

Die Operation *Split* dient der Aufteilung eines Teilobjektes in zwei. Das Aufteilen eines Knotens der verketteten Liste wird in zwei Fällen notwendig:

- *Überschreiten der maximalen Blockgrösse:* Das Verändern eines Teilobjektes verursacht das Versenden des ganzen Teilobjektes. Damit die zu versendende Datenmenge kontrolliert werden kann, ist die Länge des Textes, der in einem Teilobjekt gespeichert werden kann, nach oben beschränkt. Wird der zu speichernde Text länger als diese maximale Textlänge, so muss das Teilobjekt in zwei aufgeteilt ("gesplittet") werden, damit das System effizient bleibt, vergleiche Punkt 3 in Abschnitt 3.3.

- *Paralleler Zugriff:* Wenn bereits ein Benutzer die Schreibsperre auf einem Teilobjekt besitzt, und ein zweiter Benutzer denselben Knoten bearbeiten möchte, muss das Teilobjekt aufgeteilt werden, damit dem zweiten Benutzer der Zugriff erlaubt werden kann. Das Split muss erfolgen, weil die Sperren exklusiv für einen Benutzer sind und das ganze Teilobjekt betreffen.

In der Abbildung 4.2 ist links eine verkettete Liste mit drei Knoten gezeigt. Aus einem der beiden oben genannten Gründen wird nun der Knoten N_2 in

die Knoten $N_{2.1}$ und $N_{2.2}$ gesplittet.[1] Hinter den Knoten N_2 wird ein neuer Knoten eingefügt, der den Back-Link-Point des Knotens N_2 erhält. Somit ist das Ende der verketteten Liste des Knotens N_2 an den neuen Knoten $N_{2.2}$ umgehängt worden. Zwischen die beiden Knoten $N_{2.1}$ und $N_{2.2}$ werden zwei neue Link-Points eingefügt, die miteinander verlinkt und an je einen Knoten "angedockt" sind. Die so entstandene, geänderte Liste ist in der rechten Hälfte der Abbildung 4.2 dargestellt.

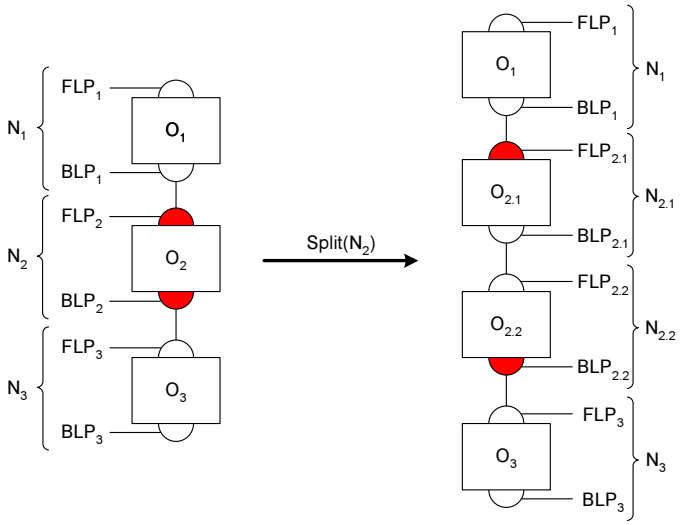

Abbildung 4.2: Exemplarisches Splitten des Knotens N_2 in die Knoten $N_{2.1}$ und $N_{2.2}$.

Szenario 4.3.1. Peer P_A besitzt ein Write-Lock auf dem Knoten N, der Cursor steht an Position C_A. Peer P_B setzt seinen Cursor an Position C_B innerhalb des Knotens N. Damit Peer P_B der Zugriff nicht verweigert werden muss, wird der Knoten N in zwei Knoten N_1 und N_2 aufgesplittet. Peer P_A erhält den Write-Lock auf dem Knoten N_1, und Peer P_B versucht den Write-Lock auf dem Knoten N_2 zu erhalten.

In Szenario 4.3.1 wird der allgemeine Fall eines Splitvorganges beschrieben. Das Problem liegt darin, dass Peer P_B den Knoten N splitten möchte, dies aber nicht kann, weil Peer P_A den betreffenden Lock besitzt. Das Problem wird gelöst, indem Peer P_B eine `Split-Request-Nachricht` an den sperrenden Peer P_A sendet, in der er diesen auffordert, den Knoten N zu splitten. Peer P_A empfängt diese Nachricht und splittet den Knoten N auf. Der Textblock des Knotens N wird an der Splitposition, die sich nach der

[1] Das Objekt O_2 wird beim Split übernommen und heisst nachher $O_{2.1}$; Der Unterschied ist rein textuell und beeinflusst das Objekt in keinster Weise (Der Textblock des Objektes wird allerdings verändert). Genauso verhält es sich mit dem Back-Link-Point BLP_2, das nachher $BLP_{2.2}$ heisst und dem Front-Link-Point FLP_2, das in $FLP_{2.1}$ umbenannt wird.

Formel 4.1 berechnet, aufgeteilt und in die beiden gesplitteten Knoten N_1 und N_2 gefüllt. Nachdem Peer P_A das Split beendet hat, sendet er im Erfolgsfall eine `Ok-Nachricht` an Peer P_B, ansonsten eine `Error-Nachricht`. Nach Erhalt der Antwort versucht Peer P_B den Knoten N_2 zu bearbeiten.

$$\begin{aligned} begin &= min(C_A, C_B) \\ end &= max(C_A, C_B) \\ splitPosition &= \left\lceil \frac{begin + end}{2} \right\rceil \end{aligned} \tag{4.1}$$

Locking Änderungen in der verketteten Liste ergeben sich beim Splitten des Knotens N_2 beim Objekt O_2 und beim Back-Link-Point BLP_2. Diese beiden Objekte müssen vom Peer gelockt werden können, der den Knoten N_2 splittet, damit die Operation erfolgreich ausgeführt werden kann. Um Deadlocks zu vermeiden, werden die Locks immer in derselben Reihenfolge beantragt: Zuerst das Lock auf dem Objekt O_2, dann das Lock auf dem Back-Link-Point BLP_2. Die Freigabe erfolgt, indem zuerst der Back-Link-Point BLP_2 freigegeben wird, anschliessend das Objekt O_2. Somit wird erreicht, dass wann immer das Lock auf dem Objekt O_2 akquiriert werden kann, alle für ein Split benötigten Locks verfügbar sind.

Spezialfälle Im allgemeinen Fall muss Peer P_B eine Nachricht an Peer P_A senden. Vom Zeitpunkt des Absendens der Nachricht bei Peer P_B bis zum Bearbeiten der Nachricht bei Peer P_A vergeht eine bestimmte Zeit t_s. Während dieser Zeit sind Veränderungen am Text und an der verketteten Liste nicht ausgeschlossen. Folgende Punkte sind zu beachten:

- *Lockfreigabe:* Es ist möglich, dass Peer P_A den Write-Lock auf Knoten N zum Zeitpunkt des Empfangs der `Split-Request-Nachricht` bereits wieder freigegeben hat. Es werden zwei Fälle unterschieden: Im ersten Fall ist das Write-Lock zum Zeitpunkt des Empfangs der Nachricht verfügbar und somit muss das Split nicht ausgeführt werden. Im zweiten Fall hat in der Zwischenzeit ein dritter Peer P_C das Lock auf dem Knoten N akquiriert. Somit wird die `Split-Request-Nachricht` an den Peer P_C weitergeleitet.

- *Änderung eines Knotens:* Während der Zeit t_s kann Peer P_A den Knoten N verändern. Dies führt dazu, dass bei der Verarbeitung der Nachricht bei Peer P_A die angeforderte Stelle C_B nicht mehr der Stelle entspricht, die Peer P_B ursprünglich gefordert hat. Falls der Knoten so verändert wird, dass sich die ursprüngliche Reihenfolge der Positionen C_A und C_B vertauscht haben, führt dies zu einer falschen Blockvergabe. Das heisst, dass der Teil des ursprünglichen Blocks freigegeben

wird, den Peer P_A nicht bearbeiten will. Dieser Fall wird dem Benutzer überlassen, der nochmals versuchen muss, seinen Cursor zu platzieren.

4.3.4 Merge

Die Operation *Merge* bildet das Pendant zum Split, indem sie zwei Knoten zu einem verschmilzt. Abbildung 4.3 illustriert den Mergevorgang. Das Ausführen dieser Methode dient vor allem der Performance, damit nicht zu viele Knoten mit sehr wenig Inhalt die Liste unnötig verlängern. Solche Knoten können bei mehrfachem Split oder beim Löschen von Zeichen entstehen.

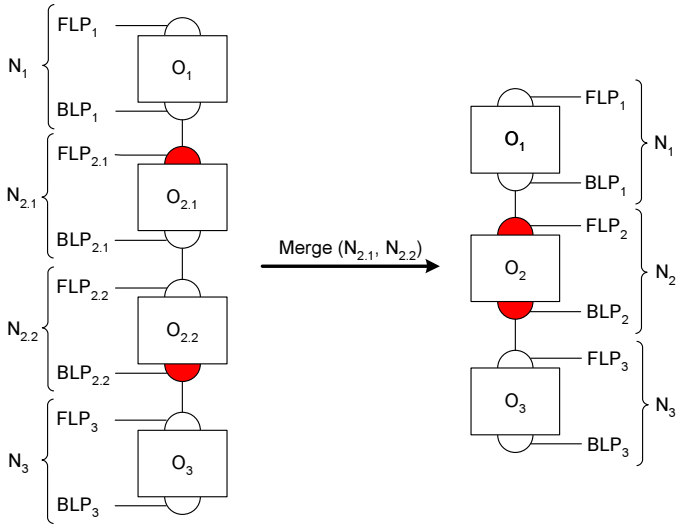

Abbildung 4.3: Exemplarisches Mergen der Knoten $N_{2.1}$ und $N_{2.2}$ zum Knoten N_2.

Der Text des zweiten Knotens wird in den ersten eigefügt und anschliessend wird das Objekt $O_{2.2}$ sowie die beiden Link-Points $BLP_{2.1}$ und $FLP_{2.2}$ gelöscht.[2]

Beim Ändern des Textes eines Knotens N_1 von T_1 nach T_2 wird der geänderte Text T_2 auf seine Länge geprüft. Beim Unterschreiten einer vorgegebenen Mindestgrösse wird der folgende Knoten in der Liste N_2 ebenfalls auf seine Grösse überprüft. Unterschreitet auch dieser die Mindestgrösse und ist das Write-Lock auf N_2 verfügbar, werden die beiden Knoten N_1 und N_2 verschmolzen.

