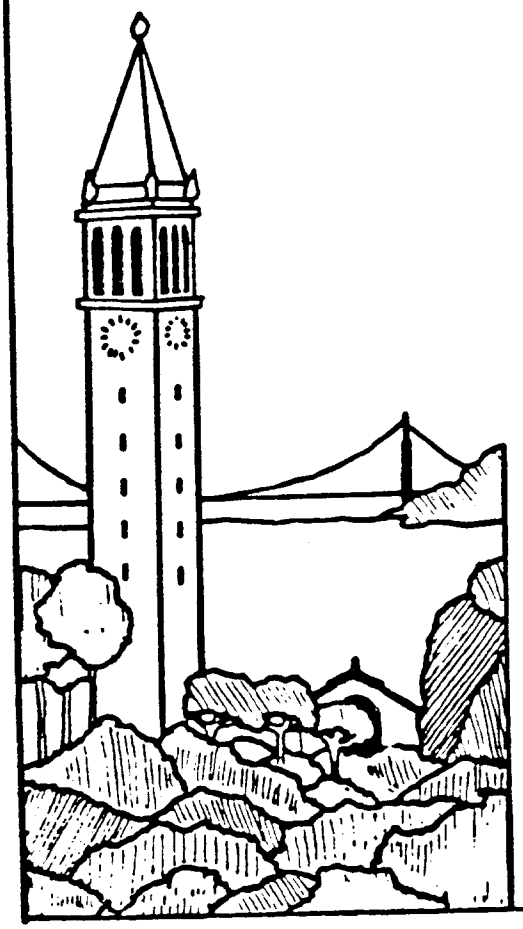
Collaborative Systems and Multi-user Interfaces

Gregg Foster



Report No UCB/CSD 87/326

October 1986

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Collaborative Systems
and
Multi-user Interfaces

by

Gregg Foster

All Hallow Even 1986

Computer Science Division
Electrical Engineering and Computer Sciences Department
University of California, Berkeley
94720

Copyright (c) 1986 Gregg Foster, All Rights Reserved

This research was supported by the Knowledge Systems Area of the Intelligent Systems Laboratory at the Xerox Palo Alto Research Center; by the state of California's MICRO program; and by Army Research Office grant DAAG29-85-K-0070, through the Center for Pure and Applied Mathematics, University of California, Berkeley.

Collaborative Systems and Multi-user Interfaces

Computer-based Tools for Cooperative Problem Solving

Gregg Foster

Submitted to the Computer Science Division October 31, 1986 in partial fulfillment of the requirements for the degree of Doctor of Philosophy

ABSTRACT

A *collaborative system* is a real-time computer-based environment for cooperative work. Computer systems have been available for some time to assist individuals with their work, but the use of computers by groups is underdeveloped. The thesis of this dissertation is that a collaborative system can be built in a principled way using network-connected workstations and that such a system can enhance group work.

Previous systems to support group work have generally either avoided computers, as in teleconferencing, or have relied on long-haul networks to support asynchronous message-passing, as in computer conferencing.

This dissertation focuses on real-time software tools to support groups working together in the same room. The Colab system and its tools explore the following properties of computer-based cooperation: the structure of the problem-solving process, the design of multi-user interfaces, social coordination, simultaneous activity, maintenance of consistent views of shared objects, and uses for digitally captured meetings.

To better understand and evaluate computer-based collaborative tools and their uses, the Colab system and the Cognoter presentation tool were implemented and used for both real and posed idea organization tasks. To test the system design and its effect on structured problem-solving, many early Colab/Cognoter meetings were monitored and a series of preliminary experiments were performed. These early observations indicated that people can and do work more efficiently and in parallel if they are given tools that help them stay focused and that help manage the added complexity of multi-user interactions.

to what there is of what I imagine there to be

Acknowledgments

Through the last few years, I have been fortunate to be associated with two great institutions: the University of California at Berkeley and the Xerox Palo Alto Research Center. People at both places demonstrate the value of collaboration.

At Xerox PARC:

Mark Stefik, manager of the Knowledge Systems Area, introduced me to the Colab meme and set this work in motion. His advice, direction, and enthusiam have been invaluable to me. I hope to repay him as he has repaid his best teachers. Stan Lanning and Daniel G. Bobrow programmed portions of the Colab and served as idea generators (many of them were good too). Steve Levy did early work on locks and keys for the Colab. John Seely Brown, head of the Intelligent Systems Laboratory, consistently encouraged the Colab project. Lucy Suchman made insightful early Colab observations and Lissa Monty made suggestions about the experimental design. Thanks also to Stan for rides, conversation, and system guruism.

At UC Berkeley:

I am grateful to my research advisor, Richard Fateman, for his support throughout my graduate career. I have appreciated his advice and his availablity for discussion of relevant or irrelevant (or irreverent) topics. Donald Glaser and Randy Katz, the other two members of my thesis committee, made incisive suggestions at critical times in my work. Gaetano Borriello and Margaret Butler read an early draft of this dissertation and improved it with their comments. Kathryn Crabtree helped me navigate the rocks and shoals of the Berkeley bureaucracy.

My Roma friends and other sophists made my stay here personally worthwhile. Especially valuable to me was the cooperation and benign competition within my prelim study group: Tony DeRose, Susan Eggers, and Mark Hill. We made it.

And Chris, she knows why.

Table of Contents

Table of Figures

Table of Tables

1

All for one, and one for all; that is our device.
— Alexander Dumas, the elder

Introduction and Overview

In this chapter it is argued that meetings are an important part of the problem–solving and decision–making process of human organizations, and that computers can be used to support real–time face–to–face meetings. The Colab meeting room at Xerox PARC and its accoutrements are presented as backdrop before spelling out the thesis of this dissertation. The research strategy of this dissertation has been to build a prototype system and use it to explore software dimensions and meeting behavior. This chapter ends by presenting a road map to the rest of this dissertation.

Introduction

Collaborative System: *a real-time computer-based cooperative work environment.*

Most human enterprises require cooperation and communication. In any task that calls for the coordination or agreement of several people — as most complex tasks and many simple tasks do — face-to-face interactions dominate the information flow and, therefore, determine the success of the enterprise. The media used to support an activity has a strong effect on its course. In group problem-solving activities the supporting media effect the participants' ability to think, communicate, and remember. Mismatches between support media and the participants' capabilities and interests account for the all-too-familiar boredom and frustration of unproductive meetings.

Technology for supporting meetings has advanced little in the last several thousand years: from sticks in the dirt and charcoal on stones to blackboards[1] and flip-charts. Computers are used increasingly to support and amplify the work of individuals, and yet when we want to work as a group we typically leave this active medium behind and move to traditional passive media.

Why have blackboards been the long-time technology of choice for meeting support? The ubiquitous blackboard brings many useful capabilities to group work. It provides a shared and focused memory: additions are visible, as they happen, to the entire group. A blackboard also serves as an area for flexible placement of text and figures, and this complements the human capability for manipulating spatial memories.

What are the limitations of blackboards as a meeting support medium? Space — both on the blackboard and in human short-term memory — is limited. Items disappear when their space is needed for something else. Rearranging items on a blackboard is inconvenient: they must be redrawn

and then the originals erased. Blackboards are also an unreliable medium for storing information. They are used in rooms that are often shared by many groups and the next group to use the room may reasonably want to use the communication and storage media, including the blackboard, and will need to clear off the work of the previous group.

Many of the things that are awkward to do with a blackboard are easy to do with appropriately programmed computers. Almost any existing computer system has a large and reliable memory when compared to blackboards or human short-term memory. Many computers have bitplanes and sketching programs with windows and other drawing aids. These systems provide much greater flexibility for rearranging text and sketches, and are capable of displaying text clearly in a variety of fonts.

Other benefits arise when the interesting events of a meeting are captured digitally and stored in a computer manipulable format. A computer-based problem-solving session can be extended in both space and time. File systems and reliable memory makes is possible to stop and start a meeting without loss of computational state. Arguments and discussions can be resumed where they left off. The history of an issue can be reviewed. Network communications make it possible for the state of a meeting to be considered or extended by other, perhaps geographically separated, groups. A digitally captured meeting record can also be used in research to record and analyze the represented activities that occurred.

In a distributed multi-station computer system, like the Colab, concurrent actions on objects of shared interest can be supported. The usual turn-taking and sequential social mechanisms are altered and can be speeded up by exploiting fast communication and parallelism. By putting all participants on an equal footing and keeping them in touch with each other, computer support of a group activity can promote a sense of community — participation in such a meeting can be less like suffering through committee work and more like pitching in at a barn-raising.

John Seely Brown [Brown83] pointed out the need, in the context of computer–based learning environments, to differentiate between the *process* of a creative effort and the *product* of such work. He also signaled the current technological opportunity to use computers in recording, representing, and communicating these underlying processes.

Computer–based meeting tools provide an opportunity to influence group problem–solving processes — to create computational support that encourages particular decision procedures, particular kinds of visualization, particular kinds of argumentation, particular kinds of interaction between the meeting participants. As it turns out, the same tools that provide new opportunities to influence meeting processes also provide unprecedented power for recording and studying meeting processes and set the stage for a more detailed and profound understanding of what makes meetings work.

The Colab project is an experimental collaborative system that attempts to understand and enhance the process of some kinds of real–time group work. To give a concrete setting for the discussion that follows, the Colab meeting room at Xerox PARC is described next.

The Colab Meeting Room

The most immediately visible facet of the Colab project is the special meeting room (see figure 1.1). The meeting room is designed for face–to–face meetings of two to six people — the size of most meetings in which group problem–solving takes place. The major portion of the room is taken up by a reconfigurable conference table with recessed workstations that are connected by a network. Each workstation has a bitmapped display, a keyboard, and a mouse. At the front of the room is a large electronic blackboard called the *liveboard*. The liveboard is touch sensitive and provides an alternative station for drawing figures or printing short pieces of text — to the system it is just another workstation. For entering large pieces of text at the liveboard, an elevated keyboard and mouse are provided on an adjacent *electern*.

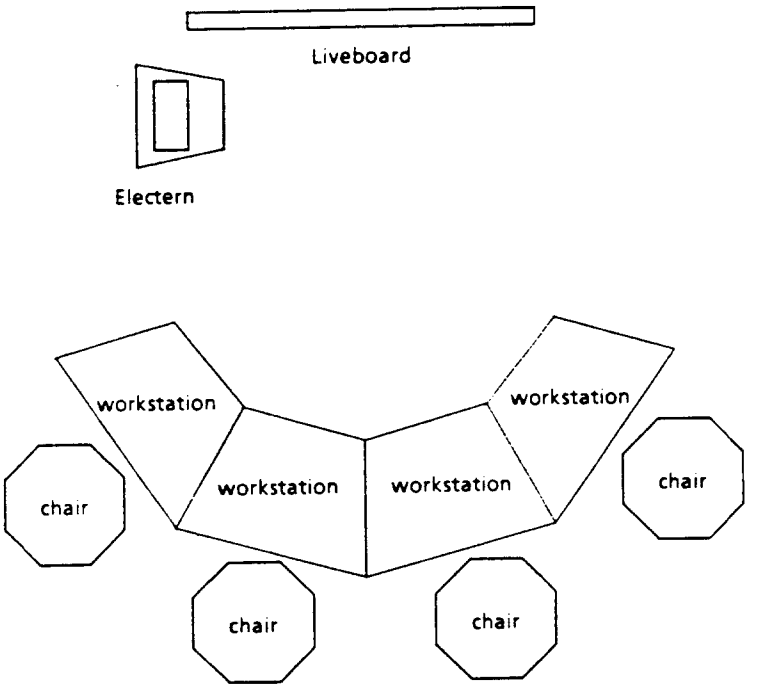


Figure 1.1. *The Colab meeting room.* Across the middle of the room is a semicircular conference table with communicating workstations interconnected by Ethernet. Each workstation is equipped with a recessed display, keyboard, and mouse. At the front of the room is a 1.5 meter by 2 meter electronic blackboard called the *liveboard*. The image on the liveboard is coordinated with the images on the workstations. The liveboard is touch sensitive and can be used for entering figures and short pieces of text. For entering substantial amounts of text, an elevated keyboard and mouse are provided at the adjacent *electern*.

An Ethernet® network is the primary means of communication among workstations. This has been augmented with a video network that allows the screen image of any machine to be displayed on any of the bitplanes in the room (including the liveboard). Although the video network was originally intended for ancillary uses of the Colab meeting room such as software demonstrations to large groups, it has proved useful in Colab meetings as well (see chapter six).

The Colab system does not depend on the special meeting room described above. It can be used anywhere with appropriate workstations connected by Ethernet. The benefits and flexibility of the pleasant surroundings, the Liveboard, and the video switch are lost, but the structure and leverage provided by the software environment (the focus of this dissertation) are still there.

The Thesis

This dissertation argues that software tools running on a network of workstations can enhance group work. The high–level goal is to explore how collaborative systems can augment usual group problem–solving practices and enable new, more effective, techniques.

I believe that the true power of computers is as a communication medium. Just as number crunching came to be seen as a special case of symbol processing, symbol processing can be thought of as a special case of communication. Related to this is the further belief that it is more important for computer systems to enhance the effective parts of the processes human beings have evolved over the centuries than it is for computer systems to suppress human failings.

Expressed throughout this work is the overarching ideal that, given the right tools, several people can coherently participate in a problem–solving or decision–making process, avoid the "committee failure problem" and yield a high quality result. In addition to physical media like blackboards and

computer screens, "meeting support" can include more subtle elements such as "process structuring".

Collaborative systems move us forward along a path of ever–expanding communication between people and other intelligent agents. While helping people to work together more effectively is a worthy goal in and of itself, it is my opinion that collaborative systems, as an assistance to Natural Intelligence, will be a fruitful step through hybrid systems toward useful Artificial Intelligence.

Research Strategy and Scope. Three questions have guided this work: Which facets and processes of cooperative work can computer systems effectively augment? What software tool features, underlying system organization, and interface presentations best support these facets and processes? And what effects do these systems have on the collaborators that use them for their work?

This dissertation is concerned with the implementation and effect of computer support for a small, but important, subset of meetings: face–to–face group problem–solving involving motivated and cooperative participants. There are several other classes of meetings not directly considered here: meetings to disseminate information, meetings to air opinions, meetings to assign blame, and so on. There is also a wide variety of meeting behavior goals: informative, obstructive, congratulatory, etc. Since this area of research is new, the pragmatic focus in this dissertation has been on the group activities that computer scientists, especially those of us working on the Colab, often participate in: meetings where participants with basically harmonious individual goals gather to make progress on a shared problem.

While it is hoped that many of the ideas developed here are extensible to other classes of meetings, detailed extensions and studies of meeting behavior in general are beyond the scope of this work. There are also many subtleties in the human factors issues: lighting, lines of sight, screen angles, distance between workstations, etc. — these things are not the immediate concern of this dissertation (though they are part of the Colab project). In

contrast to teleconferencing and computer conferencing (see chapter two), research on computer-supported face-to-face meetings focuses on problem-solving, interfaces, and communication rather than on the distribution of images and information to distributed sites (though this necessarily is considered as well).

Providing computer support for groups involves writing a lot of software: software to present a multi-user interface to several participants and software to keep distributed data consistent in a highly interactive environment. Specific requirements for software tools to support such face-to-face meetings and their implications are one of the foci of this dissertation. Software tools also use and provide models of cooperative problem-solving processes. Designers of Colab software tools script the meetings that use their software: subtly or obviously; intentionally or not. This is better done in a principled manner. These ideas were explored through a much iterated *test-redesign* cycle of the Colab system and the Cognoter tool for presentation design (see chapter four).

The Colab project is, appropriately, a collaborative project. My primary contributions have been the exploration of multi-user interface issues as described in chapter three and the design and implementation of the Cognoter meeting tool as described in chapters four and five. I also conducted the Cognoter/Colab experiments and data collection observations at UC Berkeley described in chapter six. I contributed substantially to the overall design and implementation of the Colab system and was a main participant in the experimental uses and observations that took place at Xerox PARC. The tangible evidence of my implementation effort is some thousands of lines of Interlisp/Loops code. In the few places where continuity of exposition requires presentation of work done primarily by others I have tried to make that clear.

A Map of this Dissertation

In this introduction I have outlined the motivations that led to the Colab project and to this dissertation, briefly described the Colab meeting room, and presented my thesis and research strategy.

Chapter two surveys related work that does not fit naturally into the following chapters. This chapter focuses on work dealing with computer communication and computer–based systems for group work.

In chapter three I discuss *multi–user software* and *collaborative systems.* I consider the design dimensions for software tools that support the group use of computers for cooperative work. This chapter also includes implementation details and features relevant to the design of multi–user systems in general.

Cognoter, a Colab tool for organizing and developing material for presentation, is introduced in chapter four. It is the first full–fledged tool for the Colab. The chapter explores the theory and practice of Cognoter and the motivating decisions that went into its design.

Implementation details and design tradeoffs are discussed in chapter five. I discuss the design of Cognoter, the Colab's main system entities, and some approaches to shared object consistency.

In chapter six I describe early uses of Colab and Cognoter. Cognoter was used informally and semi–formally for several months while it was being redesigned and debugged. Through this period some general principles became apparent as well as many questions about what the tools were doing and what they should be doing. These observations lead to a set of experiments using Cognoter as a test bed for the exploration of computer–based group problem–solving. The purpose of these experiments was not to prove the superiority of Cognoter over other techniques, but to lend plausibility to some of the claims made in this dissertation and to guide further understanding and development of Cognoter and future tools.

Chapter seven is the conclusion. Therein are speculations about the effects of collaborative systems on group problem–solving and about areas expected to be fruitful for further research.

Note:

[1] The term *blackboard* is used generically to refer to any wall–mounted erasable writing surface commonly used in meeting rooms. The word *blackboard*, in this work, is not intended to refer to commercially–available teleconferencing products or to programming organization techniques for artificial intelligence systems (which have adapted and redefined this term).

2

Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise.

— John W. Tukey

Related Work

This chapter surveys several early works that inspired or indirectly contributed to this dissertation. Computers have already found wide use as communication devices, especially in electronic mail systems. Computers have been used to a smaller extent in research projects involving more sophisticated systems including: computer–conferencing, hypertext, and multi–media mail conferencing. Various projects exploring cooperative work processes have also been useful to the present work. There have been a few real–time systems that share many similarities with Colab, including RTCAL, Electronic Blackboards, and the general area of Computer Conference Rooms.

Introduction

The research described in this dissertation has benefited from previous work in several disciplines. Nevertheless, there has been, so far, no comparable system dealing with computer–based real–time support for collaborative problem–solving in face–to–face situations. The main purpose of this chapter is to survey work that is significant to this dissertation, but does not arise naturally in the following chapters.

Early Work

A few observers of technology and the human condition foresaw the utility of computers in augmenting and supporting group problem–solving years before such systems were technically practicable. The following subsections highlight these early insights and projects.

Memex. In 1945, Vannevar Bush, in the course of speculating about how "we may think" in the future, introduced an imaginary system he called the "memex" [Bush45]. The memex was an early conception of the personal workstation. It included an interactive database that could save associative trails of excursions into a global encyclopedic database and recall and revisit them at a later time. Bush predicted that such a device would revolutionize human problem–solving. His description of the memex is strangely familiar:

> *On the top are slanting translucent screens, on which material can be projected for convenient reading. There is a keyboard, and sets of buttons and levers. Otherwise it looks like an ordinary desk.*

Human–Computer Symbiosis. Licklider was another early thinker concerned with the computer augmentation of individuals and groups, and how computers would change the way people worked. Among his many insights and predictions was a precursor to the Liveboard [Licklider60]:

> *The large wall display and its associated system are relevant, of course, to symbiotic cooperation between a computer and a team of men* [sic]. *Laboratory experiments have indicated repeatedly that informal, parallel arrangements of operators, coordinating their activities through reference to*

> *a large situation display, have important advantages over the arrangement, more widely used, that locates the operators at individual consoles and attempts to correlate their actions through the agency of a computer. This is one of several operator–team problems in need of careful study.*

NLS/AUGMENT. Inspired by Bush's vision, Douglas Engelbart began, in the early 1960's, to experiment with systems using computers to support collaboration. By the late 1960's, *NLS* (for *oN–Line–System*, later called *AUGMENT*) supported terminal linking, electronic mail, and the sharing of files [Engelbart68,84]. The general idea behind NLS was to provide a working environment for "knowledge workers". Quoting Engelbart [Engelbart84]:

> [AUGMENT] *permits a user to call an on–line conference of two or more people, view and edit files, add and remove conferees, pass the gavel, and transparently connect to other machines via TYMNET or ARPANET. Televiewing is usually done in conjunction with a telephone connection, and is often used to support document review and revision in a synchronous mode, where all conferees can see and discuss changes as they are made.*

Engelbart's guiding concern was to build environments to augment the human intellect. He was one of the first to understand the potential of computers as a medium for communication. He is also known also for the development of novel input devices, especially the mouse.

The Briefing Room. In the mid 1960's, Gene Roddenberry and the set designers for the classic television program *Star Trek*, envisioned (and built a mock–up of) the Briefing Room of the future [Gerrold73]. The Briefing Room (supposedly installed on the starship *Enterprise — NCC 1701*) consisted of a conference table with miniature computer screens set into it (the use of the individual workstations was never demonstrated — the screens and keyboards appear to be too small). In the middle of the conference table was a three–sided video display of remote participants. On one wall was a large screen for displaying data and images from remote sites. This room was used for (fictional) crisis problem–solving and group communication.

Electronic Mail and the New Literacy. The advent of time–shared systems such as TENEX [Bobrow72] brought file sharing, and network communications to the computer system armamentarium. This in turn made electronic mail feasible. The Advanced Research Projects Agency network (ARPANET), for example, was established so that government–funded

researchers could have access to host computers at other locations, although it came to be used predominantly for the exchange of messages among the researchers [Licklider78]. Soon after these features became generally available, observers such as Joshua Lederberg and J. C. R. Licklider reported qualitative differences in the ways that they were approaching problems and interacting with colleagues [Lederberg78] [Licklider68]. Lederberg, writing about "the New Literacy" [Lederberg78]:

> *Computer communication networks ... permit a new form of informal communication between scientists and often provide motivation and reward for timely sharing of research results. In addition, computer-based support to large distributed segments of a scientific community is made possible via users and computers interconnected by computer controlled networks.*

Terminal Linking. Once operating systems had features to support several users, various researchers began exploring system utilities to slave, split, and share terminals and display screens. *Slaving* one screen to another is easy to do if the terminals are close together (say, in the same building). This has been done for as long as video display terminals have existed. It is primarily useful for making a single display visible to larger audiences.

