NetEdit: A Collaborative Editor

Ali Asghar Zafer

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science

April 23, 2001
Blacksburg, Virginia

Keywords: Computer-Supported Cooperative Work, Collaboration, Editor, Replication

NetEdit: A Collaborative Editor

Ali Asghar Zafer

(ABSTRACT)

Centralized systems are easier to build and maintain as compared to completely distributed systems. However, distributed systems have the potential to be responsive and robust relative to centralized systems. This thesis proposes an architecture and concurrency algorithm for collaborative editing that lies between these extremes and preserves the advantages of both approaches while minimizing their shortcomings

The Jupiter collaboration system at Xerox PARC uses a 2-party synchronization protocol for maintaining consistency between two users performing unconstrained edits to the document simultaneously. The primary goal of our work has been to extend this 2-party synchronization protocol to an n-way synchronization algorithm. NetEdit is a prototype collaborative editor built to demonstrate this n-way protocol. It uses a replicated architecture with the processing and data distributed across all the clients and the server. Due to replication, the response time of the local edits performed by the users is quite close to a single user editor. The clients do not need to be aware of other clients in the system since each of them synchronizes with their counterpart at the server. All communication regarding editing operations takes place through this server. As a result this system is quite scalable (linear growth) relative to distributed systems (quadratic growth) in terms of number of communication paths required as the number of clients grow. I discuss the details of this extension and illustrate it through an editing scenario.

NetEdit uses groupware widgets (telepointers, and radarview) to distribute awareness information between participants. It supports completely unconstrained editing and allows late joining into a session. It does not assume any structure in terms of roles of participants or protocol for collaboration and thus allow users to form whatever protocol suits them. The results and conclusions derived from a preliminary usability study of NetEdit, discuss its efficacy. They also investigate the role of communication and its use in a groupware setting.

Acknowledgements

I am profoundly grateful to my advisor, Professor Clifford A. Shaffer, and co-adviser Professor Roger W. Ehrich for their support, direction and ideas throughout this project. I thank Dr. James Begole, of Sun Microsystems, for his initial help in defining the topic and later in helping to understand some of his code that was reused in my system. I am indebted to Dr Manuel A. Pérez-Quiñones, for providing background, focus and direction during the usability study of this system. I would also like to thank Natasha Dannenberg for her help in conducting and discussing conclusions derived from the usability experiments. I thank Tejas Patel, Jason Zwolak, Faria Khalid, and Pete Schoenhoff for taking notes during the usability experiment.

Finally, I express my deepest gratitude to my parents and grandparents for their unceasing moral support through the course of my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We have been through various phases of Information Technology from centralized mainframe computing to desktop systems and more recently distributed computing. With the explosive growth of Internet and pervasive use of computers, we need to think about shifting from personal computing to inter-personal computing. Since people perform a lot of tasks working together, and because a computer is used in much of their work, it is interesting to explore ways to use computers to allow this collaboration. With organizations getting global and the cost of travel for face-to-face collaboration getting higher and time consuming, it is becoming more appealing to research computer based collaboration.

Collaboration can range from asynchronous, where an interactive activity is separated by long periods of time (e.g. e-mail, discussion groups), to synchronous, where an interactive activity is simultaneous and in real-time (e.g. video conferencing). Synchronous collaborative systems are not as common as asynchronous systems. One of the most widely used synchronous collaborative technologies is telephony, where two or more people can remotely talk to each other in real-time. Computer based synchronous collaborative systems like shared whiteboards, collaborative editor etc. are only beginning to emerge recently.

This thesis describes the design, implementation and usability study of a **collaborative editor** that allows users to remotely edit a document simultaneously. It describes various architectures and design issues encountered while developing this system. This work addresses both technical issues related to the development and human factors related to the use of this system. The remainder of this chapter talks about the scope of this thesis and a summary of the organization of later chapters.

1.1 Problem Description

This work describes NetEdit, the collaborative editor, technical issues in its design and implementation, and its usability study. NetEdit allows two or more users to remotely edit a document simultaneously. The editing is completely unconstrained and users can insert and delete characters at any location. In fact, two or more users could be performing insertions and deletions at exactly the same position.

In addition to maintaining consistency of state for all users in the session, we also wish to preserve the intentions of the participants. Preserving the intention of a user is an important issue and is discussed in detail in section 4.2.3. Consistency of the state for all users can be achieved by enforcing a consistent global order for the operations, but preserving intention requires more complex processing and transformations. To achieve good response time between when the user types characters and when they get reflected in the document, we replicate the document at all the user sites. This replication introduced further complications that we had to take in to account.

The usability study of this system attempted to determine the usefulness of various group widgets like radarview, telecaret etc., during collaborative editing. These widgets help give awareness information to the user about remote participants like what part of the document they are editing, etc. Since this system does not assume any social structure during editing, the users in the session can form and follow whatever protocol suits them. We also attempted to determine the usefulness of a chat utility for communication between remote participants.

1.2 Organization

We start by discussing related systems and research done in Chapter 2. This chapter also compares some research prototypes on various parameters. Chapter 3 describes the high level architecture of the system. It discusses the interaction between its various components. The core algorithm that makes the system do its task is the topic of Chapter 4. The usability study of the system is discussed in Chapter 5. Finally, Chapter 6 draws conclusions from the work described and presents future research ideas.

Chapter 2

Background Research

This chapter reviews the current state of the art in collaborative editing and establishes the context for this thesis. It begins with a classification of collaborative systems. Various research prototypes that fall within this classification are described next. We then survey groupware widgets and conclude with a discussion on groupware architectures.

2.1 Collaborative editing systems

Collaborative systems involve several people working together, sometimes in the same place, sometimes from a distance, sometimes at the same time and sometimes at different times. Thus we can have the following four categories[37]: same time/same place, same time/different place, different time/same place and different time/different place. There exist systems in each of these categories as follows:

Same time/Different place – application sharing, teleconferencing etc.

Different time/Different place – email, voice-mail, usenet news, fax etc.

Same time/Same place – electronic whiteboards, electronic meeting-rooms etc.

Different time/Same place – desktop computers, group rooms, blackboards etc.

This thesis focuses on the “Same time/Different place” category of application. Applications in this category are sometimes referred to as synchronous applications. As opposed to this, applications in the “Different time/Different place” category are referred to as asynchronous applications. Let me first define a few terms and then discuss some of the existing systems in the first two categories.

2.1.1 Definitions

- **Document**: This is a text file that a group of one or more users edit.

- **Session**: A group of users editing the same document form a session. Thus a session consists of a file being edited and the users involved in the editing. A user can participate in more than one session at a time.

- **Having the floor** [15]: In some systems, not all users can do edits simultaneously. A user who wishes to edit must get a handle for the document, which signifies that while he/she has this handle, no one except himself/herself can edit the document. This handle gives exclusive access to the document for performing editing operations. Getting the handle is often called “having the floor”. One could get a handle, for example, by pressing a button or running some script. It depends on how the system is implemented.

- **Workspace Awareness** [24]: Workspace awareness is the information about the activity of the remote participants in the session.

- **Task Coupling** [10]: The degree to which the participants in a session work closely together categorizes them into tight coupling and loose coupling. If they are working independently on different parts of the document, then this is referred to as loose coupling. On the other hand, if they are working closely, like say brainstorming a paragraph, then their activity is tightly coupled.

- **Local/Remote Operation**: An operation is local if it is generated by the local client due to activities like typing, or a paste operation by the local user. On the other hand, an operation is said to be remote if it is generated by the remote participant and notified to the local client through some communication mechanism.

2.1.2 Asynchronous Systems

In asynchronous systems, the work is usually done in a sequential manner with versions of the document passed between group members. This requires support for communication of comments and changes within the document.

Quilt [28,12] – Quilt is an asynchronous collaborative authoring system developed at Bellcore. It forms a superstructure that manages the collaborative aspects of group authoring like coordination and information sharing. The primary mechanisms available to its users are a hypermedia system representing the document along with annotations, audit trail recording, and an integrated electronic mail and conferencing system. Annotation provides a mechanism by which users can comment to each other about the document material. Audit trail recording allows collaborators to review each other’s activities. Electronic mail and conferencing provides the communication mechanism between users. Here users have different roles in the production of the document, thereby reducing any uncertainty about the character of each other’s changes.

PREP [9] – PREP is another asynchronous collaborative writing tool, which organizes text in a columnar method. In this, successive versions of the same paragraph are arranged horizontally in columns. New text is arranged vertically in the same column. Like Quilt, PREP uses roles to delimit areas of responsibility and to control access to the document. It has explicit annotation and structured or directed messaging to provide a means for generating awareness and coordination information.

2.1.3 Synchronous Systems

Some of the challenges [8, 15, 40] faced in developing synchronous systems are as follows:

Minimize response time: The response time is the time between when the user types characters and he gets an acknowledgement of his update. The developer must strive to bring response time associated with the local actions and latency for remote actions to a minimum.

Flexible edits: Ideally, the users in the session will be given complete flexibility in where they can edit in the document to facilitate natural cooperative editing. However, the technical challenge then is to manage multiple streams of concurrent activities so that system consistency is preserved. Having a consistent global ordering of operations could help us achieve "flexible editing", but the system might not have good responsiveness. This is because each operation (say insertion of character 'A') would require round trip network latency. These operations need to be acknowledged by the coordinating server. NetMeeting[29], made by Microsoft Corporation, when used for collaboration on a Microsoft Word document allows only one person to edit at a time. This person must have the floor to do the editing. It maintains a lock at the entire document level. Thus NetMeeting is at the other extreme of our ideal for providing flexible editing.

Collaborator Awareness: This can be defined as the knowledge of the state or actions of other participants. The level of awareness that one may desire depends on the degree of engagement. Hence, depending on how focused the activity is, collaboration may vary from general awareness (where people know roughly what others are doing) to high awareness (where people work together closely). The system must be able to support both of these types of group dynamics.

Session Control: A collaborative editor should allow users to create, join, or leave editing sessions at arbitrary times. It must also ensure that each user in the session, whether he joined late or not, has the same version or view of the document.

Communication: It is essential to provide some kind of communication mechanism among individuals who are collaborating on the document. They might want to communicate about the object of their collaboration, exchange questions, revisions and acceptances, and discuss courses of action and process plans to achieve their goal.

Undo facility [35]: Since collaborative writing is an interactive activity, support for undoing changes might be provided. However, this gets tricky when we realize that there are two types of “Undo”. First, global group undo applies undo to the last operation executed, no matter who was the initiator. Secondly, local group undo affects only the operations performed by the local user. With global group undo, the operation to be executed only depends on the operation to be undone, e.g., being its inverse operation. In the case of local undo, it additionally depends on remote operations that have been executed after the last local operation.

Let us review some of the existing collaborative editing systems and how close they are to implementing the above said requirements.

ShrEdit [12]: This is a synchronous multi-user text editor, developed at University of Michigan that runs on a network of Macintosh systems. There are three types of windows in the system – Shared window, Private window and Control window. A user can have a number of these windows open at any time. Shared windows contain documents that can be edited by multiple users while private windows contain documents that only one user can see and edit. ShrEdit uses a locking mechanism at the level of text selection for concurrency control. That is the user can select certain text and lock it. Also it is not permitted for two users to have their edit cursors at the same offset in the document. It does not have telepointers, but cursor “collisions”, are indicated by an audio signal and a pop-up window.

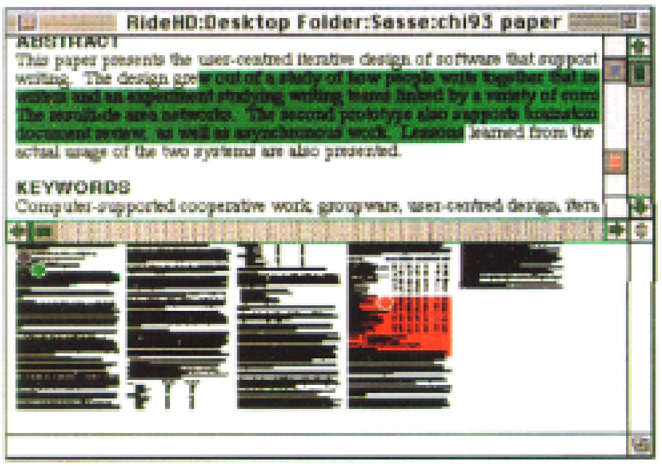
The control window shows the names of the participants in the editing session. Users can track other users i.e. see other user’s view of the document along with his edit cursor and locking areas. ShrEdit does not impose any structure upon the user’s activities. It provides equal access to all participants. However, the awareness support is quite rudimentary.

GROVE [15]: This is a text editor designed for use by a group of people simultaneously editing an outline document (tree-structured document that may be viewed at various levels of specificity) during a work session. Within a GROVE session, each user can see and manipulate one or more views of the text being worked on. There are three categories of views – private view, shared view, and public view. A private view contains elements that only a particular user can read. A shared view contains items that only an enumerated set of users can read. A public view contains items that everyone can read. The shared view window shows the images or names of the participants who are allowed and currently editing the document. Participants can enter and leave the session at any time. When a user enters or re-enters a session, he receives an up-to-date document, unless he chooses to retrieve a previously stored version.

Grove is designed to encourage and assist in tightly coupled collaboration. The default is a mode where every one can see and edit everything, and there is no locking while editing. They claim that after a learning period, it is not chaotic to do the editing simultaneously because a social protocol acts as a mediator.

SASSE [1]: This is a second prototype after SASE, which was an interactive synchronous writing tool. This version also supports asynchronous editing – an annotation mechanism is provided that allows authors to exchange notes and comments. It uses a replicated architecture with the shared document residing on each collaborator's workstation. It has a centralized communication server on Unix box that communicates with the clients using TCP/IP protocol. The clients are Macintosh workstations.

SASSE assumes that users will communicate via telephone or some external audio/video connection. It uses telepointers (described in section 2.2.1), color-coded text selections, multi-user scroll bars (described in section 2.2.2) and two kinds of views for maintaining collaborator awareness. There are two vertical scroll bars – one is the normal scroll bar of the local user and the other has multiple color codes that correspond to collaborators participating in an editing session. To avoid conflicting changes, SASSE locks text at the user text selection level. One of the views supported by SASSE is gestalt view (see Figure 2.1). It presents a condensed image of the entire document along with collaborator's positions and text selections. The observation view (see Figure 2.2) allows users to "look over the shoulder" of another user and see exactly what he is seeing and doing. Its like tracking the other participant.



ABSTRACT

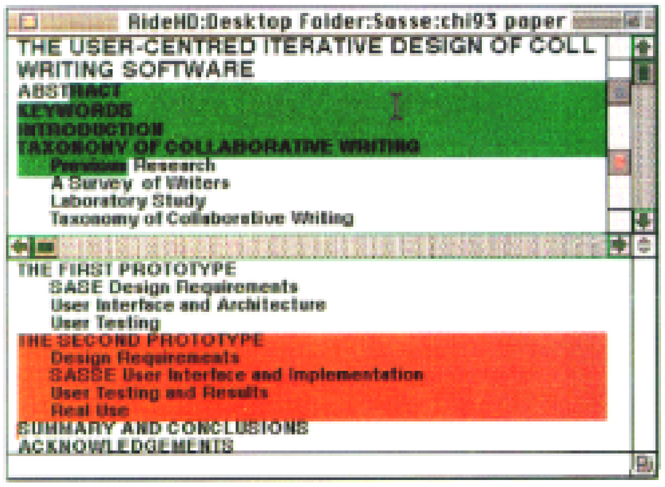
This paper presents the user-centred iterative design of software that support writing. The design grew out of a study of how people write together that in [illegible] and an experiment studying writing teams linked by a variety of comm [illegible] The worldwide area networks. The second prototype also supports [illegible] document review, as well as asynchronous work. Lessons learned from the actual usage of the two systems are also presented.

KEYWORDS

Computer-supported cooperative work, groupware, user-centred design, itera

Baecker, et al., Colour Plate 2: SASSE's normal text display for the green user appears in the upper half of the screen. A gestalt view of the entire document appears in the lower half of the screen. There are 3 users (green, gray, and red) working concurrently on the document. The red user is working midway through the document and has selected a large region of text.

Figure 2.1: Gestalt view in SASSE



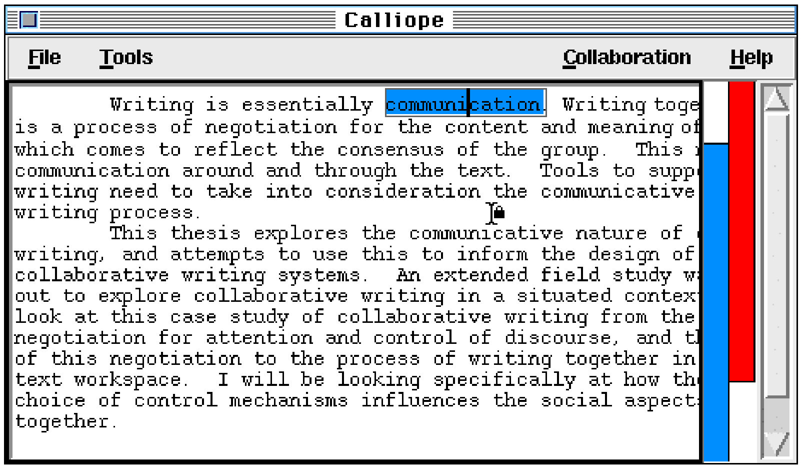
RideHD:Desktop Folder:Sasse:chi93 paper

THE USER-CENTRED ITERATIVE DESIGN OF COLL
WRITING SOFTWARE
ABSTRACT
KEYWORDS
INTRODUCTION
TAXONOMY OF COLLABORATIVE WRITING
Previous Research
A Survey of Writers
Laboratory Study
Taxonomy of Collaborative Writing
THE FIRST PROTOTYPE
SASE Design Requirements
User Interface and Architecture
User Testing
THE SECOND PROTOTYPE
Design Requirements
SASSE User Interface and Implementation
User Testing and Results
Real Use
SUMMARY AND CONCLUSIONS
ACKNOWLEDGEMENTS

Baecker, et al., Colour Plate I: SASSE is in outlining mode. The green user is working at the beginning of the document and her view appears in the upper half of the screen. An observation view of the red user appears in the lower half of the screen.

Figure 2.2: Outline view in SASSE

Calliope [30]: This is a shared editor (see Figure 2.3), developed at University of Toronto, Canada. It provides a main shared text workspace and a number of tools to enhance collaborator awareness. It uses user-selectable locking to ensure a consistent view of text at each user's workstation. That is the user can select text and optionally lock it to prevent other users from simultaneously editing it. It has facility for adding publicly available annotations that can serve as comments on specific text in the document.



```
        Writing is essentially communication. Writing toge
is a process of negotiation for the content and meaning of
which comes to reflect the consensus of the group.  This i
communication around and through the text.  Tools to supp
writing need to take into consideration the communicative
writing process.
        This thesis explores the communicative nature of 
writing, and attempts to use this to inform the design of
collaborative writing systems.  An extended field study w
out to explore collaborative writing in a situated contex
look at this case study of collaborative writing from the
negotiation for attention and control of discourse, and t
of this negotiation to the process of writing together in
text workspace.  I will be looking specifically at how th
choice of control mechanisms influences the social aspect
together.
```

Figure 2.3: Calliope Text Editor

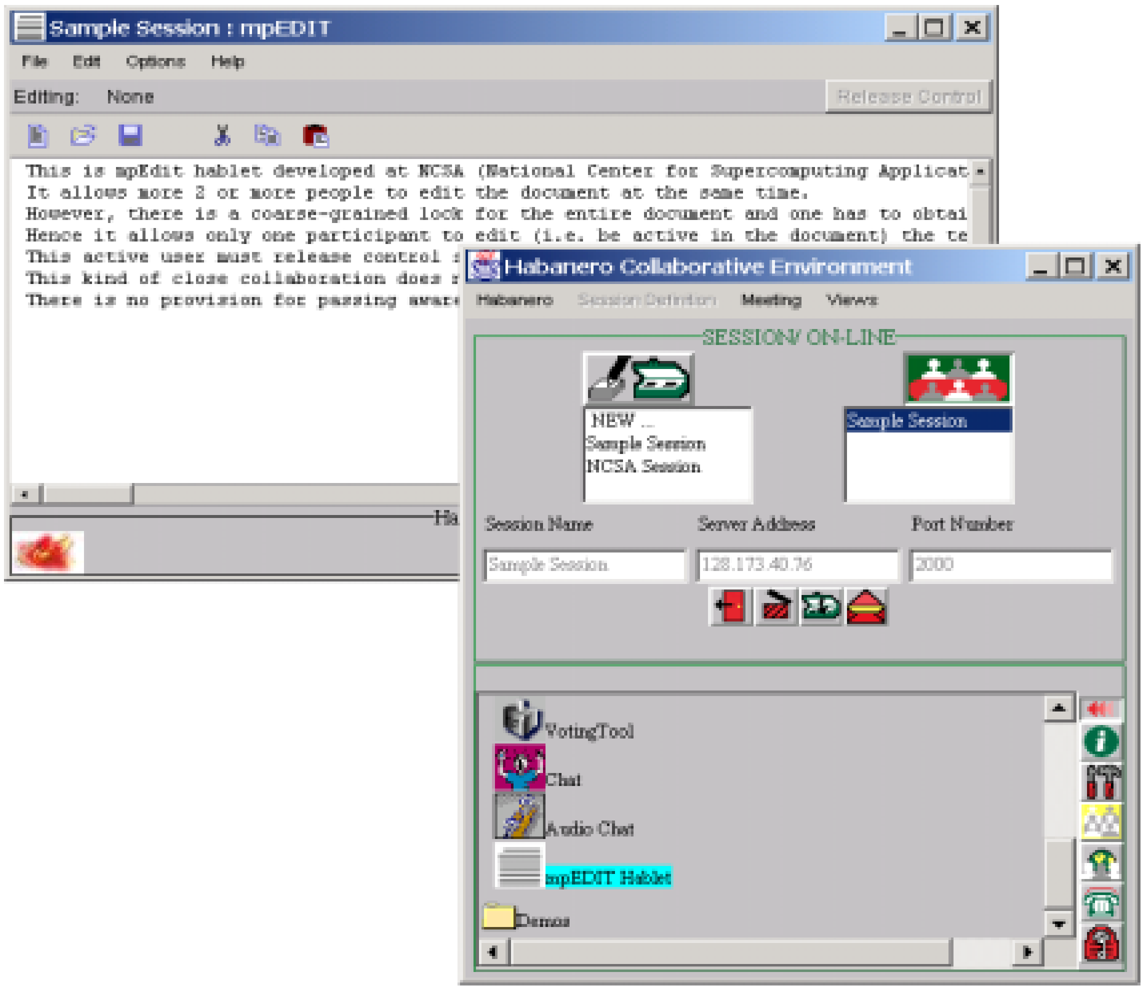
Calliope uses color-coded selections, telepointers (described in section 2.2.1), gestalt view and shared scrollbars (described in section 2.2.2) to provide awareness of the location and kind of actions being performed by the other collaborators. The color-coded scrollbars for each remote user give his approximate location or view in the document. Telepointers give more detailed information about remote participant’s position.

NetMeeting [3, 29]: This is a collaboration transparency system from Microsoft Corporation. In a collaboration transparency system [4], multiple users simultaneously collaborate with each other using a single user application. The conversion from a single user system to multi-user system takes place dynamically only for that invocation. NetMeeting only allows tightly coupled collaboration among participants. When NetMeeting is used to collaborate on a Microsoft Word document, only one person can edit at a time. The granularity of locking is the entire document and a user should obtain the floor before performing his edits. Since it supports a tightly coupled collaboration, it does not have groupware widgets like telepointer, radarview etc. Telepointers, and radarview are discussed in detail in section 2.2.