Die Beschränkung des Verschmelzens auf Knoten, die nicht gelockt sind, ist sinnvoll, weil die einzelnen Knoten adaptiv gemäss den Schreibzugriffen der Benutzer entstanden sind. Es ist sinnlos, die beiden Knoten N_1 und N_2

[2] Der Knoten wird in einen Pool eingefügt, damit bei einem späteren Erzeugen eines neuen Knotens einer aus dem Pool verwendet und mit neuem Inhalt gefüllt werden kann, vergleiche Abschnitt 4.3.2. Analog wird mit den Link-Points vorgegangen.

zu verschmelzen, wenn N_2 von Peer P_2 gelockt wird, weil dieser dann den Write-Lock auf dem Block verliert.

Zusätzlich läuft auf jedem Peer ein Merge-Prozess im Hintergrund, der periodisch alle Listen durchläuft und "zu kleine" Knoten verschmilzt.

Locking Um die Operation *Merge* ausführen zu können, müssen fünf Objekte gelockt werden können. Tabelle 4.1 listet alle Objekte auf, die ein Peer locken muss, um ein Merge zu vollziehen.

Gelocktes Objekt	Grund
Objekt $O_{2.1}$	Der Text des Objektes $O_{2.1}$ wird verändert.
Back-Link-Point $BLP_{2.1}$	Der Back-Link-Point $BLP_{2.1}$ wird nicht weiter verwendet und deshalb gelöscht.
Objekt $O_{2.2}$	Das Objekt $O_{2.2}$ wird nicht weiter verwendet und deshalb gelöscht.
Front-Link-Point $FLP_{2.2}$	Der Front-Link-Point $FLP_{2.2}$ wird nicht weiter verwendet und deshalb gelöscht.
Back-Link-Point $BLP_{2.2}$	Der Back-Link-Point $BLP_{2.2}$ wird ans Objekt O_2 verschoben und somit verändert.

Tabelle 4.1: Gelockte Objekte bei der Operation Merge

Nur wenn alle diese Objekte gelockt werden können, kann die Operation erfolgreich ausgeführt werden. Wann immer die Locks auf den Objekten O_1 und O_2 akquiriert werden können, ist sichergestellt, dass die Locks der anderen beteiligten Objekte verfügbar sind. Dies wird erreicht, indem die Locks in der Reihenfolge der Tabelle 4.1 akquiriert werden. Die Freigabe erfolgt in umgekehrter Reihenfolge.

4.3.5 Import

Diese Operation importiert ein Objekt aus einer Datei. Im Kontext der kollaborativen Textbearbeitung wird ein gewöhnliches Textdokument aus einer Datei in eine verkettete Liste eingelesen. Der aus der Datei eingelesene Text wird dabei in Blöcke zerlegt, die einzeln in die Knoten eingefügt werden. Die so entstandenen Knoten weisen einen optimalen Füllgrad auf. Optimaler Füllgrad heisst, dass der Knoten möglichst lange weder gesplittet noch verschmolzen werden muss. Bei gegebener maximaler und minimaler Blockgrösse berechnet sich die optimale Blockgrösse gemäss der Formel 4.2.

$$\text{optBlockgrösse} = \left\lfloor \frac{\text{maxBlockgrösse} + \text{minBlockgrösse}}{2} \right\rfloor \tag{4.2}$$

4.3.6 Export

Diese Operation exportiert ein Dokument in eine Datei, um es persistent auf einem Sekundärspeicher (zum Beispiel eine Festplatte) abzulegen. Alle Teilobjekte werden sequentiell in die Datei geschrieben. Das Resultat ist eine gewöhnliche Textdatei, die mit einem herkömmlichen Texteditor lokal weiterbearbeitet werden kann. Da nur eine lokale Kopie des Dokumentes abgelegt wird, müssen keine Locks beantragt werden. Das heisst, dass ein Export zu jedem Zeitpunkt erfolgen kann.

4.4 Skip-Liste

Wie in Abschnitt 3.4 erklärt, müssen innerhalb eines partitionierten Objektes Teilobjekte gefunden werden können. Der zeitliche Aufwand für diesen Vorgang wächst für die bisher dargestellte Datenstruktur einer verteilten verketteten Liste mit Link-Points linear mit der Grösse des Dokumentes. Um eine bessere Skalierbarkeit zu erreichen, kann die Liste zu einer Skip-Liste [12] erweitert werden, womit der Aufwand mit $O(log(n))$ skaliert.

Bemerkung 4.4.1. An dieser Stelle sei vermerkt, dass aus zeitlichen Gründen auf die Implementation von Skip-Listen zugunsten der Ausgestaltung der Anwendung verzichtet wurde.

4.5 Verwaltung des partitionierten Objektes

In dieser Arbeit wurde bei der Entwicklung der Verwaltung speziell die übergeordnete Anwendung der kollaborativen Textbearbeitung berücksichtigt, weshalb das entwickelte Konzept für andere Anwendungen separat angepasst werden muss. Für Details sei auf das Kapitel 5 verwiesen.

4.6 Verwaltung für ein kollaboratives Textbearbeitungssystem

Um mit einem zusammengesetzten Textdokument umgehen zu können, sind folgende Punkte von Bedeutung.

1. *Ermitteln eines Teilobjektes:* Damit in einem Dokument eine bestimmte Position verändert werden kann, muss zuerst bestimmt werden, zu welchem Teilobjekt die betroffene Position gehört.

2. *Ermitteln eines Indexes:* Damit ein neues Teilobjekt korrekt ins Objekt eingefügt werden kann, muss der Index bestimmt werden, an welchem es in die verkettete Liste eingefügt wird.

3. *Festhalten des schreibbaren Bereiches:* Da häufig viele Änderungen an einem Teilobjekt durchgeführt werden (zum Beispiel beim Einfügen eines Satzes), ist es sinnvoll, das gelockte Teilobjekt zu behalten und erst wieder freizugeben, wenn dies explizit veranlasst wird. Somit muss nicht bei jedem Schreibvorgang das betroffene Teilobjekt bestimmt werden, wie in Punkt 1 beschrieben, sondern nur dann wenn eine Eingabe über die Maus erfolgt oder eine Funktionstaste der Tastatur, die den Cursor verschiebt, betätigt wird.

4. *Wechseln des schreibbaren Bereiches:* Der schreibbare Bereich muss explizit gewechselt werden, da das bisher gelockte Teilobjekt freigegeben und ein anderes gelockt werden muss.

5. *Festhalten der Schreibposition:* Innerhalb des schreibbaren Bereiches des gelockten Teilobjektes kann die Schreibposition wie in einem normalen Editiersystem geführt werden, ohne dass die Änderungen anderer Benutzer berücksichtigt werden.

Kapitel 5

Architektur

Die in Abbildung 5.1 dargestellte Architektur des entwickelten Systems folgt dem Model-View-Controller-Paradigma [8]. Das *Modell* wird von einer verketteten Liste gebildet, wie in Kapitel 4 beschrieben.

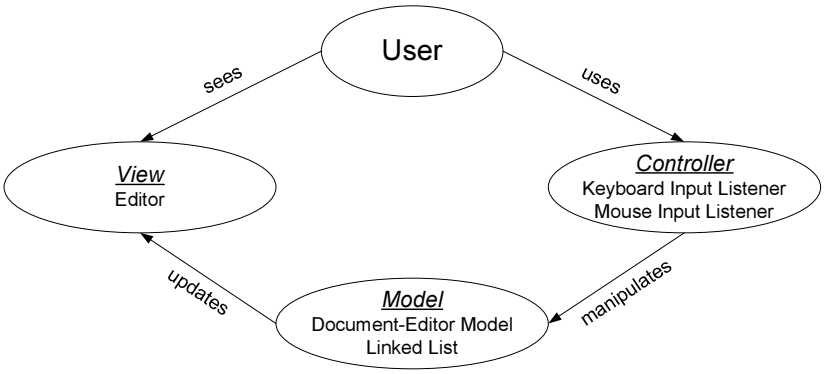

Abbildung 5.1: Die Architektur folgt dem Model-View-Controller-Paradigma.

Die View, die dem Benutzer zur *Ansicht* des Textdokumentes dient, wird durch den Editor realisiert. Mittels des Editors wird dem Benutzer der gemeinsam bearbeitete Text präsentiert.

Die vom Benutzer getätigten Eingaben, via Maus und Tastatur, werden in den entsprechenden Input-Listeners abgefangen und verarbeitet, wobei allfällige Änderungen ins Modell geschrieben werden. Die Listeners bilden den *Controller*. Dieser steuert die Navigation innerhalb des Dokumentes und modifiziert das Dokument den Eingaben des Benutzers entsprechend.

5.1 Das Modell

In Abbildung 5.2 ist das Modell des entwickelten Systems schematisch dargestellt. Das Modell ist verteilt, das heisst, dass in der vorliegenden Version

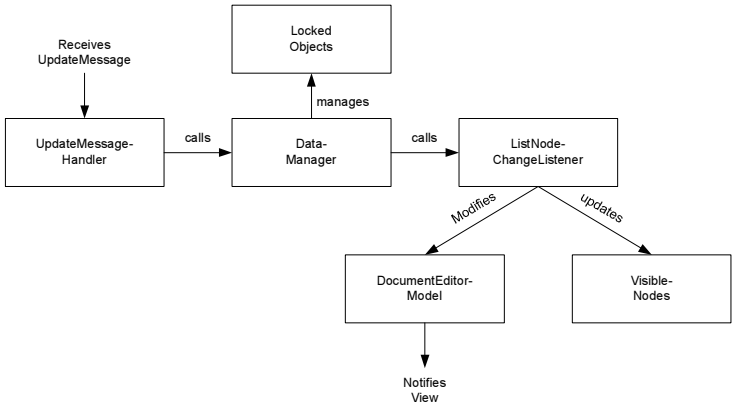

Abbildung 5.2: Schema des Modells

die verkettete Liste vollständig bei jedem beteiligten Peer repliziert ist. Im Modell müssen zwei Strukturen gewartet werden. Einerseits die Struktur `VisibleNodes`, die eine lokale Repräsentation der Liste darstellt und andererseits ein `DocumentEditorModel`, welches die direkte Grundlage für den Editor bildet.