Some features of NLS/AUGMENT, TENEX *Link*, and MIT ITS system *Output Spy* (*OS*) simply direct machine output to more than one terminal and accepts input from all linked terminals. Terminals in these schemes share a single computational environment. Characters typed from any linked terminal will appear on all linked screens; characters typed simultaneously from two terminals will interleave (and appear to the machine as a single, fast, but probably incoherent, type stream). These systems have been used to advantage for collaborative debugging.

The *talk* and *write* utilities in UNIX both allow text typed on one terminal to appear on another, but with no access to the remote environment. *Write* buffers lines of text and ships them off to the (loosely) connected target device. More interesting is *talk* (and similarly *TALKOMATIC*, CDC Plato): it splits both connected terminal screens into two parts, allowing simultaneous typing and displaying each character on both screens as it is typed. No sharing of environment is allowed in *talk*, *write*, or *TALKOMATIC*.

These utilities are generally used for messages or short negotiations lasting a few seconds or minutes.

Slave terminals and split screens allow a degree of collaboration, but primarily in a sequential fashion. True concurrency is difficult in these utilities. With *Link* and *write*, messages share the same screen space and can obscure each other. Most of these features are only used when the terminals are physically remote: they are useless in a face-to-face environment. For real-time collaborative work of any complexity, these utilities are at too low a level (keyboard input events).

Computer-based Communication Systems

Teleconferencing. Teleconferencing uses the transmission of video signals to simulate a face-to-face meeting by displaying people at geographically separated locations on one or more video monitors in a conference room. This is useful when gestures and facial features are important. There are many differences between a face-to-face meeting and a teleconference; and many more differences between a computer-based face-to-face meeting and a teleconference (most teleconferencing systems make little use of computers, though this is changing). For an overview of teleconferencing see Johansen [Johansen84] and Johansen, Vallee, and Spangler [Johansen79].

The high cost of transmitting video images has probably been the main factor preventing wider use of teleconferencing (and picture phones, etc.). *Slow-scan television*, which transmits only a fraction of the display field thereby greatly lowering the required video data bandwidth, is a promising innovation for teleconferencing applications. The Systems Concepts Laboratory at Xerox PARC (Palo Alto, California) has demonstrated this by holding meetings regularly with its satellite group in Portland, Oregon using a slow-scan system for video images of remote conference rooms.

Computer Conferencing. There has been considerable literature on the subject of computer conferencing. This sub-section draws on the surveys of Hiltz and Turoff [Hiltz78], Kraemer and King [Kraemer83], and Rice [Rice84].

Computer conferencing systems use computers as a communication medium, usually ignoring video images. They can be thought of as extending electronic mail by adding features such as special purpose editors, voting mechanisms, shared files, distribution and interest lists, and automatic archiving. Like teleconferencing, computer conferencing supports communication with geographically separated participants. It also lets participants respond in their own time; logging onto the system, viewing any new contributions, reviewing any old contributions, and composing any desired responses. Computer conferencing works better for on-going, long-term discussions than it does for short-term decision-making or real-time cooperation since the turn around time for messages is at least seconds (usually much longer) and electronic folders of messages are cumbersome to use in a real-time interaction.

Computer conferencing, despite its name, is only an indirect ancestor to this dissertation. It shares similar goals but is asynchronous rather than real-time and does not attempt to provide structure for the participants based on models of group problem-solving processes.

The main computer conferencing systems (and there have been many) are EIES [Hiltz81] (various incarnations of EIES have been used by the Department of Energy for several years), CONFER [Parnes77] (used for discussion of social issues), FORUM and PLANET [Vallee76] (now called NotePad, has been used in a variety of situations), and COM [Palme84] (for public use in Sweden).

Hypertext. *Hypertext* refers to systems that support the arrangement and re-arrangement of pieces of annotated and catalogued text into documents (i.e., a paper might be built from existing paragraphs gleaned from several sources in a hypertext network). In addition to the work of Douglas Engelbart, who saw early on the potential of organizing and

indexing disassociated text, hypertext systems include *Xanadu* [TNelson81], *TextNet* [Trigg83], and, at Xerox PARC, *NoteCards* [Halasz86] and *Annoland*. The work in hypertext and the Colab share an interest in organizing information in explicit knowledge structures for collaborative tools.

Multi-Media Conferencing. Multi-media conferencing is essentially electronic mail with protocols and peripherals to support voice, graphics, etc. *Diamond*, a multi-media message system, has facilities for creating, editing, transmitting, and managing multi-media documents [Forsdick85] [Thomas83]. A Diamond multi-media document may contain text, graphics, images (bitmaps), and speech. Other media, such as spread-sheets, are also possible. An example of a Diamond multi-media document is a map in the form of a drawing with accompanying voice directions. Users without powerful bitmapped workstations and other media peripherals, such as vocoders, will not be able to take advantage of all of Diamond's capabilities.

Cooperative Problem Solving

The Evolution of Cooperation. Evidence for the general value of immediate feedback and cooperation in face-to-face situations can be found in the *Prisoner's Dilemma* competition run by Axelrod [Axelrod81]. In this competition programs embodying different strategies for dealing with the game-theoretic Prisoner's Dilemma[1] problem ran against each other in a Round Robin tournament. "TIT FOR TAT", using a very simple reactive immediate feedback strategy (it did whatever the other guy did last time), was the overall winning program. Interestingly, TIT FOR TAT cannot beat any other strategy in a single encounter: it won the tournament by consistently doing well against all of the other strategies.

Structuring Group Decision-making. There have been attempts to establish effective techniques for group decision-making. Two examples that have seen actual use (with and without computer assistance) are the *Delphi* method [Linstone75] and the *Nominal Group* method [Kraemer83]. The Delphi method has been used for forecasting by a medium sized group (10 to 100)

dispersed in space and time [Turoff72]. The basic techniques consists of a series of questionnaires asking the participants to estimate the values of variables affecting the larger decision. Nominal Group is a consensus–forming process intended for small groups in face–to–face meetings. These meetings enforce four phases: silent generating of ideas and suggestions, presentation of the suggestions, discussion, and ranking of the ideas. The phases are related to the phases of Cognoter, but are more restrictive and rigid.

Lateral Thinking. DeBono suggests *lateral thinking* as a tool for enhancing creativity [DeBono70]. The central idea is to occasionally stop trying to think of the next step in a logical progression and to intentionally consider a tangent for a while — even a random one — to gain fresh perspective and to uncover new ideas. DeBono:

> *Lateral Thinking makes quite a different use of information from logical (vertical) thinking. For instance the need to be right at every step is absolutely essential to logical thinking but quite unnecessary in lateral thinking. It may sometimes be necessary to be wrong in order to dislocate a pattern sufficiently for it to re–form in a new way. With logical thinking one makes immediate judgements, with Lateral Thinking one may delay judgements in order to allow information to interact and generate new ideas.*

Mutual Protocols. Whimbley and Lochhead, in describing the theory behind their course in problem–solving techniques, argue that comparison of strategies is extremely important [Bransford85]. Their goal is the teaching of problem–solving strategies. Since any problem–solving involves some on–the–fly learning, their work applies to problem–solving competence as well. In their course, Whimbley and Lochhead divide the process of learning problem–solving strategies into three stages: 1. examining expert protocols; 2. working in student pairs with one taking the role of listener/checker and the other taking the role of solver; 3. devising problems for others to solve ("seeing problems from the inside out"). These stages are stages in learning and have little to do with the three phases of interaction embodied in Cognoter (see chapter four). Mutual protocols are related to Colab in that they are both concerned with face–to–face real–time interactions and the synergetic and corrective effects of group interaction.

Real-Time Systems

Interactive On-Line Conferences. *RTCAL* (Real-Time CALendar), developed at MIT by Sarin and Greif allows a group of users to "exchange information from personal calendar databases in order to schedule a future meeting" [Sarin84a,85]. *MBLINK*, another tool built on the same underlying architecture, allows a group to cooperatively edit a bitmap [Sarin84]. This work is "concerned with implementing *real-time* conferences, in which groups of users at interconnected workstations can collectively view and manipulate on-line information" [Sarin84a].

These systems have much in common with Colab. They share the domain of real-time meetings and they maintain consistent views of mutable shared data by sending messages over a local network. Sarin's work differs from Colab in that there is no attempt to explicitly support or test process models for cooperative problem-solving and that the target groups for RTCAL and MBLINK are distributed (though usually in voice communication via telephone for negotiation and discussion) [Sarin84].

Computer Conference Rooms. In 1983 Kraemer and King surveyed computer supported conference rooms [Kraemer83]. Computer conference rooms are roughly at the halfway point between computer conferencing and teleconferencing. Narrowly defined, they are meeting rooms with terminals running "decision support software" (often a database system, sometimes including some group process software like Nominal Group). Kramer and King's overall impression was that hardware difficulties were preventing these projects from finding acceptance. They cited the inaccessibility of computing resources, the unreliability of video projectors, and limited graphics capabilities as the primary obstacles to the success of these efforts. However, in the past few years, computing and projection technology have become much more reliable and much less expensive.

Electronic Blackboards. Electronic blackboards have been developed at a few research labs (AT&T and NEC have product versions). They consist of a few electronically connected glass "blackboards", thereby supplying a shared

resource. Any drawing on the board — made by someone using a special pen — is displayed on all boards. The main advance represented by these devices is that they can serve as a way to digitally capture whatever has been written on them (and broadcast the actions or markings). Otherwise, they have the usual limitations of blackboards: they are difficult to re-arrange, only simple sketches and small amounts of text can be represented, and they are computationally passive.

Now

Work in real-time cooperative systems has been hampered by lack of computational power and networking. Several recent advances in technology — including powerful personal workstations, local area networks, advanced programming environments, and distributed programming and interface technology — make it possible to develop prototype systems rapidly and to experiment readily with new computer-based tools for meetings [Sheil83].

Now that it has become less expensive to construct large systems from collections of computers connected via networks, it has become more interesting to design special programming languages to support distributed applications. For example, Liskov has designed and implemented *Argus*, a language with features that support preservation and manipulation of long-lived data [Liskov85]. Argus includes atomic actions and atomic data-types which simplify the handling of concurrency and failures and which ensure serializability and recoverability. Greif, Seliger, and Weihl used Argus to implement Seliger's CES, a distributed Collaborative Editing System [Greif86].

Other related research, especially that concerned with implementation, problem-solving techniques, and design issues, is described later in context.

Note:

[1] The Prisoner's Dilemma is a two–player game where each player makes a decision to "cooperate" or "defect" in the absence of information about what the other player is doing. In the version run in the tournament if player A cooperates and B defects then B wins 4 units and A loses 1 unit. If both cooperate then each wins 2 units. If both defect then neither gets anything. An encounter consisted of several decisions and interactions where the history of each player was available.

3

Gentlemen, I do not mind being contradicted, and I am unperturbed when I am attacked, but I confess I have slight misgivings when I hear myself being explained.

— Lord Balfour

Multi–user Interfaces

Central to collaborative systems is the concept of a *multi–user interface*, a user interface for several simultaneously active users and machines. In this chapter the basic principles of traditional single–user interfaces are considered in the context of multi–user systems. This chapter then explores specific design dimensions and concerns for multi–user software tools for interacting groups. The main ideas include task process structuring, parallel activity, mutual protocols, and WYSIWIS. Finally, generally applicable implementation techniques for multi–user interfaces are discussed. These include UIDs, BroadcastMethods, Associations, ActiveRegions, ActiveWindows, underlying communication protocols, and disciplined ways of partitioning distributed programs — ways that are easier to write and understand: user–actions, semantic–actions, and display–actions.

Introduction

As discussed in chapter one, computer systems and applications have generally been designed for single users. Computers are now being used extensively for communication, but such computer-based communications are still usually sequential and synchronous. If computer systems are to be used for parallel and synergetic cooperative work, something beyond the basics of accurate memory, computational underpinnings, and communication primitives is needed. A computer system that provides the right kind of "something more" is a *collaborative system*.

The general goal of a collaborative system design is to provide new kinds of leverage and organization to a group of people focusing simultaneously on a common objective. The effectiveness of a group working with a blackboard can be taken as the standard by which the effectiveness of a collaborative system is measured (see chapter six).

A collaborative system ideally gives each user the illusion of having immediate access to all shared objects and the ability to manipulate them at will. To stay current and avoid conflict or redundancy in this situation each participant must be able to see what everyone else is doing. All users should have equal access to shared conceptual objects (i.e., the database) and should be able to simultaneously act upon (different) portions of them.

The flexibility and potential power offered by collaborative systems don't come free. They come at the cost of more complexity in the underlying software and increased network internal communication traffic. Collaborative systems are distributed systems and any implementation must respond to the following questions. How is the data represented internally? How is the data presented and how are the presentations of the data kept up to date? How is the shared data kept in a consistent state for all users?

It is also important for a collaborative system designer to understand the effect of a system on both the communication and the communicators. In

addition to understanding the system primitives (the operations, objects and data structures), a system designer needs to identify and formalize pragmatic knowledge of meetings and group problem-solving processes. A collaborative system design inevitably expresses a group process structure.

Computer-based collaboration tools provide an environment for study of the activities of groups making decisions and solving problems. The underlying computer system is a convenient place to install automatic data gathering tools (this should only be part of an experimental system).

This chapter explores the vaguely characterized "something more" mentioned at the beginning: the interface issues and process structuring aspects of software intended for use by groups — especially as they relate to the Colab collaborative system. This chapter also presents some generally useful multi-user interface implementation issues.

Multi-user Interfaces

A *multi-user interface* is a human-machine interface coordinated for several users sharing information at the same time. This notion is different from traditional time-sharing in that with a multi-user interface any and all of the users (probably on personal workstations) can act simultaneously and in concert on parts of the same thing — time-sharing systems are designed expressly to avoid this.

For a single computer to support a multi-user interface as well as separate workstations connected by network can, the mainframe would have to have fast processors (or a *very* fast single processor — where "fast" means "much faster than a workstation"), fast context switching, fast networking, and facilities for remote mice and bit planes (implying a very large memory). The hardware to support three people using a basic Colab system costs about $30K (I hope this number looks absurdly high in the near future). A sufficiently fast time-shared system would cost at least twice that amount. In addition to the savings in cost, the workstation configuration supports simultaneous local interactions and can be incrementally expanded.

User Interface Principles and Multi-user Interfaces

Let's back up a little and describe general human-machine interface design principles and relate them to multi-user interfaces.

Time and Space. In any interface design the usual time and space tradeoffs between design goals must be made (e.g., a more complete command language may be more difficult to learn than a simpler, more primitive one; or a single long flat menu may or may not be faster to use than a tree of shorter menus). Multi-user interfaces put additional burdens on designers trying to achieve these resource allocation goals since several agents are commonly active at once.

For instance, on any computer terminal there is only so much screen space. Windows were originally conceived as a solution to this problem — they trade space multiplexing for time multiplexing. But even with windows space problems remain. One solution may be to group windows and applications together into screen groups using the Rooms technique of Henderson and Card [Henderson86].

In multi-user applications the screen space problem is exacerbated by several users competing for publicly shared space and by the correspondingly greater need to have many windows visible and active at once. The simplest approach to this problem is to get bigger screens, but this is usually not feasible and may only create more problems by displaying too much information, all of which may not be relevant to each user. Another approach is to employ compressed versions of some windows in which general activity is discernible but details are suppressed. Still another approach, described in more detail later, is to allow private views of shared data.

Models, Views and Controllers. In the Smalltalk world [Goldberg83] the user interface concerns are partitioned in a particularly useful way: into models, views, and controllers (see Figure 3.1). A *model* is the internal representation of an object in the system. A *view*[1] is how a user sees the model: it could be a window, some text, a graph, a holographic image, a combination of these, or something else completely. The user acts on a model

via *controllers*: controllers can be directly attached to input devices like the keyboard or the mouse; or they can be abstracted constructs like menus or annotated values. In a multi-user interface the model is shared with other users. Each user may have private views on and local controllers of the shared model. The model is the shared database and the views are illusory manifestations of shared objects.

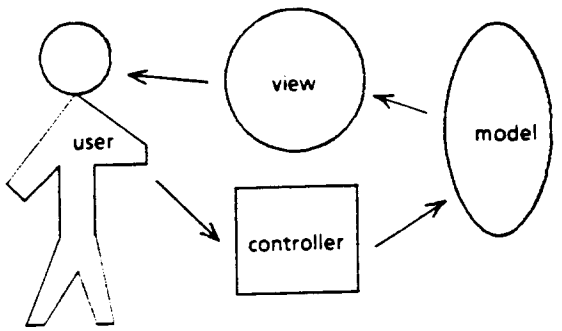


Figure 3.1. *Model–View–Controller.* An interface can be divided into three areas of concern: the underlying model (data), the view (of the data) that each user sees, and controllers that act on the data.

Point of View. How is the user injected into the data space? One way is by supporting different points of view. This implies that relationships between objects are part of the model. Participants are free to move their viewpoints around in the data space, but actually moving an object (i.e., changing its relationship to other objects — spatial or otherwise) is reflected globally in all viewpoints. This is directly analogous to physical intuition.

Attention. An interface should be designed so that attention is focused on the ideas and facets of the task and not on the interface itself. Interface tools and functions should sink below the level of consciousness. They should be on the user's side of the divide between the user and the outside world rather than entities out there somewhere to be manipulated in the hope that the proffered invocations and gyrations will produce the desired results. This is the difference between having a pencil and paper on your desk and having a pencil in your hand poised over the paper. Again, in a multi-user environment, each participant needs to be able to see what the others are doing.

Deixis. Humans almost always prefer to use pointing (literally or figuratively) and pronouns to refer to objects rather than having to compose descriptive phrases that specify objects [Bolt80]. Fundamentally, these human biases are what is behind the perceived power of menus and mice. In a multi-user environment problems of pointing and of referring to location loom large. They are discussed later in this chapter (see *pointing*) and in chapter five.

Immediacy. Rapid feedback is required for any real-time interface. The acronym *WYSIWYG* (What You See Is What You Get) was coined to describe text editors which normally display text as it will appear when printed. An analogous acronym can be used to describe an important abstraction of multi-user interfaces: *WYSIWIS* (What You See Is What I See). WYSIWIS, its variations and ramifications are discussed briefly in the WYSIWIS subsection under *Dimensions of Multi-user Software* tools below, and at length in Stefik et al [Stefik86].

Direct Manipulation. Thousands of existing programs for personal computers covering a wide variety of tasks create the useful illusion that data is being manipulated directly. This is true for text editing programs, figure drawing programs and many others. The illusion that a data object is changed immediately and directly when a user interacts with a program displaying a stylized image of the data is a powerful one [Schneiderman83]. The underlying system needs to perform the necessary computational gymnastics to maintain

the direct manipulation illusion without troubling the user. The appeal and simplicity of direct manipulation interfaces (also called *icon driven* interfaces) draws heavily on their ability to support this illusion.

> *When a system makes directly manipulable the concepts of an application, programs become more understandable. For example, some bookkeeping and accounting concepts are represented and manipulated directly in spread-sheet programs. While the state of the art has no dependable cognitive metric for how much this helps, the issue is a recurring theme in the design of languages and knowledge-based systems. It has to do with reducing the levels of abstraction that must be penetrated in order to understand system behavior* [Bobrow86].

The direct manipulation illusion is no less important in multi-user interfaces. Users should be able to use their personal interface skills and tools in a multi-user interface world. But, there will certainly be some differences. For instance, if two participants begin to make conflicting changes to a shared object or database, it would be counterproductive to keep them unaware of the conflict. In such cooperative situations the participants need to be able to observe what the others are doing so that they can coordinate their activities.

In a direct manipulation environment the image of an object and its visible relationship to other objects should signal relevant state information. Recalling the barn-raising metaphor from the introduction, if a participant wants to move a piece of lumber, a look at the piece of lumber will reveal whether or not the contemplated act is feasible. If someone else is currently doing something with the lumber, that will be immediately obvious. In the same way, multi-user interfaces must provide visual clues showing where the other participants are working.

Illusions. As mentioned above, users should have the illusion that they are directly manipulating objects and data. A consequence of this is that users make their own internal models of the machine side of the interface. For instance, a user moving (an image of) an object to (an image of) a trash can may reasonably assume it has been thrown away. Ideally the interface is so constructed that the users' guesses are valid.

Whereas a user's internal analogy or an interface designer's conscious use of a metaphor may help a user guess correctly, such abstractions may also confuse or blind the trusting user [Halasz82]. Continuing with the object in the trash can: the user may have any of several internal models of what goes on when something is put in the trash can. He may assume that, once in the can, the trash is gone forever. He may assume that the trash is still retrievable should he change his mind about it. He may assume that no further action is necessary and that the object will (somehow) be collected early some morning. He may assume that some further action is necessary to truly rid himself of the trash. Any of these assumptions may or may not be correct. Benign user illusions should be as little effected as possible by extending them to multi-user environments. The extension should be completely transparent.

Modes. There are two related aspects of modality relevant to user interfaces. Commands that work in one place should work in another and one should be able to start new commands without having to clean up after an old one. Like most things, the degree of modality is a matter of perspective and grainsize. For instance, a so-called "modeless" interface may very well use pop-up menus that monopolize the processor while waiting for the user to make a selection from the menu. Applying modality goals to the multi-user case, one participant should not, in general, be able to put the system into a mode that alters the semantics of other user's actions.

Activity. Computer systems and user-interfaces have evolved. Early systems, typified by batch mode systems, expected users to perform many *actions* before the machine gave any sort of feedback. Most current systems are of the *reactive* type: they will quickly respond to user actions (quite possibly with an error, but this is much better than silence). Future systems will be *interactive*, that is they will not only respond to user actions but will be intelligent enough to suggest future actions or to act on implication as well as on command. It is not important to draw a careful distinction between "reaction" and "interaction", but in a collaborative system it is important to be aware that there are two levels of activity: with other users (true interaction) and with the communication tools (reactive).

Extensibility. It is generally desirable for users to have the ability to customize the way information is presented and the way a system behaves toward them. In the multi–user situation things are more difficult. In general, personal extensions should not propagate across machines since personalizations on one machine may conflict with another participant's personal extensions. The related notion of private and public views is discussed below in *Dimensions of Multi–user Software*.

Exploration. It should be possible to start up the system quickly and just mess around without fear of breaking something or upsetting somebody. It should be possible to learn about the system this way, though perhaps inefficiently [Malone82]. This implies an easy to learn system with an undo facility and some protection from easily doing destructive things. In multi–user system it is important that the system be quick and easy to start — a system that takes a long time to bring up will get little use.