Flexible JAMM (Java Applets Made Multi-user) [3, 4, 5]: This is another collaboration transparency system that dynamically replaces single user objects with collaboration-aware versions. Flexible JAMM was used to dynamically convert Notepad, a java based single user editor, to allow multiple users to collaborate on the document. The converted system allowed participants to simultaneously edit different parts of the document. It provided telepointers, and a radarview, so that a user is aware of the activities of the remote participants. Telepointers and radarview are discussed in detail in section 2.2.

Habanero [6]: This was developed at NCSA (National Center for Supercomputing Applications) and is a collaborative framework that allows development of collaborative applications. It also has a number of collaborative applications (called hablets) like mpEdit, shared whiteboard, audio chat, weather visualizer etc.

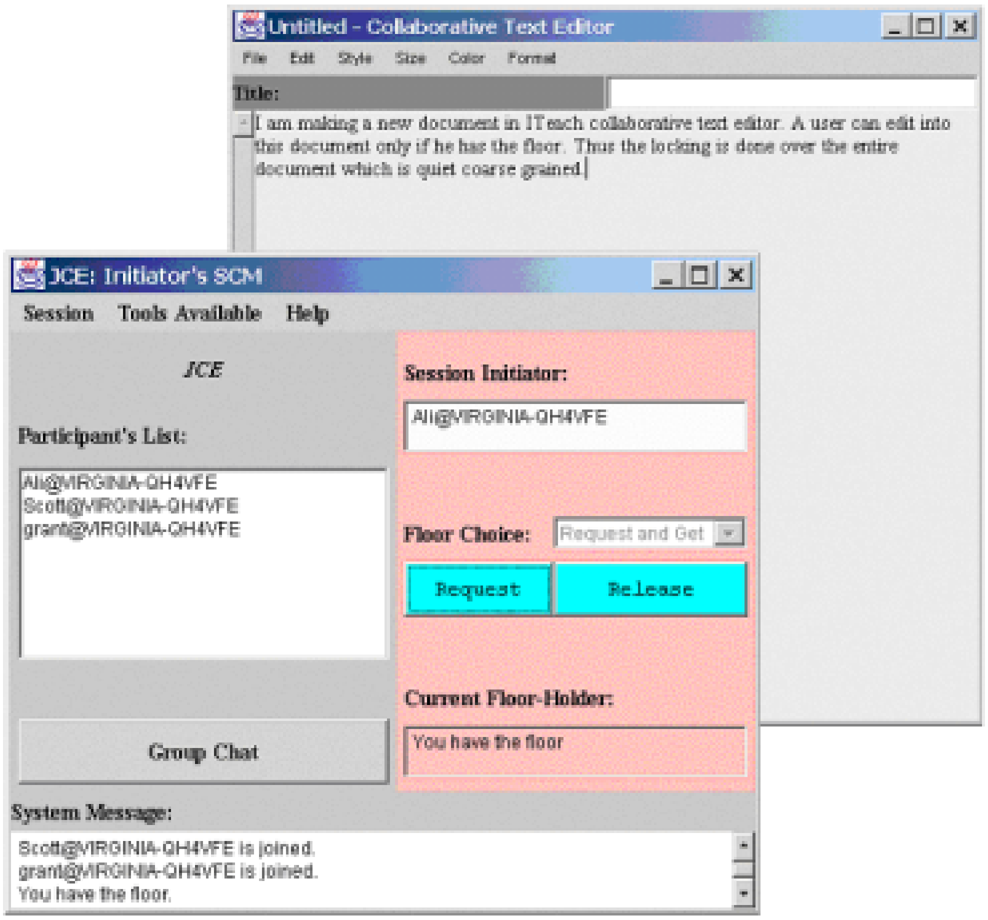
mpEdit (Figure 2.4) uses a centralized approach and tightly coupled collaboration. There is a coarse-grained lock for the entire document and one has to obtain this lock to edit the document. Hence it allows only one participant to edit the text (i.e. be active in the document) at one time. This active user must release control so that other users could make changes to the document. This kind of close collaboration does not seem to work well with the dynamics of group activity. There is no provision for passing awareness information to other users.



```
This is mpEdit hablet developed at NCSA (National Center for Supercomputing Applicat
It allows more 2 or more people to edit the document at the same time.
However, there is a coarse-grained lock for the entire document and one has to obtai
Hence it allows only one participant to edit (i.e. be active in the document) the te
This active user must release control s
This kind of close collaboration does r
There is no provision for passing aware
```

Figure 2.4: Habanero collaborative environment and mpEdit (a text editor)

ITeach [25]: This system (shown in Figure 2.5) has a multi-user text editor with a centralized architecture and tightly coupled collaboration. Its development started at Helsinki University of Technology, and then continued at the Institute for Advanced Technology in the Humanities at the University of Virginia. Like mpEdit, there is a locking mechanism and the granularity is over the entire document. Hence a user can edit a document only if he has the floor. There is no mechanism to distribute awareness information.


Untitled - Collaborative Text Editor
File Edit Style Size Color Format
JCE: Initiator's SCM
Session Tools Available Help
JCE
Request
Release
Group Chat

Title:

I am making a new document in ITeach collaborative text editor. A user can edit into this document only if he has the floor. Thus the locking is done over the entire document which is quiet coarse grained.

Participant's List:
Ali@VIRGINIA-QH4VFE
Scott@VIRGINIA-QH4VFE
grant@VIRGINIA-QH4VFE

Session Initiator:
Ali@VIRGINIA-QH4VFE

Floor Choice: Request and Get

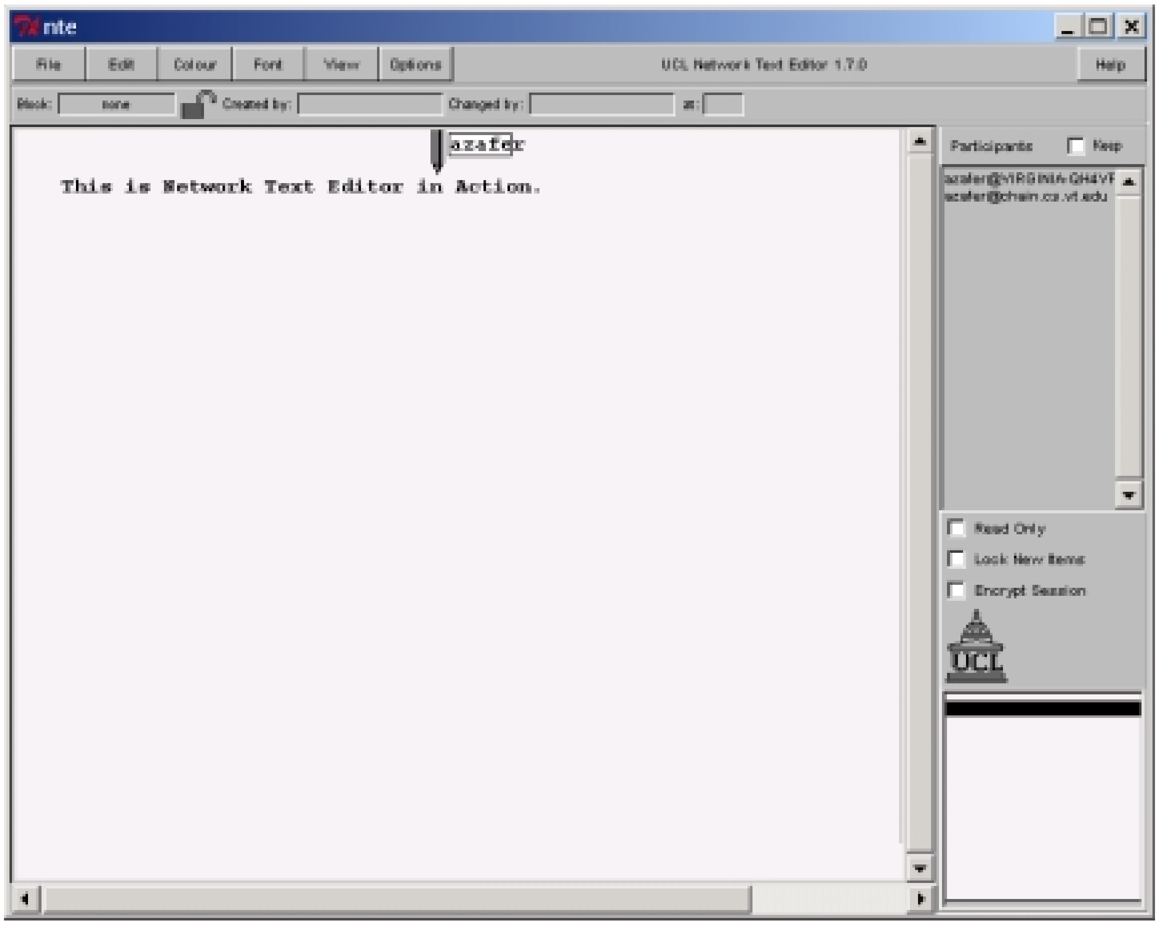
Current Floor-Holder:
You have the floor

System Message:
Scott@VIRGINIA-QH4VFE is joined.
grant@VIRGINIA-QH4VFE is joined.
You have the floor.

Figure 2.5: ITeach Server and Editor

MUSE [31]: MUSE was developed at University of Kentucky in Tcl/Tk. It is slightly better than mpEdit in that the granularity of locking is one line. Also the lines are colored differently depending on whether they have been locked or not. A line locked by a user would appear as green in his window but would appear as red in other participant's window. Apart from these color codes, there is no awareness mechanism available. It uses client-server architecture with the server managing all the locks held by various clients.

Network Text Editor (NTE) [32]: (Figure 2.6) This is a shared text editor used during conferencing. Users editing the document need to devise appropriate cooperation protocol – like establishing roles for the participant, etc. It provides no locking mechanism; the users can do the editing simultaneously without having to get the floor. Hence, this could lead to conflicts between users doing the editing concurrently. The system does not recover from these conflicts but only displays them – audio cue like beeping, when they occur. It uses telepointers for the remote participants to provide awareness information.



This is Network Text Editor in Action.

azafer@VIRGINIA-QH4VF
azafer@chain.cs.vt.edu

Figure 2.6: Network Text Editor

Table 2.1, as shown below, gives a quick preview of the features for some of these systems and compares them against each other.

	Iteach	**Habanero**	**MUSE**	nte
Cut, Copy, Paste	Available	Available	Available	Available
Undo, Redo	Not available	Not available	Not available	Not available
Groupware Widgets	No such devices. In fact the collaboration is very tightly coupled. There exists a kind of Master-Slave relationship between the holder of floor and the other terminals.	No such devices. The collaboration is very tightly coupled. There exists a kind of Master-Slave relationship between the holder of floor and the other terminals.	Color codes are used in this case. These codes are not for individual person but indicate what parts of the document different users are editing concurrently. The lines that say A is editing will appear in his window as green and in others window as pink and vice versa.	The cursor changes its appearance while editing is going on. This gives the user the illusion where other user is.
Concurrency Control	Control is through the floor. The person holding the floor is allowed to edit the document. Not otherwise.	Control is through the floor. The person holding the floor is allowed to edit the document. Not otherwise. Also, there is a minimum idle time after which the control is lost.	Here locks are used to achieve concurrency control. The minimum level of locking is one line.	There is no locking or concurrency control. The users themselves have to form some editing protocol. The system only displays the conflict but does nothing to recover from it.
Centralized	Has a central server. The documents are always stored on this server. The clients act like dumb terminals with little functionality.	Has a central server that serializes the editing operations. The document can be stored on the client or the server.	Has Client Server architecture. The Server maintains manages the locks on various clients and ensures their integrity.	NTE is a multimedia conferencing tool and not an editor per se. Hence it allows unicast session or multicast conferencing session.
Elegant to use	Not much; unless a protocol for working collaboratively is devised.	Better than ITeach. Also a protocol for working collaboratively is necessary.	A bit better than the previous two in terms of granularity of locking. Allows work on separate parts of the document by different users simultaneously	Not elegant to use. At the time of this writing, nte was in a beta test version, so it sometimes behaves abnormally.

Table 2.1: Feature comparison of research systems

2.2 Interface Support for Workspace Awareness

Collaborative editors use one of two styles of collaboration. The first is WYSIWIS (what you see is what I see)[24, 38, 3], which corresponds to tightly coupled group activity. Due to this high coupling, there is not much need to distribute awareness information, as all the participants are working at the same position or on the same artifact. The second type is called location-relaxed WYSIWIS[24, 38, 3] and corresponds to a more relaxed and flexible collaboration. Here, the participants might be viewing different parts of the document. Thus there is a need to provide appropriate widgets to make them aware of each other's activities. The following subsections discuss some groupware widgets.

2.2.1 Telepointers

Telepointers[24, 38, 3] give an indication of the remote participant's pointer location. A variant of this is a **telecaret**[3]. This corresponds to the caret position of the remote participant. Figure 2.7 shows a telecaret.

Since there might be more than two remote participants, the telecaret or telepointer could be appropriately colored according to the color of the remote participant. It could additionally be appended with the remote participants name. Whatever is the mechanism, the goal is to make explicit the precise location of the remote participant during group editing. One could also associate a semantic[18] meaning with the shape, size or the change to either pointer. For example, the shape of these pointers could change when a button is clicked. This provides more awareness information to the remote participants about each other's activities.

It is to be noted that a telepointer or telecaret is useful and needed only when the two remote participants are working in sections of the document that are close enough for these pointers to be seen.

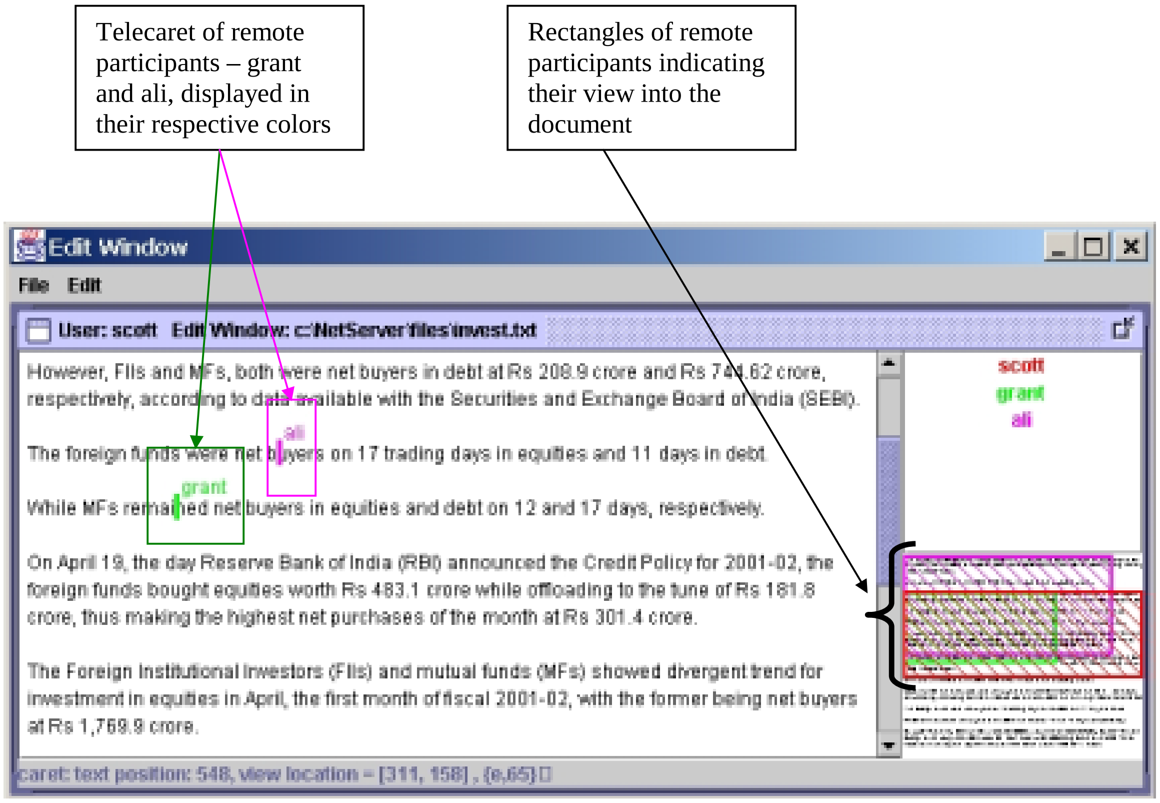
2.2.2 Multi-User Scroll Bar and Radar Views

Although telepointers give exact location of the remote participant, they do not provide the extent of view in the document of the remote participant. Multi-User Scroll Bar[24, 3] and RadarView[24, 3] help to provide this information. A Multi-User Scroll bar is a scroll bar that not only displays the local user's scroll bar but also the scroll bar of each remote participant. This was used in SASE[1] system, discussed earlier. One of the disadvantages of this is the amount of space it takes to convey this awareness information. The space overhead occurs because there is a scroll bar for each remote participant. Figure 2.3 displays a multi-user scroll bar.

A variant of this as advocated by SASSE[1] and Calliope[30] to save space, collapses all the remote participant's scroll bars in to a single scrollbar. Thus there are two scroll bars

– a local scroll bar for the local user, and a remote scroll bar displaying remote users views in to the document. The remote scrollbar has colored rectangles on them to differentiate between remote participants also provide information about the extent of their view in to the document.

A radar-view compresses the entire document into a miniature view and displays that in a window. The remote participant's view in the document can be displayed by a shaded rectangle of his color on this window as shown in Figure 2.7.


Telecaret of remote participants – grant and ali, displayed in their respective colors
Rectangles of remote participants indicating their view into the document
Edit Window
File
Edit
User: scott
Edit Window: c:\NetServer\files\invest.txt
ali
grant
scott
grant
ali

However, FIIs and MFs, both were net buyers in debt at Rs 208.9 crore and Rs 744.62 crore, respectively, according to data available with the Securities and Exchange Board of India (SEBI).

The foreign funds were net buyers on 17 trading days in equities and 11 days in debt.

While MFs remained net buyers in equities and debt on 12 and 17 days, respectively.

On April 19, the day Reserve Bank of India (RBI) announced the Credit Policy for 2001-02, the foreign funds bought equities worth Rs 483.1 crore while offloading to the tune of Rs 181.8 crore, thus making the highest net purchases of the month at Rs 301.4 crore.

The Foreign Institutional Investors (FIIs) and mutual funds (MFs) showed divergent trend for investment in equities in April, the first month of fiscal 2001-02, with the former being net buyers at Rs 1,769.9 crore.

caret: text position: 548, view location = [311, 158] , (e,65)

Figure 2.7: NetEdit's Editor window showing Telecaret and Radarview

2.2.3 Workspace Teleportals

When one is interested in seeing another person's entire view in full size, pressing a mouse button temporarily 'teleports' him to that person's location, and returning to his original view when the button is released. This technique[21, 24] allows people to 'glance' at another's work area without much effort. It works by rapidly scrolling to the remote participant's location when the mouse button is pressed, and then scrolling back again when upon release.

2.3 Groupware Architecture Analysis

Groupware systems are inherently distributed, with their components having a protocol for communication between them. Their architecture can range from completely centralized to completely replicated[17, 11]. A middle ground between them corresponds to a hybrid architecture[11] that is replicated but also has certain centralized components. The following section describes each approach and the tradeoffs involved in using them for developing a collaborative editor.

2.3.1 Centralized/Replicated Architectures

Dewan[11] associates an architecture with a replication degree – two extremes of which are centralized and replicated. Centralized architectures[17] use a single application program, residing on one central server machine to control all input and output to the distributed participants. The data and the processing details reside on this central machine. Client processes residing at each site are only responsible for passing requests to the central program, and for displaying any output sent to it from the central program. The advantage of a centralized scheme is that the synchronization is easy – state information is consistent since it is all located in one place, and events are always handled at the client processes in the same order because they are serialized by the server.

Replicated architectures[17, 11], on the other hand, execute a copy of the program at every site. Both, the data and the processing are distributed to all the remote participants. Thus each replica must coordinate explicitly both local and remote actions, and must attend to synchronizing all copies so they do not get out of step. Centralized architectures are easy to build, as it is simple to handle conflicts, concurrency issues and to maintain a single state of the data; because all of the processing and data resides in one central place. Its main drawbacks are latency, performance bottlenecks, and display issues in heterogeneous environments of remote users. A centralized scheme implies sequential processing – user's inputs are transmitted from the remote machine to the central application that must handle it and update the displays (if necessary) before the next input request can be processed. Due to this round trip delay, the responsiveness of the system is reduced and can be annoying especially in highly interactive applications like group editing.

Centralized architectures might have problems dealing with heterogeneous environments, as it is unlikely that a single process at the server can appropriately update remote clients running on (say) a Windows 95 and a Macintosh environment, as they might have different hardware capabilities – the server process tells each client how to display rather than what to display. A replicated scheme, on the other hand, implies parallel processing, where the handling of interactions and screen updates can occur in parallel at each replica. Communication is efficient, as replicas need only exchange critical state information to keep their copy of the data current. While remote activities may still be delayed, a person's local activities can be processed immediately. Processing bottlenecks are less likely – each replica is responsible for drawing only the local view, unlike the central model, which must update the graphics of all the client's screens. Heterogeneous environments can be easily handled since information regarding what to display rather than how to display is sent to all the clients. However, the cost of replication is increased complexity as issues of distributed systems like conflict management, concurrency control, etc., must be handled.

Replicated systems have an enormous growth in terms of the number of communication paths; it grows at the rate of n(n-1)/2, where n is the number of clients in the system. As compared to this, centralized systems have linear growth: n. Thus, centralized systems are more scalable in terms of communication requirements than replicated systems.

Somewhere in between are semi-replicated hybrid architectures[11] that contain both centralized and replicated components. In such systems, the data and processing might be replicated at each remote location but the communication between them might occur through a central authority. Patterson[34] advocates a centralized notification server, whose primary responsibility is to maintain the shared state, to respond to state change requests by clients, and to notify others of the change when its state has changed. It is up to the clients to decide how this data is rendered, and the display updated. Iris[26] is a collaborative editing environment and has a two-layered architecture – storage and notification layer and user interface layer. It uses a component-based approach for development.

Chapter 3

System Architecture

This chapter discusses the architecture and the implementation details of the NetEdit system. It starts by describing the functionality of the system from user's perspective. It then gives a detailed description of the server and client components. Lastly, it puts together all the pieces discussed and talks about how they interact with each other.