Der DataManager von Clippee verwaltet alle replizierten Objekte. Das Problem ist, dass diese in der Form von DataObjects gespeichert sind. Es gibt Fälle, in denen die verkettete Liste traversiert werden muss und aus Effizienzgründen ist es sinnvoll, eine aktuelle Kopie aller Objekte zu haben, die Knoten der verketteten Liste betreffen. Die Kopien der Objekte sind direkt als ListNodes verfügbar, so dass die Konvertierung entfällt.

Änderungen am Dokument laufen stets über den DataManager von Clippee. Beim DataManager wird für jeden Knoten aus der verketteten Liste ein Listener registriert, der aufgerufen wird, wenn Änderungen auf dem entsprechenden Knoten stattgefunden haben. Der aufgerufene Handler `ListNode-ChangeListener` fügt die gemeldete Änderung ins Dokument ein.

Im Zusammenhang mit unvollständiger Replikation wäre es sinnvoll, wenn die `VisibleNodes` nur aus den Knoten bestünden, die zur Anzeige im Editor benötigt würden. Da Clippee derzeit jedoch keine unvollständige Replikation unterstützt, wurde aus zeitlichen Gründen darauf verzichtet.

5.1.1 Update des Modells

Ein Update des Modells ist erstens notwendig, wenn der Benutzer lokal eine Änderung am Dokument via Editor vornimmt, indem er zum Beispiel einen Buchstaben eingibt. Andererseits können andere Benutzer, die über das Netz eingebunden sind, remote am Dokument eine Änderung anbringen, die lokal im Dokument nachgetragen werden muss. Damit das Modell nur von einem Thread bearbeitet wird, womit Fehler beim Concurrent-Write

vermieden werden können, löst jedes Update eine Update-Nachricht aus. Für eine "normale" Update-Nachricht erklärt Abbildung 5.3 mittels eines Sequenzdiagrammes die Verarbeitung.

Eine Update-Nachricht, die das Löschen eines Zeichens oder eines Teiles des Knotens betrifft, läuft analog zum oben genannten "normalen" Update ab.

Betrifft die zu verarbeitende Update-Nachricht ein Split, läuft die Verarbeitung gemäss Abbildung 5.4 ab. Bei einem Split muss die View nicht verändert werden, da sich nur die verkettete Liste ändert.

Betrifft die zu verarbeitende Update-Nachricht ein Merge, läuft die Verarbeitung gemäss Abbildung 5.5 ab. Bei einem Merge muss die View ebenfalls nicht verändert werden, da sich auch hier nur die verkettete Liste ändert.

5.2 Die Ansicht

Das Modell wird über den Editor präsentiert. In einem Textfenster wird das Dokument zur Ansicht dargestellt. Das Textfenster zeigt genau den Text des DocumentEditorModels an. Wie dieses erzeugt und verwaltet wird, ist in Abschnitt 5.1.1 erklärt.

Wird remote von einem Benutzer eine Eingabe gemacht, erscheint diese sofort farbig kodiert in der Ansicht, direkt an der Stelle, wo sie remote ins Dokument geschrieben wurde. Zum Ansehen der Änderung muss der geänderte Bereich des Dokumentes im sichtbaren Bereich des Editors liegen.

Die Ansicht ist nur ein Teil des Editors, der als Schnittstelle zwischen dem Benutzer und dem System agiert. Die Benutzerschnittstelle wird im Anhang A erläutert und die Verarbeitung von Benutzereingaben erklärt Abschnitt 5.3.

5.3 Der Controller

Gesteuert wird die Anwendung mittels den Eingaben des Benutzers, die entweder über die Maus oder die Tastatur erfolgen können. Mit der Maus kann der Benutzer den Cursor an eine beliebige Position setzen, an der er anschliessend das Dokument bearbeiten möchte. Gleiches kann er über die Tastatur erreichen, allerdings in einem beschränkten Ausmass, mit den Pfeiltasten und den folgenden Funktionstasten: `Home`, `End`, `PgUp`, `PgDn`. Tastatureingaben, die einem Zeichen zugeordnet werden, bewirken eine Veränderung des Dokumentes. Weiter sind die Tastenkombinationen `ctrl-c` fürs Kopieren, `ctrl-v` fürs Einfügen und `ctrl-x` fürs Ausschneiden reserviert.

Um Eingaben des Benutzers zu verarbeiten, werden beim Textfenster zwei Listener registriert. Der `Keyboard-Input-Listener` wird immer dann aufgerufen, wenn eine Taste auf der Tastatur gedrückt wird. Tastendrücke,