Communication. In a single user interface, the primary concern is that the human and the machine understand each other (or at least communicate to the extent that the machine follows orders). What happens when the goal is to provide a coherent interface for several people on several machines? General user interface principles and practice still apply, but there are new constraints and demands on the system. A well designed multi–user interface needs also to address the concerns of machines understanding other machines (networking and database consistency) and people understanding other people (group dynamics and communication).

Dimensions of Multi–User Software Tools

This section provides an overview of the concepts involved in the design of tools for collaborative systems. Some of the tradeoffs and goals inherent in software intended to support group problem–solving are also discussed.

Meeting Tools. One definition of *tool* is *an instrument, apparatus, or construct used to perform an operation; a means to an end* [Merriam69]. The performance of a single task on a computer often involves invoking a variety

of utilities and subsystems. For instance, the creation of a document may involve the use of a shell language, a text editor, a drawing program, and a proof-reading utility. Similarly, different activities arise in the course of a problem-solving session and these require different programs to support them. Tools for groups are different from tools for individuals in many profound ways. These tools may be called *meeting tools.*

In the Colab, a meeting tool embodies either a function, such as providing a publicly visible cursor, or a process with associated functions, such as Cognoter, a tool for organizing ideas (described in the next chapter). Meeting tools must provide a coordinated interface, a multi-user interface, for all participants. The word "coordinated" is intended to capture both the basic user-interface issues explored earlier and the need for coherent and consistent sharing of information.

The character of almost any task changes when two or three people work on it instead of a single person. For example, when two people edit a document together they have to take turns typing if they are working on a single edit buffer. If two people edit asynchronously they have to save more information than either one of them would have when editing alone (i.e., they must leave meta-messages asking each other questions, making suggestions, or indicating who did what). Another example is brainstorming: a group brainstorming needs to be concerned with various social coordination cues in addition to generating ideas.

In general, these issues derive from the changed nature of the process once it involves more than one person. They touch the potential need for consensus, persuasion, disagreement, miscommunication, speech acts, and other things that are irrelevant when only one person is working.

Structuring the Process. Software tools inevitably structure the tasks they are used for — from simple tools, like a spreadsheet program, to more complex tools, like the meeting tools that are part of the Colab. This inescapable structuring of process can be seen as an evil to be diminished or as an opportunity for enlightened assistance. Colab and its tools cautiously lean

toward the latter. The tools are designed to allow experimentation with meeting processes and the effects of computer interfaces and internal structures on group work.

> *At first sight, the idea of any rules or principles being superimposed on the creative mind seems more likely to hinder than to help, but this is quite untrue in practice. Disciplined thinking focusses inspiration rather than blinkers it* [Glegg69].

Among other things, the following chapters argue that appropriate group problem-solving techniques (whatever the supporting technology) can be more effective and less confusing than the usual free-for-alls with dominating personalities, individual fears, fuzzy or conflicting goals, and repetitious arguments.

Social Coordination. There are three factors involved in coordinating social activity in collaborative systems: *access*, *participation*, and *characterization of collaboration*. All of these factors are pursued in more detail in chapter six.

Access. In a single user system, or when a group uses traditional support technology, access to the data is serial — only one agent interacts with data objects at a time. In tools for collaborative systems, social coordination is analogous to the conventions we use in conversation for taking turns and handling interruptions. It is inherently more flexible, for example, since a visual display easily provides a space multiplexed layout that allows more than one participant to enter information at the same time. It is an open question to what extent the coordinating signals can be stylized in useful and easily recognizable ways in a computational medium across different kinds of applications.

Participation. Since each participant using Colab has control of a personal workstation and immediate access to shared objects, there is no need to wait for a turn to add or alter an idea (unless the object is already being altered at that instant — see the discussion of *busy signals* in chapter four).

When all participants can act freely on shared objects, it is more difficult for any user to monopolize the session — unintentionally or otherwise.

The *characterization of collaboration* is elaborated in the following subsections.

Parallel Activity. A key issue in the design of meeting tools is to recognize and support those activities that can be decomposed for parallel action. For parallel action to work a task must be broken up into appropriately–sized transactions that can be done more or less independently by members of the group. If the transactions are too small then they will be too interdependent and interference will preclude any substantial parallelism. For example, if a group is creating a shared text, most interactions will not be at the level of individual keystrokes (though chapter six delineates occasions when this grainsize is desirable). If transactions are needlessly large, then opportunities for synergy are lost and the participants may as well be working alone.

With the ability to act in parallel on shared objects comes the ability to come into conflict. Conflict resolution strategies will become necessary in some cases, but often social constraints will coordinate the participant's actions in a face–to–face meeting. A conflict detection system or "busy signal" (see chapter four) can also graphically warn users that someone else is already editing, or otherwise using, a shared item. Shared object and database consistency strategies are discussed in chapter five.

WYSIWIS. *What You See Is What I See*. In its strictest interpretation, WYSIWIS demands that all computer displays show exactly the same thing: the views of all participants are sized and placed identically and the images of all cursors are visible — all participants have identical points of view and reference. Direct manipulation and WYSIWIS together create the illusion that each participant in a group has immediate and personal access to shared objects. WYSIWIS is the critical idea that makes possible the sense of teamwork in the barn–raising metaphor. It recognizes the importance of being able to see what work the other members have done and what work is in progress. It

lets each participant see from an object itself whether or not it is available for interaction.

Strict WYSIWIS is a useful ideal, but in practice it is too limiting. Relaxed versions of WYSIWIS turn out to be very useful (for more on this see Stefik et al [Stefik86]). For instance, relaxing the requirement that all pointers are displayed violates strict WYSIWIS. Whereas pointing is an efficient way to refer to things in conversation, displaying the cursors of all the active meeting participants at once is usually unnecessary and distracting. A compromise position would make public pointers visible on request (see Denoter below).

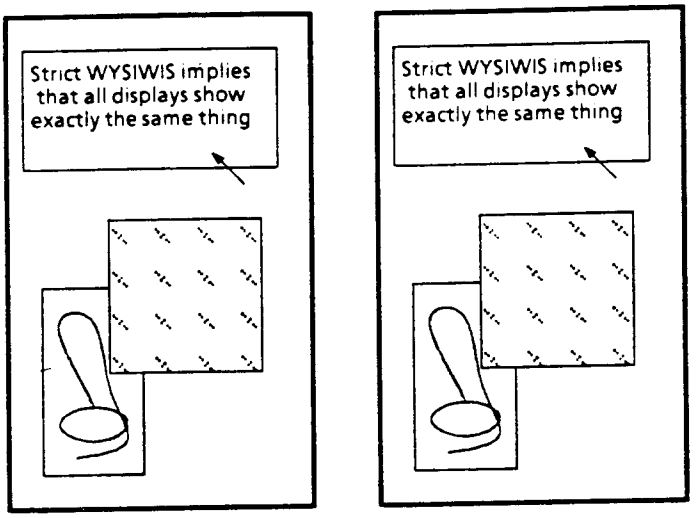


Figure 3.2. *WYSIWIS.* Strict WYSIWIS implies that all views are exactly the same.

Another WYSIWIS relaxation is to allow public views to appear at different places on different screens. This freedom to personalize screen use comes at a price: users will not necessarily have the same views of the shared

models. A participant cannot refer to screen objects by absolute position since there is no guarantee that the other participants have objects analogously placed. A related relaxation permits private views of public objects. These relaxations of WYSIWIS can be thought of as giving users the *ability* to see what everyone else is seeing without the *necessity* of seeing it (see Figures 3.2 and 3.3).

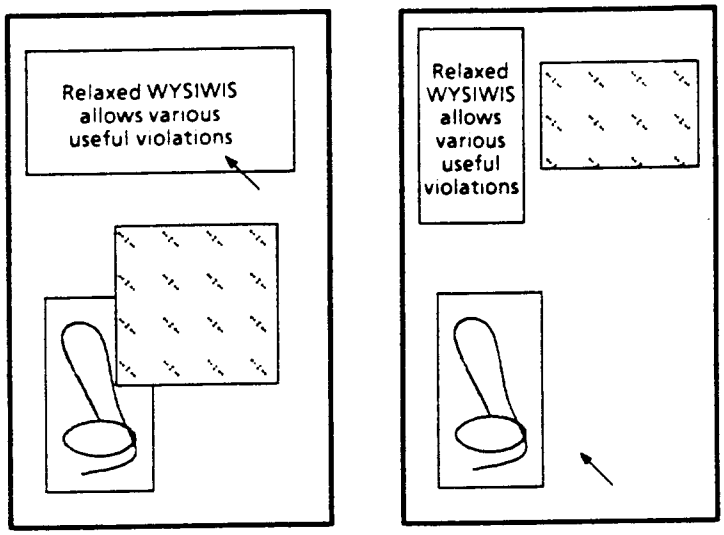


Figure 3.3. *Relaxed WYSIWIS.* Relaxed WYSIWIS allows various useful violations of WYSIWIS such as private control of views and public cursors appearing only on demand.

Public and Private. In a multi-user environment with simultaneous interaction there are two kinds of objects (and views of objects): private and public. A public object is shared with the group. Everyone can see it and, usually, do things with it. There can also be semi-public objects with access or capability lists determining who can see the object, or who can do what to the

object, respectively. A window and its contents are an example of a view of an object. There can also be private and public views of public objects. Private views and objects violate the WYSIWIS ideal.

Mutual Protocols. A plausible benefit of multi-user interfaces and face-to-face meeting support (like Colab) is the encouragement of mutual protocols. *Mutual protocols* are the articulation of the problem solving process by the participants. In a group using mutual protocols (consciously or unconsciously) individuals talk aloud about their actions, plans and goals — the group members don't just act (hopefully in concert), they discuss the process, their problem-solving strategies and tactics, what they are doing and why.

Why are mutual protocols helpful in the group problem solving process? They focus attention — people aren't as likely to wander off the point in a group setting since there is an implicit social contract to be at least somewhat directional and sensible (going off on a tangent about what to wear to the opera will usually be suppressed in a group working on an unrelated problem). Others in a group may also notice holes in one person's reasoning or facts. It is sometimes easier to criticize or edit what someone else is doing than to generate new ideas. Participants can question one another's assumptions or suggest alternatives. When each person articulates what he is doing and why he is doing it, he exposes weakness in approach or knowledge to himself and may well be drawn to more effective procedures. In an effective group session, all of the members may feel that they are building toward solutions by leaning on the others.

For mutual protocols to be an effective tool in real world problem-solving requires a certain amount of ego protection. In a hostile environment participants are less likely to participate or take chances like suggesting possibly silly ideas. Therefore, for mutual protocols to work well, the participants must be in a relatively safe environment. Such a benign environment is rare in most of today's meetings, but may be more feasible in computer mediated environments where anonymity and process structure can be invoked when desired.

Group Memory. The computer memory and WYSIWIS function as the group memory. WYSIWIS ensures a greater degree of equal access to shared objects than traditional techniques. Since each participant has easy access to anything the group has done, repetition of a point is usually unnecessary. If something has been said previously, but in the measure of a very long conversation it is now out of sight and feared to be forgotten, there should be easy ways to look back over the session history and to browse relevant contributions. The immediate public display of new ideas depersonalizes contributions by keeping them active and available.

Space and Time Revisited. A collaborative system tool can make hitherto difficult operations and options possible. For instance, meetings can be extended in space and time: since the complete computational state of a session, including history, is in the machines memory, the meeting can be stopped, shipped (if desired), and restarted — by the originating group or by another group.

Role-taking. A computer-based multi-user interface can support or suppress specialized participation in a problem-solving session. People often have preferred roles in a meeting — sometimes different roles at different phases of a meeting. Some will feel best capturing and translating the ideas of others and will function well as a scribe. Others have a critical facility (or fault) and will enjoy the role of checker and correcter. Some people are good at generating new or provocative ideas. Others are most effective extending the ideas proposed by others. In many cases a degree of role specialization will happen as a matter of course and personal preference. If the role-taking is useful for the group it can be institutionalized in software tools, if not, role-taking can be diffused throughout the tools.

Anonymity and Authorship. Using the computer as a meeting intermediary makes it possible to encourage either authorship of ideas or anonymity. Most of us are used to working with authorship information. Authorship establishes context for ideas and can be a valuable source of information — we automatically count or discount ideas depending on their source. However, in some kinds of collaborative problem-solving sessions it

may be more useful to emphasize the inherent merits of the ideas themselves. There will be places for both authored and anonymous ideas in collaborative systems.

Implementation of a Multi-user Interface

During the design and implementation of Colab and Cognoter, several new concepts and subsystems were developed that appear to be of general interest and utility for multi-user interfaces.

ActiveRegions and ActiveWindows. ActiveRegions and ActiveWindows[2] are subsystems capturing the useful notion of areas on the screen that are sensitive to mouse actions. A *Region* object knows about its screen position and extent. An *ActiveRegion* adds mouse sensitivity. ActiveRegions notice mouse events: the cursor moving into the region, the cursor moving inside them, the cursor moving out of them, and changes in the state of mouse buttons. An *ActiveRegionSet* (not surprisingly) is a set of ActiveRegions. ActiveRegionSets are hierarchical: that is, they may contain other ActiveRegionSets. An ActiveWindow contains an ActiveRegionSet and establishes the coordinate context for contained ActiveRegions. ActiveWindows, as windows, notice mouse events and pass them along to the enclosed regions under the cursor (see Figures 3.4 and 3.5).

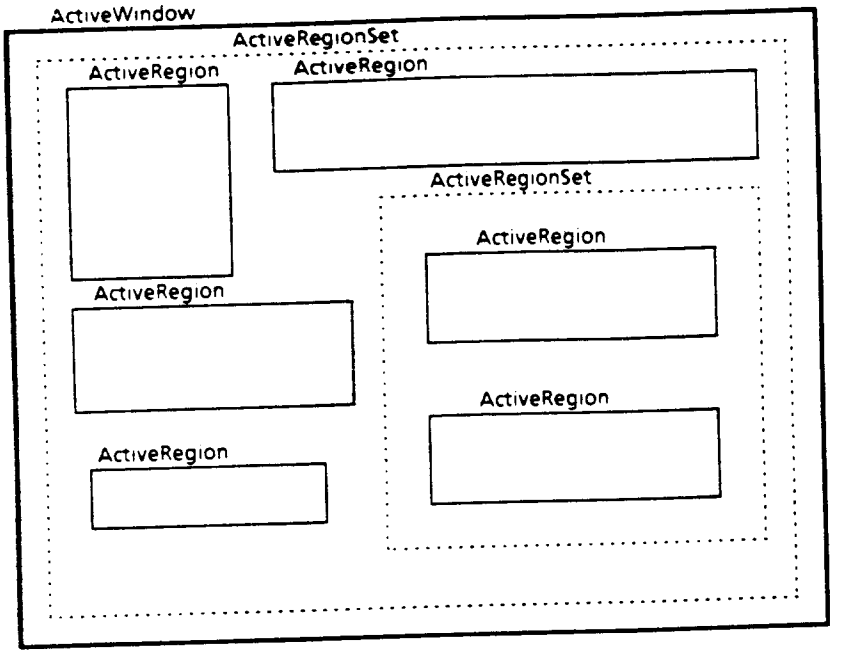


Figure 3.4. *ActiveRegion diagram.*

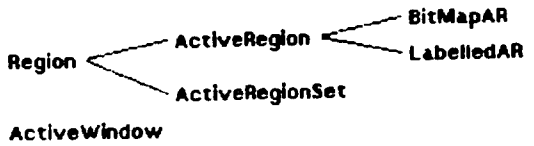


Figure 3.5. *ActiveRegion and ActiveWindow class lattice.* A Region knows about screen position and extent. An ActiveRegion adds mouse sensitivity. An ActiveRegionSet contains ActiveRegions and passes mouse events to any ActiveRegion affected. An ActiveWindow contains an ActiveRegionSet and establishes the coordinate context for contained ActiveRegions.

To determine which ActiveRegion, if any, sees a particular mouse event, the appropriate message, for example CursorMovedFN, is passed down the chain of ActiveRegionSets until it finally reaches an ActiveRegion. Figure 3.6 shows the general form of mouse sensitivity over an ActiveRegion. Each object in the hierarchy (ActiveWindow, ActiveRegionSet, ... , ActiveRegion) has a chance to respond to the mouse event and passes it along (see Figure 3.7).

```
CursorMovedFN:
        (send self BeforeARCursorMoved)
        (if (GetIV self ActiveSubregion) then
                (send (GetIV self ActiveSubregion) CursorMoved))
        (send self AfterARCursorMoved)
```

Figure 3.6. *CursorMovedFN.* Translated into English: the CursorMoved method takes whatever action desired (BeforeARCursorMoved), passes the event along to the ActiveSubregion, if any, and finally takes any post-action desired (AfterARCursorMoved).

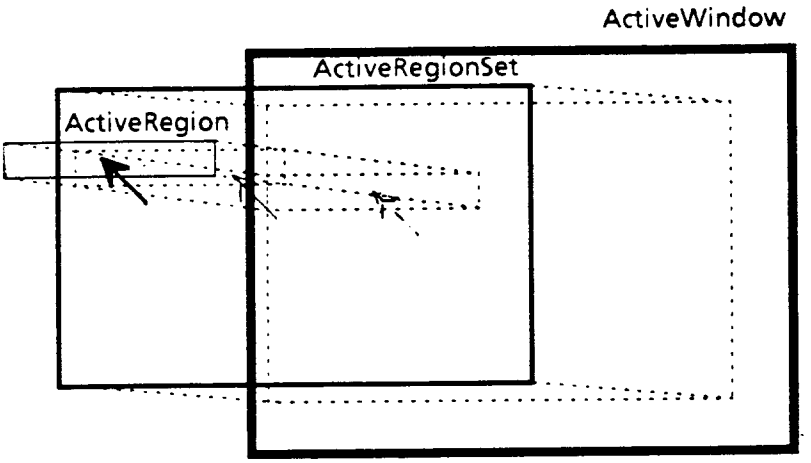


Figure 3.7. *A mouse event in an ActiveWindow.* Each object in the hierarchy (beginning with ActiveWindow, and on to an ActiveRegion, if any) may respond to the mouse event and then pass it along.

Pointing. In an early design of Colab the cursors of all participants were visible on all screens all the time. This flexibility in pointing was deemed important. There were two flaws with this idea. The first is purely pragmatic: system performance deteriorated when several mice flooded the Ethernet and process scheduler with position updates. This was doubly unacceptable since displayed cursor positions lagged so far behind that it became impossible to get acceptable cursor feedback or synchronization. Small feedback delays can cause large problems for human hand/eye coordination. Pointing must be immediate. The original motivation for displaying all cursors was the WYSIWIS ideal — poor response time made it more like "What You See Is What I Saw A While Ago".

The other problem with "constant cursors" was that most cursor motion was incidental and essentially private. Even though cursor images were personalized and could be readily distinguished, it turned out to be distracting for other participants to see foreign cursors flitting around on their screen while the owners were engaged in non–public activities. Multiply this incidental (private) cursor movement by the number of participants and it becomes unacceptably distracting. The confusion cost of this form of WYSIWIS outweighs its benefits (even setting performance difficulties aside).

Another possible approach to group pointing is multiple control of a single cursor. All users contend for a single public pointer. This fits strict WYSIWIS but forces sequentiality and violates the idea of screen control. An *additional* shared public cursor makes more sense (see Denoter below).

These early experiences with pointing showed that public pointing must be on demand only and, until rescued by improvements in network and scheduling technology, must minimize ethernet packets.

Denoter. To make a cursor display available on demand, the *Denoter* special–purpose pointing tool was developed[3] (see Figure 3.8). Denoter appears as a box of arrows that can be picked up and carried with a cursor. The arrow is personalized and appears on all machines registered with the

conversation. Even though the Denoter pointing tool can ignore position updates when it gets behind, it still suffers from some feedback delays. The screen–absolute positions of Denoter arrows highlights a drawback of allowing private views of Colab models: you can't point at things that have been privately placed or moved — one person's upper–right window may be someone else's lower–left window. Denoter could have window–relative positioning, but with the undesirable side–effect of discontinuous movement between windows and still with no guarantee that a pointed–at object actually appears in all private views.

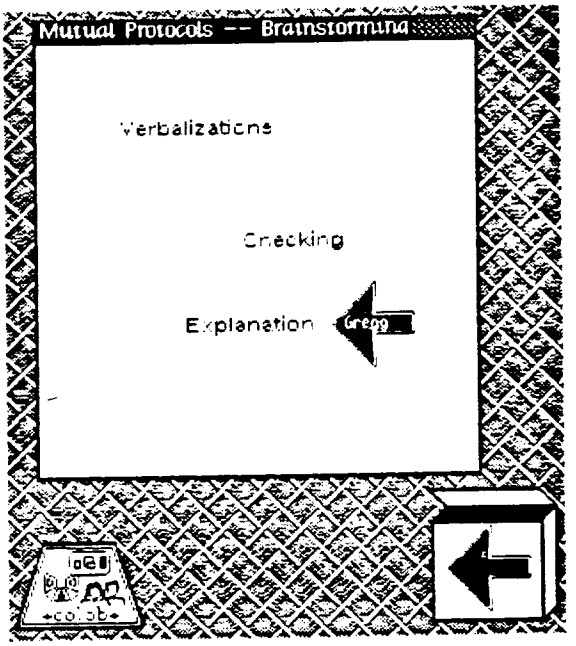


Figure 3.8. *Denoter, a Colab pointing tool.* The box of arrows will supply arrows that follow the cursor constantly (broadcasting to associated conversations) or arrows that can be dropped on the screen for long term highlighting.

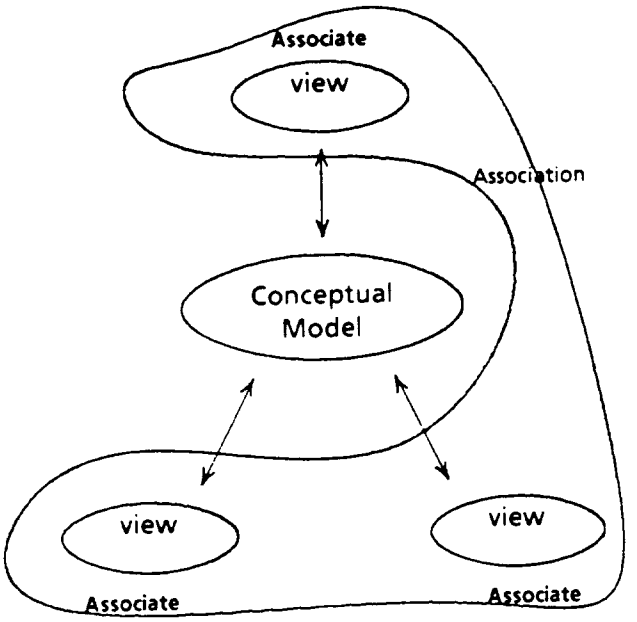


Figure 3.9. *Association/Associates*. The shared model of an object is the association. The individual views of the object at each node are the associates.