3.1 NetEdit

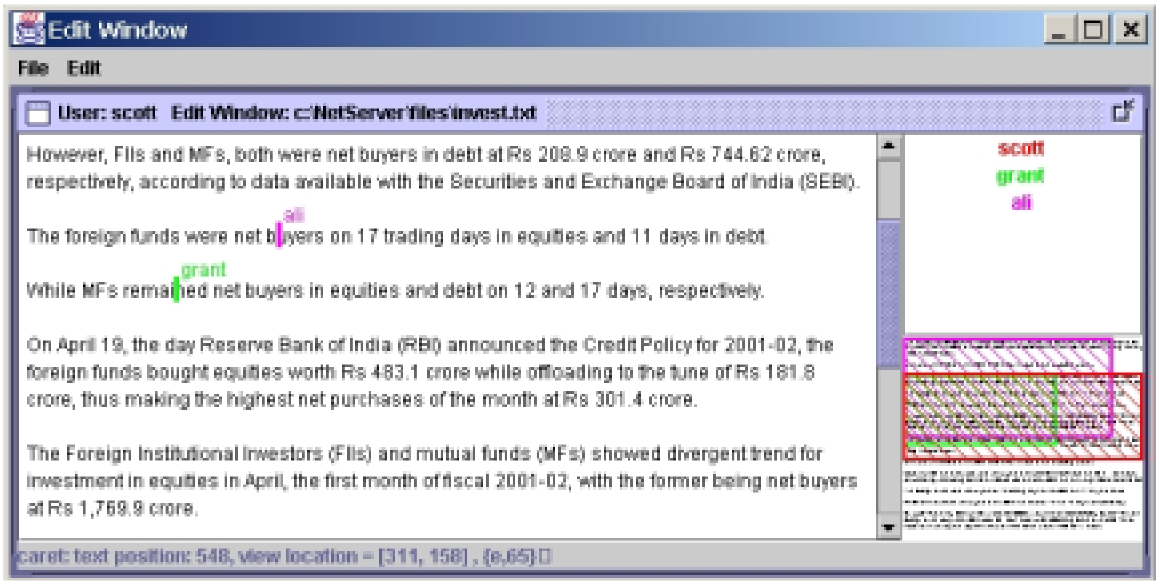
This section talks about the components of NetEdit from a user's perspective. NetEdit provides the following broad set of facilities:

- Allows editing of a text document
- Provides awareness about who else is participating in the same session
- Allows communication with collaborators through chat utility.

3.1.1 Editing a text document

All the files that users can edit through NetEdit are stored at the server. These files could be organized hierarchically in directories and sub-directories. When a person logs into NetEdit, he is presented with this hierarchical view of the files at the server. He can select any file in this hierarchy for editing. A session consists of one or more users participating in editing a particular file. He can either join an existing session or create a new session.

Figure 3.1 shows a file being edited by users scott, grant and ali. From the perspective of scott, the left section is the document window where you edit the text for the file. Apart from showing scott's cursor, it also shows telecursor locations for remote participants – grant and ali.



However, FIIs and MFs, both were net buyers in debt at Rs 208.9 crore and Rs 744.62 crore, respectively, according to data available with the Securities and Exchange Board of India (SEBI).

The foreign funds were net buyers on 17 trading days in equities and 11 days in debt.

While MFs remained net buyers in equities and debt on 12 and 17 days, respectively.

On April 19, the day Reserve Bank of India (RBI) announced the Credit Policy for 2001-02, the foreign funds bought equities worth Rs 483.1 crore while offloading to the tune of Rs 181.8 crore, thus making the highest net purchases of the month at Rs 301.4 crore.

The Foreign Institutional Investors (FIIs) and mutual funds (MFs) showed divergent trend for investment in equities in April, the first month of fiscal 2001-02, with the former being net buyers at Rs 1,769.9 crore.

caret: text position: 548, view location = [311, 158] , (e,65)

Figure 3.1: NetEdit's Editor Window

3.1.2 Awareness Information

Awareness information consists of cues that make a user aware of the other users in the system and their activities (like what sessions they are participating in, where in the document they are performing changes, etc.).

There are four places in the system where awareness information is provided to the user. First is a list of sessions that are currently active. By right clicking on any session, a user can determine the list of users participating in that session. Figure 3.2 shows three sessions active in the system. It additionally shows 3 users (scott, ali, and grant) participating in an editing session for the file C:\NetServer\files\invest.txt.

When editing any file, the right-hand side of the edit window has 2 sections – an awareness list at the top and radar view at the bottom (see Figure 3.1). The awareness list specifies all participants presently working on the current document. This list keeps changing as users log in/out of the session. The radar view contains a miniature version of the document along with a shaded rectangle for each participant of that session. This rectangle indicates his view into the document. These rectangles are color-coded and correspond with the color assigned to that participant in the awareness list section. The radar view gives an approximate location of the user's viewport in the document. To get more detailed information (e.g. the exact position of the user's cursor in the document), telecursors are used. These telecursors are also color-coded to correspond to the color of the users in the awareness list section.

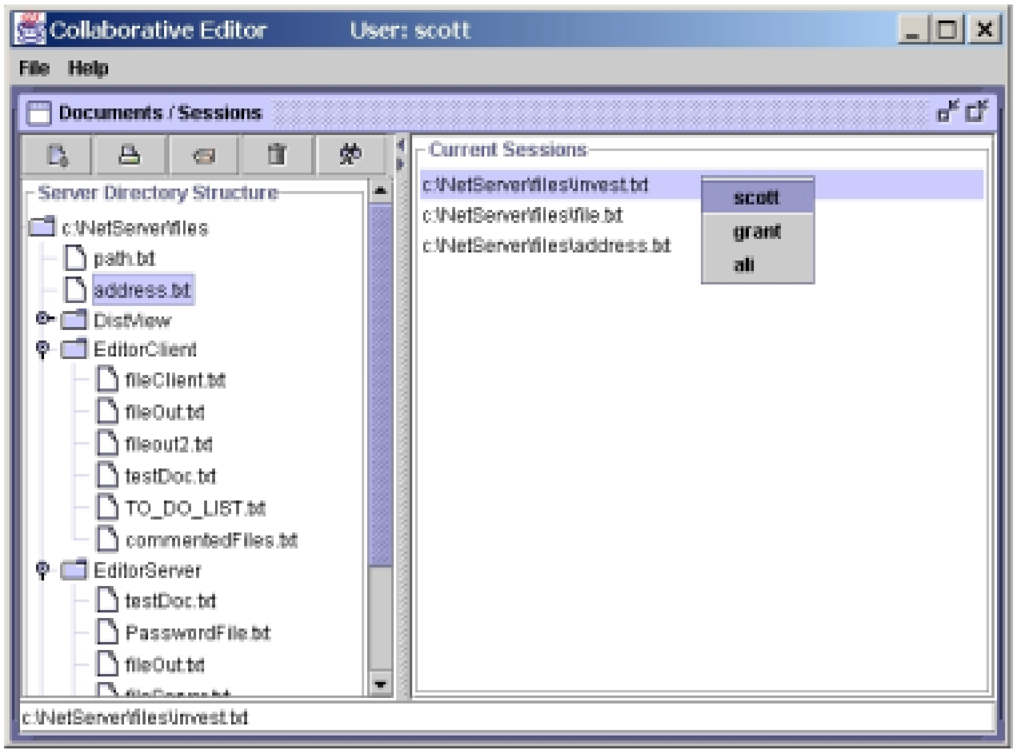
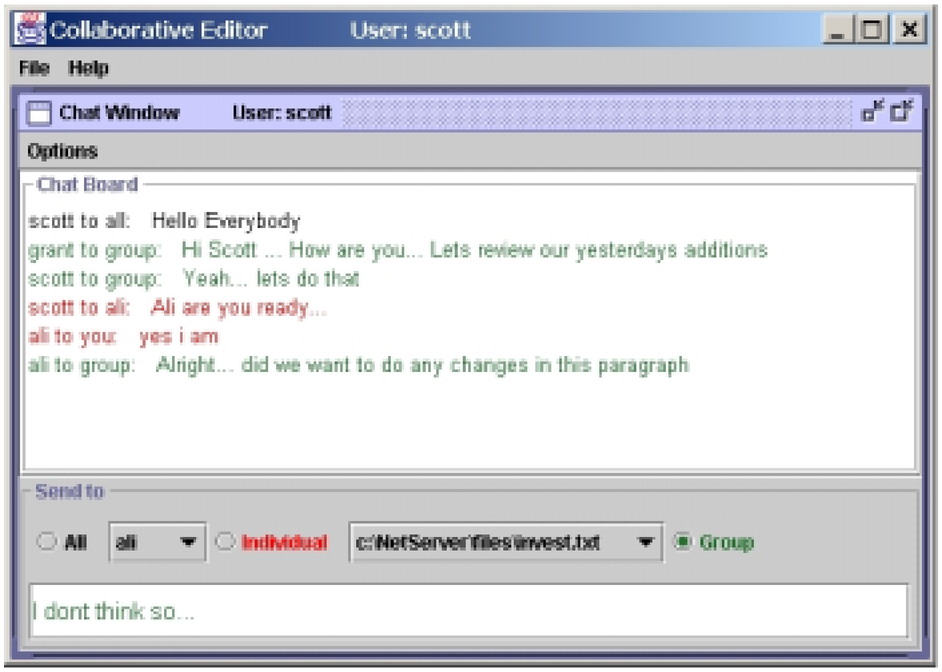


Figure 3.2: NetEdit's Documents/Sessions Window

3.1.3 Communication

NetEdit has a chat utility that allows remote users to communicate with each other. It allows sending a message to all users, sending a message to users in a certain session only or sending a personal message to some user. Figure 3.3 shows the chat window for user scott. Scott is in a session with two other users – grant and ali, for editing a file C:\NetServer\files\invest.txt.



scott to all: Hello Everybody
grant to group: Hi Scott ... How are you... Lets review our yesterdays additions
scott to group: Yeah... lets do that
scott to ali: Ali are you ready...
ali to you: yes i am
ali to group: Alright... did we want to do any changes in this paragraph

ali
c:\NetServer\files\invest.txt
I dont think so...

Figure 3.3: NetEdit's Chat Window

A user can broadcast a general message to all the users in the system. The color of these messages is black. However, he could also send a personalized message (displayed in red) to a specific user. Finally the system also supports sending a message to only a group of users in a session. The group messages are shown in green color.

3.2 Internal architectural details

This section describes the internal functioning with respect to the organization of different pieces of the system and their interaction that makes NetEdit's functionality possible. It is divided in to two subsections – Server-side components and Client-side components:

3.2.1 Server components

Figure 3.4 shows the high level architecture of the server:

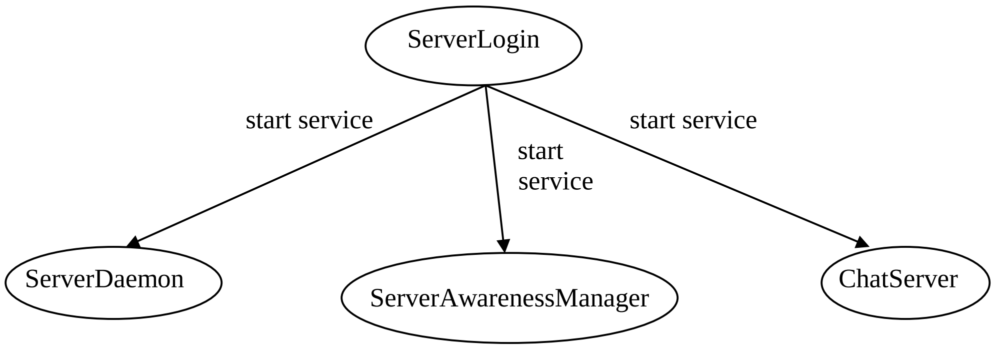


Figure 3.4: High level architecture at the Server

When the server is started, four services (ServerLogin, ServerDaemon, ServerAwarenessManager, and ChatServer) are forked. These services are threads running in the system. ServerLogin listens at a port known to the clients. The ports at which the other three services listen are chosen non-deterministically from a pool of available port numbers. Since the port numbers are allocated in this manner, this system might not work if there is a firewall between the server and a client. ServerLogin knows the ports at which the other services are listening. After establishing a socket connection with ServerLogin, the client sends the username/password to it. If this client is an authorized user, then ServerLogin sends port numbers of the other three services to the client. The client can then establish socket connections with these services directly. Thus, ServerLogin serves only as a starting point for entering into the system. The following subsections describe the functionality of ServerDaemon, ServerAwarenessManager and ChatServer.

ServerDaemon

When the client establishes a socket connection with this service, a proxy called ClientProxy is created at the server. This proxy is responsible for maintaining the client's view of the file hierarchy at the server. On behalf of the user, it executes requests like renaming a file, deleting a file, opening a file etc. Thus a proxy (ClientProxy) for each client is created and maintained by ServerDaemon. These proxies are threads forked by ServerDaemon. ServerDaemon also maintains the sessions that are currently active in the system. A session consists of one or more users participating in the editing of a document. An object EditServer is created for each session. When a client is participating in a session for editing a document, a proxy for it called EditClientProxy is created at the server. This proxy, which runs as a thread at the server, maintains communication regarding insertions and deletions from the document, with its counterpart at the client. These proxies (EditClientProxy) of all the clients together implement the core conflict and consistency management server-side algorithm for group editing. This algorithm is described in detail in the section 4.7.

Figure 3.5 shows the hierarchy of objects that implement the above two functions.

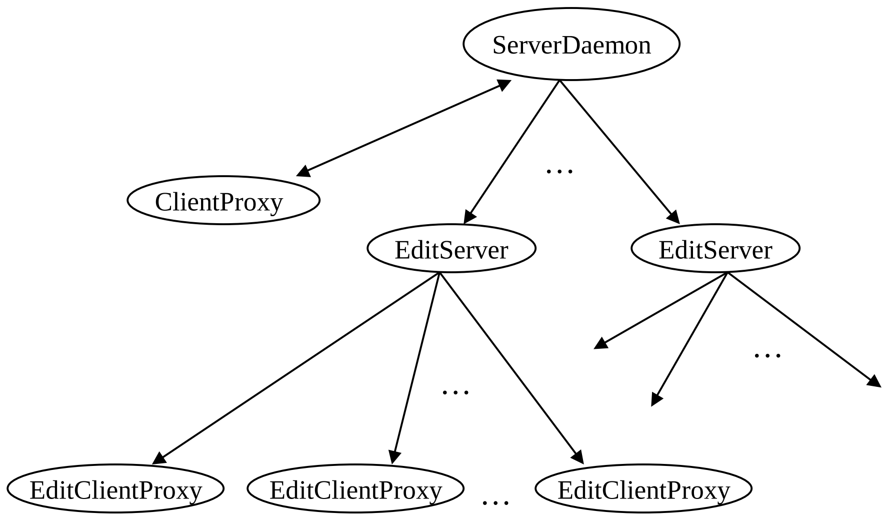


Figure 3.5: Hierarchy of threads under ServerDaemon

Thus as we see above, ClientProxy is responsible for performing administrative tasks like renaming a file, deleting a file, etc., and EditClientProxy is responsible for implementing the protocol that allows group editing.

ServerAwarenessManager

Figure 3.6 shows the architecture responsible for transferring awareness information – telecursor and radarpane updates, participant’s list, etc., to the clients. Each client, after getting an authorization from the ServerLogin, establishes a socket connection with the ServerAwarenessManager. This creates a proxy (a thread), called ClientAwarenessProxy, at the server. This proxy is responsible for transferring all the awareness information learned from other clients to this client. Each client has one ClientAwarenessProxy at the server, regardless of the number of sessions in which it is participating.

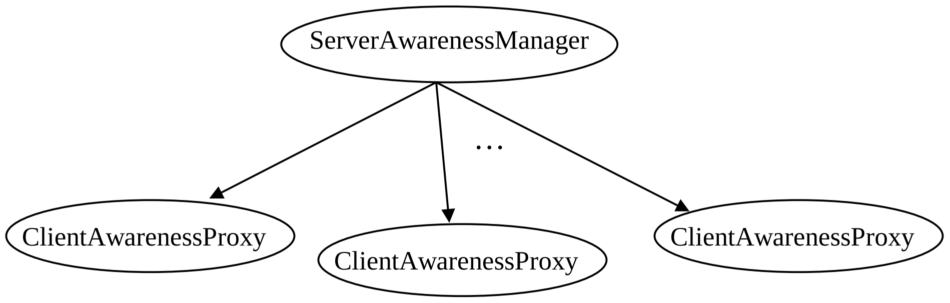


Figure 3.6: Hierarchy of threads under ServerAwarenessManager

ChatServer

This has architecture similar to ServerAwarenessManager. It is responsible for managing the chat communication system. Figure 3.7 shows the hierarchy of threads under ChatServer.

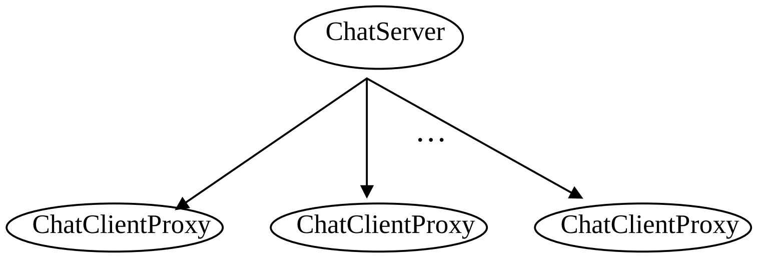


Figure 3.7: Hierarchy of threads under ChatServer

Each client, after getting an authorization from the ServerLogin, establishes a socket connection with the ChatServer. This creates a proxy (a thread) called ChatClientProxy, at the server. This proxy is responsible for transferring all the chat messages from other clients to this client depending on whether the message is a simple broadcast (message to all) or multicast (message to a group) or personalized to a user. It is also responsible for passing messages from this client to other clients depending on the type of message – broadcast, multicast or a personalized message. Each client has one ChatClientProxy at the server, regardless of the number of sessions in which it is participating.

3.2.2 Client components

Figure 3.8 shows the architecture at the client side.

ClientDaemon is an object that first establishes a socket connection with the ServerLogin and sends it username and password. If the user is authorized to use the system, ServerLogin replies back with the port numbers of other services – ServerDaemon, ServerAwarenessManager and ChatServer. The Client then forks the threads – ClientAwarenessManager, EditClient and ChatClient, shown in Figure 3.8, which connect to the appropriate services at the server. These are described further in the following subsections

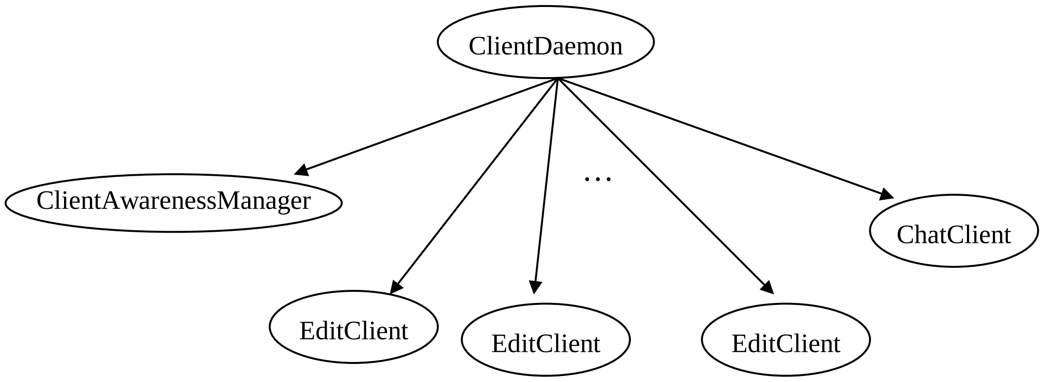


Figure 3.8: Hierarchy of threads under ClientDaemon

ClientAwarenessManager

ClientAwarenessManager is responsible for distributing the awareness information updates, obtained from the server (specifically ClientAwarenessProxy), to various graphical user interface widgets – radar views, telecursors, participant's list etc, at the client. These widgets help a user to be aware of other user's activities in the system. It establishes a socket connection with ClientAwarenessProxy at the server to continuously get the updates. It also sends updates back to ClientAwarenessProxy when the state of the client changes – like scrolling to different part of the document, exiting a session etc.

EditClient

EditClient is a thread object forked for each document that the user is participating in for editing. It communicates all the insertions and deletions through a socket connection with EditClientProxy at the server. The processing at both EditClient and EditClientProxy and the communication between them constitutes the core conflict and consistency management algorithm that manages group editing. The details of this processing and the messages that are exchanged, is discussed in detail in the section 4.7.

ChatClient

ChatClient creates the chat window to display the transcripts of the chat messages and provides a space for the user to type his messages. It establishes a socket connection with ChatClientProxy at the server and exchanges streams of chat messages with it.

3.3 Integrated View

Having discussed the server components and client components in detail, let me put them together to describe how they interact with each other. Figure 3.9 shows an integrated view of the complete system. As can be seen, the server has three main systems – chat system, awareness distributor system, and editing system. These systems are further divided into smaller components. These components communicate with their counterparts at the clients. The different types of arrows depict distinct kinds of message types being exchanged between the client and the server. The arrows that connect server components to client components indicate socket streams. Thus ChatClientProxy at the server has a socket connection with ChatClient at the client. Similar are the connections for awareness distributor system and editing system.

Figure 3.9 also shows three clients currently logged in the system. There are two editing sessions – EditServer 1 and EditServer 2, presently active and managed by ServerDaemon. Each session has a file associated with it. Thus, there are two files that are being edited by the users. All three users are participating in the first session (EditServer 1). Client 2 and Client 3 are also editing the file associated with Session 2 (EditServer 2). Due to this architecture, the system supports a many-to-many relationship between sessions in the system and users participating in these sessions. That is, a user could be participating in multiple sessions and a session could have more than one user.

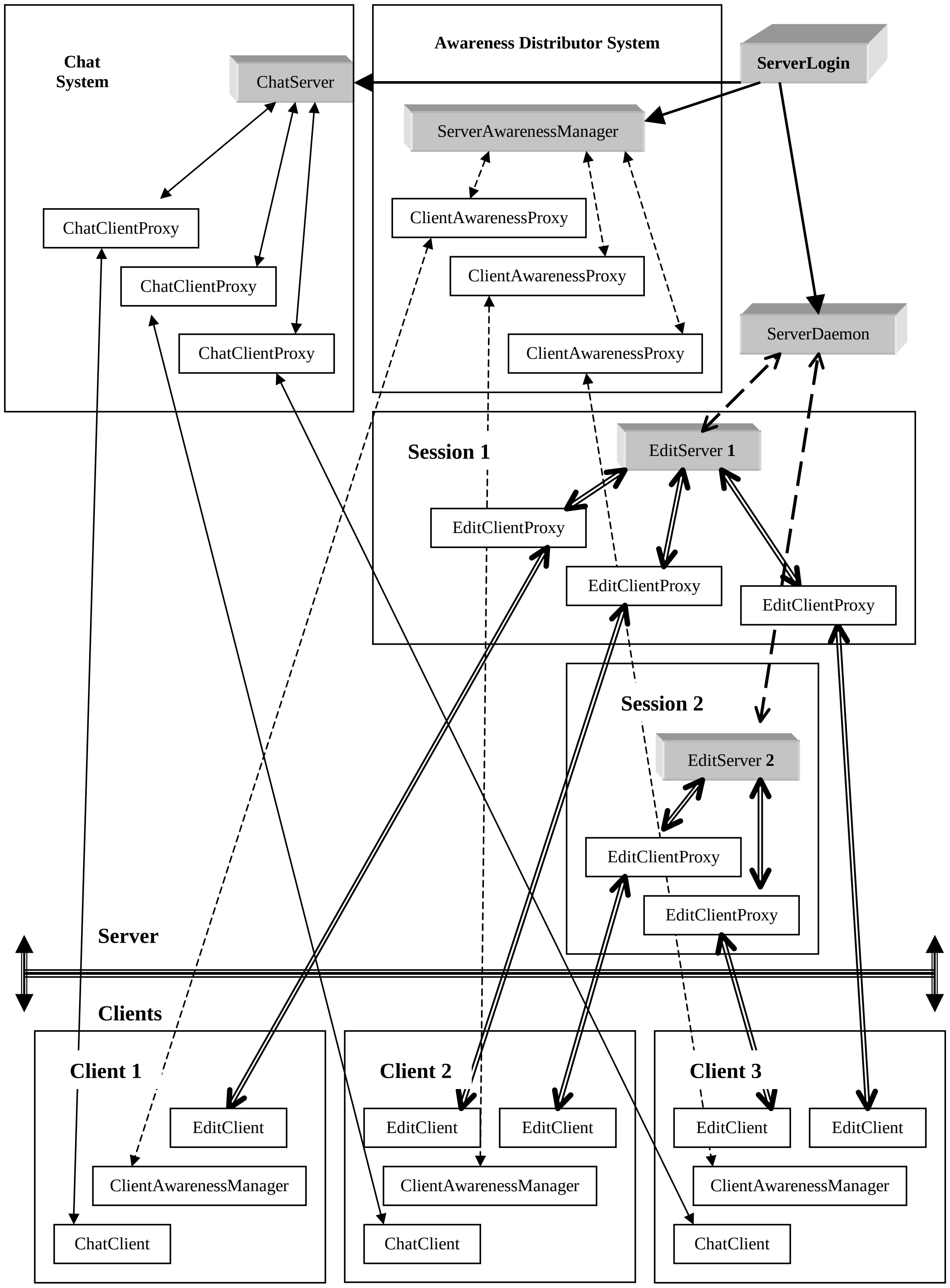


Figure 3.9: Integrated Architecture of NetEdit

Chapter 4

System Design

NetEdit is designed to have the following characteristics:

> **Real-Time response:** The response to a local user's edits must be extremely fast, as close as possible to the response when using a single user editor. Also the latency for reflecting remote users actions must be low, governed mostly by the external communication network latency.
>
> **Unconstrained editing:** Multiple users should be allowed to simultaneously and freely edit any part of the document without any prior locking.
>
> **Distributed architecture:** The users of this system should be allowed to work from remote locations, connected by different communication networks with non-deterministic latency.