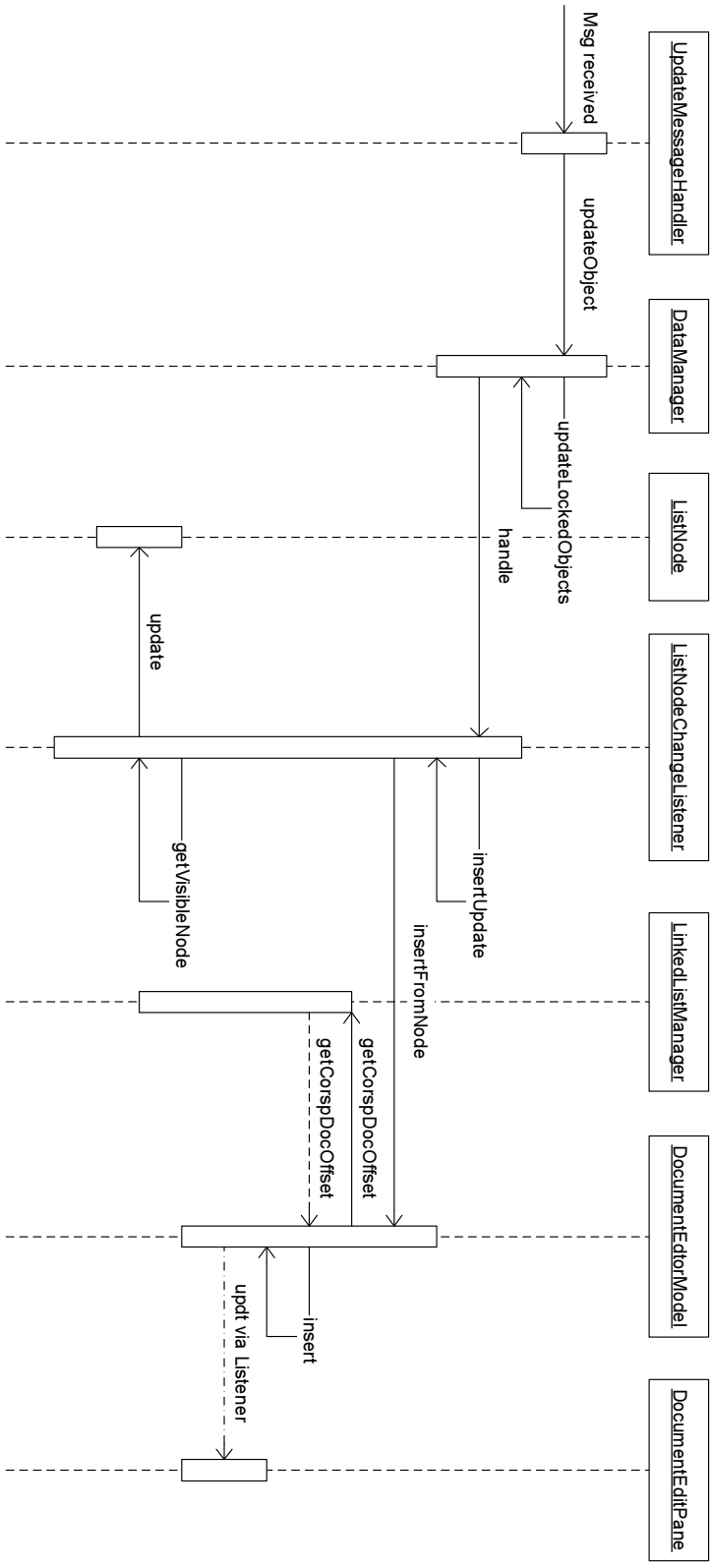

Abbildung 5.3: Sequenzdiagramm eines Updates

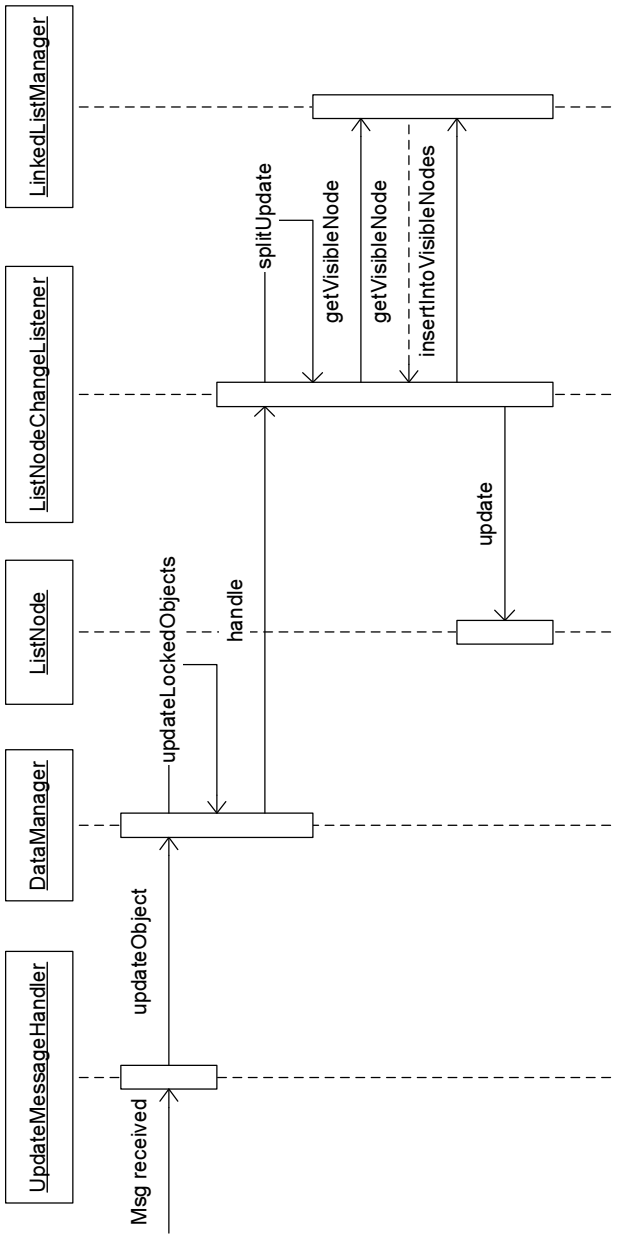

Abbildung 5.4: Sequenzdiagramm eines Splits

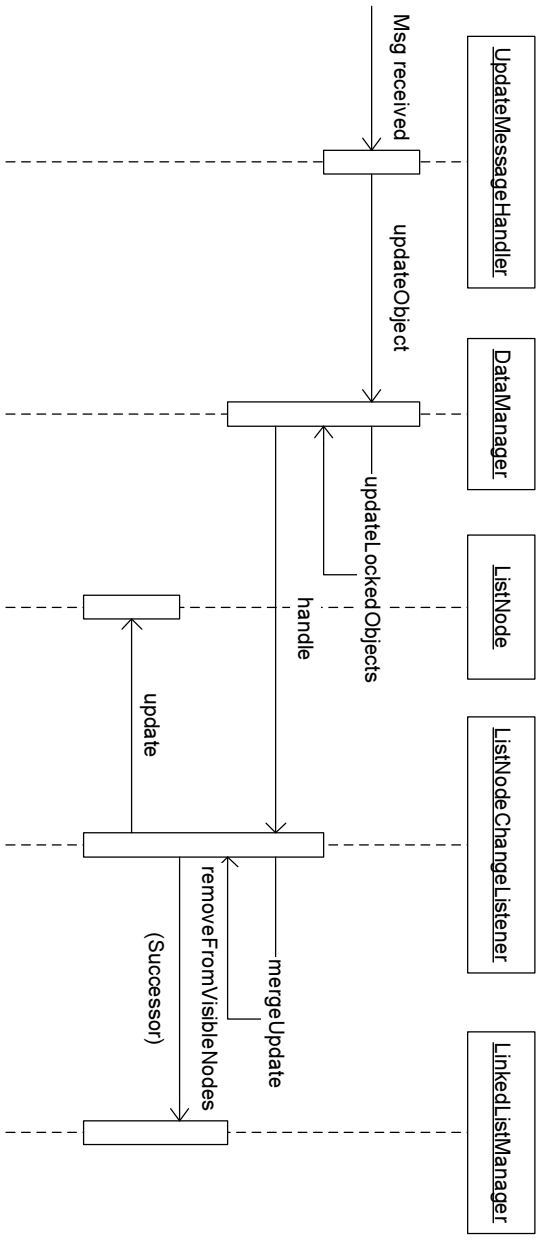

Abbildung 5.5: Sequenzdiagramm eines Merges

die eine Veränderung des Dokumentes zur Folge haben, werden vom `Keyboard-Input-Listener` entsprechend der Abbildung 5.6 behandelt. Auf diese Weise wird die Änderung in den Knoten eingetragen. Dieser sendet via DataManger allen replizierenden Peers eine Update-Nachricht, die diese gemäss Abschnitt 5.1.1 bearbeiten, womit das Dokument global verändert wird. Abschliessend führt der Knoten die Änderung bei sich lokal nach.

Tastatureingaben, die der Navigation innerhalb des Dokumentes dienen (Pfeiltasten, `Home`, `End`, `PgUp`, `PgDn`), behandelt der Keyboard-Input-Listener, indem er den Cursor an die gewünschte Stelle setzt und die entsprechenden Vorkehrungen in der Verwaltung der verketteten Liste vornimmt. Die Navigation per Maus wird durch den `Mouse-Input-Listener` geregelt. Wird mit der Maus ins Textfenster geklickt oder eine Selektion angebracht (Ziehen der Maus bei gedrückter Maustaste), ermittelt der `Mouse-Input-Listener` die entsprechende Position im Dokument und setzt den Cursor an die gewünschte Stelle.

Die Navigation innerhalb des Dokumentes verändert das Dokument selber nicht, sondern ändert nur die Cursorposition. Das Ändern der Cursorposition muss für die Verwaltung der verketteten Liste berücksichtigt werden. Die Handhabung der verketteten Liste wird in Abschnitt 5.4 genauer erklärt.

5.4 Verwaltung der verketteten Liste

In Abschnitt 4.6 wurde bereits erwähnt, wozu die Verwaltung des Dokumentes benötigt wird. In diesem Abschnitt wird die Verwaltung von Seiten der Implementation aufgegriffen.

5.4.1 Übersetzen von Offsets

Beim Erhalten eines Updates muss die Änderung eines Knotens der verketteten Liste ins Dokument übernommen werden. Dafür muss die Position direkt vor dem ersten Zeichen des Knotens bestimmt werden. Erst danach kann die Position für die Änderung innerhalb des Dokumentes bestimmt werden. Diesem Zweck dient die Methode `getCorrespondingDocumentOffset()` der Klasse `LinkedListManager`. In dieser Methode wird die gesamte Liste traversiert und die Anzahl Zeichen aufsummiert, bis der veränderte Block erreicht ist. Umgekehrt geht die Methode `getCorrespondingListNode()` vor, die zu einer Position im Dokument den zugehörigen Knoten und die Position relativ zu diesem in der verketteten Liste ermittelt.

Das Umrechnen der Offsets auf beschriebene Art und Weise skaliert linear mit der Grösse des Dokumentes. Je grösser das Dokument wird, umso mehr Blöcke weist es auf, weil ein einzelner Block in seiner Grösse beschränkt ist. Da die Struktur vom Konzept her eine Liste realisiert, ist der Aufwand für das Traversieren der Liste, welches zum Bestimmen der Offsets verwendet wird, in $O(n)$.

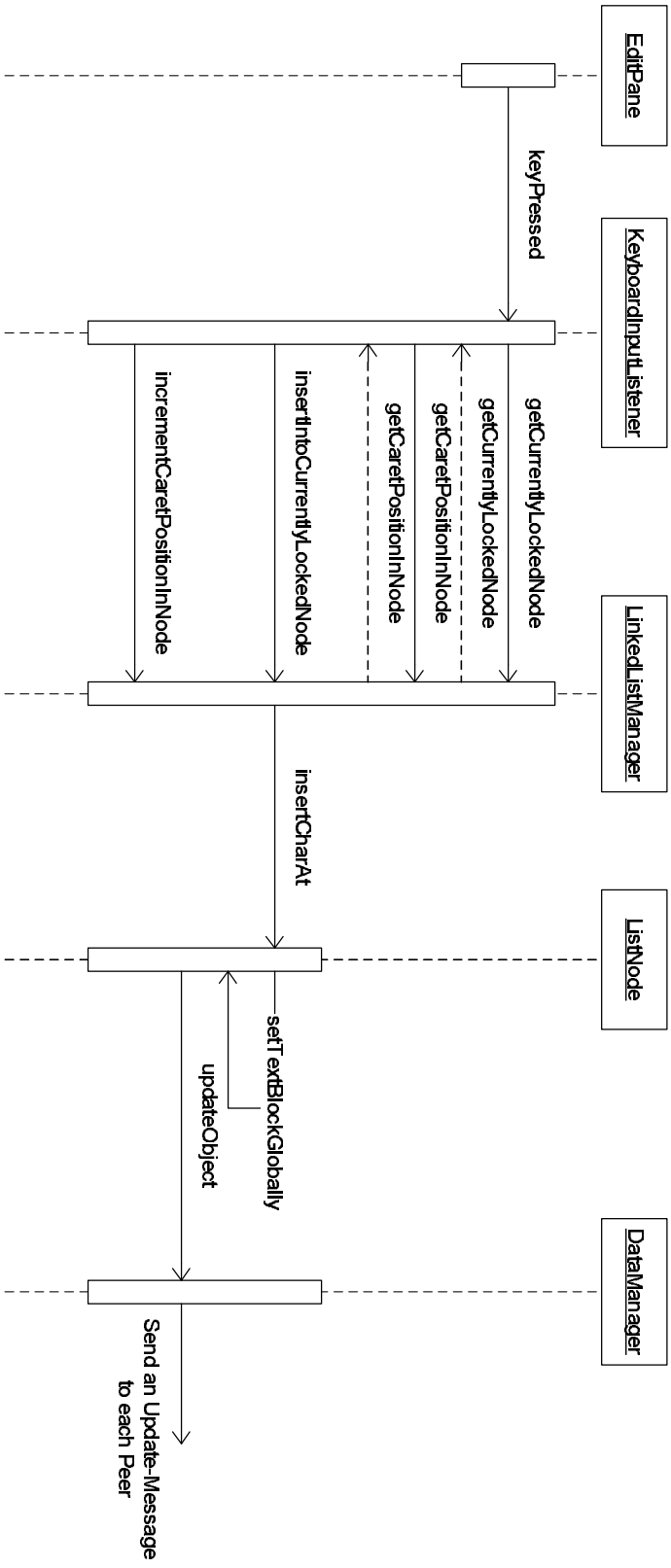

Abbildung 5.6: Sequenzdiagramm für die Eingabe über die Tastatur

Für das Traversieren der Liste wird die Struktur `VisibleNodes` verwendet, die in Abschnitt 5.1 eingeführt wird. Diese Struktur ist eine Kopie der Datenobjekte, die der DataManager von Clippee verwaltet. In Clippee werden diese als Objekte des Typs `DataObject` verwaltet, welche nicht direkt zur Traversierung der Liste verwendet werden können. Die Objekte werden in Objekte des Typs `ListNode` umgewandelt und so verwaltet, um das Traversieren zu beschleunigen.

5.4.2 Verwaltung des schreibbaren Knotens

Ist der Cursor einmal gesetzt, speichert die Klasse `LinkedListManager` den zugehörigen Knoten und die Position des Cursors innerhalb des Knotens in den beiden Variablen `currentlyLockedNode` und `positionInCurrentlyLockedNode`. Dies bringt den Vorteil, dass das System die Änderung direkt in den betreffenden Knoten schreiben kann, wenn der Benutzer ein Zeichen eingibt.

Beim Einfügen oder Löschen eines oder mehrerer Zeichen innerhalb des schreibbaren Knotens wird nur die Variable `positionInCurrentlyLockedNode` um die Anzahl der eingefügten oder gelöschten Zeichen korrigiert.

Wird der Cursor explizit, nicht durch Einfügen oder Löschen von Text, verschoben, so wird mit der Methode `changeCaretScope()` der Klasse `LinkedListManager` der schreibbare Knoten neu ermittelt. Diese Methode kümmert sich auch darum, dass ein Split veranlasst wird, wenn zwei Benutzer denselben Knoten bearbeiten möchten.

Damit der betreffende Knoten und die Position innerhalb dieses Knotens korrekt ermittelt werden können, darf das Dokument während des Ermittelns nicht verändert werden, da ansonsten die Cursorposition implizit verschoben wird, und die Ermittlung zu einem falschen Ergebnis führen kann.