Associations. The shared model of a public object, in Colab parlance, is called an *association*; an *associate* is an individual view of a public model (Figure 3.9). In the strict WYSIWIS case, changes to an association, made through an associate (or its controllers), will affect all other associates in the association. In the relaxed WYSIWIS case, the associate views may or may not be effected depending on what portion of the model they are displaying. An association has a single name across machine boundaries, a unique identifier

(UID) — each associate can be referred to by the same UID. UIDs, as implemented in Loops, have potentially useful information like the time of day, the local machine name, and the user that created the object encoded in them.

Communication protocols. The Colab communication protocol is implemented by a combination of system facilities and programming abstractions. Communication over the ethernet is supported by a layered protocol (see Figure 3.10). At the lowest level is the ethernet packet transport protocol: PUP, NS, TCP/IP, etc. The implementation of Colab has mainly ignored this level and relied on higher level remote procedure call packages.

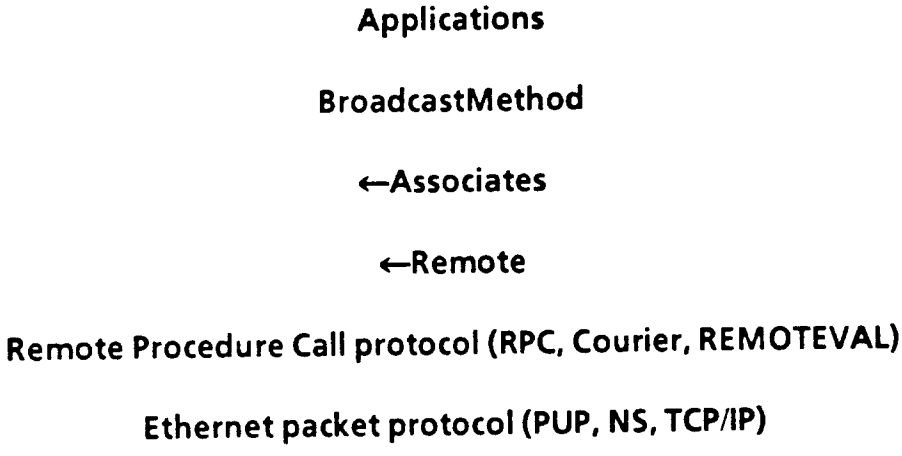


Figure 3.10. *Layers of abstraction in Colab machine communications.* The ethernet layer is concerned with transport protocols. The RPC layer supplies techniques for executing functions in remote environments. ←Remote is the only method concerned with details of the RPC layer. ←Associates uses ←Remote to broadcast changes to an entire association. BroadcastMethod is the only layer seen by an application programmer: this specifies that the method is to take effect on the association (using ←Associates) on all participating machines.

The lowest level of interest in the implementation of Colab is the protocol for remote procedure calls (RPC) [Birrell83] [BNelson81]. RPC, with appropriate preprocessing (Thompson's *Lupine* in Colab's case [Thompson84]), allows execution of ordinary functions in remote environments invisibly. It is not necessary for an application, or even an application programmer, to know whether or not a given function is executing locally or remotely. Lupine takes an ordinary function, say "MagicWords", renames it, to perhaps "OldMagicWords", and creates a new "MagicWords" that wraps the necessary broadcast protocol around OldMagicWords. MagicWords will henceforth execute in the specified remote environment. This RPC layer has been further abstracted and encapsulated so that Colab applications programmers are protected from changes in the particular choice of RPC protocol used[4].

←Remote and ←Associates. Pronounced "Send Remote" and "Send Associates", these are Lisp macros that do the right thing for the chosen transport protocol (for instance, packing strings for RPC stubs). Running on top of the RPC implementation, they supply a mechanism for sending messages to an object on a remote machine and for sending messages to all associates of an object, respectively. ←Remote is the only method affected by details in the underlying RPC mechanisms. ←Associates is made up of ←Remotes. This is the only code directly dependent on the RPC protocol. It only has to be debugged once (for each protocol) and then it can be used on faith thereafter. In fact, applications programmers never have to use ←Remote or ←Associates directly: they use the higher-level *BroadcastMethods* instead.

BroadcastMethods. This is the object-method-level abstraction that application programmers need to think about. The idea behind *BroadcastMethods* is that they extend of a basic object method in Loops to a method that broadcasts to all members of an association — a special method with an implicit ←Associates. When a BroadcastMethod is invoked on one machine the method is run on all machines involved in the conversation.

For example, suppose that *MoveTo(newPosition window)* is a BroadcastMethod of an object that moves the object, say *object7*, to a specified *newPosition* in a specified *window*. If, as a result of some user action, *object7* receives a *MoveTo* message on one machine, then *object7*'s associates (for simplicity, take *object7* as the association UID in this case) on all the other participating machines will also receive the same message and parameters (*window*, for instance, would be either a symbolic name or another association name). The details of queuing the message for broadcasting and the actual transmission to the correct machines are handled invisibly by BroadcastMethod using association data structures and methods of the enclosed method.

A BroadcastMethod works by adding an argument: an author flag that specifies the originator of the message. This flag is primarily used to prevent the message from being executed more than once by each machine and, more importantly, it stops rebroadcasting. The BroadcastMethod then packages the message and its parameters up and puts the packet on the broadcast queue to be sent out to all participating machines via the RPC protocol.

By providing a layer of abstraction, BroadcastMethods greatly simplify the organization of communication between replicated objects. For Colab a programmer normally writes an ordinary method for an tool, debugs the method for a single user on a single machine, and when satisfied that it works, simply changes the key word "Method" to "BroadcastMethod" in the method definition. This tiny change causes the method to be expanded as above with the extra argument and inserts the necessary ←Remotes and ←Associates to make all object associates find out what happened. Often the new broadcasting method just works (though there may be unforeseen bugs relating to machine interactions — usually in the multi-user interface).

Figures 3.11 through 3.13 illustrate an example concerning *Noter*, a simple message passing tool.

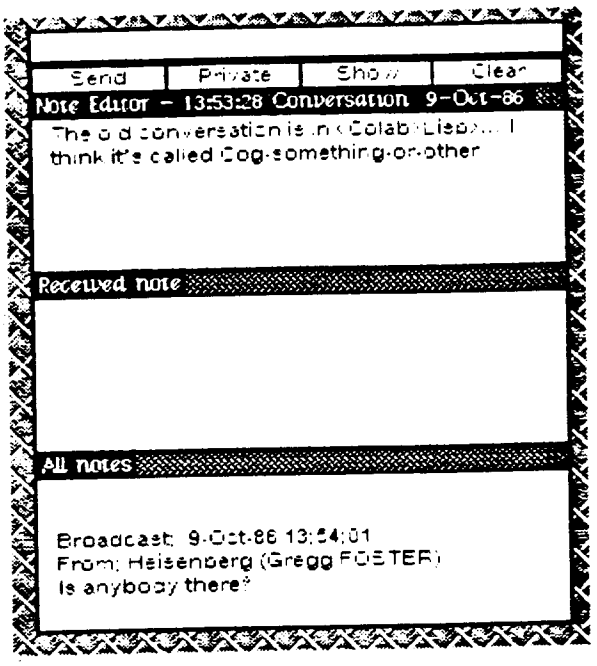


Figure 3.11. *Noter, the Colab note passing tool.* There are three main windows: the message editor, the most recently received message, and the message history. A fourth window for private messages opens if necessary. In this example, User A is about to send a message to User B...

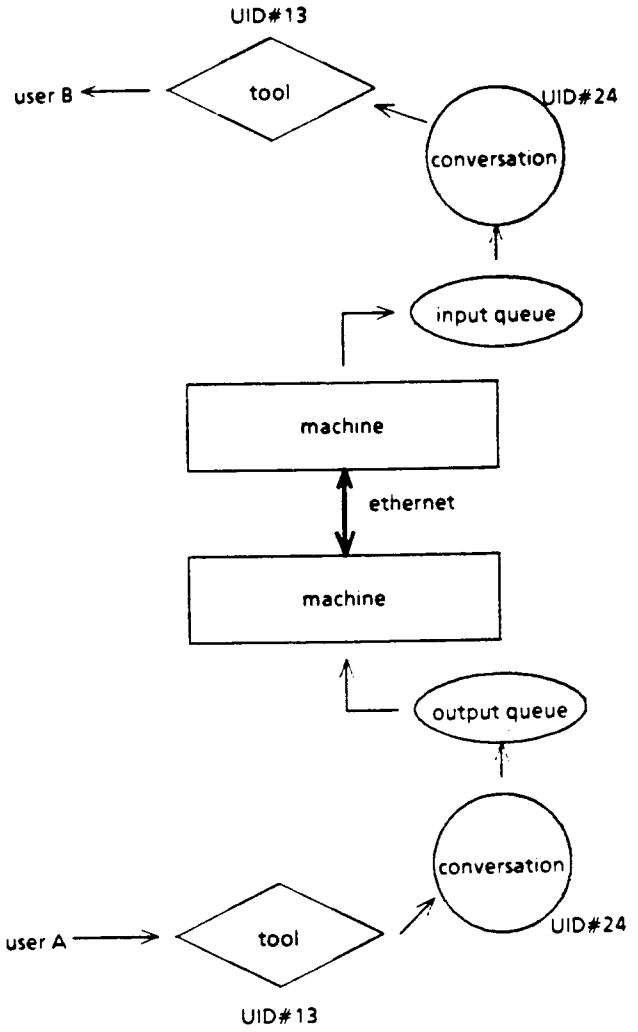


Figure 3.12. *Communication protocol diagram.* User A is sending a note to User B. User A invokes the *send* command on Noter (with a simple UID # 13). Noter#13 passes the message to its enclosing conversation (UID # 24 — see chapter five for more conversations) which in turn adds the message (containing the note) to an output queue. The message is eventually sent over the Ethernet to B's machine. B's machine fields the message and puts it on its conversation#24's input queue. B's conversation#24 gives the note display message to its Noter#13 which finally displays it for user B.

```
(BroadcastMethod
 ((Noter NoteToPrivate) self toCollaborator note)
                        (* colab: "25-Feb-85 17:44")

(* * Sends a private note to a collaborator.)

(LET ((me (@ ColabExec me)))
     (COND ((OR (EQ me toCollaborator)
                (EQ me (@ note sourceCollaborator)))
            (← self DisplayInPrivateWindow note)))
 ))
```

Figure 3.13. *BroadcastMethod Code.* This is the Loops code to send a private note to a selected collaborator. Notice how BroadcastMethods hide most of the complexity in Figure 3.12. This code, in English, takes a collaborator and a note as arguments and prints the private note in the source and target collaborator Noter private windows only.

Meta-programming and Programming Pragmatics

Programs that need to run in concert over several machines are difficult to write and debug. This truism combined with the need in Colab to experiment with the broadcast techniques themselves and to frequently change basic system code has made the creation of programming techniques and tools to simplify code development, testing, and revision essential. This section discusses some of the programming and meta-programming pragmatics developed during the implementation of Colab and Cognoter.

U-S-D. As alluded to earlier, one desirable feature of a multi-user programming effort is that it should be easy to convert a single-user tool functions into a multi-user functions. The goal is to design and test a tool on

a single machine and then be able to trivially make it available to several simultaneous users. This is not easily achieved in the general case, but if a certain amount of programming discipline is practiced it becomes possible. The factoring of concerns into functional categories clarifies the programming effort by reducing complexity and exposing hidden dependencies.

Object methods can be thought of in three categories: *user–actions, semantic–actions, and display–actions* (U–S–D — see Figure 3.14). This partitioning of method responsibilities reflects the usual pattern of actions in a single interaction with a Colab tool. It also, therefore, represents good Colab programming style. It is reminiscent of the familiar *Read–Eval–Print* top–level loop in Lisp systems and the *Model–View–Controller* partitioning mentioned before. In the early Colab programming effort, most new methods, in retrospect, tended to have aspects of these three categories sprinkled throughout them. During debugging, these undisciplined methods usually had to be extensively rewritten and refined. The useful refinements tended empirically to express these categories.

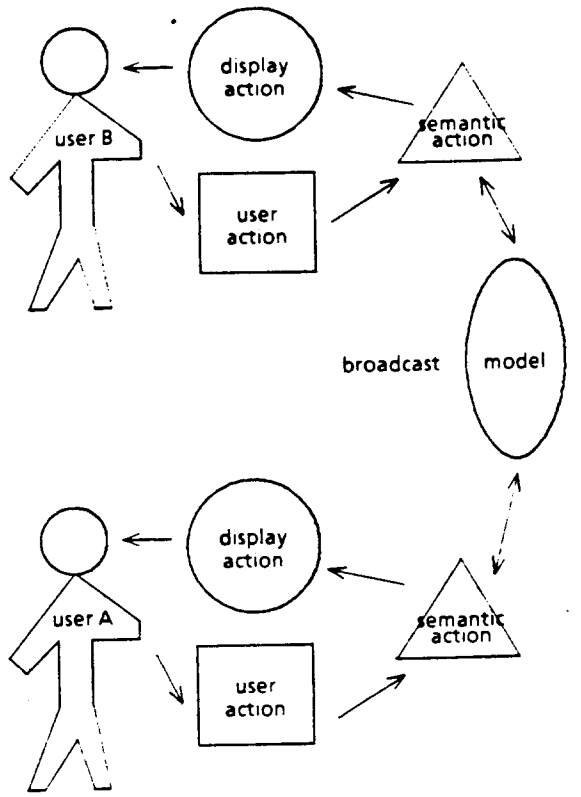


Figure 3.14. *Semantic–Actions, Display–Actions, User–Actions.*

There are also enclosing methods that glue the others together and control method transition. In existing code, the enclosing method is often the user–action method, though it may prove to be a valid fourth category that should be separated (see Figure 3.15).

```
Move(item window):
    MoveUserAction(item window):
        «user interactions yielding a new position»
        MoveSemanticAction(item window newPosition):
            «actual changes to database»
            Display(item window)
            «other display actions»
        «cleanup display actions»
```

Figure 3.15. U–S–D *example*. The three (four) method types are usually nested. The general *Move*, enclosing method, is by a participant. Move invokes *MoveUserAction* which controls the local user manipulation of the item. Once local interaction is finished, *MoveSemanticAction* is called with the proposed changes to the database. The changes are broadcast to all participants where they are installed and local displays are updated by the general purpose *Display* method.

User–action methods. User–action methods are invoked at a user's request, usually the result of a mouse action on a displayed active object. In general, through user–system–database interactions, they establish an operator and parameters (position, label, new object, etc.) for a change to the shared database. A user–action may consist of several actions (changes, adjustments, inputs, etc.) and database accesses before reaching the desired new state for broadcast. User–actions are not broadcast since the activity concerns only the initiator of the action. For example: A user wishes to add a new item to a window (taken from Connoter, see chapter four). The user buttons the window background, types in the desired label for the new item and moves a shadow of the item's region to the desired position. Once the new item is released, the user–action is over and the change to the tool model can be broadcast.

Semantic–action methods. Semantic–action methods make the actual changes to the shared database. These methods are broadcast to all associates of the changed object so that changes will propagate and copies of the shared database will be kept up to date. These are the only methods that should broadcast. BroadcastMethods calling BroadcastMethods is not allowed since this can circumvent the safeguards against circular broadcasting.

A single semantic–action may change several things in the database and may spawn several display–actions. For instance, a user–action may result in a semantic action to redisplay all items proportionally in a smaller window. This single semantic action could invoke dozens of display–actions. An additional advantage to packaging up predictable operations in this way is the evening out of the computational load over the network. However, it is not always clear whether to broadcast a single high level semantic–action with several side effects or to broadcast several smaller scale semantic–actions. The single complex action is easier on the network traffic and the process scheduler, but may be more difficult for the programmer. Continuing with the new item example above: Once the new item is in place, the change is broadcast to all associates, who update their copy of the session database. Now that the database has reflected the change, the new item can be displayed.

Display–action methods. Display–action methods update the machine displays. They need execute only locally since each machine has a copy of the session database and is the only controller of its screen. Again, a semantic–action often causes many display–actions. There may be more than one view on each datum, also causing more than one display–action. Back to the new item example: All machines, including the originating machine, individually display the new item as appropriate to the views they are supporting.

Testing and Debugging Distributed Applications. The programming discipline presented above helps make multi–user programs understandable, but tools for tracing and intercepting messages on the network, direct

network debugging, are also needed. A *ConversationViewer*[5] was created to monitor message traffic. This viewer can be attached to any Colab tool. It displays messages put onto the broadcast queues and monitors the processes used to send messages between machines. The viewer has proved useful in detecting cases of unnecessary or incorrect message sending, and especially in bringing BroadcastMethod loops to light.

Another tool for debugging distributed applications is the *PatchCollaborators* program. This program uses the RPC protocols already established by the Colab environment to propagate changes to Colab objects across machines. When debugging software running on more than one machine, it is frequently useful to make changes on a single machine and broadcast the changes to the other machines so debugging can continue (on the rare occasions that all bugs weren't anticipated in the first iteration...).

Summary

This chapter introduced collaborative systems, real–time computer–based systems for cooperative work. Multi–user interfaces were introduced and considered in the context of established user interface principles and their implications. The major section of the chapter considered the attributes of computer–based tools to support cooperative problem–solving.

In the course of implementing the Colab system several ideas useful to the design of multi–user interfaces were developed: ActiveRegions and ActiveWindows for mouse sensitivity over screen regions, the Association abstraction for treated shared models as local objects, and the BroadcastMethod abstraction for object communication. The partitioning of method functions into user–actions, semantic–actions, and display–actions led to a clearer and more maintainable system.

The next chapter describes Cognoter, a tool for planning presentations. Cognoter is a multi–user Colab tool embodying many of the attributes (and difficulties) described in this chapter. The following chapters discuss the issues

and concepts that arise in the design, implementation, and use of Colab and Cognoter.

Notes:

1 According to Peter Deutsch, a *view* is "a function to compute something that could appear on the screen". When the view is defined this way, there is a fourth partition to consider: the display *media*. A medium in this context is the basic image transformations and clipping. It is a bitmap, not necessarily the screen bitmap. Views always display through a medium.

2 ActiveRegions and ActiveWindows are being superseded by the *Wegion* work going on at Xerox PARC. Future versions of Colab and Cognoter will likely take advantage of this work.

3 Stanley Lanning did most of the implementation of Denoter.

4 As this chapter was finished the system was converted to the *Courier* network communication protocol with almost no effect on higher levels and no effect on applications.

5 Kudos to Stanley again, for implementation of the ConversationViewer.

4

The best way to have a good idea is to have lots of ideas.

— Linus Pauling

Cognoter, a Tool for the Colab

Cognoter is a program that helps a cooperating group of people to organizing their thoughts for a presentation, e.g., a paper or talk. Cognoter provides a multi-user interface and a structured meeting process. An annotated graph of ideas is built up by the group in three phases: *brainstorming* for idea generation, *ordering* for idea organization, and *evaluation* for choosing what will be finally be presented. Interesting aspects of Cognoter include direct spatial manipulation of ideas and their order relationships, support of parallel activity, and incremental progress toward a total ordering of ideas.

Introduction

Cognoter[1] is a meeting tool for preparing presentations — talks, papers, memos, anything in which ideas must be organized so that they can be understood. At the end of a successful meeting using Cognoter the participants will have an annotated outline of ordered ideas and associated text. Cognoter has been used to prepare outlines for several talks and papers, including this dissertation.

Many of the multi-user interface and collaborative system concepts that were introduced in a general way in the previous chapter are reified here. In addition to any actual usefulness that Cognoter might find as an organization tool, the design of Cognoter was intended as a test-bed for exploration of these concepts.

In an environment where there is little cost associated with trying things out, things tend to get tried out. Meeting tools, such as Cognoter make it easy to re-arrange items, and to alter their relationships to each other. They encourage a breadth of approach. Such flexibility is useful over a range of applications.

The underlying philosophy behind Cognoter is two-fold. On one hand a tool should not be too prescriptive. People in a group should be able to jot down ideas as they think of them, without regard to order or relevance, and then play around with the ideas and their relationships until they are satisfied with the overall content and organization. On the other hand, some active assistance, a supportive environment that guides consensus and funnels progress toward a coherent organization, is also desirable. Cognoter combines these two points of view.

Whereas the previous chapter was primarily prescriptive, this chapter is mainly descriptive: it describes the theory and practice of Cognoter. Implementation discussion is in the next chapter and discussion of Cognoter's use in real-life, its successes and failures, can be found in chapter six.

Comparison to Idea Processors. Cognoter is similar in some ways to currently available "idea processors". These include commercially available personal computer programs like ThinkTank™ [O'Connor84], and research projects like the Notecards system developed at Xerox PARC [Halasz86]. All of these share the goal of organizing ideas. All express an organizational model and display ideas graphically.

The most important difference between Cognoter and most other idea processing tools is that Cognoter is designed for simultaneous use by multiple participants (though the organization process it embodies is also useful for a single user). It is also designed to manage the complexity of organizing ideas in more direct ways than existing idea processors. Cognoter divides the organization process into smaller and different kinds of steps. In Cognoter, independent decisions can be made independently, ideas can be generated and simply "put on the table" without concern about their position in relation to other ideas. The steps for organizing ideas are incremental and efficient. Cognoter separates the concerns of idea generation, ordering, and evaluation.

In ThinkTank, ideas are always organized in an outline — there is no place else to put them. When a new idea is added the user must also decide, at creation time, where it comes in the scheme of things. In most idea processors it is a simple matter to change an item's position, but it is not readily apparent when an item is only provisionally placed. Items appearing in nonsense order in an outline look no different than carefully ordered items.

ThinkTank and NoteCards support well-known metaphors for organization (outlines and file cards, respectively). Whereas Cognoter can display organized ideas in an outline format, a goal of its design was to find more powerful ways to display, consider, and manipulate ideas. Cognoter does more than supply an active reflection of a known model (such as an outline), as shown in the next section it also assists the organization process.