This chapter reiterates the architecture of the components involved in group editing, discusses the problems in the design of a synchronization algorithm, surveys some existing concurrency algorithms, and explains the motivation behind this work. It further discusses our approach and concludes with an example illustrating how our algorithm works.

4.1 Session Architecture

Let me reiterate the high-level architecture of a session, as discussed in the Chapter 3, that involves editing of a document by one or more users. It has a star topology as shown below, in Figure 4.1.

Before proceeding forward, let me describe the terms "user", "client" and "server". A user is the person who is doing the editing. He is also referred to as a participant. The client is the client-side interface or the graphical user interface tool in which the editing occurs (see Figure 3.1). It corresponds to EditClient in the session architecture shown in Figure 4.1. The server is a service that holds a session. This corresponds to EditServer (see Figure 4.1). Although EditServer is shown as a single component, it actually consists of many sub-parts that will be discussed in detail later.

Whenever a user joins a session for editing a document, his client establishes a socket connection with EditServer. EditServer sends a copy of the document in its current state to the client. The client then displays this document in the edit window and also maintains it, as the local user and remote users make edits to the document. Thus if there are four users in a session, there will be five copies of the document in the system – one at each user's workstation and one at the server. The users make changes to their local copies of the document. Local edits are applied directly to the local copy, buffered, and sent to the server to distribute it to the remote clients. The server receives similar updates from all the clients. It processes these operations in real-time, updates its own copy and sends the processed updates to the clients so that they can do further processing and update their copy. The processing done at the clients and the server makes sure that all the contentions and conflicts are resolved and the copies at all the clients and the server are consistent.

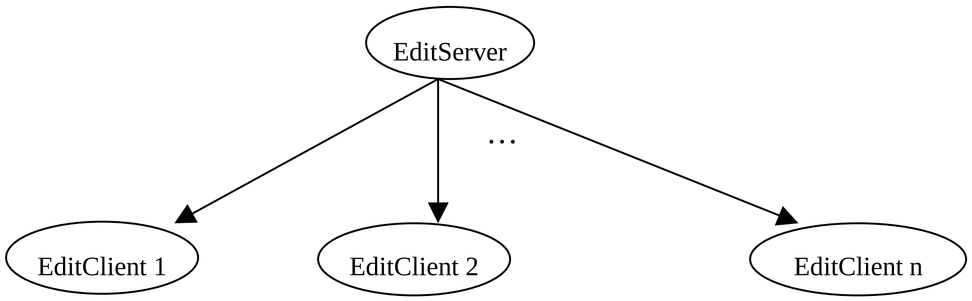


Figure 4.1: Architecture of objects (threads) involved in an editing session

4.2 Issues in the design of a concurrency control algorithm

To achieve good responsiveness from NetEdit, a replicated architecture is used. A copy of the document resides at all the clients. Thus there is a replication of the document at all the distributed sites. Due to the use of this architecture, maintaining consistency between replicated copies is a difficult task. To appreciate the complexities in dealing with this problem consider the following scenario.

Suppose there are three clients C1, C2 and C3 that are performing the edits on behalf of their users as shown in Figure 4.2:

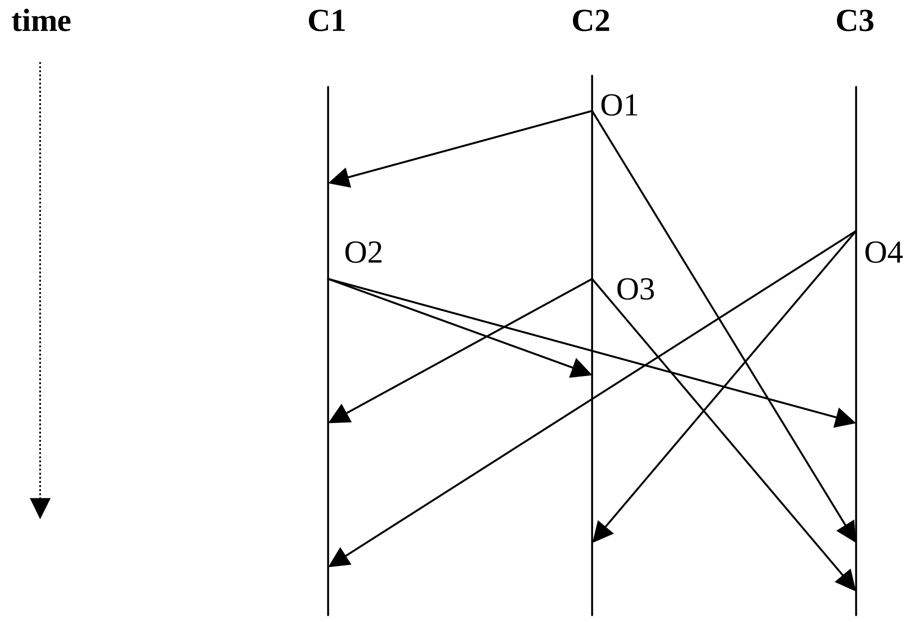


Figure 4.2: Sample editing scenario

In this example, O1, O2, O3 and O4 represent four operations to modify the document. Suppose that the edits are applied to the local copy immediately after they are generated and sent to the remote clients where they are applied in their original form (without transformation of any kind). Transformation (discussed in detail in section 4.4) consists of changing the offset of the operation at which it is applied in the document, so that the new offset is consistent with the execution of other concurrent local operations. As a result the following inconsistency problems might arise.

4.2.1 Divergence

Due to non-deterministic communication delays, the messages might arrive and be executed in different order at different clients. In Figure 4.2, the order of execution of messages is O1, O2, O3, O4 at C1; O1, O3, O2, O4 at C2; O4, O2, O1, O3 at C3. These messages O1, O2, O3 and O4 could be insertions and deletions and hence are not commutative. Thus the state of the document at the three clients will be different. This is unacceptable for a group-editing tool such as this.

4.2.2 Causality Violation

As shown in the above time-line diagram (Figure 4.2), O2 is executed after the execution of O1 at C1. Thus O2 is causally dependant on O1. As a result, it is necessary that this order of execution be preserved at all the clients. However, due to non-deterministic communication delay, this cannot be guaranteed. But out-of-causal-order execution should be prohibited.

According to Lamport [27], let me formally define the causal ordering relation on operations in terms of their generation and execution.

Causal Ordering Relation " –>": Given two operations Oa and Ob, generated at sites i and j, Oa –> Ob if and only if

1. i = j, and the generation of Oa happened before the generation of Ob.
2. i is not equal to j, and the execution of Oa at site j happened before the generation of Ob
3. there exists an operation Ox, such that Oa –> Ox and Ox –> Ob.

Dependent operations: Given any two operations Oa and Ob, Ob is said to be dependant on Oa if and only if Oa –> Ob.

Independent operations: Given any two operations Oa and Ob, they are said to be independent (occurring simultaneously or concurrently) if and only if neither Oa –> Ob, nor Ob –> Oa. This can be expressed as Oa || Ob.

It is to be noted that the above two inconsistency problems can be solved easily by enforcing a total ordering on the messages. This can be done through the use of a central coordinator who time-stamps each message before sending to the clients who then execute the instructions based on this total order. But the drawback of this approach is that, it does not have good responsiveness (round trip delay before the local operation can be applied to the document stored locally) and also cannot solve the third inconsistency problem, intention violation, described in the next section.

4.2.3 Intention Violation

As shown in Figure 4.2, O2 and O3 were generated and applied when the document was in the same state at both C1 and C2. O2 was generated without any knowledge of O3 and vice-versa. Thus O2 is independent of O3 and vice-versa (see the definition of independent operations above). Due to this concurrent generation of operations, the actual effect of an operation at the time of its execution may be different from the intended effect at the time of its generation. To understand this better, let us consider the following example:

Suppose the document at C1 and C2 before the execution of O2 and O3 is “ABCDEFGH”. Let O2 be insert[“1234”, 4] (insertion of “1234” at offset 4). Let O3 be remove[2, 4] (delete 2 characters at offset 4 – removal of “EF”). After the execution of both these operations and preserving the intention of C1 and C2, the final state after the execution of O2 and O3 should be “ABCD1234GH”. However the state at C1 would be “ABCD34EFGH”. This violates the intention of operation O2 as “12” that it inserted are not there in the document and also the intention of operation O3 since “EF” is yet present in the document.

It should be noted that the above three inconsistency problems, divergence, causality violation, and intention violation, are independent of each other and the resolution of one does not guarantee the resolution of others. Also the problem of divergence and intention violation are of different nature. Applying some serialization technique can solve the former but the latter needs some transformation to the operations before applying to the document.

4.3 Consistency model

Having talked about the above problems let me state the model (from Sun et al. [8]) of consistency that must always be maintained by cooperative editing system.

Convergence: At quiescence, when all the operations that were generated are applied at all the clients, all the copies of the replicated document must be the same.

Causality preservation: For any pair of operations Oa and Ob, if Oa –> Ob, then Oa is applied to the document before Ob at all the clients.

Intention preservation: For any operation O, the effects of applying O on documents at all the clients are the same as the intention of O, and the effect of applying O does not change the effects of independent operations.

The convergence property ensures that the state of the document is same at all the clients at the end of an editing process. The causality property ensures that dependant operations are always applied in their causal order at all the clients. Both these properties are necessary for the correctness of the system. The intention preservation property ensures two things. First, the effect of applying an operation at the remote clients is the same as the effect of its application at the local site at the time of its generation and secondly, the execution of independent operations do not interfere with each other.

4.4 Transformation of a message

Transformation[14] consists of changing the offset of the message at which it is applied in the document, so that the new offset is consistent with the execution of other concurrent local operations. As noted earlier, imposing some global order on the operations can take care of convergence and causality. But to take care of intention preservation, this ordering is not sufficient. It is necessary to transform independent operations with respect to each other appropriately. This is explained as follows.

Consider the operations O2 and O3 from Figure 4.2. Both these operations have originated when the document is in the same state at C1 and C2. Lets say the document contains “ABCDEF” before the execution of either O2 or O3. Also consider that O2 is insert[“123”, 2] and O3 is insert[“abc”, 4]. When C1 receives O3, the document at C1 after applying O2 but before executing O3 is “AB123CDEF”. Since (1) O2 and O3 were independent operations, (2) Both O2 and O3 were generated from the same state of the document and (3) the offset of O3 is greater than the offset of O2, O3 needs to be transformed with respect to O2 at Client C1. The transformation essentially consists of adding 3 (size of O2) to the offset of O3. Thus O3 now gets transformed into insert[“abc”, 7]. This when applied to the document gives “AB123CDabcEF”. Similarly, at client C2, O2 will be transformed with respect to O3 giving the final state of the document at C2 as “AB123CDabcEF”. Thus we see that the intentions of both O2 and O3 were preserved. Similar transformations are required for delete-delete, insert-delete and delete-insert combinations. It is to be noted that when you have delete as one of the operations, it might involve subtraction as well.

It is important to realize that the transformation of two operations is warranted only when they originate from the same state of the document. To emphasize this point, consider operations O3 and O4. O3 has seen the effect of execution of O1 but this is not the case with O4. Hence these two operations in their original form cannot be transformed.

4.5 Concurrency control in groupware systems

Some of the algorithms developed to perform unconstrained editing are discussed in this section. These algorithms use a replicated architecture as far as processing and data is concerned. They use optimistic concurrency control where the basic idea is to take an operation executed in some past state and to transform it in a way so that it can be applied to the current state. However, they assume there is already some mechanism for exchanging messages between participants. For example, one could have a centralized communication server and each participant would communicate with the other through this server, or some other mechanism for exchanging messages.

4.5.1 Distributed operational transformation algorithm

Ellis et al. suggested a concurrency control algorithm for group editing based on distributed operational transformation[14, 7]. Here there is no central entity. All the clients have a copy of the document that is being edited collaboratively. They update their own copy. The operations that they apply to their local copy are also sent to all other clients. When the remote clients receive these operations, the state of their local document might be in conflict with the incoming operation. Hence these incoming operations are transformed i.e. their offset changed (incremented or decremented) depending on whether they are inserts or deletes and the type of conflict. Since the processing of these remote and local operations (remote and local operations are defined in section 2.1.1) is distributed at all the clients and the remote operations get transformed when they occur concurrent to local operation, this algorithm is referred to as distributed operational transformation (dOpt).

However, two research groups[8, 36], working independently on this problem, discovered a flaw in this algorithm. When two users are participating in editing a document, and one of them issues and executes more than one operation concurrently with an operation by the other user, the document at both sites becomes inconsistent i.e. the state of the document stored locally at both the user's workstations becomes different. For example, O2 and O3 (Figure 4.2) are generated in the same state and can be transformed against each other. However, O2 and O4 (Figure 4.2) are generated from different state of the document and hence cannot be transformed against each other. The dOpt algorithm did not take this into account and did the transformation in both cases. Both of these research groups have proposed a solution – Reduce approach and Adopted approach, as discussed below, to this problem.

4.5.2 REDUCE approach

REDUCE [8, 7] approach performs two types of transformations to the incoming messages: inclusion transformation and exclusion transformation. Inclusion transformation involves transformation of messages when they are generated from the same state of the document. Exclusion transformation involves more processing since here messages are not generated from the same state of the document. It also maintains a history buffer (HB) that keeps track of all the operations that have been executed and is used to perform the transformations.

As in the above algorithm, there is no central entity. All the clients have a copy of the document that is being edited collaboratively. They update their own copy. The operations that they apply to their local copy are also sent to all other clients. When a new operation O at a client’s site is causally ready for execution, the following is done: First, all the operations in HB that causally follow this operation are undone to restore the document in a state before their execution. Next, operation O is applied to the document. Finally all the undone operations are redone. These undo/do/redo operations must all be done as a single transaction.

4.5.3 Jupiter approach

The Jupiter collaboration system[33, 7] was developed at Xerox PARC. They describe a two-way synchronization protocol that allows 2 participants to be in sync with each other during editing. They further suggest that one can use this 2 party synchronization protocol to achieve n-way synchronization. That is, a client does not synchronize with other clients. Instead each client synchronizes only with the server. Hence, due to this 2 party (client and server) protocol, one can achieve n-way consistency protocol for group editing. It uses a 2 – dimensional state space graph, instead of a HB as in REDUCE approach, to keep track of paths to follow during operation transformation. The state graph ensures that the pair of messages, undergoing transformation, has originated from the same state of the document. The state space is discussed in detail in section 4.7.1.

4.5.4 ADOPTED approach

In the ADOPTED approach[36, 7], there is no central entity. All the clients have a copy of the document that is being edited collaboratively. They update their own copy. The operations that they apply to their local copy are also sent to all other clients.

In addition to performing the above discussed transformation when the two operations are generated from the same state of the document, the ADOPTED approach maintains an N-dimensional interaction model graph to keep track of all valid paths of operation transformations. This N-dimensional interaction model graph can be viewed as a generalization of the 2 dimensional state space graph in the Jupiter algorithm. By choosing the right paths in this interaction model, the algorithm ensures that any pair of operations involved in the transformation originate from the same state of the document.

4.6 Motivation for this work

This section discusses the differences between the synchronization algorithms, their advantages and disadvantages, the motivation behind our work, and finally gives a brief overview of our protocol. It is interesting to observe that the Reduce and Adopted approaches are completely distributed with no central entity. Since in a collaborative system, no client has ownership of the document, it is not clear in these approaches where the document will be stored and maintained. However, both systems are more resilient to failure as compared to the centralized Jupiter system – a failure of one client does not affect the other collaborating participants. And, due to the distributed nature of Adopted and Reduce approaches, their algorithm is also more complex than Jupiter algorithm.

In Reduce and Adopted approach, each client needs to know all the other participants in the system and establish a communication path with them. Hence this architecture is not very scalable. To cope with this enormous growth in communication paths, they might communicate with each other through a central server. If this is done, then the communication server becomes the bottleneck and its failure will cause the entire system to fail. As compared to this the Jupiter algorithm uses a centralized component for communication of messages, processing and storage of the documents. However the clients are not dumb terminals, but need to do their part of processing to the messages received from the server. Thus although the architecture is centralized, it involves replication of the documents at all the clients. The Jupiter collaboration approach discusses a 2-way synchronization protocol for unconstrained editing of a document between two participants. They further suggest how this could be extended to n-way synchronization.

Our goal is a collaborative editing system that is scalable, simple, and resilient to failure. The 2-way synchronization protocol developed for the Jupiter collaboration system served as a good starting point for us to achieve n-way synchronization. By our literature review in this field, it appeared that almost all the group editing systems were built before 1995 and were not being maintained by the research groups. For example, SASSE[1], discussed in Chapter 2, was built in the early 90's for Macintosh workstations. It is our thinking that the current state of the art in distributed technology and programming environment makes it relatively simple and efficient, as compared to early 90's, to develop collaborative editing systems.

Thus in this thesis, we worked out the details of extending this two-way protocol to multi-way protocol. We used multiple two-way component synchronization to achieve multi-component synchronization protocol. This is shown in Figure 4.3 as follows:

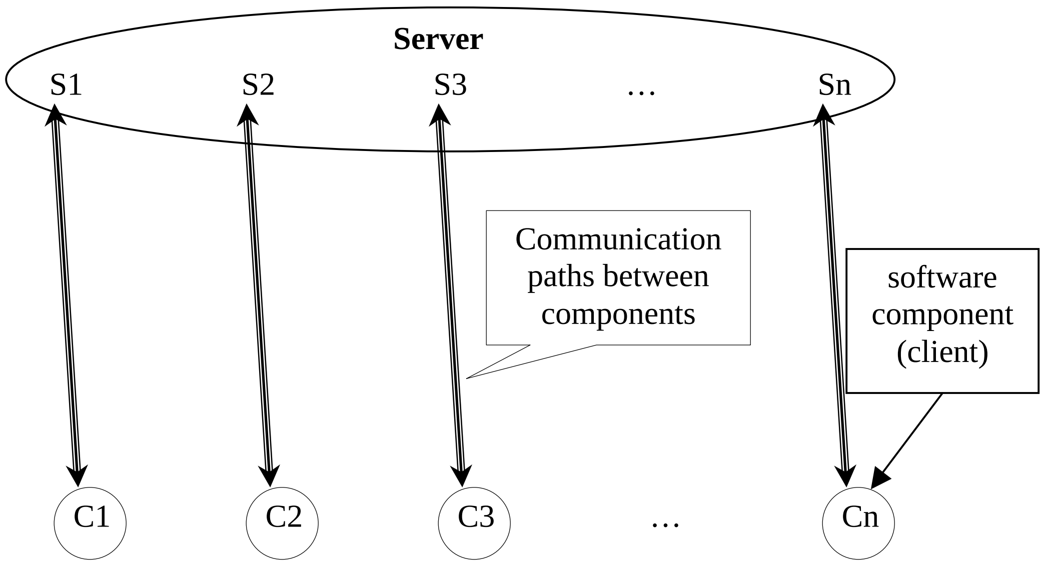

Server
S1
S2
S3
…
Sn
Communication paths between components
software component (client)
C1
C2
C3
…
Cn

Figure 4.3: Multi-component synchronization

Here C1, C2, C3, … Cn correspond to the software components (clients in our context), while S1, S2, S3, … Sn are their counterparts at the server. The arrows show the communication paths between components. C1 synchronizes with S1, C2 with S2 and so on. We developed a communication protocol between S1, S2, S3 … Sn so that C1, C2, C3 … Cn synchronize with each other without each being aware of the other. Thus we see that C1, C2, C3, … Cn coordinate with each other through their counterparts at the server.

4.7 Algorithm details

This section describes the multi-party synchronization protocol discussed in brief earlier. It starts with discussing the data structures used, then formally specifies the algorithm and finally ends with an example illustrating its steps.

4.7.1 Data Structures

All the clients and the server (one for each client) maintain a state space [33] as shown in Figure 4.4.

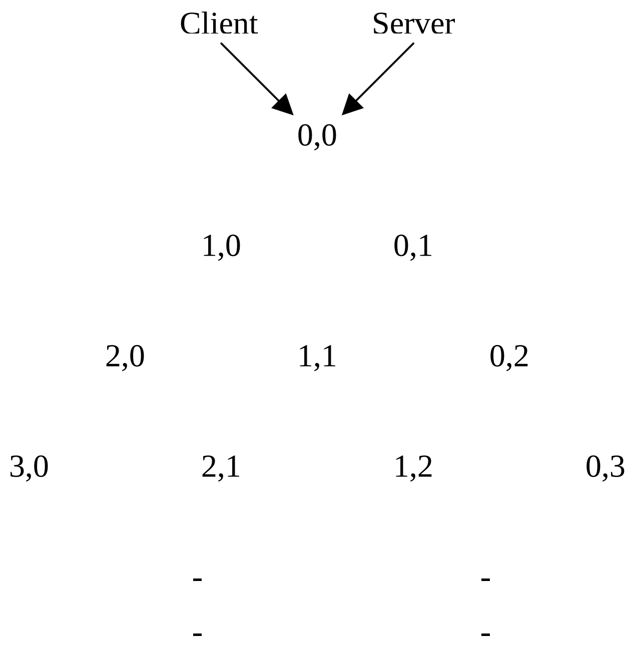


Figure 4.4: State space

The state space is used so that each client-server pair could maintain information of where the other is, relative to it, in the editing process. Both the client and the server pass through this state space as they process messages. Each state is labeled with the number of messages from the client and server that have been processed to that point. For example, if the client is in the state (2,1), it has generated and processed 2 messages of its own, and has received and processed 1 message from the server. Similarly if the server is in state (0,2), it has generated and processed 2 messages of its own, and has received and processed 0 messages from the client. Hence the second component corresponds to the server while the first corresponds to the client. If the server and client process the messages in the same order, then they will follow the same path in the state space graph.