5.5 Collaboration Awareness

Das System bietet Hilfestellung für die Zusammenarbeit in zweierlei Hinsicht. Im rechten Bereich des Editors ist eine Liste sichtbar, die alle verbundenen Benutzer mit ihren Benutzernamen anzeigt, Abschnitt A.2 zeigt einen Screenshot des Editors. Beim Klicken auf den Benutzernamen, wird der Block im Dokument hervorgehoben, welchen dieser Benutzer momentan editiert. Der Block wird nur hervorgehoben, falls dieser im sichtbaren Bereich des Textfensters liegt.

Von einem Benutzer eingegebener Text wird im Textfenster farblich kodiert. Mit derselben Farbe ist der Name des Benutzers in der Benutzerliste dargestellt. Jedem verbundenen Benutzer wird eine Farbe aus einer beschränkten Farbpalette zugeordnet, die zur Codierung verwendet wird. Ist die Palette erschöpft, werden die zur Verfügung stehenden Farben nochmals verteilt. Somit können mehrere Benutzer farblich gleich kodiert sein.

5.6 Dokumentenordner

Um mit der Anwendung mehr als ein einziges Dokument zu verwalten, wird ein einfaches Filesystem benötigt. Das Filesystem hat nur eine Ebene, auf der alle verfügbaren Dokumente verwaltet werden. Das Filesystem ist so realisiert, dass ein spezielles Objekt (Dokumentenordner) existiert, das die Schlüssel der Startknoten der einzelnen Dokumente enthält. Solange ein Dokument existiert, wird der erste Knoten immer der erste bleiben.

Ein Dokument existiert im Normalfall im Hauptspeicher der beteiligten Peers. Um ein Dokument bei einem Benutzer in dem Sinne persistent zu machen, dass es ein komplettes Herunterfahren des Clusters übersteht, kann es in eine gewöhnliche Textdatei exportiert werden. Die Textdatei kann lokal mit einem herkömmlichen Texteditor weiterverarbeitet werden. Zu einem späteren Zeitpunkt kann die Datei in ein laufendes System importiert und somit wieder in Zusammenarbeit editiert werden.

Auf den Dokumentenordner wird nicht sehr oft schreibend zugegriffen, nämlich nur dann, wenn ein neues Dokument angelegt wird. Aus diesem Grund ist es vertretbar, dass bei jedem Schreibzugriff das ganze Objekt gelockt wird. Ist beim Anlegen eines Dokumentes der Ordner von einem anderen Peer gelockt, wartet das System auf die Freigabe des Locks und versucht anschliessend das Dokument anzulegen. Dies kann zu einer kurzen Verzögerung führen, wenn viele Dokumente von Benutzern gleichzeitig erzeugt werden.

Kapitel 6

Fazit

In diesem Kapitel werden sowohl das entwickelte Modell für partitionierte Objekte als auch das implementierte System kurz reflektiert; zudem wird ein allgemeiner Rückblick auf die Arbeit gegeben.

6.1 Beurteilung des Modells und des Systems

Die entwickelte Struktur für partitionierte Objekte ist auf ein kollaboratives Textbearbeitungssystem spezialisiert und ist dafür gut geeignet. Das Modell sollte mit wenig Aufwand so erweiterbar sein, dass beliebige Objekte, welche die Eigenschaft der Fragmentierbarkeit aufweisen, als partitionierte Objekte in Clippee verwaltet werden könnten.

Das darauf aufgebaute kollaborative Textbearbeitungssystem *ShaDoW* setzt partitionierte Objekte erfolgreich ein, um Dokumente zu verwalten. Das uneingeschränkte Editieren von Dokumenten durch verschiedene Benutzer läuft flüssig, und die Benutzer werden mit der nötigen Hilfe bei der Zusammenarbeit unterstützt.

Generell kann gesagt werden, dass das entwickelte System gute Ansätze für ein überzeugendes, kollaboratives System liefert.

6.2 Rückblick auf die Arbeit

Es war spannend und fordernd, während der letzten vier Monate bei der Gruppe für "Distributed Computing" zu diplomieren. Die Aufgabenstellung liess mir viele Freiheiten in der Ausgestaltung meiner Arbeit. Deshalb konnte ich die Schwerpunkte sehr selbständig bestimmen und auf meine Interessen anpassen. Das Resultat ist die praktische Umsetzung des entwickelten, theoretischen Modells der partitionierten Objekte in einem kollaborativen Textbearbeitungssystem.

Die Entwicklung eines kollaborativen Textbearbeitungssystems war fordernd: Einerseits war die Erarbeitung eines Konzeptes anspruchsvoll und

andererseits wurde ich mit verschiedensten programmiertechnischen Schwierigkeiten bei der Implementation konfrontiert. Dank der Unterstützung der Gruppe konnten die geforderten Aufgaben gut gelöst werden. Auch der Einsatz des Client/Peer-Systemes Clippee als Grundlage für meine Arbeit war spannend, weil die Technologie für mich neu und der Lerneffekt somit gross war. Der Einsatz von Clippee lief bis auf ein, zwei kleine Probleme reibungslos ab.

Über alle vier Monate beurteilt, war die vorliegende Arbeit eine gute Wahl und die verbrachte Zeit spannend und lehrreich.

Kapitel 7

Ausblick

In der vorliegenden Arbeit wurde eine erste Version eines kollaborativen Textbearbeitungssystems geschaffen. Dieses Kapitel liefert Anhaltspunkte für Verbesserungen und Erweiterungen des bestehenden Systems.

7.1 Uneingeschränktes Editieren

Mit dem erarbeiteten theoretischen Modell zur Verwaltung der Dokumente ist es möglich, diese uneingeschränkt zu editieren: Zwei Benutzer können ihren Cursor genau an dieselbe Stelle setzen und das Dokument bearbeiten. Dies ist möglich, indem ein Benutzer den Block bearbeitet, der hinter der betreffenden Stelle liegt und der andere den Block vor der Stelle. Um drei oder mehr Benutzer an genau derselben Position Zeichen einfügen zu lassen, können an dieser Stelle neue, leere Knoten in die verkettete Liste eingefügt werden, so dass jeder Benutzer wieder einen Knoten bearbeiten kann. Führt ein Benutzer B_A an einer Position P eine Löschoperation aus, deren zugehöriger Knoten N bereits von einem anderen Benutzer B_B gelockt wird, muss der Knoten N gesplittet werden, so dass Benutzer B_A den Knoten erhält, der nach dem Split die Position P speichert. An derselben Position kann nur ein einziger Benutzer zu einer gegebenen Zeit eine Löschoperation ausführen. Dies ist durchaus sinnvoll, denn ein einzelnes Zeichen kann nur von einem Benutzer gelöscht werden. Eine weitere Löschoperation beträfe das vorhergehende Zeichen.

Aus zeitlichen Gründen wurde darauf verzichtet, im implementierten System zwei Benutzer genau dieselbe Stelle bearbeiten zu lassen. Wird dies dennoch versucht, erscheint der Cursor rot und der zweite Benutzer kann das Dokument nicht editieren.

7.2 Zugriffsrechte auf dem Dokument

Ein sehr interessanter Punkt wäre die Realisierung von Zugriffsrechten auf einem Dokument. Es soll möglich sein, dass nur ausgewählte Benutzer ein Dokument bearbeiten können und anderen der Zugriff verweigert wird. Der Zugriff soll aber nicht nur für ein ganzes Dokument erteilt werden können, sondern auch für einzelne Bereiche des Dokumentes. Damit soll zum Beispiel ermöglicht werden, dass ein Manager in einem Bericht einzelne Teile an seine Mitarbeiter delegiert und dazu definiert, welcher Mitarbeiter welchen Teil bearbeiten kann.

Die Realisierung von Zugriffsrechten bedingt, dass sich ein Benutzer mit seinem Benutzernamen und Passwort beim System anmeldet. Die Idee wäre, dass der Benutzer, der ein neues Dokument erzeugt, *Master* für dieses Dokument wird. Der Master kontrolliert die Zugriffsrechte auf dem gesamten Dokument. Er kann das Dokument segmentieren und spezifisch pro erstelltem Segment Benutzer zum Editieren berechtigen.

In diesen Bereich fällt auch das exklusive Locking. Dabei wird erreicht, dass ein Benutzer den alleinigen Zugriff auf einem Block erlangt und dieser nicht weiter gesplittet werden kann.

7.3 Persistente Datenhaltung

Der in Abschnitt 5.6 beschriebene Dokumentenordner ermöglicht das Verwalten von mehreren Dokumenten auf einer Ebene. Bei steigender Anzahl von Dokumenten ist es von Interesse, die Dokumente besser ordnen zu können, ähnlich wie dies mit Dateien in einem Dateisystem möglich ist.

Dokumente, die ein vollständiges Herunterfahren des Clusters (Ausschalten aller Peers) überstehen sollen, müssen von einem Benutzer von Hand in eine Textdatei auf seine Festplatte exportiert und nach dem erneuten Aufstarten wieder ins System importiert werden. Das System könnte diese Aufgabe automatisch vornehmen oder sogar noch einen Schritt weiter gehen: Die Knoten der verketteten Liste könnten in einer Art Datenbanksystem und nicht nur im Hauptspeicher verwaltet werden.

7.4 Hilfsmittel für die Zusammenarbeit

Wie in Abschnitt 5.5 beschrieben, unterstützt das vorliegende System bereits einige Hilfsmittel, welche die Zusammenarbeit erleichtern. Folgende weitere Hilfsmittel wären nützlich:

- *Telepointers:* Telepointers [16, 3, 9] geben Aufschluss über die Position des Mauszeigers eines entfernten Benutzers. Eine Variante des Telepointers ist das Telecaret, welches die Position des Eingabezeichens (Carets) des entfernten Benutzers anzeigt. ShaDoW unterstützt

bisher keine Telepointers, fügt allerdings die Eingaben mit verschiedenen Farben ins Dokument ein.

- *Radarview:* Obwohl Telepointers die exakte Stelle eines entfernten Benutzers anzeigen, liefern sie keine Information über den sichtbaren Ausschnitt des Dokumentes beim entfernten Benutzer. Radarviews [16, 3, 9] stellen diese Information zur Verfügung, indem sie eine Miniaturansicht des gesamten Dokumentes anzeigen, wobei die Sichten aller Benutzer mit einem farbigen Rahmen gekennzeichnet sind. ShaDoW unterstützt bisher keine Radarviews, markiert allerdings den Block eines Benutzers beim Anklicken dessen Benutzernamens.