Scenario. This chapter has a running example of Cognoter in action concerning three mythical collaborators, planning to write a parallel version

of this chapter (see Figure 4.1). The example lives in the figure captions. Whereas the example figures are actual screen snapshots of Cognoter, the example session has been simplified to isolate the points being made. Also the window arrangements have been simplified and overlapped to allow compact figures.

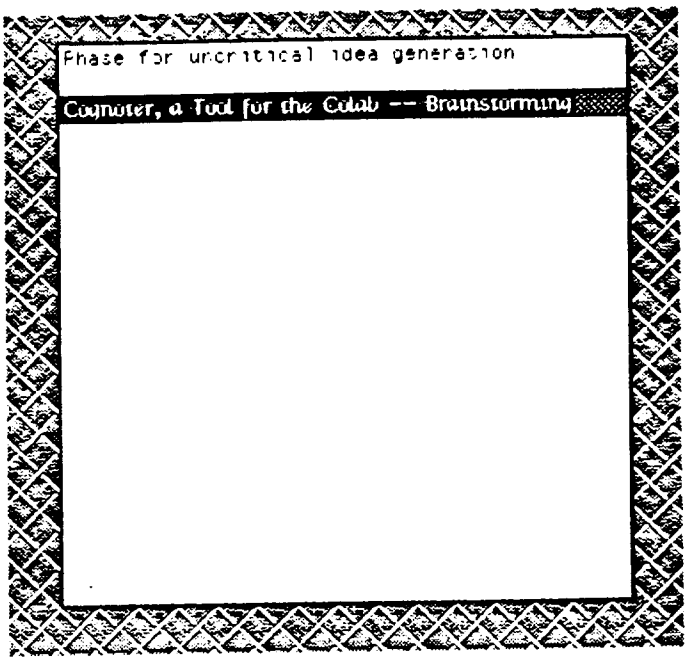


Figure 4.1. *Tabula Rasa*. The tool is initialized. ***Example***: Before starting their session, the trio planned to write a chapter about Cognoter. They agree to meet in the Colab. Since three machines are ready, running fresh Colab environments, one of the collaborators starts Cognoter and chooses "Cognoter, a Tool for the Colab" as the session title. The other two collaborators are added to the tool conversation and the group is ready to brainstorm. The above window is seen on all participating displays.

Cognoter's Problem–solving Process

The organizational techniques embodied by Cognoter are similar to techniques that have long been used without computer support. Participants come to the session with a general goal in mind, something like: "Let's use this tool to plan a paper about the stuff we've been talking about recently". But a typical group will not have a clear notion of the best framework to present their ideas or even what the key ideas are.

How does a group get past the blank page? Starting at the best guess for a beginning and diving into a depth–first approach to an outline is almost certainly wrong: "What's I?" "Now, what's I.A?" Better, but similarly misguided is a breadth–first approach where the group attempts to generate the handful of major topics: "What's I?" "What's II?" A more flexible approach, including bottom–up, top–down, and middle–out techniques, is needed.

A group planning a presentation needs to do several things. First, relevant ideas must be accumulated. The participants need to decide which ideas are related and how the ideas go together. They need to determine the presentation dependencies between ideas: which ideas should come before which other ideas. Finally, they need to decide which ideas or groups of ideas are at the wrong level of detail or are irrelevant to the presentation.

Phases. Cognoter organizes a meeting into three phases: *brainstorming*, *ordering*, and *evaluation*. Each phase emphasizes different kinds of activities. As the group advances through the phases, the set of possible actions is expanded; for instance, brainstorming, emphasized in the first phase, is still possible in the last phase. Groups that find the rigid enforcement of phases too confining can skip immediately to the last phase where all operations are possible.

One of the goals of work with Colab is to experiment with various structures and techniques for group problem solving. The particular three phases mentioned above are the current best guess, based on successful traditional techniques and the expected strengths of computer–based tools.

These three phases are not useful for all kinds of problem-solving: different meeting processes are needed.

Other researchers have described similar phases for problem-solving. In their description of general writing tasks, Hayes and Flower independently developed three analogous phases [Hayes80]. They refer to them as the *Generating*, *Organizing*, and *Translating* phases. In his *tecretarial* (technical secretary) group facilitation work, DeKoven divides sessions into three phases that he refers to as: *Collect*, *Connect*, and *Correct* [DeKoven86]. Von Oech suggests that the *germinal* phase has been generally neglected in our educations [vonOech85]. According to von Oech, the germinal phase consists of idea generation (brainstorming in Cognoter) and idea manipulation (ordering). He goes on to describe the *practical* phase: critical evaluation (evaluation) and execution (outline generation). Polya describes an approach to solving mathematical problems in four phases: *Understanding the Problem* — clearly understand what is required; *Devising a Plan* — see how the items are connected and decide on an approach to the problem; *Carrying out the Plan* — do it; *Looking Back* — review and discuss the solution (or lack of solution) [Polya57].

Brainstorming phase. The goal of the brainstorming phase is to get many ideas "on the table" for possible inclusion in the presentation. Too many ideas are better than too few — it is easier to prune than to generate. Since the goal is quantity, participation by all members of the group must be encouraged and any actions that would inhibit the flow of ideas should be discouraged. Ideas are represented in Cognoter by short descriptive *items* that are displayed in a public window. Items are not evaluated or deleted in this phase and, at first, little attention is paid to details of organization.

This theory of brainstorming is reflected in Cognoter's software and in its "rules of the game". Participants can act simultaneously, adding new idea items as they think of them to a Cognoter window (see Figure 4.2). In the brainstorming phase there is only one window for all items (in later phases any number of subwindows are permitted).

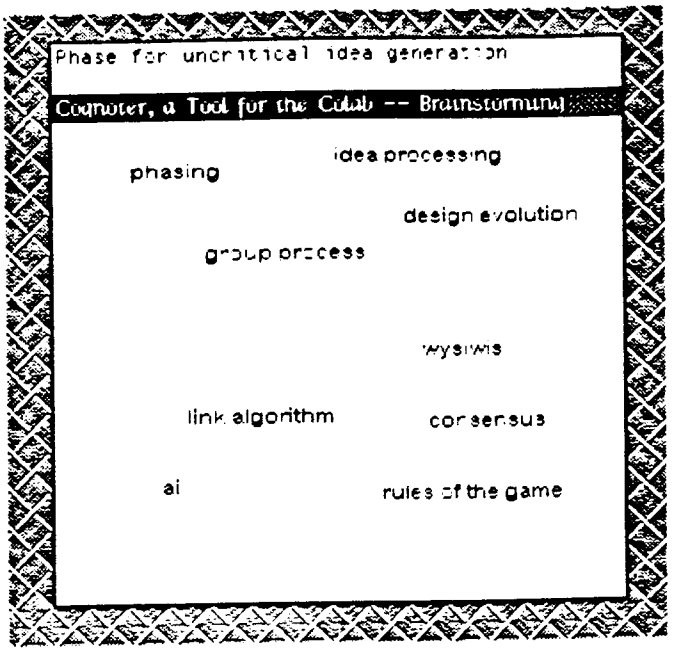


Figure 4.2. *Cognoter in action: Brainstorming.* This figure shows the main Cognoter window early in the session. The goal of this phase is to generate as many ideas as possible with little regard for their positioning. ***Example***: Here, the collaborators have begun brainstorming. They simultaneously add items by clicking the mouse in the background of the window and typing in a short title or phrase that stands for the idea. As soon as items are entered, they appear on all screens. If no one were actually typing at the moment of this snapshot, this window would be seen just like this on all screens.

Participants may attach *supporting text* to any item by selecting the item and using a private editor. Supporting text is used to clarify or amplify an item appearing in the main idea window. Once text is attached to an item it can be displayed publicly or further edited by any participant (see Figure 4.3). Items with text attached to them are displayed in a bold font.

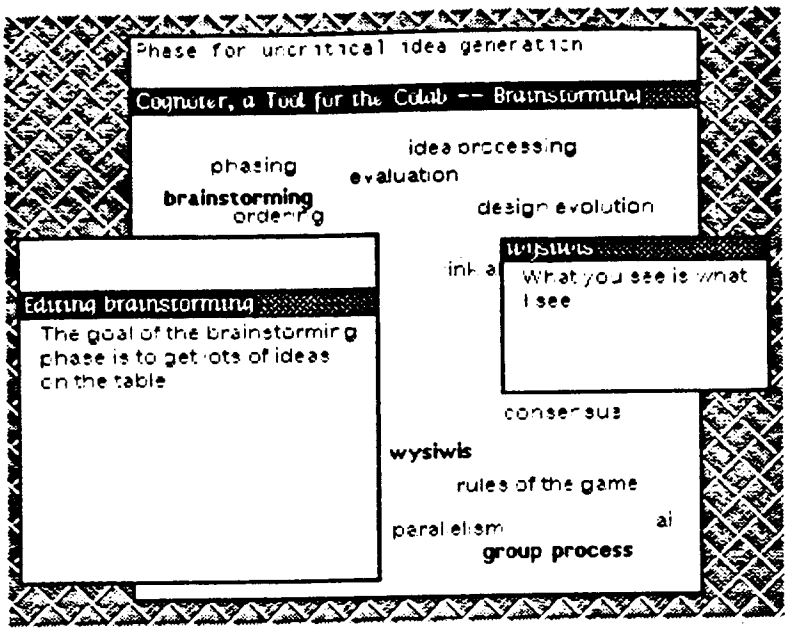


Figure 4.3. *Supporting Text.* In addition to adding new items, participants can also amplify items by attaching supporting text. *Example*: The collaborators have continued to brainstorm. Supporting text explaining ideas in more detail is entered by selecting the item with a mouse and then using a text editor in a separate window. Above, on the left, one of the participants has just finished editing the text attached to the item ***brainstorming***. Text supporting item ***wysiwis*** is being publicly displayed to the right. Items with text attached to them are displayed in boldface. Notice in the upper left that the collaborators have moved or placed related items "phasing", "brainstorming", "ordering", and "evaluation" together.

Items cannot be deleted in this phase, and it is against the rules of the game to verbally criticize ideas. They can be moved freely, but there is little other organization during this phase. It is time to move on to the Ordering phase when the main window is too full, a jumble of ideas begging for organization.

Brainstorming Phase Operations

Tool Operation	Description
NextPhase	Enter the next phase
Help	Get some help
AddItem	Add a new item
AddNewRelation	Add a new relation type item
Display	Redisplay the items
RedisplayAll	Redisplay the entire association
Shape	Reshape the private view
ShapeToFit	Fit the display neatly around the items
Scrunch	Shrink the display window and ShapeToFit
Spread	Expand the display and ShapeToFit
Item Operation	**Description**
Edit	Edit this item's attached text
EditLabel	Edit this item's label
Copy	Copy this item
Move	Move this item
Show	Show the text attached to this item

Table 4.1. *Brainstorming Phase Operations.* Tool operations affect the tool in a global sense. Item operations affect the item selected. Indentation in the table indicates submenu items. There are *accelerators* (faster, but less obvious ways to do things) for the most frequently used and time-critical operations, such as AddItem and Move.

Ordering phase. Once a group of Cognoterers has a window full of items, they are ready to put them into order. There are two basic operations added in this phase: asserting that one idea should be presented before another and asserting that several ideas belong together. Both of the ordering operations, *linking* and *grouping*, support incremental decision-making. An aggregate of small ordering decisions about what comes before what and what goes with what can yield a total order of the ideas being considered. A visual representation of ambiguities in the current

ordering constraints can serve as a guide to participants that more ordering constraints are needed.

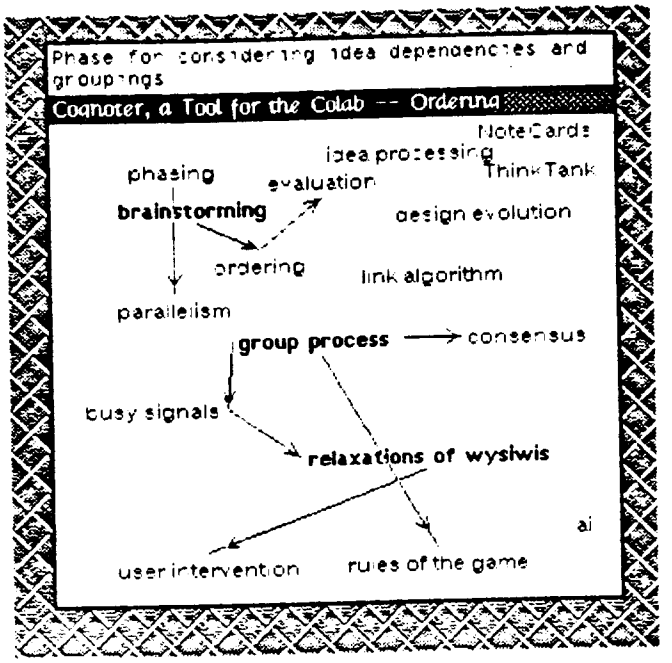


Figure 4.4. *Links establish the order of presentation.* The order of ideas is established incrementally by linking items. The semantics of a link are that the item at the tail should be presented before the item at the head. Links can be added or removed through item operations. Items will usually have one or more links to other items. *Example*: The collaborators have begun ordering their ideas. One of the collaborators has decided that *group process* should be presented before *rules of the game* and connected them with a link. The order of several has yet to be specified (*group process* and *evaluation*, for example, are relatively unordered).

Participants indicate precedence by linking items: a link is a suggestion that the item at the link tail should be presented before the item at the link head. This may well be accompanied by verbal discussion: "I'm putting *group*

process before *rules of the game* since we'll need to motivate *rules of the game* before we assert it." Linking is represented visually by directed arrows between items as shown in Figure 4.4.

The item moving operation makes it possible to discuss grouping operations before actually doing them by moving items near each other before clustering. Thus, spatial clustering provides a suggestive intermediate indicator of organization before formal divisions are agreed upon.

Items can also be clustered into groups[2] as shown in Figure 4.5. When items are grouped, they are replaced in the Cognoter window by a single new group item (surround by brackets). The items that were grouped can be displayed and manipulated by *opening* the group item. An opened item displays the contained subgraph of items in a separate window. A link to or from a group item is treated like a link to or from the whole contained subgraph. Items can be moved across group window boundaries with links and display being adjusted accordingly.

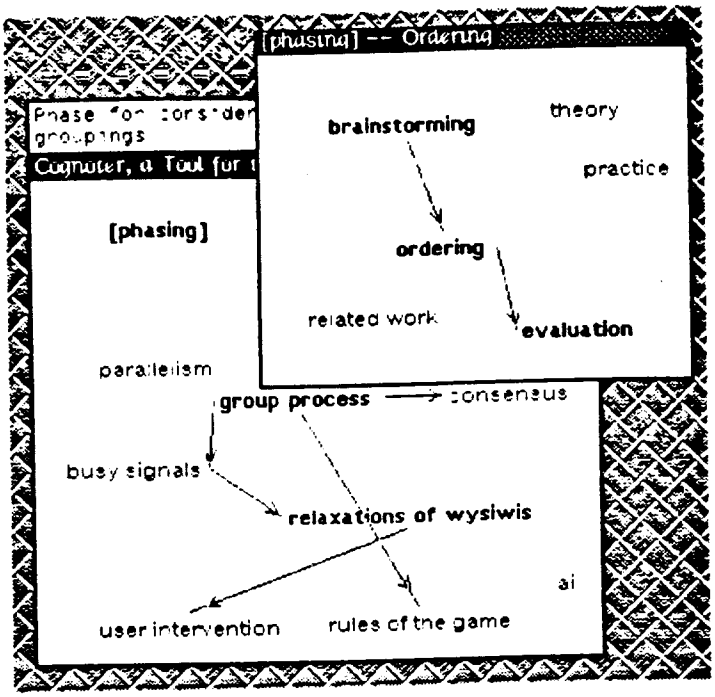


Figure 4.5. *Groups describe the hierarchy of ideas.* Items that should be taken together can be grouped. The items are replaced with a new item with brackets surrounding it to indicate that it is a complex item. Each group has an associated window for displaying the items it contains. *Example*: A collaborator has converted "phasing" into a group item and opened it (upper right), showing the partially ordered ideas contained within it. Some of the original ideas have been moved into it and some new ones have been added.

Cognoter provides operations that allow items to be ordered incrementally. The link-forming operation organizes the ordering task so that a partial ordering of items is refined stepwise towards a complete ordering. Transitivity and grouping operations make it possible to organize the ideas efficiently with a small number of links. Optionally the places where the ordering is over- or under-constrained can be indicated (see Figure 4.6). The groups and links are used collectively in the final phase to determine a

complete order of idea presentation. Circular or contradictory linkings can be carried along and resolved when desired.

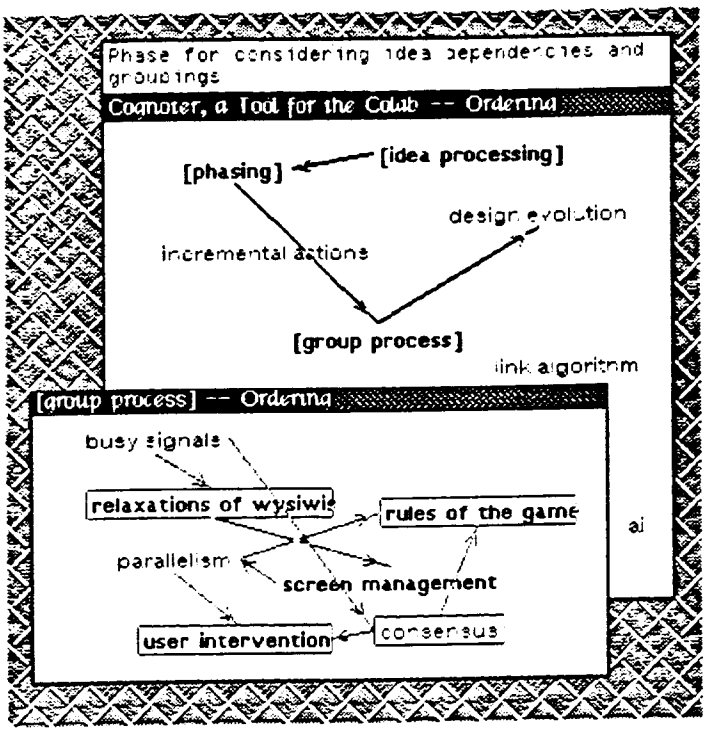


Figure 4.6. *Items ambiguously ordered can be automatically highlighted.* Ambiguously ordered items can be highlighted to guide linking and grouping. *Example*: The group has continued to group and link. Ambiguity highlighting has been turned on inside the opened item "group process". The order of the boxed items cannot be completely determined by the existing links. If a link were added from "rules of the game" to "user intervention", the order of both of those items would then be determined. Boldly drawn links indicate a link to or from a group item — these are treated specially in some cases.

Ordering Phase Operations

Operation	Description
NextPhase	Enter the next phase
Help	Get some help
AddItem	Add a new item
AddNewRelation	Add a new relation type item
Display	Redisplay the items
RedisplayAll	Redisplay the entire association
Shape	Reshape the private view
ShapeToFit	Fit the display neatly around the items
Scrunch	Shrink the display window and ShapeToFit
Spread	Expand the display and ShapeToFit
Ambiguities	*Highlight incompletely ordered items*
NoAmbiguities	*Stop highlighting incompletely ordered items*
Neaten	*Arrange items into pseudo-outline form*
Scramble	*Arrange items randomly*
Group	*Collect chosen items into a group item*
Ungroup	*Replace a group by its enclosed items*
Item Operation	**Description**
Edit	Edit this item's attached text
EditLabel	Edit this item's label
Convert	*Convert the type of this item*
ConvertToGroup	*Convert this item to a group item*
ConvertToRelation	*Convert this item to a relation type*
ConvertToItem	*Convert this item to a regular item*
Copy	*Copy this item*
Move	Move this item
Show	Show the text attached to this item
Open	*Open up a group item*
Link	*Link this item to the next chosen item*
UnLink	*Unlink this item from the next chosen item*

Table 4.2. *Ordering Phase Operations.* Tool operations affect the tool in a global sense. Item operations affect the item selected. Indentation in the table indicates submenu items. Entries in *italics* are new to the ordering phase.

Evaluation phase. In the evaluation phase the final form of the presentation is determined. In this phase the participants prepare the complete organization of the paper or talk. Participants should review the overall structure, reorganizing the ideas as needed, filling in missing details, and putting aside peripheral and irrelevant ideas. Critical analysis, deletion, and outline generation are best considered after brainstorming and ordering are mainly complete.

There are several reasons to delay deletion until this phase. One reason is for the liberating effect it has on idea generation. Criticism or deletion in the brainstorming phase tends to inhibit participation, since most people don't like to be criticized and will feel that they must generate arguments to defend their ideas henceforth. Another, related, reason is that arguing against (or for) ideas too soon will slow the generation process down.

The evaluation phase is also a good time to consider re–organizations of various kinds because there is a tangible basis for discussion. For instance, an argument than an idea is in the wrong place is more compelling when other places for it are visible. An argument to delete an idea because it is irrelevant is much more compelling when it is obviously not linked to the rest of the presentation. A claim that an idea is too trivial is more convincing when competing ideas are right there displaying their virtues. A complaint that there are too many ideas is more convincing when all the ideas can be displayed.

Most of the ordering operations are based on local information. The evaluation phase, with the ideas essentially ordered, is a good time to consider the more global elements of the presentation: Does it cover the right amount of material? Have key terms been defined? Is it too long? Is a glossary or appendix needed?

Cognoter provides a facility for systematically generating an outline (see Figure 4.7). Outline generation is delayed until this phase since it is not useful until the item ordering is largely complete. Items with no incoming links are potential starting points for the presentation. Cognoter can assist in

the ordering process by focusing attention on ambiguously ordered (or unordered) items (see Figure 4.8). The presentation graph can be displayed in outline format, with or without the attached text, by successively displaying and removing *beginning* items, items with no in–links. Items with no outgoing links are potential endpoints for the presentation. Items with no links at all are probably irrelevant to the presentation (though they may have served as a taking off point for other ideas in the graph).

[phasing]

Each phase emphasizes different kinds of activity.
One goal is to experiment with various structures for group problem solving.

***related work**

DeKoven's collect-connect-correct. Hayes and Flower had an analogous three phases. Polya and Platt.

***[brainstorming]**

Get as many ideas on the table as possible

theory

Generation only. No deletion or criticism. Too many ideas better than too few.

practice

Users act simultaneously. One window at first. Supporting text can be added to items.