The algorithm labels each message with the state the sender was in just before the message was generated. The recipient uses these labels to detect conflicts. One can transform two concurrent messages only when they are generated from the same state of the document, or special handling is required.

The clients and the server (one for each client) also maintain a buffer that contains operations that have been generated and applied locally but have not been acknowledged by the other party.

4.7.2 Description of the algorithm

This description is divided in to 2 parts – Client and Server, as follows:

Client

When the client receives a message, say s1, from the server with state space value of (a1, a2) then you search its buffer from the beginning (i.e. the oldest entry in it) and start discarding those messages (with state space (b1, b2)) from it (buffer) such that b1 < a1. These are those messages that server has already received and processed.

Next transform s1 with respect to the next message (the first message after the discarded messages) in the buffer. This is the message that was executed in parallel to s1 and also when the document was in the same state. It might be that there are no messages left in the buffer after discarding; in that case you simply apply the message directly to the document. Otherwise, call this transformed message $s1'$. Next transform $s1'$ with each remaining message in the buffer in order until you reach the end as follows

$s1'$ = transforms ($s1'$, next message in the buffer);

Apply the final transformed message to the document in its current state. While you are transforming s1 with the messages in the buffer, also transform those messages (ones in the buffer) say c2, c3, c4 … into $c2'$, $c3'$, $c4'$… and store them accordingly in the buffer along with updated state space values (i.e. with their original state space values except that the server component – second component of the 2 valued tuple, in each will be incremented by one). The state of the client now goes from (x, y) to (x, y+1). This procedure is repeated for all the messages that it receives from the server (i.e. for remote operations).

The operations that are generated locally are applied to the document directly and also stored in the buffer with proper state values (i.e. the state the document was in when the local operation was generated). After applying this local operation, the client moves from state (x, y) to (x+1, y).

Server

Assume that there are 4 clients (C1, C2, C3 and C4) in the system and S1, S2, S3 and S4 respectively are proxies for them. Let the buffers of S1 through S4 be named as q1, q2, q3 and q4 respectively. Suppose S1, which maintains communication with C1, is in state (x1, y1). Similarly S2, S3 and S4 are in states (x2, y2), (x3, y3), and (x4, y4) respectively.

Suppose that a message c1 comes from client C1 having state space value (a1, b1). S1 will search its buffer (q1) from the beginning (i.e. the oldest entry in it) and start discarding those messages that have state space (u1, v1) from it (q1) such that v1 < b1. These are those messages that client C1 has already received and processed.

Next transform c1 with respect to the next message (the first message after the discarded messages) in the buffer q1 of S1. This is the message that was executed in parallel to c1 and also when the document was in the same state. It might be that there are no messages left in the buffer q1 after discarding; in that case you simply apply c1 directly to the document. Otherwise, call this transformed message $c1'$. Next transform $c1'$ with each remaining message in the buffer (q1) in order until you reach the end as follows

$c1'$ = transforms ($c1'$, next message in the outgoing queue(q1));

Apply the final transformed message (say m1) to the document in its current state. While you are transforming c1 with the messages in the buffer (q1), also transform those messages (ones in the buffer q1) say s2, s3, s4 … into $s2'$, $s3'$, $s4'$… and store them accordingly in the buffer (q1) along with updated state space values (i.e. with their original state space values except that the client component – second component of the 2 valued tuple, in each will be incremented by one). Also increment the client component of the state for S1, that is, its state changes from (x1, y1) to (x1+1, y1).

Add m1 to the buffers of C2, C3 and C4. The state stored in the buffer q2 for message m1 would be (x2, y2). Similarly m1 in q3 and q4 will have state (x3, y3) and (x4, y4) respectively associated with it. This corresponds to the state that S2, S3 and S4 respectively were when the message m1 was processed by S1.

m1 will then be sent to all the clients, that is, m1 with state value (x2, y2) will be sent to C2; m1 with state value (x3, y3) will be sent to C3; m1 with state value (x4, y4) will be sent to C4. S2 goes to state (x2, y2+1). S3 goes to state (x3, y3+1). And S4 goes to state (x4, y4+1). It must be noted that all the above must be executed as one atomic operation.

Similar processing is required for messages from other clients as well. At quiescence, S1 must have state same as the state of client C1. Similarly S2 must have state same as the state of client C2 and so on for all the clients in the system.

4.7.3 An example of the algorithm

This section discusses the steps of the algorithm by taking a specific example.

Suppose there are 2 clients in the system. Hence there will be 4 state graphs that are maintained – 2 by the clients (one each), and 2 by the server (one for each client). Assume that the initial state of the document at all these entities is (0, 0) and the buffers are empty. Also assume that the document in this state contains “ABCDEF”. They are as shown in Figure 4.5.

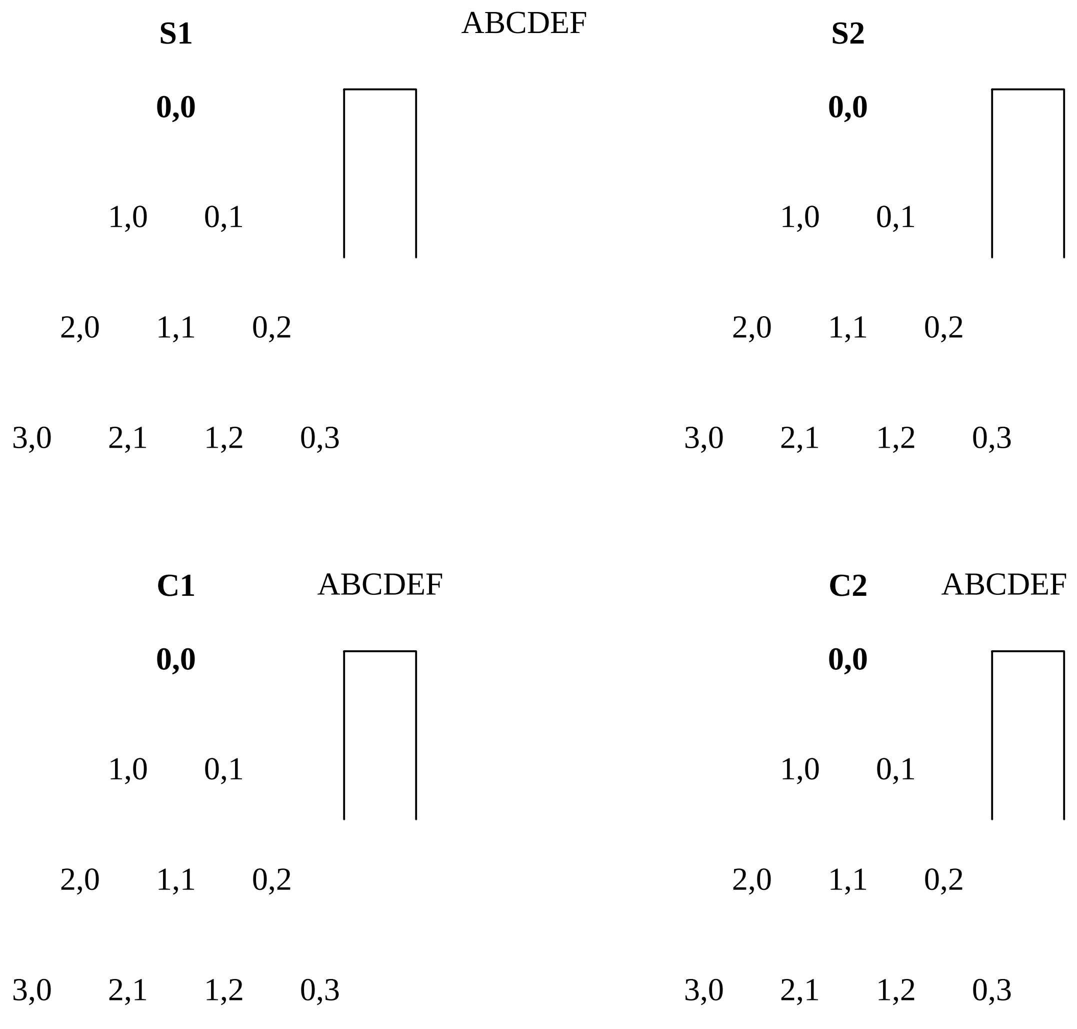


Figure 4.5: Initial state of the system

Each entity’s current state in the state graph is displayed by bold font and buffers are shown as empty. At this point, suppose that C1 performs insert[“a”, 2] (insertion of character ‘a’ at offset 2 in the document) and C2 performs remove[1, 4] (removal of a character at offset 4 in the document). Both these local operations are performed at the local copy of the document. This causes them to move to a new state and the buffers also get modified as shown in Figure 4.6.

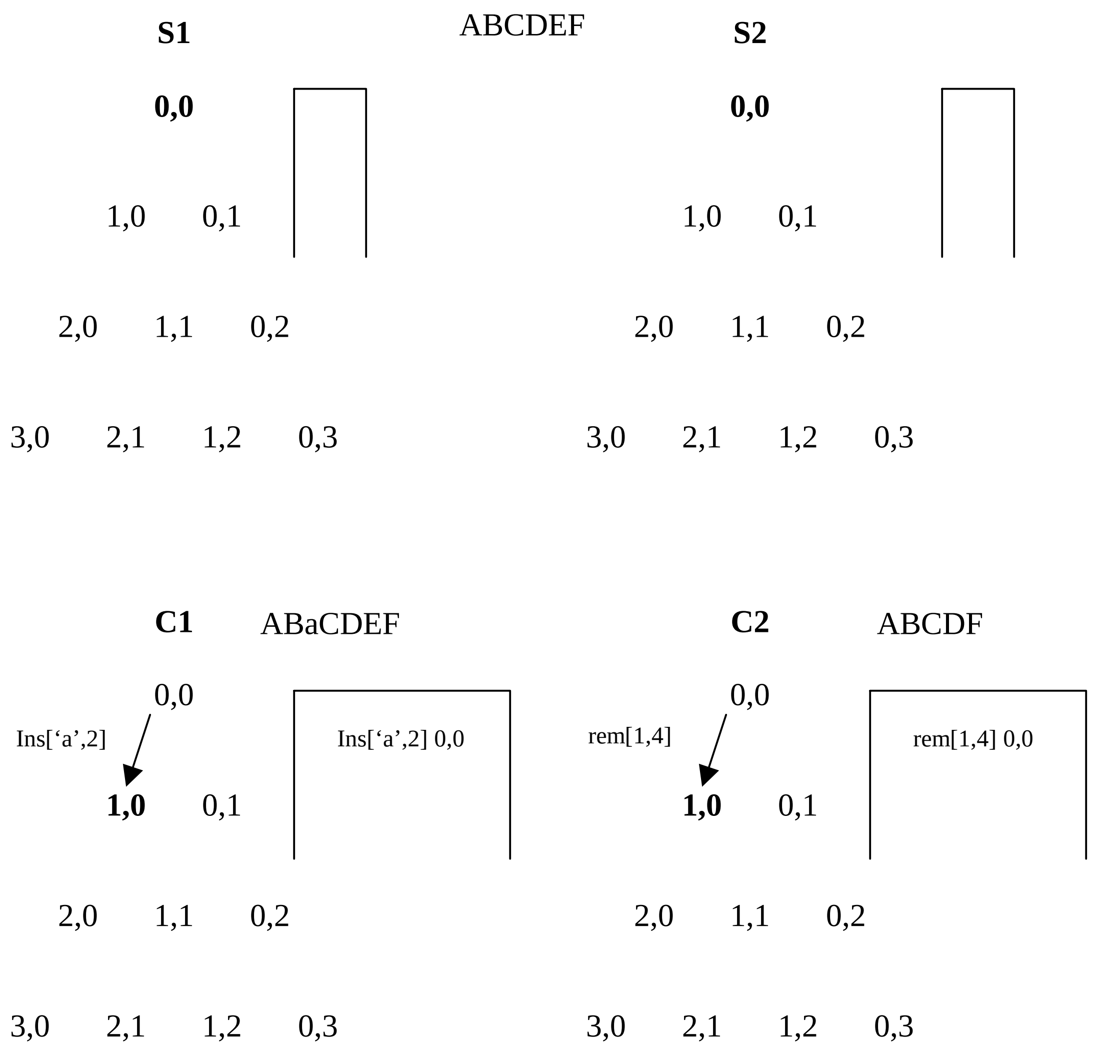


Figure 4.6: C1 and C2 perform an insert and remove operation respectively

The state of (1,0) of C1 and C2 implies that both have generated and processed 1 message of their own, and has received and processed 0 from the server. After applying to the local copy, both C1 and C2 send their operations to S1 and S2 respectively. It is to be noted that all local operations are buffered at the point of their generation and no remote operation is buffered.

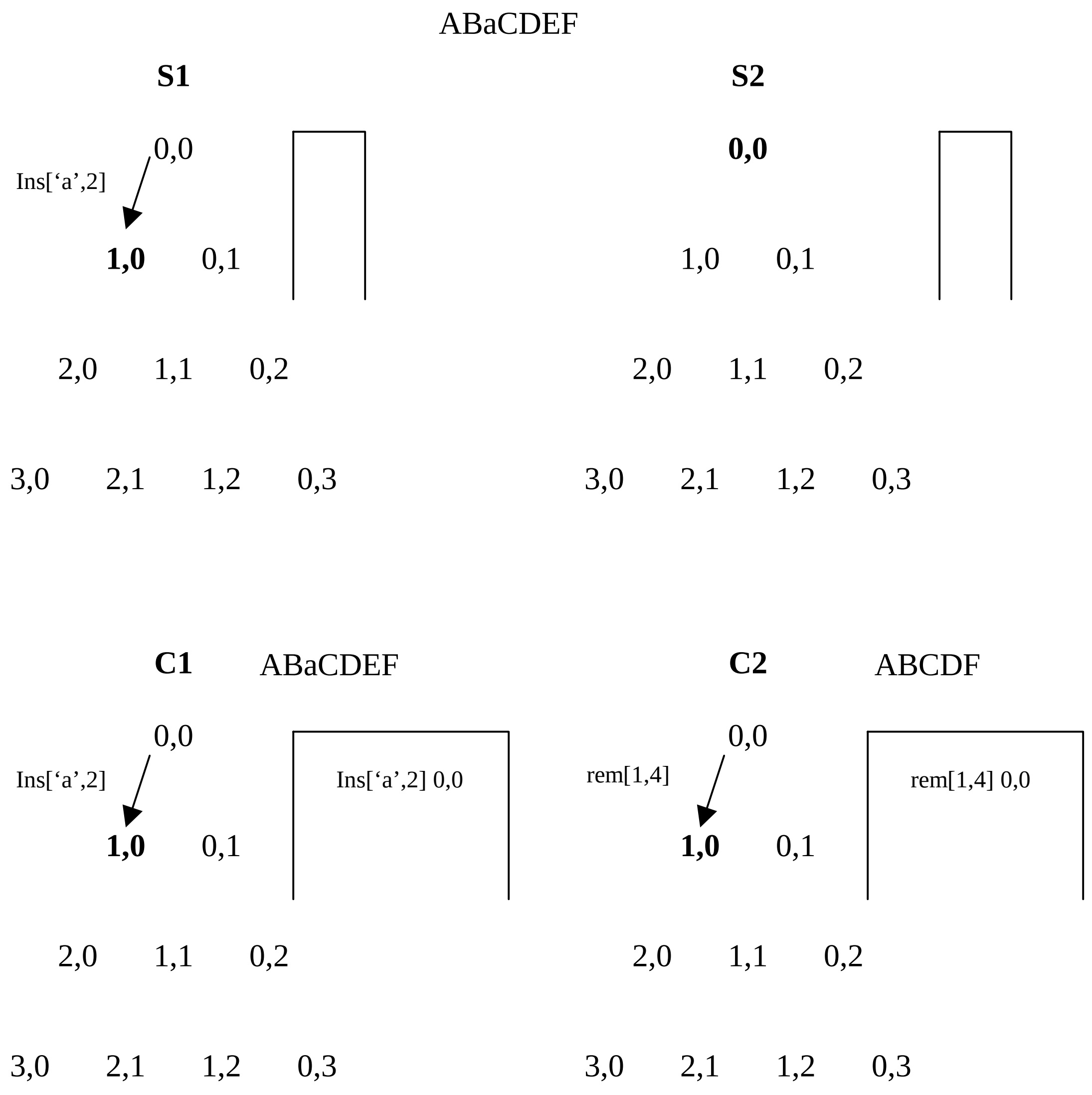


Figure 4.7: S1 receives and processes insert

All messages that are going from clients to server get serialized at the server and are processed in this sequence. Suppose that C1's message (ins['a', 2] 0,0) (henceforth called M1) reaches server before C2's message. The same final result would have been obtained if we had assumed otherwise. M1 is a remote operation for S1 with respect to C1. Hence M1 will not be buffered at S1. As can be seen from Figure 4.7, M1 causes S1 to go from 0,0 to 1,0 (generated and processed 0 messages of its own, and has received and processed 1 message from the client C1). M1 is first transformed with respect to operations in the buffer at S1, and then this transformed message is applied to the document and handed over to S2 for further processing and also to send it to C2.

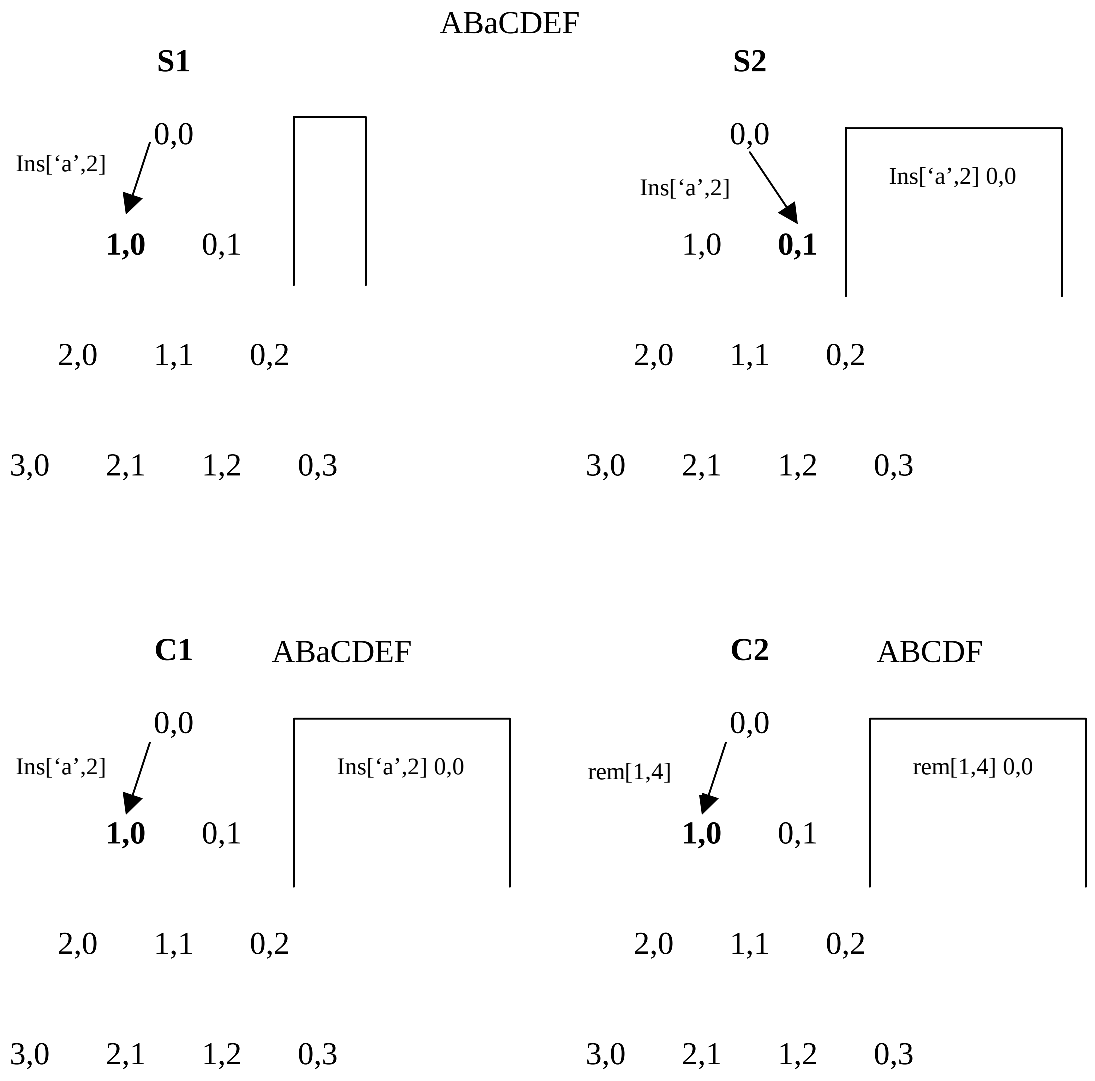


Figure 4.8: Insert processed by both S1 and S2

With respect to C2, M1 is a local operation at S2. Hence it causes S2 to go from 0,0 to 0,1 (generated and processed 1 message of its own, and has received and processed 0 messages from the client C2). Since M1 is a local operation at S2 with respect to C2, it is also buffered at S2 as shown in Figure 4.8. This message is then sent to C2.

Suppose the server now receives the operation from C2 (rem[1,4] 0,0 – henceforth called M2). M2 will first be transformed against all the messages in the buffer of S2 that have not been seen by C2. This is done by removing all messages, from the buffer of S2, whose server-component of their state is less than the server-component of the message. After this elimination, M2 will be transformed with all the remaining messages in the buffer. These are those messages that were executed in parallel with M2.

After transformation, M2 would become rem[1,5] (henceforth called $M2'$). While M2 was being transformed, the messages in the buffer are also transformed with respect to M2 and stored back into the buffer. The result is shown in the buffer of S2 in Figure 4.9. M2 is then applied to the document and passed on to S1 for further processing and also to send to C1. S2 now goes in to state 1,1 (generated and processed 1 message of its own, and has received and processed 1 message from the client C2).