- *Chat:* Um den Benutzern einen vom Dokument unabhängigen Kommunikationskanal zur Verfügung zu stellen, könnte ein Chat angeboten werden. Der Chat sollte mindestens die Funktionalität bieten, dass entweder allen Benutzern oder nur denjenigen, die momentan ein bestimmtes Dokument bearbeiten, eine Nachricht zugestellt werden kann.

7.5 Ausbau zu einer Client-Anwendung

Zur Zeit ist ShaDoW eine Peer-Anwendung. Das bedeutet, dass ShaDoW nur auf Peers von Clippee läuft, und somit alle Instanzen der Anwendung vollständig miteinander verbunden sind. Um die Netzlast zu verringern kann die Anwendung zu einer Client-Anwendung erweitert werden. Dabei verwalten die Peers die Dokumente, welche durch Instruktionen der Clients bearbeitet werden. Mehrere Clients verbinden sich mit einem Peer, der die Änderungen "seiner" Clients auf die anderen Peers repliziert. Im vorliegenden Falle einer Peer-Anwendung speichert jede ausführende Instanz von Shadow alle Dokumente, die der Clippee-Cluster verwaltet. Somit kümmert sich die Instanz um unbenötigte Objekte, die das System unnötig belasten. Dieser Punkt kann mit dem Ausbau zu einer Client-Anwendung wesentlich verbessert werden. Es ist zu beachten, dass der Ausbau mit viel Aufwand verbunden ist, da das aktuelle System noch keine Ansätze in diese Richtung aufweist.

7.6 Performanz

Im bestehenden System blockieren Update-Nachrichten. Das heisst, vom Absenden einer Update-Nachricht bis zum Empfang der Antwort kann der sendende Thread keine weiteren Arbeiten verrichten. Zudem wird erst eine Antwort auf eine ankommende Update-Nachricht zurückgesendet, wenn die Nachricht vollständig bearbeitet ist. Eine mögliche Verbesserung sorgt dafür, dass eine Update-Nachricht von einem Peer entgegengenommen und

in eine Warteschlange eingereiht wird, wonach direkt eine Antwort zurück gesendet wird. Die Update-Nachricht kann anschliessend von einem anderen Thread bearbeitet werden, ohne dass der Sender der Nachricht lange auf eine Antwort warten muss.

In einem weiteren Schritt werden Änderungen lokal direkt geschrieben und dann global nachgetragen. Schlägt das globale Ändern fehl, muss lokal die Änderung rückgängig gemacht werden.

Durch die Anwendung einer unvollständigen Replikation kann die Netzlast unter Umständen extrem reduziert werden. In diesem Zusammenhang müsste mindestens auf jedem Peer der sichtbare Bereich des Dokumentes repliziert werden. Weiter kann die Netzlast reduziert werden, indem nicht jedes veränderte Zeichen einzeln geschrieben wird, sondern diese in gepufferten Updates zusammengefasst werden.

7.7 Editierfunktionen (Skiplist)

Die Navigation innerhalb des Dokumentes lässt sich verbessern, indem man eine Zeile direkt mittels deren Zeilennummer anspringen kann. Um Zeilen effizient anspringen zu können, ist es sinnvoll, die Datenstruktur zu erweitern und eine Skiplist [12] zu implementieren. Diese könnte zum Beispiel auch für die Verwaltung des Dokumentes herangezogen werden, vergleiche Abschnitt 5.4.

Des Weiteren sind die bereits bestehenden Editierfunktionen durch die gängigen Funktionen Undo und Redo erweiterbar, die das Arbeiten mit einem Dokument erheblich vereinfachen.

7.8 Automatisches Verbinden von Peers

Beim Starten des Systems muss immer angegeben werden, ob ein neuer Cluster erzeugt oder einem bestehenden Cluster beigetreten werden soll. Für den Fall des Beitretens muss die Rechneradresse eines beteiligten Peers bekannt sein, damit die Verbindung zu diesem Peer hergestellt werden kann.

Damit bestehende Peers in einem LAN automatisch gefunden werden, kann ein *Discoveryprotokoll* implementiert werden, welches das Aufstarten der Anwendung vereinfacht.

Mit der Bereitstellung eines ähnlichen *Rendezvous-Peers* wie im JXTA-Projekt [7] wäre es sogar möglich, bestehende Cluster automatisch im Internet zu finden.

Literaturverzeichnis

[1] K. Albrecht, R. Arnold, and R. Wattenhofer. Clippee: A Large-Scale Client/Peer System. Technical Report 410, ETH Zurich, July 2003.

[2] Apple Computer, Inc. Rendevous. http://www.apple.com/macosx/jaguar/rendezvous.html.

[3] James Begole, Mary Beth Rosson, and Clifford A. Shaffer. Supporting worker independence in collaboration transparency. In *ACM Symposium on User Interface Software and Technology*, pages 133–142, 1998.

[4] James Begole, Craig A. Struble, Clifford A. Shaffer, and Randall B. Smith. Transparent Sharing of Java Applets: A Replicated Approach. In *Symposium on User Interface Software and Technology (UIST'97)*, pages 55–64, NY, 1997. ACM Press.

[5] The Coding Monkeys. Hydra. http://hydra.globalse.org/, 2001.

[6] Microsoft Corporation. NetMeeting. http://www.microsoft.com/windows/netmeeting/, 2001.

[7] B. Traversat et al. The Project JXTA Virtual Network. http://www.jxta.org/docs/JXTAprotocols.pdf.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9] Carl Gutwin, Mark Roseman, and Saul Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *Computer Supported Cooperative Work*, pages 258–267, 1996.

[10] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. Structuring distributed applications as fragmented objects. Technical Report INRIA 1404, nelly@sor.inria.fr, anonymous FTP nuri.inria.fr [128.93.1.26], 1992.

[11] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. *Fragmented objects for distributed abstractions*, pages 170–186. IEEE Computer Society Press, 1994.

[12] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.

[13] Chengzheng Sun and Clarence A. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Computer Supported Cooperative Work*, pages 59–68, 1998.

[14] Marteen van Steen, Philip Homburg, and Andrew S. Tanenbaum. The architectural design of Globe: A wide-area distributed system. Technical Report IR-422, Netherlands, 1997.

[15] P. Wilson. *Computer Supported Cooperative Work - An Introduction.* Intellect Books, 1991.

[16] Ali A. Zafer, Clifford A. Shaffer, Roger W. Ehrich, and Manuel Perez. NetEdit: A Collaborative Editor.

Anhang A

Die Benutzerschnittstelle

In diesem Kapitel sind die einzelnen Bildschirme beschrieben, mit denen der Benutzer konfrontiert wird. Einige Screenshots sind exemplarisch für mehrere ähnliche abgebildet.

A.1 Startbildschirm

Der Startbildschirm, wie in Abbildung A.1 dargestellt, erscheint als erste Anzeige beim Aufstarten des Systems. In diesem Dialog werden die folgenden zwei Fälle unterschieden:

- *Erzeugen eines neuen Clusters:* Wenn noch kein anderer Peer aufgestartet wurde, oder ein von bereits bestehenden Clustern unabhängiger Cluster gebildet wird, ist der untere Teil des Dialogs auszufüllen. Im Feld mit der Bezeichnung "loginName" wird der Benutzername eingetragen. Dieser erscheint später in der Benutzerliste. Im Feld mit der Bezeichnung "peerPort" wird der Port eingegeben, auf dem das System auf eingehende Verbindungen reagiert. Es ist darauf zu achten, dass der eingegebene Port noch frei ist.

- *Verbinden mit einem bestehenden Peer:* Existiert bereits ein Peer, und soll sich das System damit verbinden, ist der obere Teil des Dialogs auszufüllen. Die ersten beiden Felder haben dieselbe Bedeutung wie im oben beschriebenen Fall. Im Feld mit der Bezeichnung "ConnectIP" ist der Rechnername des Computers einzugeben, mit dem sich das System verbinden soll. Weiter wird im Feld mit der Bezeichnung "ConnectPort" der Port für die Verbindung angegeben.

Hinter jedem Textfeld ermöglicht eine Checkbox das Abspeichern des eingegebenen Wertes in einer Konfigurationsdatei, so dass die Werte beim nächsten Start bereits eingefügt werden, wie man diese beim letzten Mal abgespeichert hat.

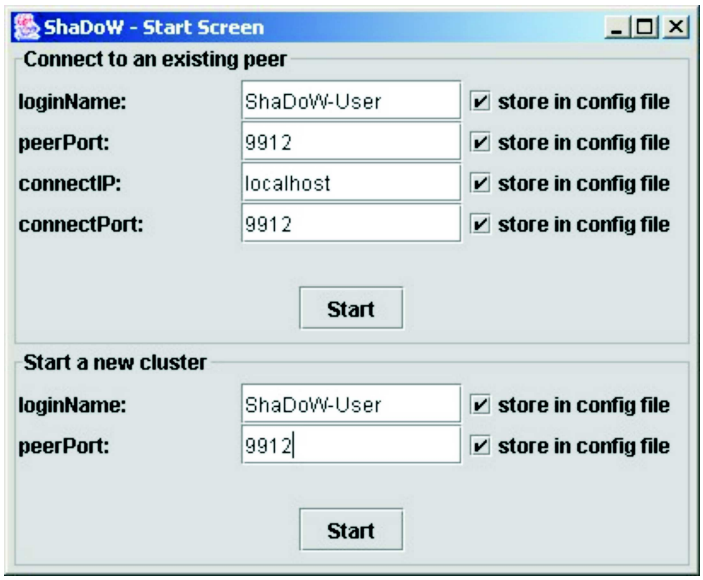

Abbildung A.1: Screenshot des Startbildschirmes

A.2 Arbeitsansicht

Die in Abbildung A.2 dargestellte Oberfläche dient im Normalfall als Benutzerschnittstelle zum Bearbeiten des Dokumentes. Diese besteht aus den im Folgenden erklärten Komponenten.

Textfenster Im Textfenster wird das Dokument wie in einem herkömmlichen Texteditor bearbeitet. Zusätzlich erscheinen Änderungen von entfernten Benutzern farblich kodiert.

Benutzerleiste In der Benutzerleiste werden alle verbundenen Benutzer aufgelistet, inklusive dem Benutzer, der die Anwendung gestartet hat. Die Benutzernamen sind in der Farbe dargestellt, mit der die Eingaben der zugehörigen Benutzer kodiert im Textfenster erscheinen.