Figure 4.7. *A portion of a Cognoter Outline.* When desired, Cognoter will display an outline. The outline can be displayed for the whole presentation graph, without or without attached text, or for any subgraph. Items arbitrarily ordered by the outlining algorithm are starred.

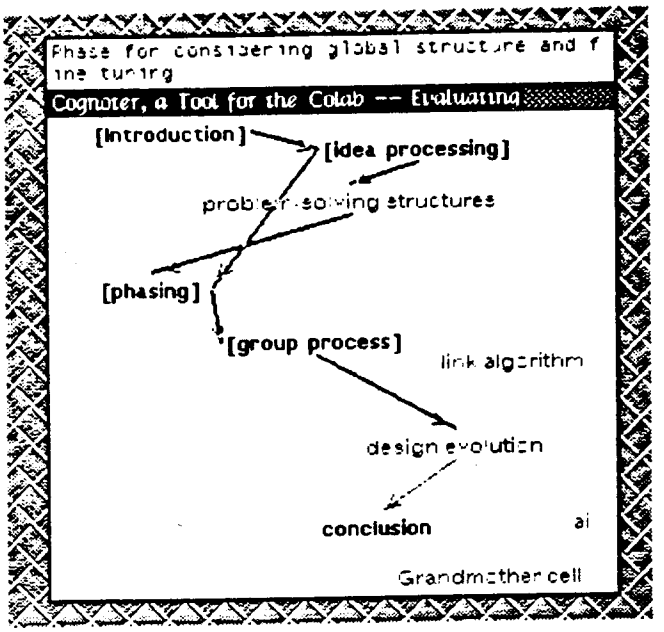


Figure 4.8. *Evaluation of an idea graph.* In the Evaluation phase the graph is scrutinized for overall structure and unlinked items are considered. *Example*: Above, the item "link algorithm" has no links and needs to be inserted into the ordering or declared a side issue. "Grandmother cell" is probably irrelevant and a candidate for deletion (though it may have served some purpose in the session by stimulating other ideas).

Evaluation Phase Operations

Operation	Description
Help	Get some help
SeeDeletes	*Display previously deleted items*
AddItem	Add a new item
Display	Redisplay the items
RedisplayAll	Redisplay the entire association
Shape	Reshape the private view
ShapeToFit	Fit the display neatly around the items
Scrunch	Shrink the display window and ShapeToFit
Spread	Expand the display and ShapeToFit
Ambiguities	Highlight incompletely ordered items
NoAmbiguities	Stop highlighting incompletely ordered items
Group	Collect chosen items into a group item
Ungroup	Replace a group by its enclosed items
Outline	*Display the outline so far*
TextToo	*Display the outline with all attached text*
Item Operation	**Description**
Edit	Edit this item's attached text
EditLabel	Edit this item's label
Convert	Convert the type of this item
ConvertToGroup	Convert this item to a group item
ConvertToItem	Convert this item to a regular item
Copy	Copy this item
Move	Move this item
Show	Show the text attached to this item
Open	Open up a group item
Link	Link this item to the next chosen item
Unlink	Unlink this item from the next chosen item
Delete	*Delete this item*

Table 4.3. *Evaluation Phase Operations*. Tool operations affect the tool in a global sense. Item operations affect the item selected. Indentation in the table indicates sub–menu items. Italics show entries new to the evaluation phase. The relation and neaten operations are not shown above to conserve space.

Cognoter as a Multi–user Interface

To make the shared database simultaneously accessible to all the members of a group, Cognoter provides a multi–user interface.

WYSIWIS Interfaces. Recall that strict WYSIWIS (What You See Is What I See) demands that all screen images are exactly the same: all views are sized and placed identically and the images of all cursors are visible. The WYSIWIS ideal for multi–user interfaces must be addressed in a system, like Cognoter, that supports a multi–user interface.

The general issue of WYSIWIS and the need for relaxations in the multi–user interfaces is treated in Stefik et al [Stefik86]. The current implementation of Cognoter addresses WYSIWIS in simple ways. For instance, Cognoter relaxes strict WYSIWIS because it provides both private and public display space. The Cognoter windows, those windows where the links and items are displayed, are public, but the outline display and item editing windows are private. Visual cues indicate whether a Cognoter window is public or private.

Even in a multi–user interface, it is important that users have a high degree of control of their displays. Cognoter provides private placement of public windows. This freedom of screen use comes at a WYSIWIS cost: users will not necessarily have the same views of the shared models. Participants can not refer to screen objects by absolute position.

Busy signals and Social Conventions. When more than one user is able to interact with shared objects conflicts can occur. This is a key problem in the overall design of Colab, but largely avoided in Cognoter through the use of *busy signals*. Cognoter helps participants avoid conflict by signaling potential conflict (see Figure 4.9). Busy items are greyed–out in all views when being edited or moved or grouped. These busy signals do not make conflict impossible, but makes them avoidable, by relying on the participants to notice that an item is being changed.

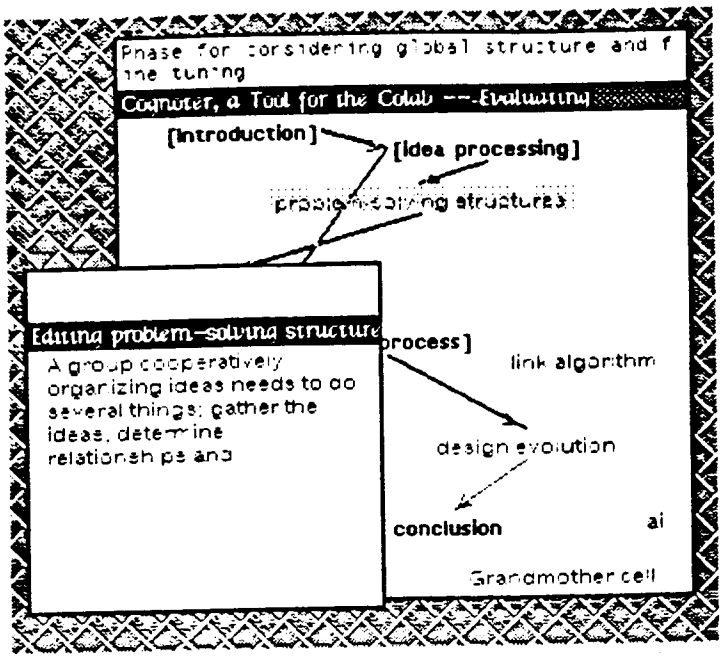


Figure 4.9. *A busy item.* Conflicts can occur when more than one user is able to interact with shared objects. Cognoter warns users that someone else is doing something to a item by highlighting the item. These busy signals put some of the burden of conflict detection onto the users in a fairly natural way. ***Example***: One of the collaborators is in the process of editing the item "problem–solving structures". The greying–out of the item appears on all displays warning the other participants that the item is busy.

In a face–to–face meeting social conventions come into play. While using Cognoter people can verbally gain exclusive access to a shared object, "I'm going to knock the introduction into shape", or suggest non–interfering subtasks: "Why don't you work on the conclusion." Cognoter is intended to support these kinds of behaviors (indeed, it depends on them in the current implementation of conflict avoidance).

Another convention is semi-reserving the left side of the displays for private activity. This partially avoids the problem of remote competition for screen space.

Cognoter and Meeting Processes

As a meeting tool, Cognoter inevitably reflects a philosophy and model of meeting processes. By making some things explicit and ignoring others, meeting processes are inadvertently (or deliberately) biased. For example, Cognoter users must take the phases into account: they can either follow the urged path or consciously react against it. On the other hand, Cognoter (for better or worse) is not involved in policing the technical level of the presentation — this must be worked out by the participants.

Parallelism and Equal Access. Cognoter users at personal workstations have the potential to simultaneously handle different parts of a task. For example, during the brainstorming phase, participants often add items simultaneously to the shared database (and all displays). In the ordering phase, participants frequently partition items into sets order the sets in parallel. In all phases, it is usual for participants to add attached text to different items simultaneously.

In Cognoter sessions a characteristic pattern of activity occurs, especially in the ordering phase. Users interact verbally for a few minutes, discussing things and making short plans of action. This is followed by a period of intense individual interaction with the system. Gradually, over the course of minutes, the group tends to lose track of what the others were doing and the session returns to verbal interchange for summarization and focusing (see chapter six).

Incremental actions. The ability of a tool to support incremental progress is very important. It is key to the rapid and synergetic interactions. The parallel actions that we see in Cognoter are not at the grainsize of hours — they are the interactions that make up the give and take of participants in a rapid problem-solving context. Interactions range from a few seconds to a

few minutes, and the shorter ones must happen quickly or they will slow down the meeting. Many small contributions and local decisions about idea ordering taken in sum constrain large scale organizational decisions. Large scale organizational decisions may turn out to be sufficiently constrained that they simply do not have to be made at all.

Consensus. Cognoter serves as a focus of attention and, since it supports only a single version of the idea organization it, perforce, maintains consensus. This will not be the correct approach for all applications (in fact, some other research suggests otherwise [Kerr82] [Rice84]). Other applications may wish to delay consensus. For instance, *Argnoter*[3] [Stefik87], a Colab tool under development for considering competing proposals, seeks to delay consensus to highlight the differences between several competing proposals.

The "Rules of the Game". People who agree to cooperatively solve a problem are likely to implicitly agree to the "rules of the game" — especially if they think that playing the game will help them work more effectively. Cognoter establishes a working framework both in the software and in the implicit or explicit rules of the game. In effect, it both carries and presupposes certain attitudes about the way that meetings are done. When tools like Cognoter become widely used, they may have an important effect on large organizations as carriers of problem–solving "culture".

Limitations of Cognoter. Some important parts of the group problem–solving process are not captured in Cognoter. For example, Cognoter has no representation of the goal for the presentation other than a title. On the flexibility side this is good, but it also allows the group to wander off the point. Cognoter does not handle a specification of the audience for the presentation. When using Cognoter participants get little help at keeping the technical or verbal level of the presentation at the appropriate level. The current version of the tool does not provide the ability to attach supporting arguments to links or deletions. Experimentation with these is left to future work (see chapter seven).

Design Evolution

Cognoter began as a desire to automate a particular reasonable–seeming method of organizing material for presentation. Although the basic idea for phasing existed early on [Stefik84], the first implementation had no phases, only the ability to put idea items into a shared window, attach text to them with local editors, publicly display items and their text, and move the items around. It was expected that any phasing would be handled verbally by the participants.

It quickly became apparent that a more flexible and definite ordering technique was needed (rudimentary organization in the initial implementation was achieved by moving items roughly into a left to right, top to bottom order). Links were added as a precedence indicator.

Once links were added, the tool became marginally useful. With greater utility came increased use and increased demands for improvement. Useful work could be done and it became necessary to be able to generate more useful output and hardcopy than a simple snapshot of the screen. Outline format display and hardcopy features were introduced.

More sophisticated ordering techniques were needed for the generation of real outlines. Complex items (groups items) were added. At this point the crude tool was finished and was actually more useful than a piece of paper and a pencil. Nothing (!?) remained to do except clean up the details, straighten out the user interface, speed things up, and fix the things that were being done the wrong way.

For Cognoter to be used regularly it was necessary to enhance item movement (allowing movement across windows and, therefore, across groups), the ability to save and restore the state of a session, and features to guide and gauge completeness (ambiguity highlighting and notations indicating that attached text was present). The menu structures were rationalized several times and operations were introduced to allow more control of display complexity: *Spread*, *Scrunch*, and *ShapeToFit* are examples.

Summary

Cognoter is a meeting tool that supports a group of people who are organizing ideas for presentation. It is the first computer-based meeting tool to be regularly used in the Colab. The implementation and design of Cognoter has provided a better understanding the design of multi-user tools and computer-supported meeting processes.

A key motivation for computer-supported meetings is the possibility of parallel activity by participants. To support this, Cognoter provides a multi-user interface which gives all participants equal and immediate access to the shared database of the meeting. Cognoter's interface is based on the WYSIWIS abstraction, which ideally enables all users to see the same written information and where other participants are pointing.

Computers, as an active medium, allows the capture of some aspects of meeting processes. Meeting tools, such as Cognoter, make it possible to experiment with meeting processes in principled ways. Cognoter has three phases that guide a presentation–planning meeting from the generation and articulation of ideas through an annotated outline of the presentation. At each phase of the meeting, progress towards the goal is achieved through small, incremental actions that ultimately lead to a complete ordering of the ideas to be presented.

The first generation of Cognoter is now finished. The next chapter discusses implementation details of Cognoter and Colab. Informal observations on Cognoter's early use and the results of some experiments to explore Cognoter's multi–user interface and meeting processes are presented in chapter six. Extensions to Cognoter and speculation on future directions of this work appear in chapter seven. Appendix C is the Cognoter Users' Guide.

Notes:

1 The name *Cognoter* comes from a combination of *Cogno–ter*, "thinker", *Cog–noter*, "thought noter", and *Co–gnoter*, "knowing together".

2 Interestingly, there turns out to be the same reticence in making a group consisting of a single item in Cognoter as there is in creating an outline sub–section consisting of a single entry (Either the Mrs. Grundys of the world seem to be winning: "You can't have a **II.B.1.a** without also having a **II.B.1.b**", or this is a recognition of a fundamental distinction).

3 The name *Argnoter* is intended to suggest "Argument Noter".

5

So many out of the way things had happened lately that Alice had begun to think that very few things, indeed, were really impossible.
— Lewis Carroll

Implementation and System Objects

Here the object–based implementation, concepts and subsystems of Cognoter and Colab are delineated. The Colab Executive is the high–level interface to the system. Conversations are associations of machines, collaborators, and meeting tools. Cognoter is dissected and its parts described: its windows, items, links, group items, and display algorithms. The C–graph representation used by Cognoter is compared to the usual O–tree representation of outlines. After exploring several different database consistency schemes a combination of "busy signals" and "social locking" techniques was settled upon for the current implementation. This technique cannot guarantee consistency but is surprisingly effective in Cognoter's special domain of cooperative participants in face–to–face communication.

Introduction

Several assumptions determined the design of the Colab and Cognoter software. Foremost were the assumptions that each human participant is at a personal workstation, that the computers of the participants are connected together by a communications network, and that all machines run the same software kernel. Also, as explained later, by limiting the domain of the project to support of face-to-face situations, the software design was affected in some unexpected ways.

Though other design spaces for collaborative systems are certainly possible — for example, a system might be built using a time-sharing system or software might support a heterogeneous computing environment — the chosen approach has proven workable and makes sense, directly reflecting as it does the conceptual model of each node as an intelligent agent: a person augmented by computational support. This networked workstation approach is flexible and open-ended, suiting the experimental goals of the Colab project. With this arrangement it is possible to add special software to processors or to add processors to perform special functions, like statistics gathering.

The software for Colab and Cognoter is written in Loops [Bobrow83], an object oriented extension of Lisp [Sannella83]. Loops is similar to Smalltalk-80 [Goldberg80] in that programs are organized as objects and control is expressed in messages passed between objects. Objects can hold private data and private methods specifying operations to perform or they can inherit default data and methods from parent objects. As implemented, the Colab system runs on Xerox Lisp Machines connected by an Ethernet [Metcalfe76] network.

This chapter describes the implementation of the Colab system and the Cognoter brainstorming tool. Since the Colab system is implemented in an object-oriented language, the main system objects and classes are presented in some detail, along with an exploration of some of the object consistency

issues that arose in the course of the programming effort. For object–oriented programming argot see the glossary (appendix A).

Colab Implementation and Objects

This discussion of the implementation of Cognoter and Colab begins by considering the relationships between the system classes and objects (see Figure 5.1 for a high–level view). Most objects in the Colab system are associates. Recall from chapter three, that when any associate in an association changes, new values are broadcast to all associates using an RPC–based communication protocol (see BroadcastMethods in chapter three).

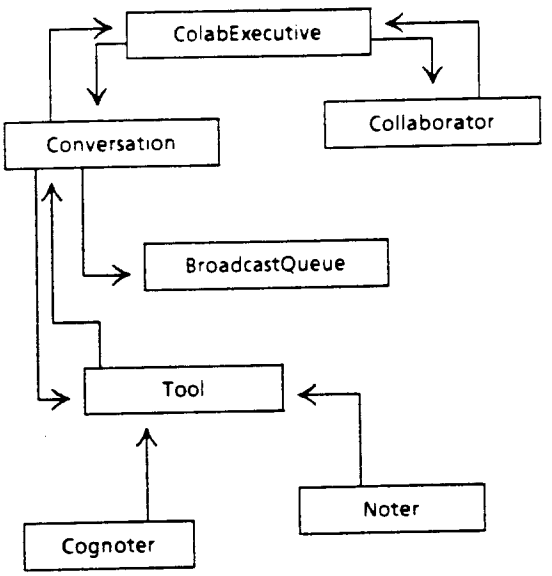


Figure 5.1. *Diagram of illustrative high–level object classes and their inter–connections.* Colab consists of many classes of system objects — this diagram is the basic road map. The links mean "has a pointer to".

The implementation of the system can also be considered operationally. The system is designed as an "operating system" in steps: initially supporting an individual tool, then supporting several tools together in a single session, then the dynamic addition of tools to an existing session, and finally supporting multiple sessions. After the single machine case is delineated, message passing protocols are added and a similar sequence of steps is followed in the multi-machine case. Beyond this are the requirements for the sharing of common subsystems and dynamic system reconfiguration.

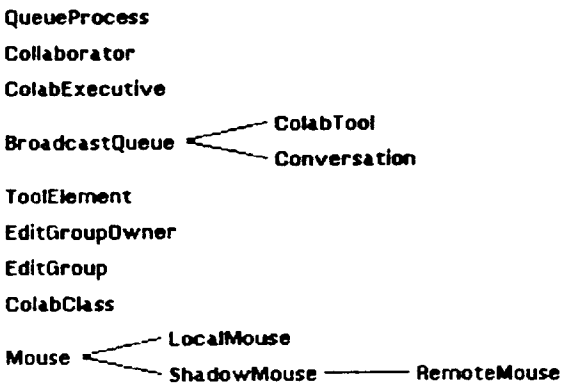


Figure 5.2. *High-Level Object Class Lattice*. This is the Loops class inheritance browser for some of the main system objects.

Colab Executive. The *ColabExec*, an instance of the *Colab Executive* class, is the high-level interface to the Colab environment. There is only one ColabExec per machine. The ColabExec is a special kind of object. It is not quite an association (see chapter three) since the ColabExec in each machine environment keeps its own information and doesn't broadcast internal state, but it is like an association in that all ColabExecs share the same name so they

can communicate easily with each other in an association–like way. The ColabExec serves as the top–level user interface to the system, dealing with things like adding new conversations and displaying currently known participants (see Figure 5.3). It also handles some behind–the–scene chores such as registering new participants and distributing changes to the state of the conversations.

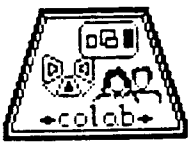


Figure 5.3. *ColabExec Icon.* The icon represents the people, the software (windows), and the room (trapezoidal tables). Buttoning the icon brings up a menu of global operations.

As shown in Figure 5.4, the ColabExec object contains the known session collaborators (any of which may or may not be involved in individual conversations), the active conversations, icon information (position, bitmap), and menu interface.

ColabExecutive

MetaClass: *Class*

Supers: *IconWindow*

InstanceVariables: (*me* «Collaborator»)

(*conversations* «ListOfConversations»)

(*collaborators* «ListOfCollaborators»)

(*LeftButtonItems* «ListOfMenuItems»)

(*icon* «Bitmap»)

Methods: (... *AddConversation* ... *ChangeCollaborators* ... *RestartRPC* ...)

Figure 5.4. *A condensed version of the Colab Executive class description.* The ColabExec, the only instance of the ColabExecutive class on each machine, contains the collaborators known in the session, the active conversations, and information about the icon interface and functionality.

Conversations. Technically, a *conversation* is defined as an association of machines, collaborators, and tools. A conversation object (instance of the Conversation class) holds a description of the participants involved in the interaction and the tools that they are using (see Figure 5.5). In looser Colab parlance, a conversation is a set of machines, Colab tools, and participants working together to solve a problem in a meeting. People speak of joining, leaving, saving, or restarting a conversation. There are several ways this might be implemented, in Colab the Conversation class is a sub–class of BroadcastQueue and therefore can run processes on each workstation that field broadcast messages or package local messages for broadcast.

Conversation

MetaClass: *Class*

Supers: *BroadcastQueue*

InstanceVariables: (*collaborators* «ListOfCollaboratorObjects»)
(*toolsInUse* «ListOfTools»)
(*title* «String»)
(*audience* «ListOfVoyeurs»)

Methods: (... *AddCollaborator* ... *AddTool* ... *NotifyColabExec* ... *PutSnapshot* ...)

Figure 5.5. *A condensed version of the Conversation class description.* A conversation holds a description of collaborators that it will broadcast to and the tools that are active.

New participants can be added to a conversation in progress through the ColabExec. All the other participants will find out about the newcomer just as the newcomer finds out about existing participants. The newcomer's machine installs copies of the current state of objects that represent the database and, if desired, a history of the conversation. Current implementation allows participants to be brought up to date by either playing back the conversation history from the beginning at high speed or by loading a saved state (history would start with the GetSnapshot command, see below, that restored the saved state). Conversations may also have attached to them *voyeurs*, unfortunately named objects that can eavesdrop on the network or conversation activity — examples include debugging aids and data collection monitors (it is possible ensure a degree of privacy by turning off the voyeurs).

Collaborators. A *Collaborator* object is a description of a participant in a Colab session (see Figure 5.6). It encapsulates network information and user

information. A Collaborator object is created for each participant — "participant" in this case refers to a human being and a workstation together.

Collaborator

MetaClass: *Class*

Supers: *Object*

InstanceVariables: (*machine* «NetAddress»)
(*machineName* «String»)
(*userName* «Atom»)
(*realName* «String»)
(*remoteMouse* «MouseObject»)

Methods: (... *Describe* ... *MakePrettyDescription* ...)

Figure 5.6. *A condensed version of the Collaborator class description.* A Collaborator object holds a description of the human–machine pairs in a Colab session.

Session State. Session or tool state can be saved and restored using the standard methods *PutSnapshot* and *GetSnapshot*, respectively. These methods will usually need to be specialized for each Colab tool since the default versions inherited from ColabTool save everything about every object. For tools of any complexity saving static data will be wasteful of space and inflexible. If the contents of every instance variable were saved in Cognoter, for example, then Interlisp window or font descriptors would be written out. But these descriptors may need to be rebuilt when installed into a new environment. Along the same line, it is often better to recreate particular object instantiations. The usual approach is for each tool to have a

PutSnapshot method that stores enough structured information for its GetSnapshot method to rebuild the tool state.