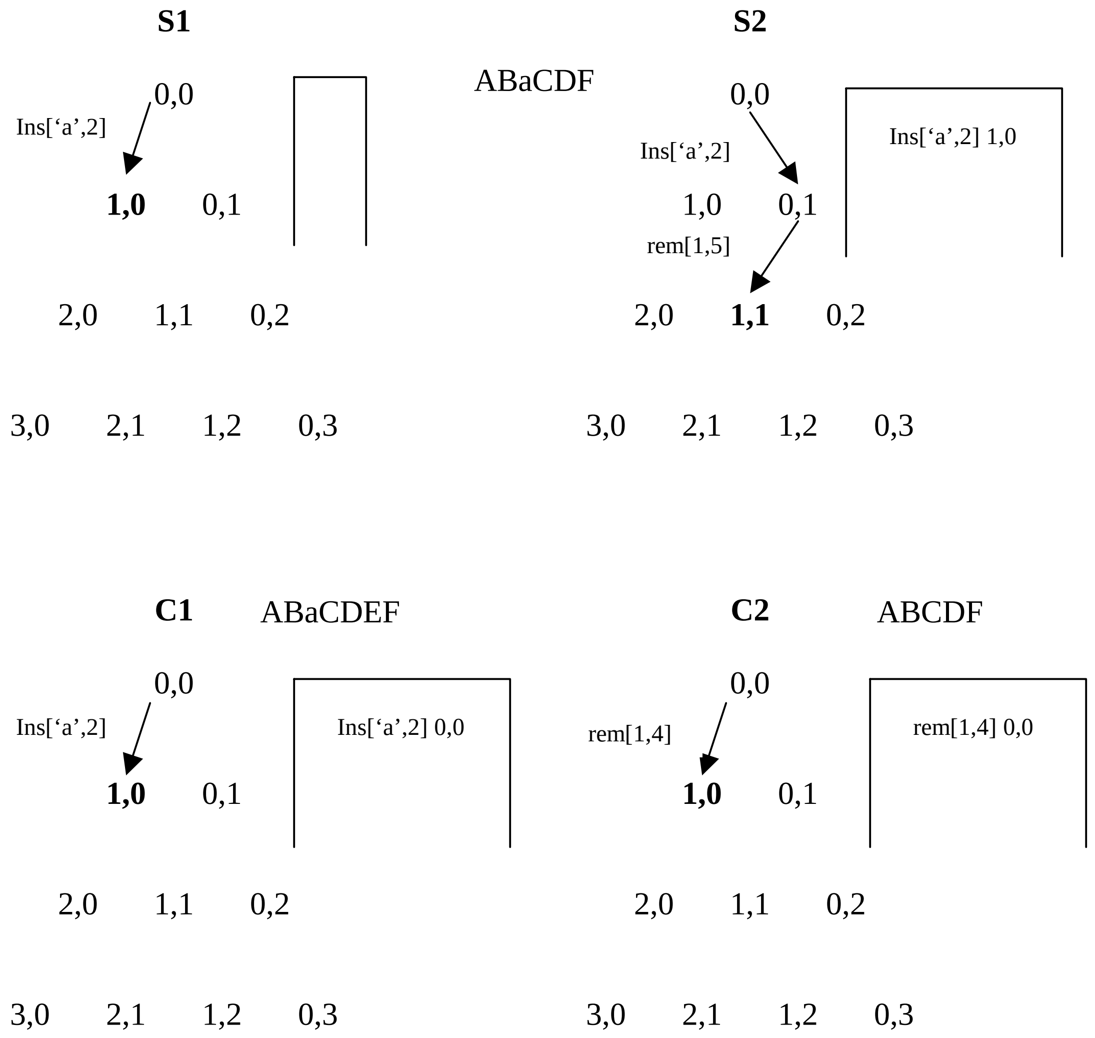


Figure 4.9: S2 receives and processes remove

Note that since $M2'$ was a remote operation for S2, it is not stored in its buffer. But for S1, it is a local operation as far as C1 is concerned. Hence it causes it to go from 1,0 to 1,1 (generated and processed 1 message of its own, and has received and processed 1 message from the client C1). It is also stored in its buffer and then sent to C1 as shown in Figure 4.10.

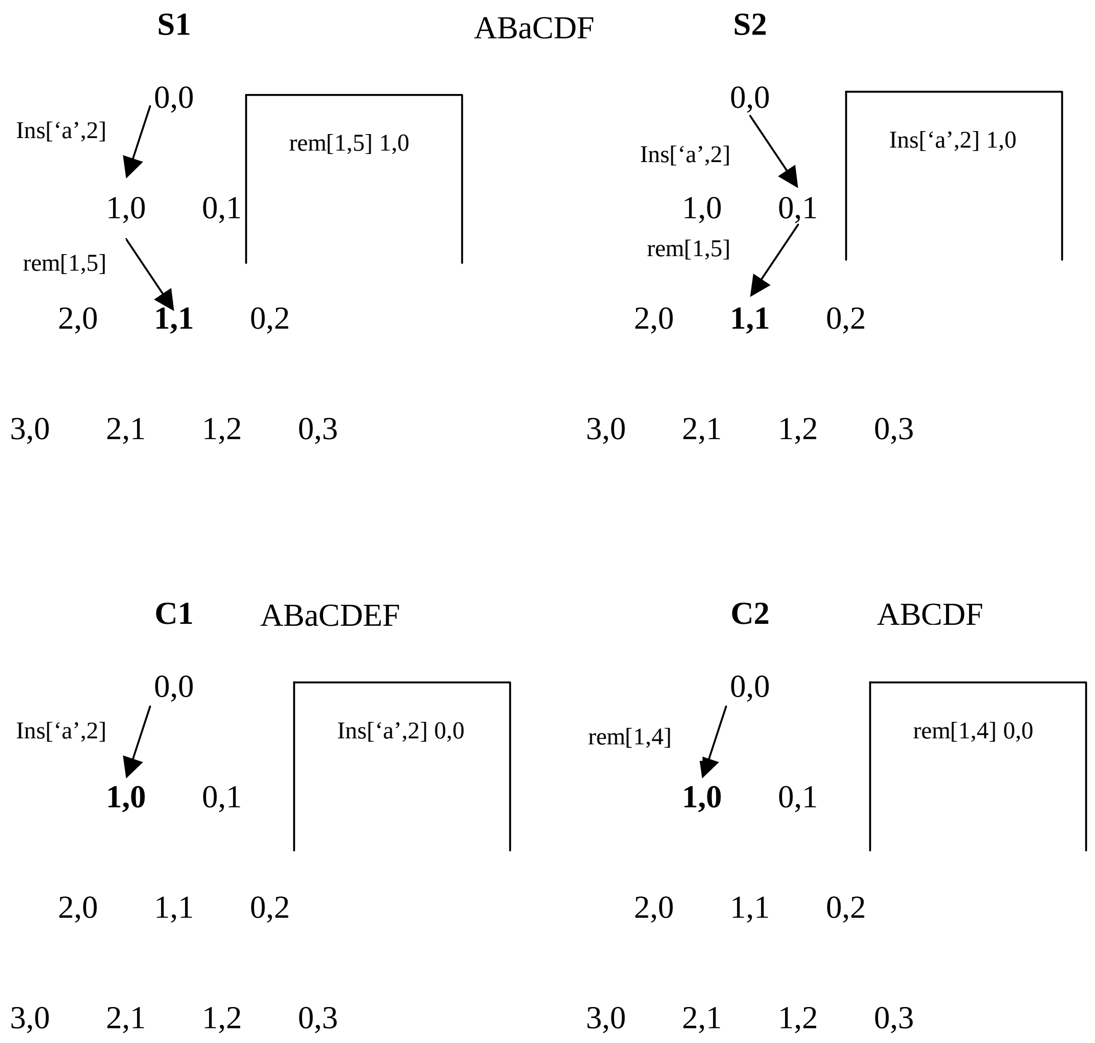


Figure 4.10: Remove processed by both S1 and S2

Note that the state information sent with $M2'$ to C1 is 1,0 and not 0,0 that it originally had when it was sent from C2 to S2. This is because $M2'$ is a local operation for S1 and the state that S1 was in when this local operation is generated is actually sent to C1. When C1 receives $M2'$, it does all the operations similar to those that S2 had done when it received M2 from C2. That is, it first removes all the operations from the buffer that have been seen by S1 because the effect of those operations is present in the received operation. This is to obtain all the operations that were executed in parallel to the received operation. On doing this, the saved message in C1’s buffer gets deleted since its client-component of the state is less than the client-component of $M2'$. This is shown in Figure 4.11.

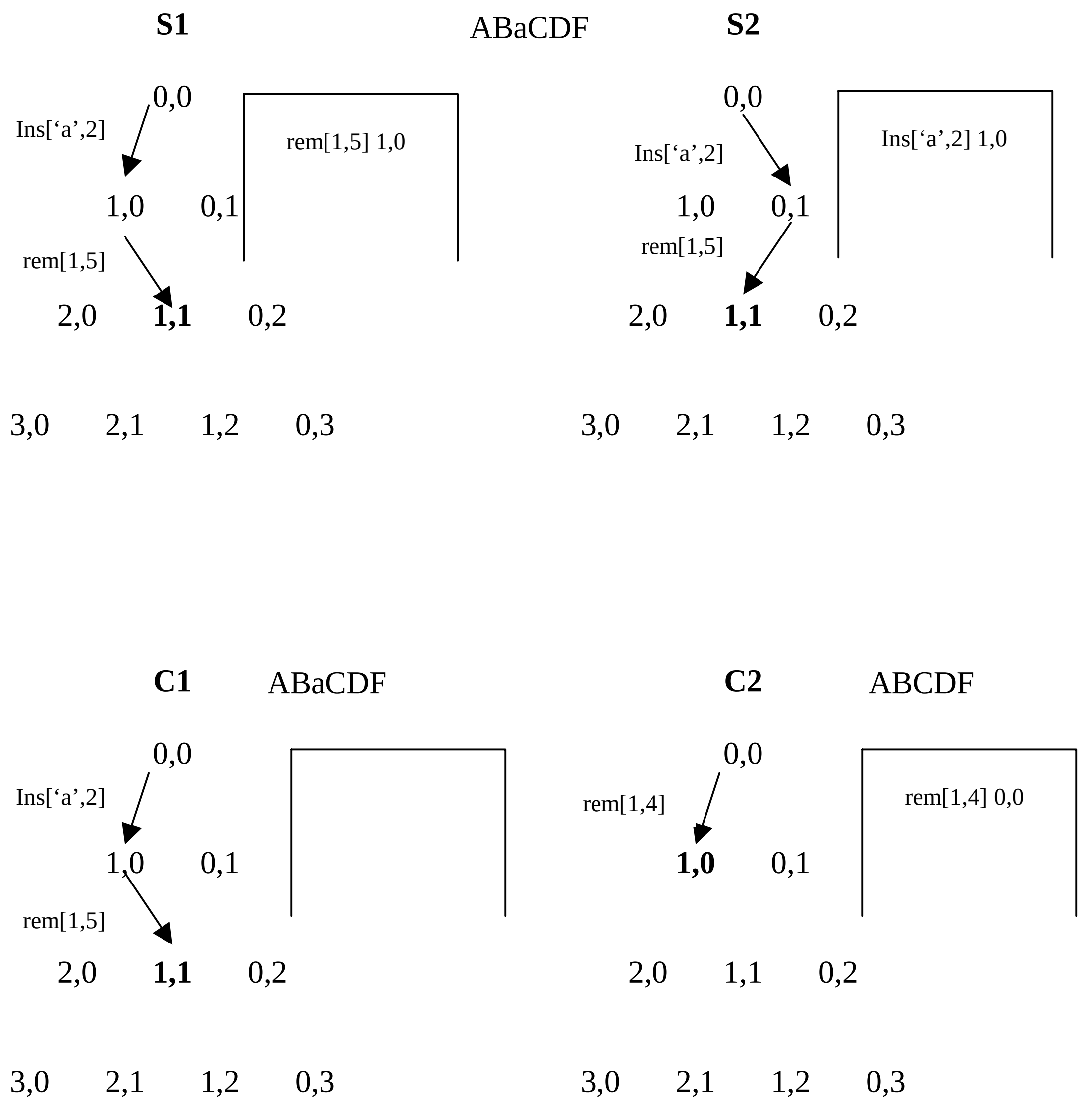


Figure 4.11: Remove processed by C1

Since there are no more messages left in the buffer, $M2'$ is applied directly to the local copy of C1. Since this was a remote operation for C1, it is not stored in its buffer. Also note that C1 now moves to state 1,1. To complete the entire cycle, suppose that the operation ins['a', 2] 0,0 (called as M1) sent by S2 to C2 is now received by C2. Once again, M1 will be transformed with messages in C2's buffer. Since M1's offset is less than M2's offset, it will remain unchanged. It is then applied to the document at C2. This causes C2 to go to state 1,1 as shown in Figure 4.12:

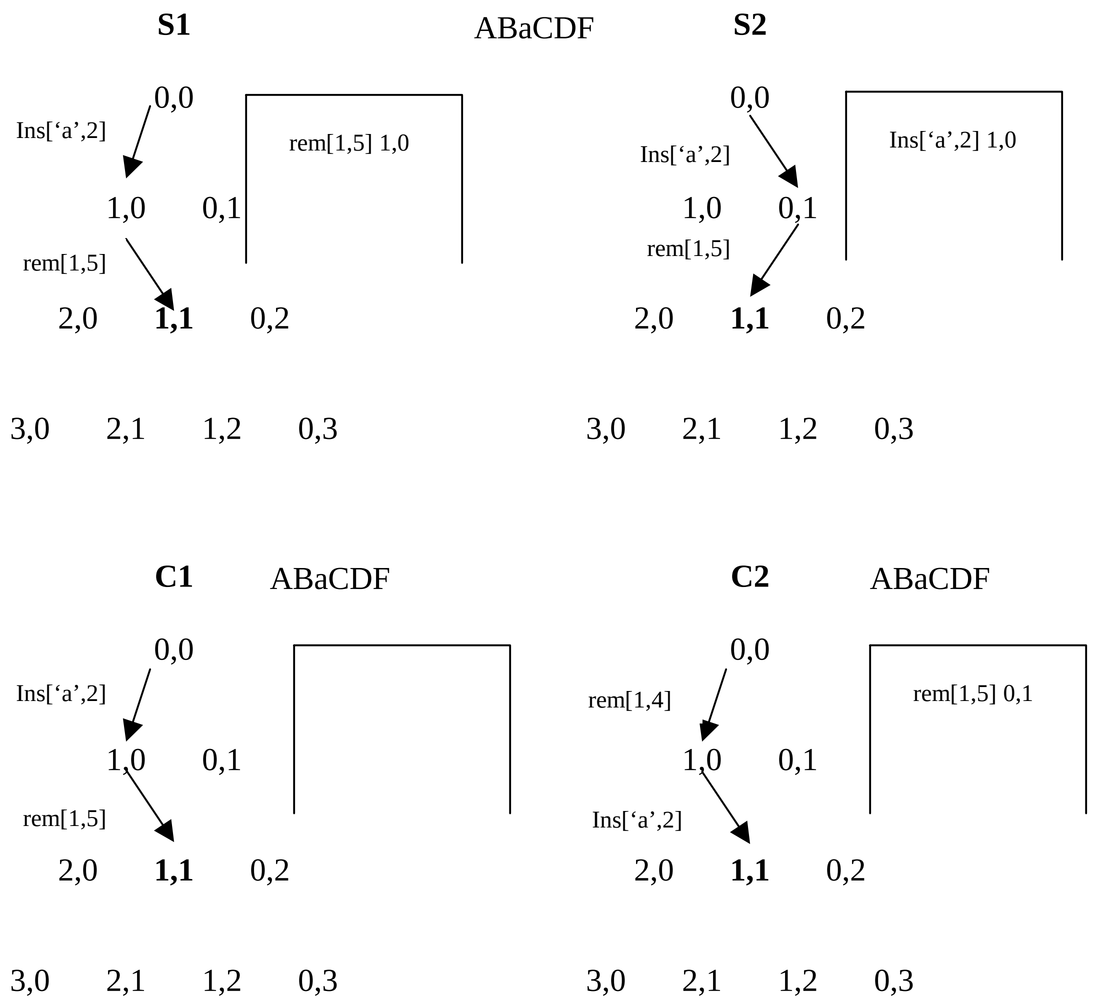


Figure 4.12: Insert processed by C2

All the parties are now in the same state and the document at each is also the same. This is a condition that occurs at quiescence when all the messages that have been generated are applied at all the clients. This implies that the first condition of the consistency model is satisfied. It is easy to observe that the way a client sends a message to other clients (i.e. through centralized server where they are serialized), the causality property between messages is preserved. Since a transformation engine does some bookkeeping and transforms multiple independent messages with respect to each other, the intention preservation property is also preserved. It is interesting to observe how S2 and C2 traverse different paths in the state graph, but ultimately converge to a common state. This is because they both execute operations in different order. The above example showed that the two were displaced from each other by just one operation, but in practical situations, they might deviate by multiple operations. The key that makes them converge to a common state is the buffering of operations at each site.

Chapter 5

Usability Study

This chapter describes our goals in the usability study for NetEdit, talks about the methodology for collecting the data, discusses the results, and finally derives conclusions from the analysis.

5.1 Usability goals

When users work in groups in the same physical space, they are aware of each other’s activities through what they see and hear. Since in a collaborative system the participants are geographically separated from each other, workspace awareness is an important functionality[15, 21, 24, 18] that should be supported. Telepointers and radarview are the primary widgets in NetEdit that provide this information to the users. Apart from external modes of communication like telephone etc., the participants in NetEdit communicate with each other through a chat window.

In our study, our goal was

1. To determine the efficacy of these awareness widgets during group editing, whether the users were able to correctly interpret their change, whether the users were distracted by them, etc.

2. To study the amount of use of the chat window and whether it was sufficient for communication during the group activity.

3. To determine the user-friendliness of other functionality in the system.

5.2 Methodology

We used observation, self-reporting, questionnaire and discussions to gather information for addressing our questions. This section describes the participant groups, the experimental setup and the task that the participants were asked to perform.

5.2.1 Participants

We did the experiment with three groups – two groups consisting of three participants each and one group consisting of four participants. One of the groups, with three participants, acted as pilot subjects to test our experimental setup, and task to be performed. The three participants in the pilot study were graduate students (one female and two males) in computer science. All other participants (males) were also majoring in computer science (six undergraduate students from an undergraduate level Human Computer Interaction class and one graduate student).

5.2.2 Experimental setup

Participants worked on Windows machines from separate closed rooms. The only way they could communicate with each other was through the chat utility provided in the system. An experimenter was assigned to each participant to observe his/her activities, and take notes from whatever he/she said during the course of the experiment. The participants were asked to speak aloud their intentions and any remarks they had while performing the assigned task. All their communication and editing activities were logged.

5.2.3 Task details

The task was to write a document no longer than 3 pages as a group. The topic of the document was their evaluation of the usability of the interface of NetEdit. The interface consists of 3 components – a Documents/Sessions window, a Chat window and an Editor window. Each user was responsible for doing the usability evaluation of one of these components. They had to decide amongst themselves, after they were assigned to their workstations, which component each of them would evaluate. Even though they were evaluating one window, the final document was to be organized based on the following eight characteristics.

1. Visibility, Mapping, Feedback
2. Did the responses from the system make sense
3. User control and freedom
4. Recognition rather than recall
5. Aesthetic and minimalist design
6. Ability to help users recognize, diagnose, and recover from errors
7. Online help and documentation
8. Gulf of evaluation and execution

Hence this exercise made them work in a loosely coupled manner when they were doing their own evaluation and in a tightly coupled manner when they were putting together all the details. The entire process was required to be completed within 45 minutes. All participants were familiar with the above usability principles.

5.2.4 Procedure

Participants were introduced to the entire experimental setup and asked to sign the consent forms. They were then given a demonstration of NetEdit, briefly describing its components, widgets and how they worked. Finally the task was explained to them and they were sent to their workstations in different rooms. Each participant was assigned an experimenter, who observed his/her activities and noted any remarks they said aloud. The participants were asked to speak aloud their activity/intentions while performing the task.

After around 45 minutes, participants were given a questionnaire that analyzed their experience with the system. Some responses in the questionnaire and any eccentric activity we observed were then discussed briefly with them.

5.3 Results

This section discusses the notes taken by the experimenters during the editing session, some of their observations, the evaluation document produced by the participants, and the responses in the questionnaire.

5.3.1 Notes/General observation

Eight out of ten participants had not used a collaborative tool before. The participants started by playing around with the system. They liked radarview and also found it useful during the editing process. Initially when they were concentrating on their own evaluation, radarview was used often to determine what other participants were doing. Telecursor gained importance when the groups started combining their work together. In fact, one of the participants kept watching the telecursors of other users to find out what they were doing and infer their intentions. He remarked that it was difficult during editing to have to continually switch back and forth between the chat and editing window. He further suggested that a voice channel for communication would have been more effective. These observations were substantiated by the responses from the questionnaire and the brief discussion we had with the participants after the experiment.

Two different styles of working were observed. One of the groups with three participants made sections in the edit window and each were working in their own section. But they took some time to decide on this organization. They were allowed to communicate only through NetEdit – Chat window. The frustration of not being able to get themselves organized was clearly indicated by their messages in the chat window. They had initially started preparing the document as if they were using a single user editor. And it took them a little while to realize that in order for the group activity to be effective, they needed to work a little differently. It clearly showed that after some learning of not only how to use a collaborative tool but also how to work remotely in groups, the session can be productive. It all came down to establishing a social protocol for editing the same document.

The editing started getting chaotic when they began combining their evaluation into a single coherent document. Some participants tried to move their work closer to the work of other participants without realizing that even these other participants were trying to do the same thing. Thus all of them were trying to move their text in between the text of other users. Although there were a lot of chat messages between them so that they could get synchronized, it seemed that communicating using chat window was too slow. By the time a user typed some chat message and returned to the edit window, the state of the document was changed. Hence his comments were no longer valid. This caused a lot of irritation between participants. Also, to avoid having to switch back and forth between a chat window and an edit window, they started communicating using the edit window itself. Section 5.3.2 discusses the usage of chat window by the two groups in more detail. The document now started to look a little confusing, as lot of chat messages got inserted between their texts. They needed a high bandwidth communication mode like audio conferencing tool within the system. However at the end of the session, they were able to produce a document that was a little unorganized but had some valuable comments. Some of these comments are discussed in section 5.3.3.

We had explicitly imposed a structure on the group with four users. One of the participants was to act as a director overlooking the activities of all the participants and guide the preparation of the document. However, this group could not get oriented and instead focused on what they were doing. One of the participants in this group felt a need for the system to provide tools to automatically know the intentions of his group members. It seemed to us that since there were four participants (a bigger group than other groups), identifying and synchronizing the activity was more difficult. And the slow mechanism for communication (chat messages) exacerbated the problem.

One of the problems that three users complained of was quite interesting and a good research issue. This system tried to preserve the intentions of the users as explained in section 4.2.3. Hence whenever there were characters inserted or deleted from the part of the document above the point where a user is working, the user notices a sudden movement of his cursor location. This movement is especially significant when a new line character is inserted. This is because the entire line of the user along with his cursor suddenly goes to a new line. When this happens, he looses the context of his surrounding text and gets confused. We thought this could be solved in two ways. First is that of a gradual change of his position so that the sudden movement is avoided. This means that the incoming remote operations might be processed further before applying to the document. One could use the cloud burst model[15] for this. In cloud burst model, the local operations appear in the window immediately but a cloud appears over remote operations. The text associated with these remote operations is then progressively revealed as the cloud starts fading. Secondly, the JScrollPane and JTextPane widgets that contain the document could be modified or rewritten so that they grew both ways instead of just growing in one direction (bottom). Thus if characters are inserted above the local users cursor position, the document would grow upwards but if the characters are inserted below the local users cursor position, the document would grow downwards.

5.3.2 Chat usage pattern

Figure 5.1 shows distribution of the number of chat messages against time, during the experiment. The time on the graph is divided in five-minute intervals. Thus Group 1 had three messages passed around during the first five minutes and seven messages during the next 5 minutes. Group 1 had three participants and Group 2 had four participants.