Das Klicken auf den Benutzernamen hebt den Block, welchen der zugehörige Benutzer momentan bearbeitet, farbig hervor. Dazu muss allerdings der entsprechende Teil des Dokumentes im Textfenster sichtbar sein.

Beim Positionieren des Cursors über dem Benutzernamen erscheint dessen IP-Adresse sowie der Port, auf welchem die Anwendung auf Eingaben wartet.

Menü Die Menüleiste bildet einen zentralen Punkt beim Bearbeiten eines Dokumentes. In der Menüleiste werden alle verfügbaren Menüs direkt oberhalb des Textfensters aufgelistet. Die Menüeinträge der Menüleiste sind in

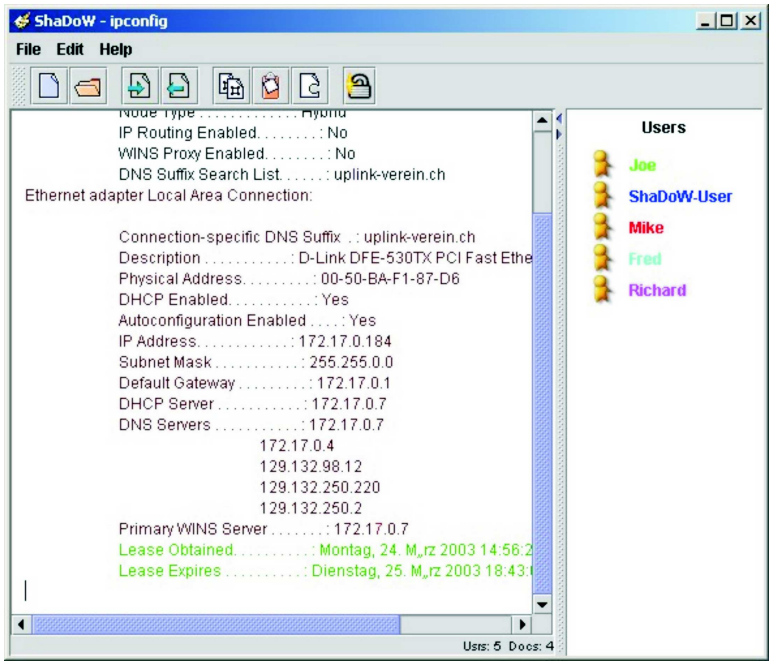

Abbildung A.2: Screenshot während des Bearbeitens eines Dokumentes

den folgenden Abschnitten erklärt.

File-Menü Das File-Menü gruppiert Menüpunkte zum Verwalten der Dokumente sowie zum Beenden der Anwendung. Die Tabelle A.1 erklärt die enthaltenen Untermenüs.

Edit-Menü Das Edit-Menü gruppiert Menüpunkte zum Bearbeiten des geöffneten Dokumentes. Die Tabelle A.2 erklärt die enthaltenen Untermenüs.

Help-Menü Das Help-Menü gruppiert Menüpunkte, die dem Benutzer beim Arbeiten mit dem System hilfreich sind. Da das System ziemlich einfach zu bedienen ist, besteht das Help-Menü nur aus einem Eintrag, der in Tabelle A.3 erklärt wird.

Toolbar Die Toolbar soll das Arbeiten mit dem System vereinfachen. Die Toolbar ist eigenständig, das heisst, sie kann als eigenes Fenster irgendwo auf dem Bildschirm platziert werden. Standardmässig erscheint sie oberhalb des Textfensters und bietet über verschiedene Icons schnellen Zugriff auf ausgewählte Menüpunkte.

Menüpunkt	Erklärung
New Document	Öffnet einen Dialog, um ein neues Dokument anzulegen. Abschnitt A.3 erläutert den Dialog.
Open Document	Öffnet einen Dialog, um ein bestehendes Dokument zu öffnen. Abschnitt A.3 erläutert den Dialog.
Close Document	Schliesst das offene Dokument.
Import Document	Öffnet einen Dialog, um ein neues Dokument aus einer Textdatei zu erstellen. Abschnitt A.4 erläutert den Dialog.
Export Document	Öffnet einen Dialog, um das offene Dokument in einer Textdatei abzulegen. Abschnitt A.4 erläutert den Dialog.
Exit	Beendet die Anwendung. Es ist darauf zu achten, dass alle Dokumente verloren gehen, wenn der letzte Peer beendet wird. Zur persistenten Sicherung können Dokumente in Textdateien exportiert werden.

Tabelle A.1: Menüpunkte des File-Menüs

Menüpunkt	Erklärung
Copy	Kopiert den selektierten Text in die Zwischenablage.
Paste	Kopiert den Inhalt der Zwischenablage an die Stelle des Carets.
Cut	Löscht den selektierten Text, nachdem dieser in die Zwischenablage kopiert wurde.

Tabelle A.2: Menüpunkte des Edit-Menüs

Menüpunkt	Erklärung
About	Zeigt einen Dialog mit Informationen über den Autor und das System.

Tabelle A.3: Menüpunkte des Help-Menüs

Statusbar Die Statusbar liefert Informationen über den Zustand des Systems und ist unterhalb des Textfensters platziert. Der rechte Bereich der Statusbar gibt Aufschluss über die Anzahl im System verfügbarer Dokumente sowie die Anzahl verbundener Benutzer. Im linken Bereich erhält der Benutzer Informationen, wie er mit dem System interagieren kann.

A.3 Erstellen eines neuen Dokumentes

Der in Abbildung A.3 dargestellte Dialog dient dem Erstellen eines neuen Dokumentes. Der obere Bereich, der mit "Available Documents" überschrieben ist, listet alle Dokumente auf, die bereits im System verfügbar sind. Im unteren Bereich kann im Textfeld mit der Bezeichnung "Document Name" der Name des neu zu erstellenden Dokumentes eingegeben werden. Dabei ist darauf zu achten, dass der Name eines Dokumentes eindeutig sein muss. Das heisst, es darf noch kein Dokument existieren, das den eingetragenen Text bereits als Namen trägt. Wird dennoch versucht, einem neuen Dokument denselben Namen wie einem bereits existierenden Dokument zu geben, so reagiert das System mit dem nochmaligen Anzeigen des Dialogs und der Fehlermeldung "Document already exists" im oberen Bereich des Dialogs. Das neue Dokument wird erstellt, wenn die Schaltfläche mit der Bezeichnung "New" betätigt wird. Mittels der Schaltfläche mit dem Symbol "X" in der oberen rechten Ecke des Dialogs kann dieser geschlossen werden, ohne dass ein Dokument angelegt wird.

Oben wird exemplarisch beschrieben, wie ein neues Dokument angelegt wird. Ganz ähnlich geht der Benutzer vor, um ein bestehendes Dokument zu öffnen. Über den Menüeintrag "Open Document" aus dem Menü "File" gelangt der Benutzer zum Dialog fürs Öffnen eines bestehenden Dokumentes. Der einzige Unterschied des Dialogs zu demjenigen in Abbildung A.3 besteht darin, dass die Schaltfläche zum Öffnen des Dokumentes mit "Open" bezeichnet ist. Das Anklicken eines Dokumentes in der angezeigten Liste kopiert dessen Name in das Textfeld für den Dokumentennamen. Wird die Schaltfläche mit der Bezeichnung "Open" betätigt, versucht das System, das Dokument mit dem eingegebenen Namen zu öffnen. Existiert dieses nicht, erscheint der Dialog von neuem mit der Fehlermeldung "Document does not exist". Ist das Dokument vorhanden, wird dessen Inhalt im Textfenster sichtbar, und der Benutzer kann mit dem Bearbeiten des Dokumentes beginnen.

A.4 Exportieren eines Dokumentes

Über den Menüeintrag "Export Document" im Menü "File" gelangt der Benutzer zum in Abbildung A.4 gezeigten Dialog, der zum Exportieren eines Dokumentes dient. Der Benutzer kann eine Datei bestimmen, in welche das

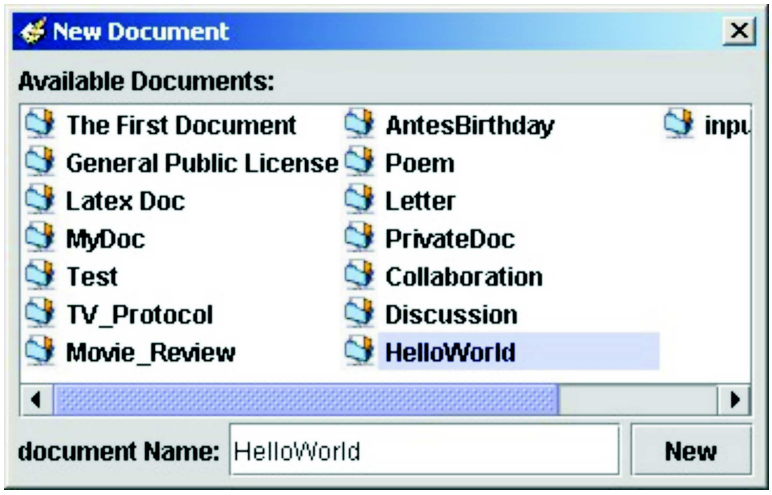

Abbildung A.3: Screenshot des Dialogs zum Erzeugen eines Dokumentes

geöffnete Dokument exportiert werden soll. Durch Betätigen der Schaltfläche mit der Bezeichnung "Export" wird das Exportieren eingeleitet, und der Erfolg der Aktion wird dem Benutzer über die Statusbar mitgeteilt.

Das Importieren eines Dokumentes aus einer Textdatei läuft ganz ähnlich ab. Der erscheinende Dialog ist analog zu dem in Abbildung A.4, nur dass jetzt die ausgewählte Datei bereits existieren muss. Der Inhalt der gewählten Datei wird in einem neuen Dokument, das den Namen der Datei trägt, geöffnet. Ist der Name bereits vergeben, muss die Datei temporär umbenannt werden, damit das Dokument importiert werden kann.