Cognoter Implementation and Objects

The theory and practice of Cognoter was discussed in the previous chapter. This section looks at the implementation details of Cognoter. Any Colab meeting tool is implemented as a collection of objects. Like all Colab tools, an instance of the Cognoter object has interface aspects and tool functions. Figure 5.7 shows the classes for the main Cognoter objects. The theory and practice of Cognoter were discussed in the previous chapter. The main parts of Cognoter are illustrated in Figure 5.8.

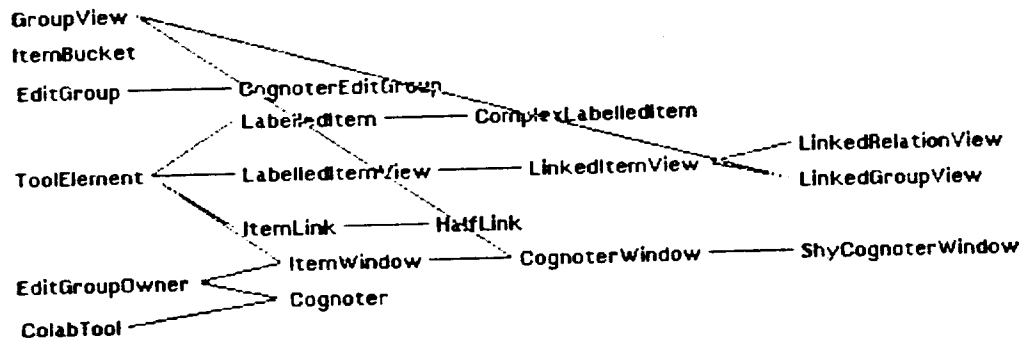


Figure 5.7. *Cognoter object class browser.*

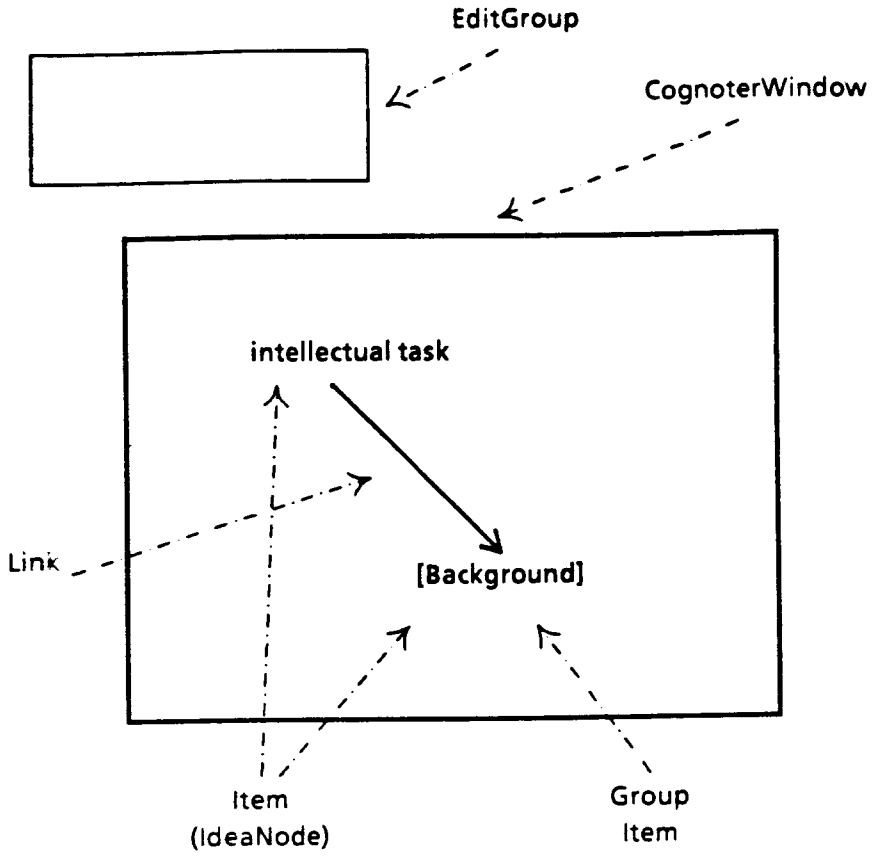


Figure 5.8. *The Basic Parts of Cognoter's Interface.* A CognoterWindow is where the items are manipulated. An EditGroup is a subtool supplying editing and text display functions. An Item (IdeaNode) is the basic unit. A Group item encloses a group of items. A Link specifies the order of two items.

Cognoter. An instance of the *Cognoter* class is really an interface between its various subtools and the rest of the Colab system (see Figure 5.9). A Cognoter object is the only part of the tool that communicates with the enclosing conversation. Requests to broadcast generated by a part of Cognoter is forwarded to the Cognoter object which in turn forwards them to

the enclosing conversation object. A Cognoter object contains pointers to all the large scale parts: the windows and edit groups. It also takes care of global tool state, such as session phase.

Cognoter

MetaClass: *Class*

Supers: (*ColabTool EditGroupOwner*)

ClassVariables: (*ObjectTypes* «TypePropList»)
(*MenuName* «String»)
(*Phases* (Brainstorm Order Evaluate))

InstanceVariables: (*mainWindow* «CognoterWindow»)
(*itemWindows* «ListOfCognoterWindows»)
(*outlineWindow* «Window»)
(*garbageWindow* «ShyCognoterWindow»)
(*phase* «phaseName»)
(**editGroup* «EditGroup»)
(**conversation* «Conversation»)
(**initiator* «Collaborator»)
(**objectsForSnapshot* «ListOfObjects»)

Methods: (... *DisplayOutline* ... *GetSnapshot* ... *NextPhase* ... *Quit* ...)

Figure 5.9. *A condensed version of the Cognoter class description.* Contains the parts of the tool: the windows, the edit groups,... (* means inherited).

CognoterWindow. An instance of *CognoterWindow* (also called a CognoterWindow when the difference between class and instance is unambiguous) supports most of the interactive tool functionality (see Figure

5.10). It supplies the command menus and invokes most changes that happen in the course of tool use. In addition to tool menus, a CognoterWindow contains a list of items within it and their links.

CognoterWindow

MetaClass: *Class*

Supers: (*ItemWindow GroupView*)

InstanceVariables: (**items* «ItemList»)
(**tool* «CognoterInstance»)
(**title* «string»)
(*font* «FontInformation»)
(*spawningItem* «Item»)
(*ambiguitiesFlg* NIL)
(*rightButtonItems* «MenuItemList»)
(*titleItems* «MenuItemList»)
(*middleButtonItems* «MenuItemList»)
(*leftButtonItems* «MenuItemList»)

Methods: (... *AddLink* ... *EditAttachedText* ... *GroupItems* ... *ShapeToFit* ...)

Figure 5.10. *A condensed version of the CognoterWindow class description.* Menus, Items,... (* means inherited).

Items and Item Views. A complete specification of an item requires two instances: one for its model and one for its view (technically *item* refers to an item view and *theItem* refers to an item model). The underlying model of a Cognoter item is an instance of *ComplexLabelledItem*. It contains the basic (display invariant) information about an item: the label, any attached text,

and a list of its views. An item view is an instances of *LinkedItemView*. It contains, in addition to a pointer to back to its model, display information: which font to use, the region the item should use to display, window coordinates, and any links to or from the item (see Figure 5.11).

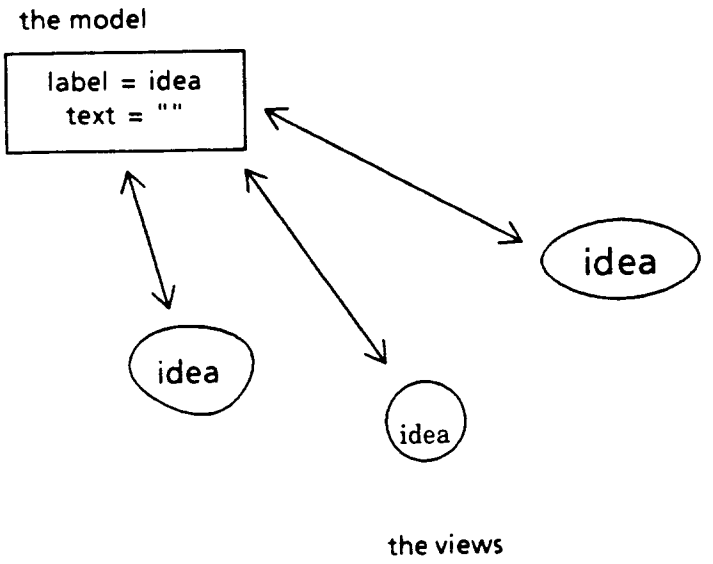


Figure 5.11. *Items: the model and the views.* ComplexLabelledItems and LinkedItemViews are each specialization of more primitive classes (LinkedItemView, for example, is a specialization of *LabelledItemView* which is a specialization of *LabelledActiveRegion* which was begat by *ActiveRegion* which was begat by *Region* which was begat by ...). Instances of LabelledItem contain the basic information held by an item: label, attached text, and its views. Instances of LinkedItemView, in addition to a pointer to the LabelledItem it is displaying, contain display information: label font, display region, and links.

An example may clarify: Suppose supporting text was to be attached to an item. The item view would be buttoned. "EditText" would be chosen from the menu (supplied by the enclosing CognoterWindow). The

CognoterWindow would send an EditText message to the item model of the item view (e.g., (send (GetIV item theItem) EditText)) which brings up an editor for the text. Since items with attached text appear in bold face, the item model sends its views a Display message.

GroupViews. GroupView is a mixin class for LinkedGroupView and CognoterWindow, adding a list of contained item views. Instances of LinkedGroupView (group items) provide the hierarchical structure of Cognoter. They display just as any other item does except for the added brackets that indicate their complex nature.

Items that have been grouped are displayed only on request. The group item can be opened (a new CognoterWindow is created) to display the contained items (see Figure 5.12) and make them available for interaction.

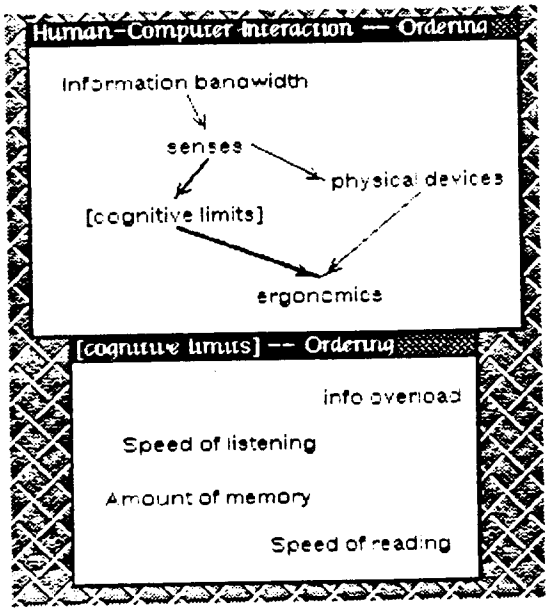


Figure 5.12. *A group item and its contents.* [cognitive limits] is an opened group item.

ItemLinks. An instance of ItemLink, a link, established an ordering between two item views (see Figure 5.13). A link contains information about the item it is coming from and the item it is going to. It knows how to draw itself by looking at the window coordinates of the items it connects. The link arrowheads are backed off the end of the link by 10% to avoid overwriting existing arrowheads — so items would have to be in exactly the same place for their links to completely overwrite each other. As implemented, links contain both model and view data and functions — this doesn't matter in the current implementation but will be changed in the future.

ItemLink

MetaClass: *Class*

Supers: *ToolElement*

InstanceVariables: (*fromItem* «Item»)

(*toItem* «Item»)

(*displayWindow* «Window»)

Methods: (... *Arrowhead* ... *Draw* ... *Quit* ... *To* ...)

Figure 5.13. *A condensed version of the ItemLink Class.* Links know where they are coming from and where they are going to. They use the window coordinates of the items they link to calculate the coordinates for display.

Grouping Link Algorithm. When a group item is formed, strictly contained items retain any links with each other, but any links across CognoterWindows are broken. If there were any inlinks from items outside

the group to any of the contained items then an inlink to the new group item is formed from each outside item. If there were any outlinks from contained items to items outside the group then a new group item link is formed to each of them (this is described more clearly in Figure 5.14).

Groups are *ungrouped* analogously. A shrunken version of the contents of the group item is put in place of the group item. Links from the outside to the newly ungrouped items are determined by inspecting the decommissioned group item. Any inlinks to the group item become inlinks to the *beginning set* (defined below) of the contained items, and similarly for outlinks and the *ending set* of the contents.

As implemented, grouping followed by ungrouping is not invariant. The simplest example of this is the following: Imagine four items **A**, **J**, **K**, and **Z**. **A** is linked to both **J** and **K**, **J** is linked to **K**, **J** and **K** are both linked to **Z**. If **J** and **K** are grouped into **[G]**, then links from **J** and **K** to **A** and **Z** are broken and new links are formed from **A** to **[G]** and from **[G]** to **Z**. If **[G]** is now ungrouped, new links will only be formed from **A** to **J** and from **K** to **Z** (the link between **J** and **K** is unchanged).

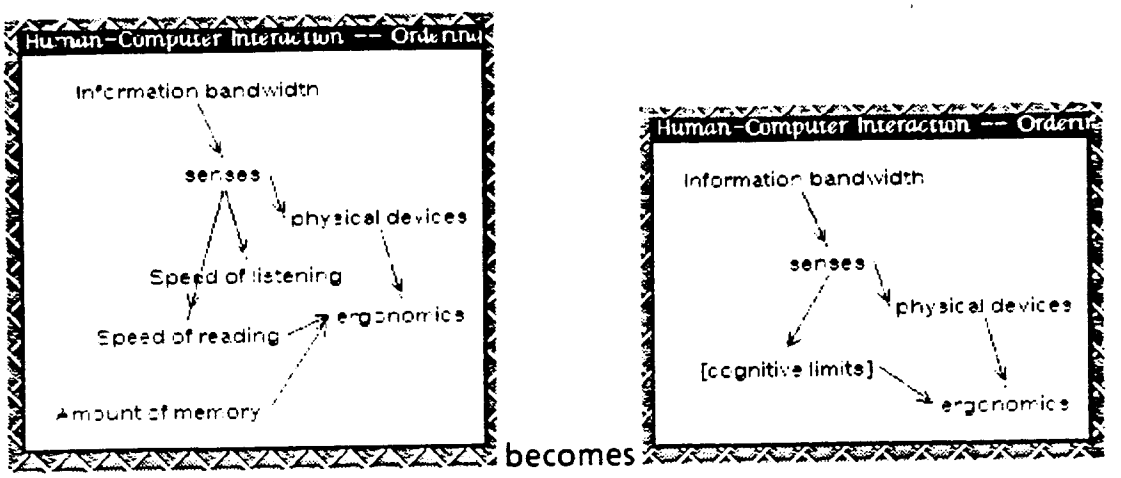


Figure 5.14. *Links in Grouping*. Links between contained items are kept intact. In–links to contained items from outside items become in–links to the group item. Out–links from contained items to outside items become out–links from the group item.

Outline Algorithm. An outline is generated, with or without the attached text, by successively removing and displaying items in the *beginning set* (items with no in–links). In a completely specified graph with no ambiguities there will be only one beginning item at each stage. If an item is a group item then that item is traversed in the same way, but is displayed with an indentation in the outline window. Figure 5.15 shows the algorithm used for Cognoter's outline display.

```
DisplayOutline (cognoterWindow, title, textToo):
  ;; get the items and order them by taking successive "beginnings"
  ;; OrderItems behaves much like the Ambiguities Algorithm below
  items ← OrderItems(GetItems(cognoterWindow)

  ;; GetOutlineWindow creates a new outlineWindow if necessary
  outlineWindow ← GetOutlineWindow()

  ;; print the heading
  Print(outlineWindow, title)

  ;; and the items...
  ;; PrintOut will plumb any group items and PrintOut with
  ;; appropriate indentation (using PrintOutAux)
  for i in items do
     PrintOut(i, outlineWindow, globalItemFont, textToo)
```

Figure 5.15 *Simplified version of the algorithm used for Cognoter's outline display.*

Since the Outline command is handled by a CognoterWindow, outlines are made in the context of invocation. This means that outlines can be made of any subgraph in the hierarchy of CognoterWindows.

O–trees and C–graphs. An outline can be represented as an *O–tree*, a tree whose nodes are decorated with attached text. An outline is actually a forest made into a tree by adding a false root node to connect all roots of trees that make up the forest. The outline is traced by traversing the O–tree in preorder. The depth of the O–tree corresponds to the depth of the outline (see Figure 5.16).

Title

I. Heading
attached text

II. Heading

A. Subheading

1. Point

2. Point
attached text

B. Subheading

C. Subheading

III. Heading

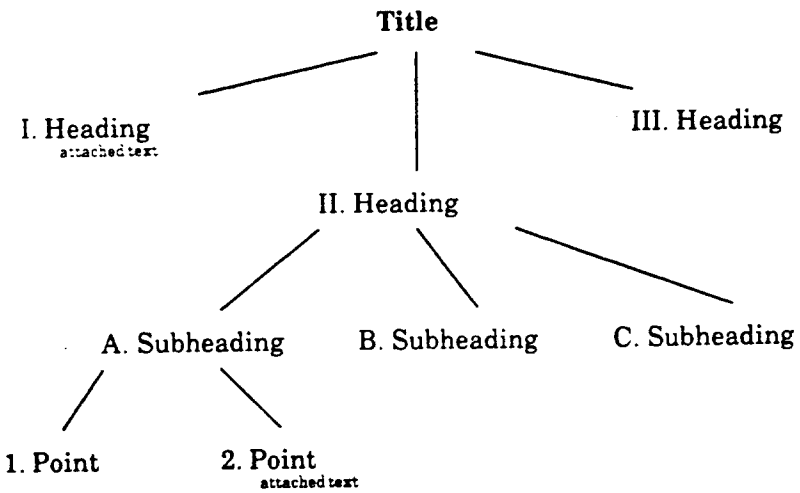


Figure 5.16. *An outline and the corresponding O-tree.* An O-tree is a decorated tree. An outline is generated from an O-tree by traversing it in pre-order.

At the center of Cognoter is a way of organizing ideas called a *Cognoter graph* or *C-graph* (C-graph is used to refer to the abstraction and

Cognoter graph is used for the view of a C-graph seen on a display during a Cognoter session — an Outline is to an O-tree as a Cognoter graph is to a C-graph — see Figure 5.17). A C-graph is similar to an O-tree except that redundant connections between nodes are allowed and there is no need for the additional false root to enforce ordering. An outline can be generated from a C-graph by traversing the graph in pre-order, making sure to visit each node only once. A Cognoter graph consists of two basic node types: item-nodes and group-nodes. The other major elements in a Cognoter graph are the links between the nodes, text attached to either kind of node, and the containing item windows (a view of the items in a group).

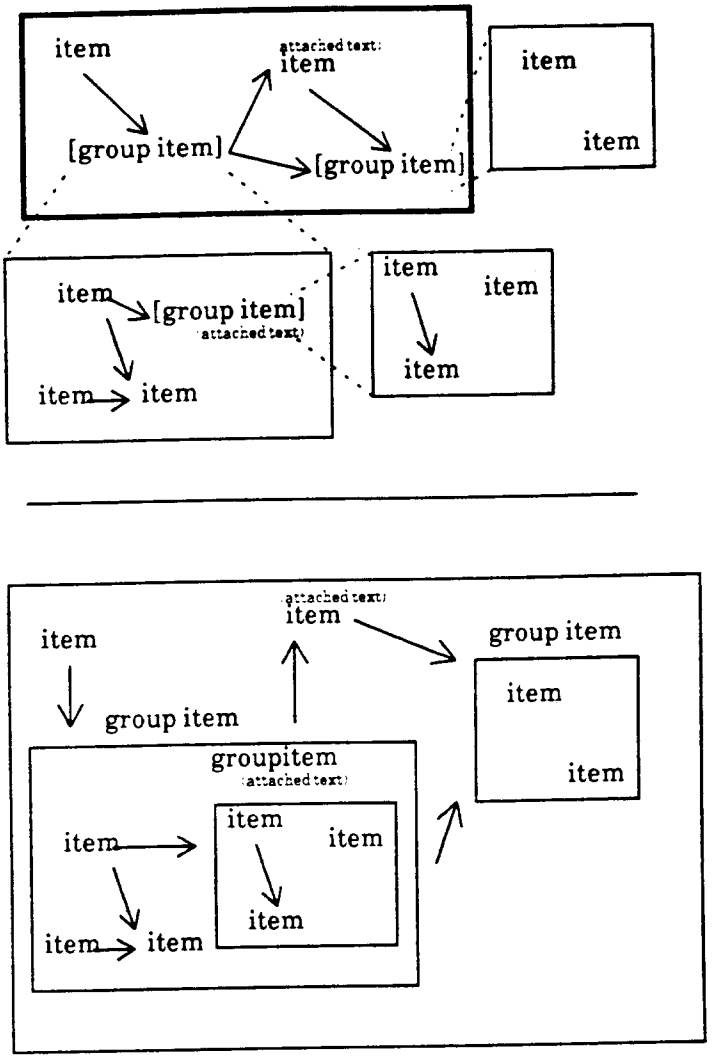


Figure 5.17. *A C-graph and the corresponding Cognoter graph.* A C-graph is a decorated directed graph. An outline can be generated from a C-graph by traversing it in pre-order, but only visiting each node once.

People are generally familiar with the basic use of outlines and outlining. It can be argued that an outline is a good design form to use when

the goal is to generate an outline, since an outline exists at all phases in the design. However, the "always have an outline" constraint is also a weakness. The outline format forces decisions too quickly: you must choose a place in the outline to put an idea at the moment the idea is created. An outline is also difficult to rearrange. Even when, in an attempt to overcome the need to decide order at creation time, ideas are placed in random order with the plan of scrutinizing and ordering later, the outline format still *implies* an ordering (possibly false) at all times. It is not clear which ideas have been ordered and which only appear to be in order.

A Cognoter graph, while having much to offer, is not a traditional structure for ideas and it requires some effort to learn to think and work in C–graph terms. This newness itself, if not overly difficult, can be an advantage — providing a fresh way to look at things. A Cognoter graph permits the delay of outline decisions since order need not be specified at node creation time. Since items can be simply generated without concern for where they fit into the existing nodes, a Cognoter graph lends itself well to incremental decision–making. Items that are not ordered or are only partially ordered are obvious in a Cognoter graph. A C–graph is able to hold more information than an O–tree: this additional information is the allowed redundant links that specify relationships between items (see Figure 5.18). Information useless at one moment, may become useful later. For instance, when a link between two items is broken, formerly redundant links may become germane.