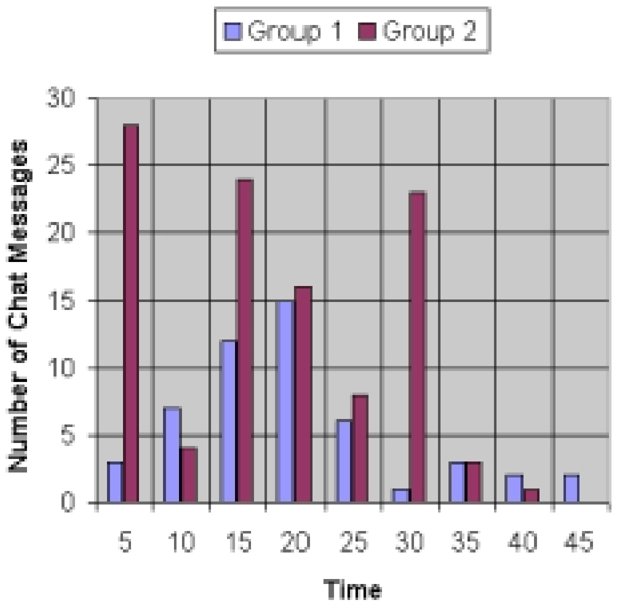


Figure 5.1: Chat usage graph

Group 1 started by playing for a little while with NetEdit; exploring its widgets, going through different options on various windows etc. There were messages for deciding who would write what and how they would later combine their evaluation. As they proceeded with the activity, the number of chat messages kept increasing for the first 20 minutes. After this, they began to combine their work and contrary to our notion, the chat activity started decreasing significantly. This is because the users were getting annoyed by continually having to switch back and forth between the chat window and the edit window. They preferred to communicate with each other through the edit window itself. That explains the low activity in the chat window during the latter 20 minutes of the experiment.

Group 2 started by discussing with each other how they would get organized. This is explained by heavy activity in the chat window during the first five minutes. Then they went on to exploring NetEdit – its features, different windows etc. Even with this group, we see a dip in the number of messages during the latter part of the experiment for the same reason explained above. This group was a little unorganized. They continued to chat for extended periods then shift to doing only editing and then again come back to only chatting and so on. They emphatically felt the irritation of switching between chat and edit windows.

5.3.3 Evaluation document

The document produced by the two groups was quite unorganized. The participants spoke aloud what they were about to do, their thoughts, assumptions, etc. during the entire experiment. The experimenters captured almost all this feedback during the editing session.

Some important comments that the document had are “green color is hard to see”, “the rename icon is confusing… it needs a better image”, “This window does not refresh itself fast enough”, “sometimes the people's cursors sit on top of what I am trying to write and it makes it hard to see”, “wonder what happens when you delete a file that people are working in…haha…kicked everyone out”, etc.

The participants enjoyed working with NetEdit, particularly because for many of them this was their first experience with a collaborative tool. The comment in the document “sometimes the people's cursors sit on top of what I am trying to write and it makes it hard to see” was quite noteworthy. Two participants complained about getting distracted by the telecursors. However, both of them felt that it provided important information and suggested that it be customizable according to the look and feel of the user’s choice.

5.3.4 Questionnaire responses

The participants felt that the response time of the editor was 3.0 on a scale of 1 to 5 with 1 being extremely slow and 5 being very fast. They had mixed reactions over the efficacy of telecursors and radarview. Some preferred telecursors while others thought radarview was more useful. But they definitely wanted both of them. These widgets seemed to distract a few of them (2 participants), while for others (8 participants) it complimented their editing activity. Six participants felt the need to color-code the text with the color assigned to them, to identify who wrote what text; but they also wanted an option to turn the coloring off.

The amount of use of chat utility was 3.2 on a scale of 1 to 4 with 4 being extensive usage and 1 being not used at all. This rating is amply substantiated by our observations during the experiment. The chat system has three modes of sending a message to the remote users of the system. This was to address the issue of the users getting overwhelmed with messages. Although the participants felt that it was important to have these modes, they did not like the coloring scheme used for distinguishing different types of messages. They were confused between the coloring scheme of these messages and the colors in the edit window. They did not realize that both were independent of each other.

Their overall experience with the system was 3.2 on a scale of 1 to 5 with 1 being completely confusing and 5 being extremely intuitive. The first thing they wanted to be changed was a better and faster communication mechanism. They felt that their group activity would have been much better if they were able to communicate as they do while working from the same physical space.

5.4 Analysis

This section analyzes the data we collected during the usability experiment of NetEdit.

The coloring scheme in the chat window and edit window needs to be made more intuitive. One could have more than one chat window – say one global where every one can communicate with each other and the other local to each session attached to the edit window. The chat window specific to a session could color-code the messages with the color assigned to the participants in that session. This would resolve the difficulty of identifying which message came from whom.

As the users insert text in the edit window, it should be colored with the color assigned to them. However there should also be an option to turn this coloring off. The need for this mechanism was realized during the observation and discussion with the participants. Whenever the participants wanted to communicate with each other about certain text written by someone, they had to first identify who wrote that text, and then talk about that text. The earlier round of messages for discovery simply added to the volume of messages being passed around without serving any useful purpose.

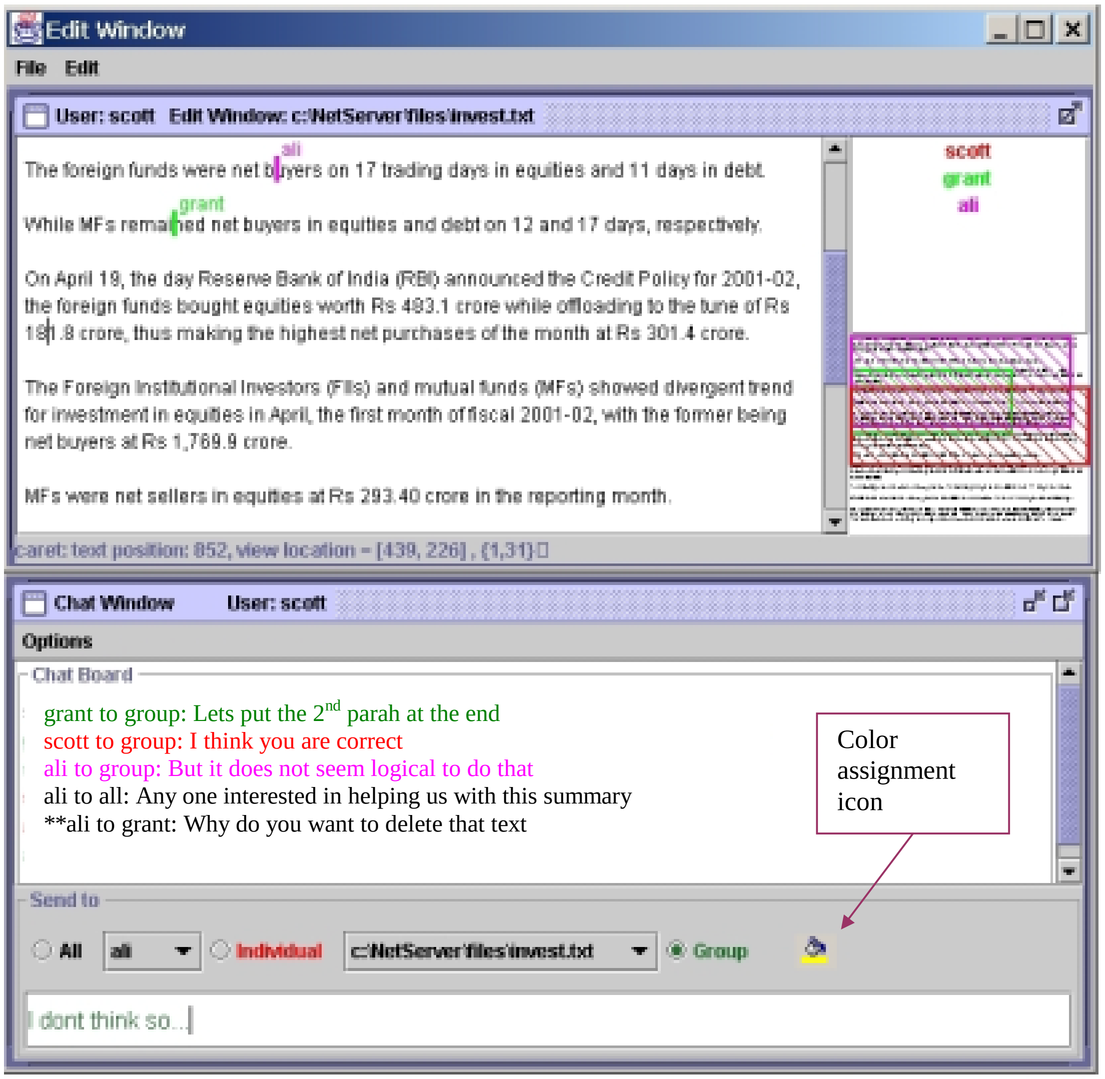
There was an earnest need for a faster communication mechanism between participants. One quick solution to this problem that was suggested is to have another chat window, attached to the edit window, specific to each session. This would avoid having the users to continually switch back and forth between typing some text in the edit window and typing messages in the chat window to communicate with the participants in their session. The other option was to incorporate an audio channel in NetEdit for communication between participants.

A gradual change in the cursor position of a user when characters (especially new line character) get inserted before his location in the document. Two solutions to this problem were discussed in Section 5.3.1.

5.5 Redesign of the interface

This section does a brief redesign of the interface to address some of the usability concerns discovered during the experiment. Figure 5.2 shows the edit window modified to now incorporate a chat window. The chat messages appear in the color of the participant who sent them. The chat “send to” control panel also displays an icon for assigning colors to the participants in a session. The messages that are addressed to all appear in black in the chat window. Individual messages have two asterisks before them.

Thus there is a chat window associated with each session. The user is given the flexibility of assigning the color he/she likes, to each remote participant. This might avoid confusion between the colors in the edit window and those in the chat window as they are now controlled by the user and are not independent of each other as was the case earlier. Since now the chat window is below the edit window, there is no switching between windows. They both appear, in front, in the same parent window. And since there is a chat window specific to each session, message overload problem is also addressed – as chat messages belonging to a session will only appear in the chat window associated with that session.



The foreign funds were net buyers on 17 trading days in equities and 11 days in debt.

While MFs remained net buyers in equities and debt on 12 and 17 days, respectively.

On April 19, the day Reserve Bank of India (RBI) announced the Credit Policy for 2001-02, the foreign funds bought equities worth Rs 483.1 crore while offloading to the tune of Rs 181.8 crore, thus making the highest net purchases of the month at Rs 301.4 crore.

The Foreign Institutional Investors (FIIs) and mutual funds (MFs) showed divergent trend for investment in equities in April, the first month of fiscal 2001-02, with the former being net buyers at Rs 1,769.9 crore.

MFs were net sellers in equities at Rs 293.40 crore in the reporting month.

caret: text position: 852, view location = [439, 226] , (1,31)

grant to group: Lets put the 2nd parah at the end
scott to group: I think you are correct
ali to group: But it does not seem logical to do that
ali to all: Any one interested in helping us with this summary
**ali to grant: Why do you want to delete that text

I dont think so...

Figure 5.2: Redesigned Edit Window

The text inserted by the remote participants in the edit window can be colored with the color assigned to them. However, one might provide a button or a menu option to turn this coloring off. This is not to say that one must preserve the coloring over the lifetime of the document but should manage the coloring for an active session.

The above said features are quite simple to implement, as there is no global information that needs to be maintained. It requires modification only at the client's end.

Incorporating an option for an audio mode can significantly improve the communication between remote participants. However, this might require some effort both at the server and at the client. A balance would also need to be determined since many users doing audio communication could cause a lot of confusion.

Chapter 6

Conclusion

This chapter summarizes the benefits of computer-based collaboration for group editing, discusses the contribution of this work and suggests ideas for future research.

6.1 Benefits of computer-based Collaboration for group editing

Due to the explosive growth of Internet and pervasive use of computers, it is important to explore ways to use computers for group activities. Shared document editing could be important for organizations that are global in nature and have geographically distributed teams. Instead of passing a document from one team member to another for review, all of them could brainstorm simultaneously. This is a cost-effective tool that can save travel time and make a team more productive.

This tool also provides a platform for studying how people work in groups. This study could lead to the creation of group work environments that result in more productive activity from group members.

6.2 Contribution of this work

The research in collaborative editing systems can be categorized into (1) application level – where the focus is coordination and consistency management between remote software components, and (2) human level – where group dynamics, collaborative widgets etc. are studied in detail.

This thesis has primarily focused on the application level, although a preliminary usability study of the groupware widgets and group dynamics is also performed. A number of distributed algorithms [8, 36, 7] for achieving consistency between collaborators exist. These algorithms do not scale, as the number of participants grow and are also quite complex. On the other hand, Jupiter collaboration system [33, 7] discusses a 2-way synchronization protocol for group editing. As suggested by them, we explored the details of extending this 2-way protocol to n-way synchronization protocol. We implemented a prototype in Java 2.0 (jdk1.3.0), NetEdit, to demonstrate the mechanics of our extension.

6.3 Future Work

This research has brought to light many future ideas categorized into software areas and usability areas. The software area relates to improvements in fault tolerance, efficiency, firewall issues etc., while usability area discusses ideas about studying group dynamics during group activity, the semantics of version control, etc. Although we have this categorization, the ideas discussed are inter-dependant and require acumen from both software and usability domain.

6.3.1 Software Areas

- **Fault Tolerance**: The system has a centralized architecture with the processing distributed between clients and the server. However, if the server fails, then the entire system comes to a halt. Thus there is no secondary server that could mirror all the data and operations, and dynamically replace the faulty server. It will be interesting to explore this possibility, keeping in mind that while the switch is taking place, there could be operations being performed by the clients. Also, it needs to be researched whether the existing algorithm will be able to scale to this requirement or if it would require significant fundamental changes.

- **Algorithm Extension**: The core algorithm, described in section 4.7, that manages consistency of the document at all the clients processes single characters. However, it is seen that multiple characters inserted or removed in succession, in a short period of time, go through the same kind of transformation. Hence it will be interesting to modify this algorithm so that strings of arbitrary length, instead of single characters, can be transformed. This might potentially increase the speed, as lesser number of operations now needs to be processed, and also reduce the memory space required, as lesser number of operations would be needed to be stored in the buffers.

- **Improving Efficiency**: While implementing this prototype, we noticed two bottlenecks that affected the performance of the system. One is the bandwidth used for passing messages between clients. There are large numbers of small messages that are being broadcasted to the clients. These messages include operations being performed by the users, awareness information such as change in caret position etc. One could aggregate some of them and thus potentially improve the bandwidth utilization. However, one needs to explore the optimum point beyond which the response time becomes noticeable by the remote users. The other bottleneck is the slow display refresh mechanism of the java virtual machine. Some of the widgets where this was noticeable were JTextPane (which holds the document being edited), and JScrollPane (which provides scrolling support). It will be interesting to explore ways to address this issue.

- **Supporting Large Documents**: The JTextPane widget that holds the document during editing has a limitation that it can only support files that are a couple of megabytes long. For bigger size documents, it stops responding and behaves abnormally. Reading only a window's worth of text at a time from a larger document and displaying it could potentially overcome this limitation. However, application level caching might be required, by exploiting the locality of reference property, to reduce the latency involved in frequent disk accesses.

- **Firewall and Security**: Firewalls can be configured to allow certain specified traffic while preventing other traffic to pass through. NetEdit allocates ports from a pool of free ports non-deterministically. Thus one does not have control on what ports or range of ports could be used. This causes access problems if one is using NetEdit from behind a firewall. It will be important to explore ways to solve this and make NetEdit operate in versatile environments. NetEdit currently has a single password file that stores the login ids and passwords of users. One could replace this with more robust and fine-grained security and authentication capabilities. The channels of communication between the clients and the server could be encrypted using state of the art encryption algorithms.

- **Editor features**: There is support for cut, copy and paste operations during editing. However, it is important to support other features like page or paragraph alignment, different styles and sizes of font, etc. It might be interesting to import images in to the document and allow them to be manipulated collaboratively. The system provides two groupware widgets – telecaret and radarview for providing awareness information during text editing. It will be interesting to investigate whether these widgets would suffice during image manipulation or one would require a fundamentally new set of groupware widgets.

- **Communication**: Our preliminary usability study of NetEdit revealed that the existing communication mechanism between participants through a chat window is extremely slow and unacceptable. The various options to address this issue need to be explored. For example one could provide an audio channel between participants in the same session and have a chat window for communicating with participants in other sessions etc. These two modes are suggested to avoid the user from getting overwhelmed by audio messages.

- **Web Enabling**: Currently, the system is an application written in Java 2.0 (jdk1.3.0). One could web enable this system by converting to applets or through some other mechanism taking into account the Java security model.

6.3.2 Usability Areas

- **Version Control and Undo facility**: Keeping track of different versions of the document and reverting to an earlier version can be an important utility. However, it is difficult to determine, to which version one must roll back. Is it the version where only the operations done by the local user must be reverted or is it the version where both local and remote operations need to be undone. It really depends on the type of group activity. If it's tightly coupled collaboration – like brainstorming a paper where all the participants are working very closely, then one might want to revert to a previous version undoing both local and remote operations. However, if the collaboration is loosely coupled, where the participants are working in different parts of the document, one might want to rollback to a previous version undoing only local operations. This was our observation during the usability study of the system and needs to be more thoroughly studied. It is interesting to realize that this might be more of a social issue than a technical problem.

- **Text Coloring Scheme**: The system does not color the inserted characters with the local color of the participant. It is interesting to explore whether this coloring is necessary and if it allows more awareness information to be shared between users. One might also study whether it is necessary to retain this coloring over the lifetime of the document or it should go away after each session. We assign an unused local color to each remote participant during an editing session. Thus the same participant might have different colors in different sessions. Is this confusing? Is there a need for consistent global color assignment so that the participant has the same color in all the sessions? It might be interesting to explore these questions more thoroughly by a comprehensive usability study.

- **Studying Group Mechanics**: Since the system does not assume or impose any protocol for group activity, the participants can form the editing structure that suits them. These structures could be decided explicitly at the beginning of the session or could be formed implicitly as the group activity proceeds. It might be interesting to observe this behavior with different types of groups collaborating in a tightly coupled or in a loosely coupled manner. The system logs all the activities including the messages exchanged by the users. This log could be used to replay the entire group activity and perform further usability analysis.

References

1. Baecker, R.M., Nastos, D., Posner, I.R., and Mawby, K.L. “The User-centred Iterative Design of Collaborative Writing Software”, Proceedings of InterCHI'93, ACM, 1993, 399-405, 541.

2. Bardram, Jakob, “Designing for the dynamics of cooperative work activities”, Proceedings of the ACM, CSCW 1998, Pages 89 – 98.

3. Begole J., “Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application – Sharing Systems ”, PhD Thesis, Virginia Polytechnic Institute and State University, Department of Computer Science, 1998.

4. Begole J., Craig A. Struble, Clifford A. Shaffer, and Randall B. Smith, “Transparent Sharing of Java Applets: A Replicated Approach”, Proceedings of the ACM, UIST, 1997, Pages: 55 – 64.

5. Begole, James , Rosson, Mary Beth, and Shaffer, Clifford A., “Supporting Worker Independence in Collaboration Transparency”, Proceedings of the ACM, Symposium on UIST 1998, Pages: 133 – 142.

6. Chabert Annie, Ed Grossman, Larry S. Jackson, Stephen R. Pietrowiz and Chris Seguin, “Java object-sharing in Habanero”, Communications of the ACM, June 1998/Vol. 41, No. 6, Pages 69 – 76.

7. Chengzheng Sun, Clarence (skip) Ellis, “Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements”, Proceedings of the ACM, CSCW 1998, Pages: 59 – 68

8. Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang and David Chen, “Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems”, ACM Transactions on Computer-Human Interaction, pages 63-108, March 1998.

9. Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok and James H. Morris, “Computer Support for Distributed Collaborative Writing: Defining parameters of Interaction”, Proceedings of CSCW ’94 Computer-Supported Cooperative Work (Chapel Hill, NC, USA, October 1994).

10. Dewan Prasun and Choudhary Rajiv, “Flexible User Interface Coupling in a Collaborative System”, Proceedings of ACM CHI 1991, Conference on Human Factors in Computing Systems. Pages 41 – 48.

11. Dewan, Prasun, “Architectures for Collaborative Applications”, In Beaudouin-Lafon, M. (Ed.), Computer Supported Cooperative Work, Trends in Software Series 7, John Wiley and Sons Ltd. 1999.

12. Dourish, Paul, and Bellotti, Victoria, “Awareness and Coordination in Shared Workspaces”, CSCW November 92 proceedings of ACM.

13. Du Li, Muntz, Richard, “COCA: Collaborative Object Coordination Architecture”, Proceedings of the ACM, CSCW 1998, Pages 179 – 188.

14. Ellis, C. A., S. J. Gibbs, “Concurrency Control in Groupware Systems”, Proceedings of the ACM, SIGMOD Conference on Management of Data, May 1989, Pages: 399 – 407.

15. Ellis, C.A., Gibbs, S.J., and Rein, G.L. “Groupware: Some Issues and Experiences”, CACM 34(1), 1991, 38-58. Reprinted in Baecker, R.,M. Readings in Groupware and Computer-supported Cooperative Work: Facilitating Human-Human Collaboration, Morgan Kaufmann, 1993.

16. Fish, Robert S., Kraut, Robert E., Leland Mary D. P., and Cohen, M., “Quilt – A Collaborative Tool for Cooperative Writing”, Proc. COIS’88 Office Information Systems (Palo Alto, CA, March 1988).

17. Greenberg, Saul and Mark Roseman, “Groupware Toolkits for Synchronous Work”, In Beaudouin-Lafon, M. (Ed.), Computer Supported Cooperative Work, Trends in Software Series 7, John Wiley and Sons Ltd. 1999.

18. Greenberg, Saul, Carl Gutwin and Mark Roseman, “Semantic Telepointers for Groupware”, Proceedings of OzCHI November 1996 Sixth Australian Conference on Computer-Human Interaction.

19. Grudin, Jonathan, “Eight Challenges for Developers”, Communications of the ACM, January 1994, Vol 37, No. 1, Pages: 93 – 105.

20. Gutwin, C., and Greenberg, S. “Workspace Awareness for groupware”, CHI 96, Conference Companion (Vancouver, 1996), 208-209

21. Gutwin, C., Greenberg, S. and Roseman, M. (1996). “Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. in Sasse”, R.J., A. Cunningham, and R. Winder, Editors. People and Computers XI (Proceedings of the HCI'96), pages 281 - 298, Springer-Verlag. Conference held at Imperial College, London, August 20-23.

22. Gutwin, C., Greenberg, S., & Roseman, M., “Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation”, People and Computers XI (Proceedings of the HCI'96) Springer-Verlag, Pages 281-298.

23. Gutwin, C., Greenberg, S., and Roseman, M. “Workspace Awareness support with Radar Views”, CHI 96, Conference Companion (Vancouver, 1996), 210-211

24. Gutwin, C., S. Greenberg and M. Roseman, “A usability study of awareness widgets in a shared workspace groupware system”, Proc. CSCW, ACM Press, New York, November 1996, Pages 258 - 267.

25. ITeach, A Collaborative editing environment, at Institue of Advanced studies in Humanities.
URL: http://jefferson.village.virginia.edu/ITeach/

26. Koch, Michael, Jürgen Koch, “Using Component Technology for Group Editors - The Iris Group Editor Environment.” Proceedings of Workshop on Object Oriented Groupware Platforms, Lancaster, UK, G. H. ter Hofte, H. J. van der Lugt(ed.), September 1997.

27. Lamport, L, “Time, clocks and the ordering of events in a distributed system”, CACM 21(7), Pages: 558 – 565, July 1978.