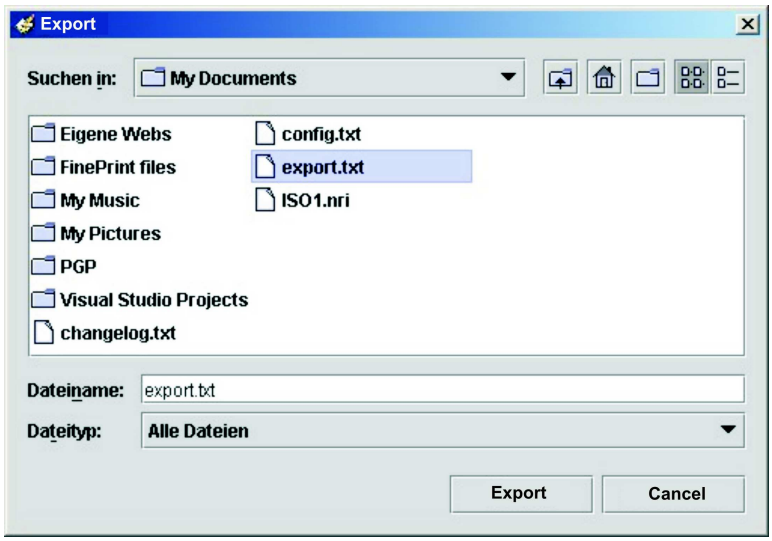

Abbildung A.4: Screenshot des Dialogs zum Exportieren eines Dokumentes

A.5 Keyboard Shortcuts

Die Tabelle A.4 listet die speziellen Tasten der Tastatur auf und was der Benutzer mit ihnen bewirkt.

Shortcut	Wirkung
ctrl-c	copy
ctrl-v	paste
ctrl-x	cut
Delete	Löscht das Zeichen nach dem Cursor. Ist Text selektiert, wird dieser gelöscht.
Backspace	Löscht das Zeichen vor dem Cursor. Ist Text selektiert, wird dieser gelöscht.
Pfeil aufwärts	Bewegt den Cursor um eine Zeile aufwärts.
Pfeil abwärts	Bewegt den Cursor um eine Zeile abwärts.
Pfeil links	Bewegt den Cursor um eine Stelle nach links.
Pfeil rechts	Bewegt den Cursor um eine Stelle nach rechts.
Pfeil Home	Bewegt den Cursor an den Anfang der Zeile.
Pfeil End	Bewegt den Cursor ans Ende der Zeile.
Pfeil PgUp	Bewegt den Cursor um einen Bildschirm aufwärts.
Pfeil PgDn	Bewegt den Cursor um einen Bildschirm abwärts.

Tabelle A.4: Keyboard Shortcuts

Anhang B

Konfiguration

Die Anwendung lässt sich zentral über die Konfigurationsdatei einstellen. Die Tabelle B.1 listet alle Schlüssel und die dafür möglichen Werte auf und gibt dazu eine Erklärung. Im anschliessenden Listing ist exemplarisch ein Beispiel einer möglichen Konfigurationsdatei abgebildet.

Tabelle B.1: Schlüssel und Werte für die Konfigurationsdatei

Schlüssel	Wert	Erklärung
`MIN_BLOCKSIZE`	Integer aus dem Intervall [0, `MAX_BLOCKSIZE`)	Definiert die minimale Anzahl von Zeichen, die in einem Knoten der verketteten Liste gespeichert werden.
`MAX_BLOCKSIZE`	Integer aus dem Intervall (`MIN_BLOCKSIZE`, 4294967295]	Definiert die maximale Anzahl von Zeichen, die in einem Knoten der verketteten Liste gespeichert werden.
`LOGIN_NAME`	String	Definiert den Namen des Benutzers, erscheint in der Benutzerleiste, s. Abschnitt A.2.
`PEER_PORT`	Integer aus dem Intervall [0, 65535]	Definiert den Port, auf dem das System auf eingehende Verbindungen reagiert.
Fortsetzung auf der nächsten Seite...		

Tabelle B.1: Fortsetzung Schlüssel und Werte

Schlüssel	*Wert*	*Erklärung*
`CONNECT_IP`	gültige Rechneradresse	Definiert die Rechneradresse für die Verbindung auf einen anderen Peer.
`CONNECT_PORT`	Integer aus dem Intervall [0, 65535]	Definiert den Port für die Verbindung auf einen anderen Peer.
`LIST_NODE_ MERGER_PERIOD`	Integer	Definiert den Zeitabstand in Millisekunden zwischen zwei Durchläufen des List-Node-Mergers, vgl. Abschnitt 4.3.4. Ist die Periode auf 0 gesetzt, wird der Merger ausgeschaltet.
`TEXT_COLOR_ FADER_MODE`	Integer aus dem Intervall [0, 2]	0 schaltet den Text-Color-Fader aus. 1 färbt das ganze Dokument schwarz und 2 dekrementiert die Farbe einzelner Zeichen.
`TEXT_COLOR_ FADER_PERIOD`	Integer	Definiert den Zeitabstand in Millisekunden zwischen zwei Durchläufen des Text-Color-Faders.
Fortsetzung auf der nächsten Seite...		

Tabelle B.1: Fortsetzung Schlüssel und Werte

Schlüssel	*Wert*	*Erklärung*
`TEXT_COLOR_ FADER_DECREMENT`	Integer aus dem Intervall [0, 255]	Definiert um wieviel der Farbwert der Schriftfarbe bei einem Durchlauf des Text-Color-Faders heruntergesetzt wird. Die Dauer, bis der Text völlig schwarz wird ergibt sich aus dem Dekrement und der Periode des Faders. (Nur im Modus 2)
`HIGHLIGHT_ TIME`	Integer	Definiert in Millisekunden, wie lange ein gelockter Block markiert wird, wenn ein Benutzername doppelt angeklickt wird.
`LOCKS_RELEASER_ PERIOD`	Integer	Definiert die Zeit in Millisekunden, nach der alle Locks, die der Peer aktuell hält, freigegeben werden. Beträgt der Wert 0, werden die Locks nicht automatisch freigegeben.

Listing der Konfigurationsdatei

```
#============================================================================
# ShaDoW: A Shared Document Writing System
# Configuration File
#
# Diploma Work by Martin Meier, me1er@gmx.ch
# 23.08.2003
#============================================================================


#
# Global configurations for all peers (only effective on the first peer)
# ----------------------------------------------------------------------------

#
# Minimal size of a text block of a list node
# Should be an integer within the interval [0, MAX_BLOCKSIZE)
#
MIN_BLOCK_SIZE = 5

#
# Maximal size of a text block of a list node
# Should be an integer within the interval (MIN_BLOCK_SIZE, 4294967295]
#
MAX_BLOCK_SIZE = 20
```

```
#
# Local configurations for each peer itself
# --------------------------------------------------------------------------

#
# Name for the login. Will appear in the GUI.
#
LOGIN_NAME = ShaDoW-User

#
# The port this peer listen to other peers.
#
PEER_PORT = 9912

#
# Startup. Connection to another peer.
#
CONNECT_IP = 129.132.130.173
CONNECT_PORT = 9912

#
# Period for the list node merger background process in millis.
# If 0 or less, the merger will be disabled.
#
```

```
LIST_NODE_MERGER_PERIOD = 300000

#
# Text color fading enabled.
#   0: disabled
#   1: change the color of the whole document
#   2: change the color per character
#
TEXT_COLOR_FADING_MODE = 2

#
# Period for the text color fader in millis.
#
TEXT_COLOR_FADER_PERIOD = 10000

#
# Decrement for the text color fader.
# Each RGB of the text color will be decremented by the given decrement
# in one period of the text color fader.
# Integer: [0,255]
#
TEXT_COLOR_FADER_DECREMENT = 50

#
# Determines how long the locked node will be highlighted.
# Time in milli seconds.
```

```
#
HIGHLIGHT_TIME = 1000

#
# Specifies the period of the locks releaser.
# If the last change in the document is the specified time ago,
# the system releases all locks hold by this peer.
#
# Time in milli seconds.
# 0: locksReleaser is disabled;
#
LOCKS_RELEASER_PERIOD = 50000
```

Anhang C

Aufruf und Debugging

Dieses Kapitel beschreibt den Aufruf des Systems sowie die erweiterte Funktionalität der Anwendung, die insbesondere fürs Debugging gedacht ist.

C.1 Starten der Anwendung

Die Anwendung lässt sich auf verschiedene Arten starten. Die gewöhnliche Methode ist das Starten der Java-Anwendung ohne Argumente:

```
java -jar shadow.jar
```

Weiter besteht die Möglichkeit, den Startbildschirm zu umgehen und die benötigten Parameter direkt auf der Kommandozeile zu übergeben. Mit der Angabe eines weiteren Argumentes wird ein Standarddokument automatisch geöffnet. Die einzelnen Argumente werden in Tabelle C.1 erklärt.

```
java -jar shadow.jar [
        <peerPort>
        [<connectIP> <connectPort>]
        <loginName>
        [loadDirect]
    ]
```

Unter Windows besteht die Möglichkeit, die Anwendung durch einen Doppelklick auf das File “shadow.jar” zu starten. Dies entspricht dem Aufruf ohne Argumente.

C.2 Konsoleneingaben

Wurde das System von der Konsole aus gestartet, nimmt das System Eingaben über die Konsole entgegen. Diese Möglichkeit ist vor allem zum Debuggen sinnvoll. Dabei wird ein Zeichen gefolgt von `ENTER` eingegeben. Tabelle C.2 listet mögliche Zeichen für die Eingabe und deren Auswirkungen auf.

Argument	Bedeutung
peerPort	Definiert den Port, auf dem das System auf eingehende Verbindungen reagiert.
connectIP	Definiert die IP-Adresse für die Verbindung auf einen anderen Peer. Wird dieses Argument angegeben, muss auch das Argument "connectPort" angegeben werden und der Peer wird in den bereits bestehenden Cluster aufgenommen.
connectPort	Definiert den Port für die Verbindung auf einen anderen Peer. Dieses Argument kann nur angegeben werden, wenn auch das Argument "connectIP" angegeben wird.
loginName	Definiert den Namen, mit dem sich der Benutzer beim Cluster anmeldet. Dieser Name wird in der Benutzerleiste sichtbar sein, siehe Abschnitt A.2.
loadDirect	Wird dieses Argument angegeben, lädt das System beim Aufstarten direkt ein Standarddokument.

Tabelle C.1: Argumente für den Aufruf von ShaDoW

Eingabezeichen	Auswirkung
c	Beendet die Anwendung.
f	Listet den Inhalt des Dokumentenordners auf; die Namen aller Dokumente im System werden ausgegeben.
m	Der List-Node-Merger wird angestossen; dieser führt einen Durchlauf aus, für die Beschreibung des Mergers siehe Abschnitt 4.3.4. Dabei spielt es keine Rolle, ob der Merger im System ein- oder ausgeschaltet ist, vergleiche Kapitel B.
n	Der Knoten, den der Linked-List-Manager als Schreibbereich für diesen Peer verwaltet, wird ausgegeben.
p	Die Struktur `VisibleNodes` wird ausgegeben.

Tabelle C.2: Konsoleneingaben