28. Leland, Mary, Fish, Robert and Kraut, Robert, “Collaborative Document Production Using Quilt”, Proc. CSCW ’88 Computer-Supported Cooperative Work (Portland, Or., September 1988).

29. Microsoft NetMeeting, April 2001.
URL: http://www.microsoft.com/windows/netmeeting/default.asp

30. Mitchell, A., “Communication and Shared Understanding in Collaborative Writing”, MS Thesis, University of Toronto, Department of Computer Science, 1996.
URL: http://home.pacific.net.sg/~kamitchell/portfolio/calliope.html

31. Mullick, Sachin, Raphael A Finkel, MUSE, A Collaborative editor. Masters Project. University of Kentucky.
URL: http://www.cs.engr.uky.edu/~raphael/studentWork/muse.html

32. Network Text Editor, an application developed at Networked Multimedia Research Group, University College London.
URL: http://www-mice.cs.ucl.ac.uk/multimedia/software/nte/

33. Nichols, David A., Pavel Curtis, Michael Dixon, and John Lamping, “High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System”, ACM, UIST 1995, pages: 111 – 120.

34. Patterson J. F., Day M. and Kucan J. “Notification servers for synchronous groupware”, Proceedings of the ACM, Conference on Computer Supported Cooperative Work, 1996.

35. Prakash, Atul, Michael J. Knister, “Undoing Actions in Collaborative Work”, Proceedings of ACM on Computer Supported Cooperative Work 1992, Pages 273-280

36. Ressel, Matthias, Doris Nitsche-Ruhland, and Rul Gunzenhauser, “An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors”, Proceedings of the ACM, Conference on Computer Supported Cooperative Work, 1996, Pages: 288-297.

37. Schlichter, Johann, Michael Koch, Martin Bürger, “Workspace Awareness for Distributed Teams”, Proceedings of Workshop Coordination Technology for Collaborative Applications, Singapore, Wolfram Conen(ed.), 1997.

38. Stefik, M., D. G. Bobrow, G. Foster, S. Lanning and D. Tatar, “WYSIWIS revised: early experiences with multiuser interfaces”, ACM transactions on Office Information Systems, April 1987, Pages 147 – 167

39. Yun Yang, Chengzheng Sun, Yanchun Zhang, Xiaohua Jia, “Real-Time Cooperative Editing on the Internet”, IEEE Internet Computing pages 18-25, May-June 2000.

40. Yun Yang, Chengzheng Sun, Yanchun Zhang, Xiaohua Jia, “Real-Time Cooperative Editing on the Internet”, IEEE Internet Computing, May-June 2000, Pages 18-25.

Appendix A

A.1 Installation Instructions

The following are the installation instructions for downloading and setting up the environment for executing NetEdit.

Installing the Server:

This can be done in two ways as follows:

From Internet

1. First download the server side files from the following URL: http://csgrad.cs.vt.edu/~azafer/thesis/thesisPage.html. There will be a link to the .zip file at the top of the page.

2. Create a new directory named NetServer under the root directory say C:\, and go to this directory.

3. Extract the contents of the .zip file in the directory created in Step 2.

4. If you do not have java runtime environment installed, then go to the following URL:http://www.java.sun.com/j2se/1.3/download-windows.html
   Download Java 2 SDK, v 1.3.0 Software for Windows 95/98/2000/NT 4.0 as a one large bundle. Then install this environment by double clicking on the downloaded file and following the instructions. Select the default directory and drive to install the program.

5. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the java runtime environment). This may be done directly or by executing the .bat file, server.bat.

   Direct Method:

   Type the following 2 commands from the DOS console:

   path=%path%;C:\<directory in which java was installed>\bin;

```
set CLASSPATH=%CLASSPATH%;C:\<directory in which java was
installed>\src.jar;C:\<directory in which java was
installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
```

Easy method:

Type **C:\NetServer\server** from the DOS console:

6. Go to the directory C:\NetServer created in Step 2 and give the following command from the console:

   ```
   java   ServerLogin <Server address> <Port number>
   ```

   Here Server address corresponds to the IP address of the machine on which the above steps are being performed. The port number can be any free port (say 2500).

   e.g. java ServerLogin thyme.cs.vt.edu 2500

7. Create a directory C:\NetServer\files. Any files to be edited should be placed in this directory. The name of this directory can be changed by adding the following to the file C:\NetServer\server.cfg:

   ```
   FILES: <file directory path>
   ```

From CD ROM

1. Create a new directory named NetServer under the root directory say C:\, and go to this directory.

2. Copy the contents of Server directory on the CD ROM in the directory created in Step 1.

3. Install Java runtime environment by double clicking on the file j2sdk-1_3_0_02-win.exe and following the instructions. Select the default directory and drive to install the program.

4. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the java runtime environment). This may be done directly or by executing the .bat file, server.bat.

   Direct Method:

   Type the following 2 commands from the DOS console:

   ```
   path=%path%;C:\<directory in which java was installed>\bin;
   ```

```
set CLASSPATH=%CLASSPATH%;C:\<directory in which java was installed>\src.jar;C:\<directory in which java was installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
```

Easy method:

Type **C:\NetServer\server** from the DOS console:

5. Go to the directory NetServer created in Step 1 and give the following command from the console:

   ```
   java ServerLogin <Server address> <Port number>
   ```

   Here Server address corresponds to the IP address of the machine on which the above steps are being performed. The port number can be any free port (say 2500).

   ```
   e.g.  java   ServerLogin thyme.cs.vt.edu 2500
   ```

6. Create a directory C:\NetServer\files. Any files to be edited should be placed in this directory. The name of this directory can be changed by adding the following to the file C:\NetServer\server.cfg:

   ```
   FILES: <file directory path>
   ```

Restarting the Server:

In case the server crashes or needs to be restarted, do the following:

1. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the java runtime environment). This may be done directly or by executing the .bat file, server.bat. This step may be skipped if the path and CLASSPATH are still set up from a previous session.

   Direct Method:

   Type the following 2 commands from the DOS console:

   ```
   path=%path%;C:\<directory in which java was installed>\bin;

   set CLASSPATH=%CLASSPATH%;C:\<directory in which java was installed>\src.jar;C:\<directory in which java was installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
   ```

   Easy method:

Type **C:\NetServer\server** from the DOS console:

2. Go to the directory NetServer and give the following command from the console:

   ```
   java ServerLogin <Server address> <Port number>
   ```

   Here Server address corresponds to the IP address of the machine on which the above steps are being performed. The port number can be any free port (say 2500).

   e.g.
   ```
   java   ServerLogin thyme.cs.vt.edu 2500
   ```

Appendix A

Installing the Client:

This can be done in two ways as follows:

From Internet

1. First download the client side files from the following URL: http://csgrad.cs.vt.edu/~azafer/thesis/thesisPage.html. There will be a link to the .zip file at the top of the page.

2. Create a new directory named NetClient under the root directory say C:\, and go to this directory.

3. Extract the contents of the .zip file in the directory created in Step 2.

4. If you do not have the Java runtime environment installed, then go to the following URL:http://www.java.sun.com/j2se/1.3/download-windows.html
Download Java 2 SDK, v 1.3.0 Software for Windows 95/98/2000/NT 4.0 as a one large bundle. Then install this environment by double clicking on the downloaded file and following the instructions. Select the default directory and drive to install the program.

5. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the Java runtime environment). This may be done directly or by executing the .bat file, client.bat.

   Direct Method:
   Type the following 2 commands from the DOS console:

   ```
   path=%path%;C:\<directory in which java was installed>\bin;
   ```

   ```
   set CLASSPATH=%CLASSPATH%;C:\<directory in which java was installed>\src.jar;C:\<directory in which java was installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
   ```

   Easy method:

   Type **C:\NetClient\client** from the DOS console:

6. Go to the directory NetClient created in Step 2 and give the following command from the console:

   ```
   java ClientDaemon
   ```

From CD ROM

1. Create a new directory named NetClient under the root directory say C:\ and go to this directory.

2. Copy the contents of Client directory on the CD ROM in the directory created in Step 1.

3. Install java runtime environment by double clicking on the file j2sdk-1_3_0_02-win.exe and following the instructions. Select the default directory and drive to install the program. Please note that you have to install the Java runtime environment only if you don’t have it installed already.

4. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the Java runtime environment). This may be done directly or by executing the .bat file, client.bat.

   Direct Method:

   Type the following 2 commands from the DOS console:

   ```
   path=%path%;C:\<directory in which java was installed>\bin;

   set CLASSPATH=%CLASSPATH%;C:\<directory in which java was installed>\src.jar;C:\<directory in which java was installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
   ```

   Easy method:

   Type **C:\NetClient\client** from the DOS console:

5. Go to the directory C:\NetClient created in Step 1 and give the following command from the console:

   ```
   java ClientDaemon
   ```

Restarting the Client:

1. Set up the PATH and CLASSPATH as follows (assuming you have selected the default locations for installation of the Java runtime environment). This may be done directly or by executing the .bat file, client.bat. This step may be skipped if the path and CLASSPATH are still set up from a previous session.

   Direct Method:

Type the following 2 commands from the DOS console:

```
path=%path%;C:\<directory in which java was installed>\bin;
```

```
set CLASSPATH=%CLASSPATH%;C:\<directory in which java was installed>\src.jar;C:\<directory in which java was installed>\lib\tools.jar;C:\<directory in which java was installed>\lib\dt.jar;.;
```

Easy method:

Type **C:\NetClient\client** from the DOS console:

2. Go to the directory C:\NetClient and give the following command from the console:

```
java ClientDaemon
```

A.2 Questionnaire

Questions	Responses
1. The response time from the editor after you do the edits like insertions/deletions, etc was: Please circle your choice. a. Extremely noticeable … Very slow b. Noticeable, slightly unacceptable c. Neutral d. Noticeable, but acceptable e. Not noticeable … Very fast f. Do not know	3.0 (5.0 being "Extremely noticeable" and 1.0 being "Not noticeable")
2. Rank the awareness widgets in order of your liking; with 1 being the widget you liked the most and 4 being the least favorite. ____ Telecursors ____ RadarView ____ Active participant's list in an Editing session ____ Active sessions list	Radarview(13) Telepointers(18) Active Participants list(19) Active Sessions list(26) [The number in bracket indicates the sum of ranks assigned. Hence lower the sum better is the ranking]
3. Rank the widgets in order that, you think, made you collaborate more efficiently with your group members; with 1 being the widget you liked the most and 4 being the least favorite. ____ Telecursors ____ RadarView ____ Active participant's list in an Editing session ____ Chat Window	Telepointers(15) Chat Window(21) Radarview(23) Active Participants list(28) [The number in bracket indicates the sum of ranks assigned. Hence lower the sum better is the ranking]
4. Rank the widgets in order of usefulness; with 1 being the most useful widget and 4 being the least useful. ____ Telecursors ____ RadarView ____ Active participant's list in an Editing session ____ Chat Window	Telepointers(16) Radarview(22) Chat Window(22) Active Participants list(27) [The number in bracket indicates the sum of ranks assigned. Hence lower the sum better is the ranking]
5. Did you notice the logging in and logging out of participants from your session? Please circle your choice. a. Yes b. No c. Do not know	4 out of 10 – Yes 6 out of 10 – No

6.	The movements of the shaded rectangles in the radarpane were: <Please circle your choice> a. Very helpful b. Quite helpful c. Slighly helpful d. Neutral e. Totally useless f. Do not know	3.5 (5.0 being "Very helpful" and 1.0 being "Totally useless")
7.	The movement of the telecursors allowed me to identify what each person was editing. Please circle your choice. a. Yes … (agree) b. No … (disagree) c. Do not know	10 out of 10 – Yes 0 out of 10 – No
8.	Rate the usefulness of "Telecursors" in identifying what each person was editing. Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	4.2 (5.0 being "Very useful" and 1.0 being "Totally useless")
9.	Rate the usefulness of "Active participant's list" to be aware of other active participants in the session? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.7 (5.0 being "Very useful" and 1.0 being "Totally useless")
10.	Rate the usefulness of "RadarView" to know other participants view into the document? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.9 (5.0 being "Very useful" and 1.0 being "Totally useless")

11.	Would you have preferred if the text inserted by the remote participant appeared in the same color as is assigned to him? Please circle your choice. a. Yes. I would have preferred b. No. I would not have preferred. c. Do not know	3 out of 10 – Yes 4 out of 10 – No 3 out of 10 – Do not know
12.	Did you feel the need to know "who wrote what text"? Please circle your choice. a. Yes b. No c. Do not know	8 out of 10 – Yes 2 out of 10 – No
13.	How was your experience with this editing window in terms of knowing its purpose? Please circle your choice. a. Extremely intuitive b. Quite intuitive c. Slightly intuitive d. Neutral e. Slightly confusing f. Quite confusing g. Extremely confusing h. Do not know	5.2 (7.0 being "Extremely intuitive" and 1.0 being "Extremely confusing")
14.	Was the directory listing (left-hand component) in Collaborative Editor Document/Sessions window intuitive? Please circle your choice. a. extremely clear b. slightly clear c. neutral d. slightly confusing e. extremely confusing f. Do not know	4.5 (5.0 being "extremely clear" and 1.0 being "extremely confusing")
15.	Was the session list on the right-side of Collaborative Editor Document/Sessions window useful? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.7 (5.0 being "Very useful" and 1.0 being "Totally useless")

16.	Rate the usefulness of knowing the history of who was in the session, when, and other details? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.6 (5.0 being "Very useful" and 1.0 being "Totally useless")
17.	The names of the sessions are files that are associated with that session. Was this what you expected? Please circle your choice. a. Yes b. No c. Do not know	9 out of 10 – Yes 1 out of 10 – No
18.	As a continuation of question – 17, would you have preferred an alias instead of a filename for the session name? Please circle your choice. a. Yes b. No c. Do not know	0 out of 10 – Yes 10 out of 10 – No
19.	How was your experience with this Collaborative Editor administrative window in terms of knowing its purpose? Please circle your choice. a. Extremely intuitive b. Quite intuitive c. Slightly intuitive d. Neutral e. Slightly confusing f. Quite confusing g. Extremely confusing h. Do not know	5.4 (7.0 being "Extremely intuitive" and 1.0 being "Extremely confusing")
20.	Did you use the chat utility during your editing? Please circle your choice. a. Used extensively b. Used somewhat c. Barely used d. Not used at all e. Do not know	3.2 (4.0 being "Used extensively" and 1.0 being "Not used at all")

21.	Was the chat utility effective in communicating with other collaborators? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.7 (5.0 being "Very useful" and 1.0 being "Totally useless")
22.	Were you being overwhelmed by the chat messages that were coming? Please circle your choice. a. Extremely overwhelmed b. Quite overwhelmed c. Slightly overwhelmed d. Neutral e. Not overwhelmed at all f. Do not know	3.9 (5.0 being "Extremely overwhelmed" and 1.0 being "Not overwhelmed at all")
23.	Did you like the 3 modes of sending and receiving a message to/from other participants in the system? Please circle your choice. a. Yes b. No c. Do not know	6 out of 10 – Yes 4 out of 10 – No

24.	Rank the 3 modes of sending the message based on how much you used each of them; with 1 being the most used option and 3 being the least used option.
	____ Send to All
	____ Send to group
	____ Send a personalized message to an individual
		All(12)
		Group(18)
		Individual(24)
		
		[The number in bracket indicates the sum of ranks assigned. Hence lower the sum better is the ranking]

25.	Did you find it difficult to continually have to switch between editing a document and sending chat messages to other participants? Please circle your choice. a. Yes b. No c. Do not know	10 out of 10 – Yes 0 out of 10 – No
26.	As a continuation of question 25 – Would you have preferred if the chat window were part of the Editing Window so as to avoid this continuous switching back and forth? Please circle your choice. a. Yes b. No c. Do not know	10 out of 10 – Yes 0 out of 10 – No

27. Rate the usefulness of color-coding of the message types in distinguishing them? Please circle your choice. a. Very useful b. Useful c. Neutral d. Not useful e. Totally useless f. Do not know	3.2 (5.0 being "Very useful" and 1.0 being "Totally useless")
28. How was your experience with Chat window in terms of knowing its purpose? Please circle your choice. a. Extremely intuitive b. Quite intuitive c. Slightly intuitive d. Neutral e. Slightly confusing f. Quite confusing g. Extremely confusing h. Do not know	5.6 (7.0 being "Extremely intuitive" and 1.0 being "Extremely confusing")
29. Have you used any similar collaborative tool before? Please circle your choice. a. Yes b. No c. Do not know	2 out of 10 – Yes 8 out of 10 – No
30. Was the system more tuned to what you would want from a collaborative editor? Please circle your choice. a. Yes b. No c. Do not know	4 out of 10 – Yes 1 out of 10 – No 5 out of 10 – Do not know
31. How was your experience with System as a whole? Please circle your choice. a. Extremely intuitive b. Quite intuitive c. Slightly intuitive d. Neutral e. Slightly confusing f. Quite confusing g. Extremely confusing h. Do not know	4.4 (7.0 being "Extremely intuitive" and 1.0 being "Extremely confusing")

A.3 User Manual

This section describes the user manual for NetEdit. It begins by explaining the login procedure, and then discusses other windows and functionality of the system.

A.2.1 Login procedure

When the application is initially started, a login window appears asking for username and password. It might also additionally ask for the server name and its port number. Give the server name (e.g. thyme.cs.vt.edu) and port number (e.g. 2500) given by your administrator. The server name (IP address of the machine on which the server is running) and port number (an integer) is asked only once, when the application is started the first time after installation. Later invocations of the application ask only for username and password.

A.2.2 Chat Window

Chat window initially appears in the right hand side of the screen. At the top is a white pane that logs all the messages that you send or receive from other users. Next comes the “send to” control panel that has three modes of sending the chat messages. Users type their messages in the text field at the bottom of the screen.

“Send to” control panel

This panel contains three options as follows:

"Individual" checkbox: This combo-box lists the usernames of all the currently logged-on users who have their chat window open. By clicking the check box ("Individual") beside this combo-box causes the system to send the typed messages only to the individual whose name is selected in the combo-box.

"Group" checkbox: This combo-box lists the sessions that are active in the system. A session corresponds to a file being collaboratively edited by a group of individuals. By clicking the check box ("Group") beside this combo-box causes the system to send the typed messages only to the users of the session selected in the combo box.

"All" checkbox: By clicking the check box ("All") causes the system to send the typed messages to all the users who are currently logged and have their chat window open.

Shortcut keys

1. You can reply to the sender of the last message by typing "\r message or /r message"

2. You can talk to an individual by typing "\t message or /t message". The recipient of this message is the user whose name appears in the first combo list.

3. You can send a message to the group by typing "\g message or /g message". The recipients of this message are the users who are participating in the session shown in the second combo list.

4. You can send a message to all the users by typing "\a message or /a message".

Audio cues

A user can set up audio cues (beep from the system) to inform him when a message from specified group of individuals or message to specified sessions arrives.

A.2.3 Documents/Sessions Window

Documents/Sessions window appears in the upper left side of the screen. It has two parts that display dynamic content reflecting the server file system state (File Explorer) and the sessions (Session List) currently active in the system. They are described as follows:

File Explorer

This is the left part of the Document/Sessions window. It displays the hierarchical directory structure and the .txt files residing at the server under the directory given by the root of the tree. It contains five buttons at the top that allows one to create a new file, create a new directory, open an existing file, deleting a file or directory (along with all the files in it) and renaming a file or directory. These operations are described further as follows:

Create a New File: First select the directory in which to create the file. Then either right click on the selected directory and choose “New File” menu item or click the first button from left on the toolbar for creating the new file. A dialog then appears asking to choose a name for this file. Give a name and press “Enter” key on the keyboard or click the “Enter” button at the right end of the dialog. An “Edit Window” for editing this file appears on the screen. The details of the “Edit Window” are given in section A.2.4.

Create a New Directory: First select the directory in which to create the new directory. Then either right click on the selected directory and choose “New Directory” menu item or click the second button from left on the toolbar for creating the new directory. A dialog then appears asking to choose a name for this directory. Give a name and press "Enter" key on the keyboard or click the "Enter" button at the right end of the dialog. An acknowledgment whether this operation was successful or not is displayed on the screen. If the operation is successful, then the directory hierarchy is updated accordingly.

Open an existing file: There are three ways in which this can be done:

- Double click the file you want to open.
- Right click on the file you want to open and select "Open File" menu item from the popup menu list.
- Select the file you want to open by left clicking on it and then click the third button from left on the toolbar for opening the file.

Deleting a file: This can be done in two ways as follows:

- Select the file you want to delete by left clicking on it and then click the fourth button from left on the toolbar for deleting the file.
- Right click on the file you want to delete and select "Delete File" menu item from the popup menu list.

Deleting a directory: This can be done in two ways as follows:

- Select the directory you want to delete by left clicking on it and then click the fourth button from left on the toolbar for deleting the directory.
- Right click on the directory you want to delete and select "Delete Directory" menu item from the popup menu list.

It is important to note that when you are deleting a directory it will recursively delete all the files and subdirectories under that directory.

Renaming a file: This can be done in two ways as follows:

- Select the file you want to rename by left clicking on it and then click the fifth button from left on the toolbar for renaming the file.
- Right click on the file you want to rename and select "Rename File" menu item from the popup menu list.

A dialog asking for a new name will popup. Give a new name and press "Enter" key on the keyboard or click the "Enter" button at the right end of the dialog. An acknowledgment whether this operation was successful or not is displayed on the screen.

Renaming a directory: This can be done in two ways as follows:

- Select the directory you want to rename by left clicking on it and then click the fifth button from left on the toolbar for renaming the directory.

- Right click on the directory you want to rename and select “Rename Directory” menu item from the popup menu list.

A dialog asking for a new name will popup. Give a new name and press “Enter” key on the keyboard or click the “Enter” button at the right end of the dialog. An acknowledgment whether this operation was successful or not is displayed on the screen.

Session List

This is a list of files on the server that are currently being edited by active users. More than one user could be editing a file in this list. To start editing a file for which a session is already going on, double click on that file in this list. You could also search the file in the File Explorer window and open that file. In any case you will join the active session for that file. To see the users who are in a session editing a certain file, right click on that session. A list displaying the usernames of the users who are editing that file will appear.

A.2.4 Edit Window

Edit Window has 3 sections. The left section is the place where you write your text. It also displays telecursors of the remote participants. The right-top section displays a list of users (in different colors) who are in a session with you, editing the same file. The right-bottom section is the RadarPane. It displays a rectangle for each user showing his view in to the document. This gives one an idea of where the other users are relative to his location in the document, which may span multiple pages. The rectangle for a user say Jack is of the same color as the color of his name in the right-top section of the edit window.

Cut operation: First select the text you want to Cut and put on the clipboard. Then press CTRL + X or select the Cut option from Edit Menu

Copy operation: First select the text you want to Copy and put on the clipboard. Then press CTRL + C or select the Copy option from Edit Menu

Paste operation: Go to the position where you want to put the previously copied or cut text. Then press CTRL + V or select the Paste option from Edit Menu

Vita

Ali Asghar Zafer was born in Mumbai, India on September 23, 1976. He began his undergraduate studies at University of Mumbai, in August 1994. As an undergraduate, he participated in one year of industrial training program, working for Datamatics Ltd. in Mumbai, India.

He received his Bachelor of Engineering in Computer Engineering from University of Mumbai in May 1998. He worked for a year with Tata Infotech Ltd., a consulting firm in India, before coming to Virginia Tech for pursuing graduate studies. His interests are in distributed computing, concurrency management, and networking. He will be graduating with a Masters of Science degree in Computer Science in May 2001.