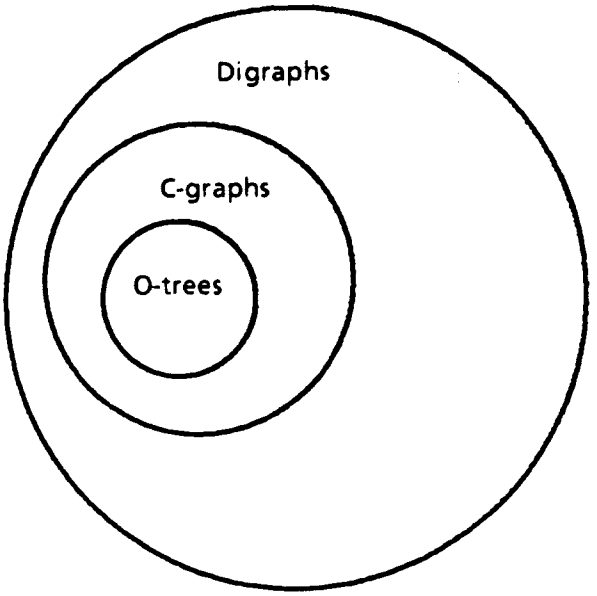


Figure 5.18. *Cognoter graphs contain more information than outlines.* Any outline can be completely represented as a Cognoter graph, but not every Cognoter graph can be represented completely when cast as an outline.

Other, more subtle, information is captured in a Cognoter graph (though not in the abstract C–graph): e.g., the real distance between items on the display. Flexible positioning and re–positioning of items allows a continuum of subtle relationships. Items physically closer together are implicitly more closely related that items farther apart.

Ambiguity Algorithm. During the course of a session, the items in a CognoterWindow will often be incompletely ordered. Items whose order in incompletely specified can be highlighted on request. The topological sort algorithm used for ambiguity highlighting and for outline generation is shown in Figure 5.19 (Figure 5.20 shows an example).

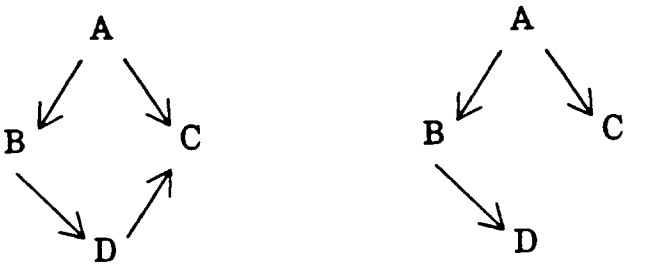
```
Ambiguities (items, ignoredLinks):
  ;; add each item with no inlinks to the beginning list
  beginnings ← NIL
  for i in items do
    if (NoInLinks?(i)) then
      beginnings ← Union(beginnings, i)

  ;; no beginning list ⇒ there's a cycle —punt
  if (Length(beginnings) < 1) then
    for i in items do
      FlagAsAmbiguous(i)
    beginnings ← items

  ;; more than one beginning ⇒ they are ambiguous
  if (Length(beginnings) > 1) then
    for b in beginnings do
      FlagAsAmbiguous(b)

  ;; remove beginnings from item list
  itemsLeft ← NIL
  itemsLeft ← SetDifference(items, beginnings)
  ;; if any items are left, then recurse
  if itemsLeft then
    ;; in any case, ignore any links attached to beginnings
    ignoredLinks ←
      Union(ignoredLinks, CollectLinks(beginnings))
    Ambiguities(itemsLeft, ignoredLinks)
```

Figure 5.19. *Ambiguity Algorithm (simplified version).*



unambiguous ambiguous

Figure 5.20. *Ambiguity Algorithm Example.* If D→C is missing then the relationship of B to C and D to C is unknown.

Linearization Algorithm. Linearization of a Cognoter graph makes it look neater (see Figure 5.21). Linearization rearranges the display in a CognoterWindow by placing the longest path down the left of the window. Other shorter paths grow off this backbone to the right and down. The graph can be displayed with the display of redundant links suppressed. A linearized graph is easier in many cases to understand and is closer to an outline in form. Unfortunately linearization also destroys any meaning attached to idiosyncratic placement of nodes. The linearization algorithm is similar to the ambiguities algorithm above.

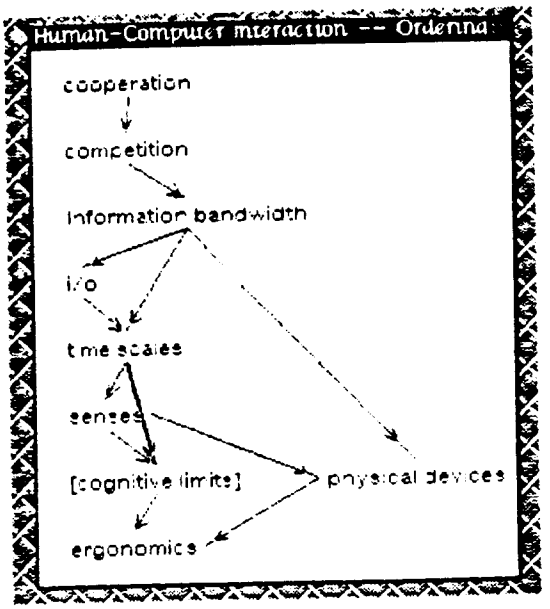


Figure 5.21. *A Linearized Cognoter graph.* A completely graph with the order of items completely specified would appear as a single column. The relative ordering of *[cognitive limits]* and *physical devices* is still ambiguous. Redundant links remain, for instance between *information bandwidth* and *time scales.*

Gone, but not Forgotten. Nodes that are deleted disappear from the display windows but they are not destroyed. They are moved to a closed garbage window. The garbage window is an instance of a specialization of CognoterWindow called a *ShyCognoterWindow* because it will only open when asked very nicely. In other respects, a garbage window is a normal CognoterWindow. The contents of the garbage window may be examined on demand.

Shared Objects and Database Consistency

Cognoter enables people working together to share and jointly revise information. To consider the properties of this shared workspace it is useful to consider it as a a database management system for a concurrent database. The specifics of data structure and grainsize is left unspecified: a Colab datum may be anything from an integer to a bitmap to a piece of executable Lisp code. The design of this database its management system is a starting point for understanding implementation issues and programming techniques that have been used specifically for Cognoter and, more generally, for the Colab.

In addition to the usual database constraints of robustness and correctness Colab shared databases also have some special requirements. Overall, since Colab is a real-time application, real-time response is necessary: changes to the database and display must happen quickly and updates must propagate quickly. Since complex displays of information in the database are being maintained, data access must be very fast. Since several displays will be showing the same data, the database must converge to a consistent state very rapidly. It should not be possible for accidental actions to have catastrophic actions on the shared database. A Colab application should be able to survive the loss of a participating machine (whether the loss is intentional or not — i.e., the system should be able to survive both planned exits and unexpected crashes).

What were the various approaches to database management? Several schemes for conflict avoidance and database consistency were tested experimentally.

Centralized Database Model. The first database scheme to be considered seriously was a single centralized database (see Figure 5.22). This *centralized model* has been used for many other applications (including RTCAL by Sarin and Grief [Sarin83]). Since there is only one database, concurrency control is straightforward. Nearly simultaneous changes to the database are handled by well-known database transaction mechanisms [Bernstein81]. Since all participants necessarily use the same data for display,

screen consistency is assured. However, the centralized model and object servers in general were rejected after feasibility testing because distributed displays could not be serviced fast enough from the remote database. Also, since Colab was to be a real-time system of networked workstations with expected bursts of interaction, the peak communication load between individual workstations and the centralized database was expected to be high enough to introduce unacceptable delays in both display and database updating.

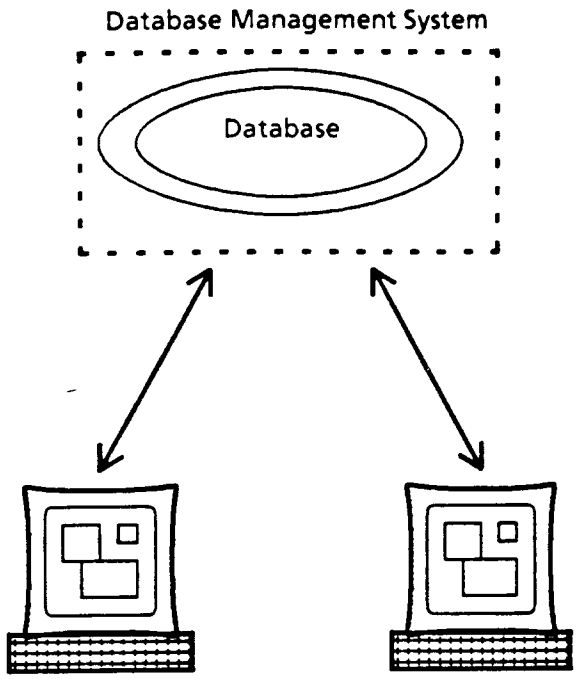


Figure 5.22. *Centralized Database Model.* There is one central database that all participants must modify and use as a source of display data.

Centralized–Lock Model. After considering the basic centralized model it was clear that, to approach real–time performance, the database must be replicated at each workstation. This approach is feasible since only a small database will be generated during the course of a meeting. The next model considered was the *centralized–lock model* (see Figure 5.23). In centralized–lock model each workstation maintains a copy of the database, but can only make changes to the database when it has global ownership of the item (i.e., a lock on it). Ownership of an item is obtained by conversing with a centralized lock server. By locality arguments, this model ameliorates the slow data retrieval problem of the pure centralized model since each workstation will tend to already have the locks it needs to update the database. Data for displays is always retrieved from the local cache, avoiding network communication delays. The cache is updated whenever changes made on any workstation are broadcast. Bernstein and Goodman discuss several variations on this basic approach [Bernstein81].

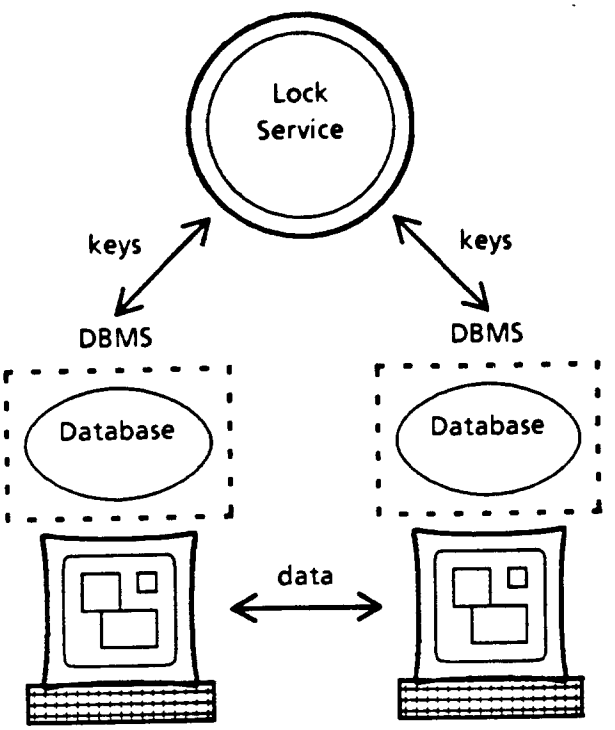


Figure 5.23. *Centralized-Lock Model.* The database is replicated at each workstation. There is a centralized lock server that allows changes to the database only with the appropriate key. Legal changes are broadcast to all workstations.

In the centralized-lock scheme the grainsize of the data is very important. It might be reasonable to provide locks for data ranging from the entire database to individual words of text. If the grainsize is small enough, work on the database can proceed in parallel, but if too small a large number of locks must be obtained to do significant work. If the data grainsize is large, it is easy for once participant to make sweeping changes, but often at the cost of locking others out. In an early implementation, the grainsize was at the level of windows. This was much too large. It essentially forced sequential action. All parallel activity was lost, and the frustration of waiting to get a

lock was added. Unfortunately a smaller, item-level, grainsize would have been too slow.

An improvement to this scheme (unimplemented) would be to add time-stamping. With a little database management machinery to check time-stamps , changes to the database could be serialized. This would ensures that replicated databases would converge to the same state, even if the changes were received out of order. For transactions requiring ownership of multiple locks, the the usual caveats about avoiding deadlock apply [Coffman71] [Hansen73]. One solution is to require transactions needing more than one lock to acquire them all before proceeding.

Roving-Keys. The centralized lock server described above can be a communication bottleneck. The *roving-key model* (see Figure 5.24) reduces network communication load further by locking all items and distributing the key granting responsibility (keys are initially given to the machine beginning the conversation). When a machine obtains a key on a datum it also obtains the ability grant the key to someone else. Thus, when a machine needs a datum it checks to see if it already has the necessary key. If not, it must get the key (and the future granting ability) from whichever other machine currently has it. The postulated main advantage of this model is a locality argument that machines would tend to acquire working sets of keys they need: most key requests would be satisfied by looking in the local machine environment.

An apparent flaw in the roving-keys model, that keys may be held by unknown machines requiring polling the entire network, can be largely avoided by having each machine storing the last known location of each key it touches. Machine A may need a key that it no longer has, but at least it knows that it gave it to machine B. B either has the key and grants it to A or it may not have the key any more and forwards the key request to C. If A had never owned the needed key, it would query the machine that first started the conversation.

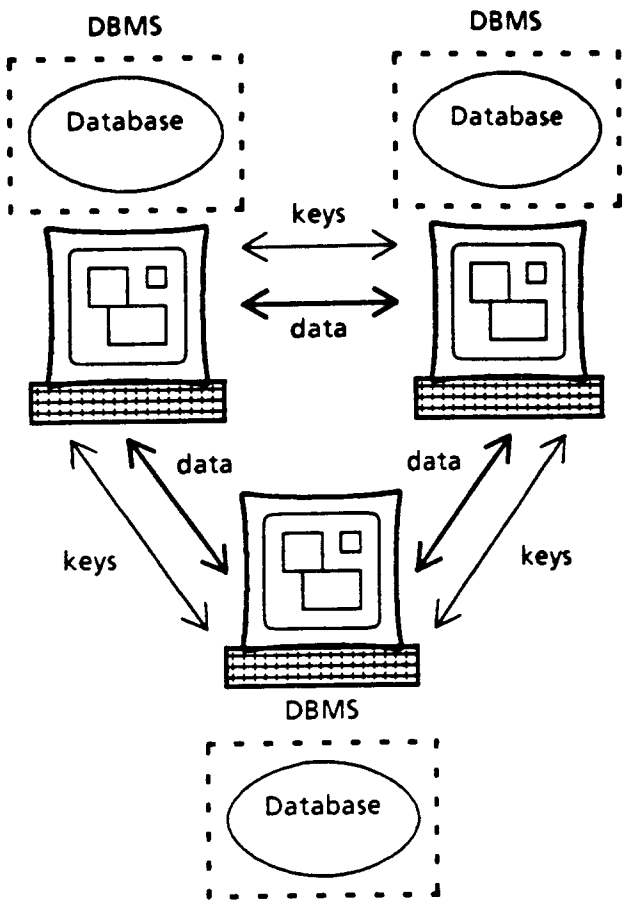


Figure 5.24. *Roving-keys Model.* This scheme dispenses with the central lock server by distributing not only keys to data, but the ability to grant keys. If a machine does not already have a key that it needs, it queries the machine it last knew had the key to for the current location of the key.

The roving-keys and the centralized-lock models were only partially implemented — partly because other issues were more important in the early phases of the project and partly because of a limitation of the Interlisp process scheduler. This process scheduler is non-preemptive. In the Interlisp environment there is no way to limit how long a system process may run

before yielding to other processes. Early tests showed that the system could be brought to a halt by tying up a single processor that had obtained some key locks (!). Future Interlisp releases will have a preemptive scheduler.

Cooperative Model. The need for real-time performance led away from the overhead of data locking. Since the observed time for a round trip remote function call was so long (see table 5.1 and 5.2), techniques and communication models that minimized remote function calls were necessary (eschewing data locks and replicating the database, for instance). This model works surprising well in practice considering its shortcomings. It is essentially the database model used in the Colab at this writing.

Average RoundTrip Times

	RemoteEval			
Machines Client–Server	Simple Call Wait	Simple Call NoWait	Long Call Wait	Long Call NoWait
Dorado–Dorado	111	91	145	112
DLion–Dorado	244	204	N/A	N/A
Dorado–DLion	431	428	N/A	N/A
Dorado–Dolphin	659	550	980	602
Dolphin–Dorado	265	220	426	294

(times in milliseconds)

Table 5.1. *Round trip communication time for function calls using the REMOTEVAL package.* The simple call was "1". The long call was a concatenation of several long strings. Wait means wait for the return value. NoWait means wait only for an acknowledgment (the return value is presumably not of interest). These times are approximately 100 times slower than those reported by Birrell and Nelson [Birrell83] using Cedar. The degradation is mainly due to the non–preemptive scheduler in Interlisp.

Average RoundTrip Times

	Courier			
Machines Client–Server	Simple Call Wait	Simple Call NoWait	Long Call Wait	Long Call NoWait
Dorado–Dorado	41	N/A	50	N/A

(times in milliseconds)

Table 5.2. *Round trip communication time for function calls using the COURIER package.* The simple call was "1". The long call was a concatenation of several long strings. Courier is about twice as fast as REMOTEVAL; the non–premptive scheduler is still the main communication bottleneck.

Originally considered as a practical compensation for the non–preemptive scheduler in Interlisp, the *cooperative model* (see Figure 5.25) makes the critical assumption that all participants have non–hostile intent and are trying to cooperate. This model can fail miserably in an adversarial environment. In this model each machine maintains its own copy of the database. Changes to the virtual global database are installed by broadcasting the change to the replications without any synchronization.

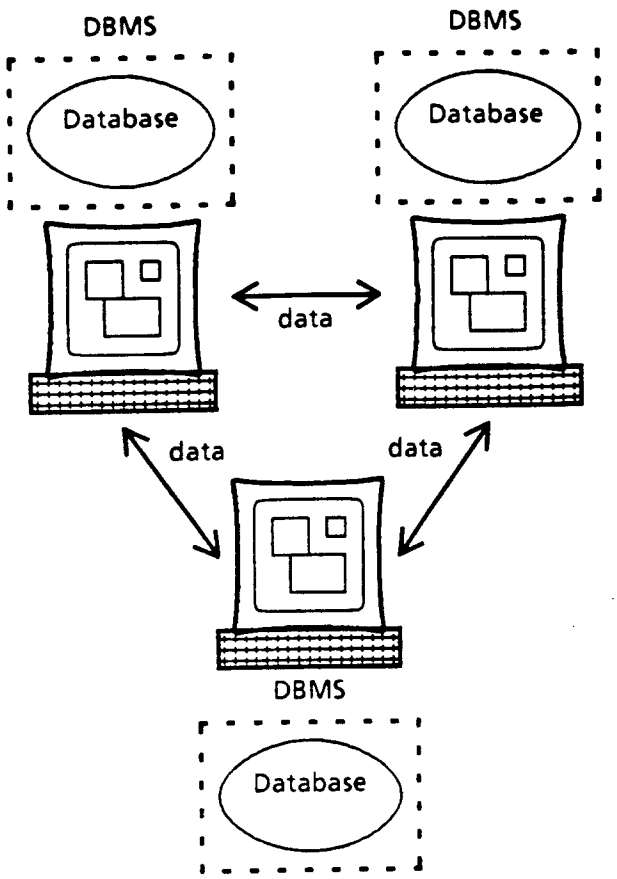


Figure 5.25. *Cooperative Model.* Changes to the virtual global database (replicated at each workstation) are installed by broadcasting the change without any synchronization. This scheme relies on human time-scale events and cooperative behavior.

Considered theoretically, this approach is dangerous and irresponsible, rife with potential conflict and race-conditions. In practice it is possible to lose work, which is unacceptable in production database systems. If two participants make nearly simultaneous changes to the single datum, there is a race to see which change will take effect (and which is lost). Worse, the outcome of the race can be different on different machines. These

shortcomings of the cooperative approach are counterbalanced by several factors.

Real-time performance. One helpful factor has already been mentioned: this approach, especially when combined with semantic-action packaging (see chapter three), is fast and permits real-time system response.

Independent changes. Another mitigating factor of the cooperative approach is that for most sequences of changes to a Colab session database the order of the changes is irrelevant. Almost all changes in the early uses of Colab were independent.

Humans in the loop. Since human beings are on the critical path for alterations to the database, changes take place on human time scales and, therefore, simultaneous changes to data will be very rare. Nearly simultaneous interactions can be avoided with busy signals (mentioned earlier) that alter the appearance of objects and warn participants that an item is already undergoing alteration. As a further mitigation, the participants are aware of several kinds of actions that have the potential to cause conflicts and will *voice lock* a portion of the shared data. "I'm going to expand the nodes in the Related Work group."

Social coordination. Colab tools, Cognoter in particular, are designed to coordinate the actions of participants. One reason to focus on face-to-face meetings is to exploit social mechanisms for dividing up the work and for reaching or maintaining consensus. The busy signal greying-out of a displayed object is an example of Colab software making use of existing social coordination mechanisms.

Discovery and recovery. In real systems, there is an inevitable delay between the moment someone starts to alter an item and the propagation of busy signals to other participants. It is possible that two users will begin conflicting work on an object at very nearly the same moment. However, the busy signal ensures, at least, that the two participants can quickly *discover* that they are beginning to work in a conflicting way. Allowing them to

recover before they have invested very much time. In a face-to-face meeting conflict resolution can be quickly negotiated once noticed.

Compare the cooperative model to the design constraints of telephone systems: it is considered acceptable behavior for some percentage of phone calls placed to fail (by dropped connection or wrong number). Such failures require the user to redial. Little work (other than dialing) is lost and, with little or no error detection/correction, the system is fast. People are willing to assume that they have made a mistake in dialing if failures don't happen very often[1]. Users even tolerate systems that are known to be unreliable so long as work tends not to be lost. Once a connection is established, different, more reliable mechanisms, are used to maintain it.

Use of the cooperative model has shown that data conflicts occur rarely in human time-scales. Therefore, the cost of system interruptions for conflict negotiation should be small, probably much smaller than the cost of maintaining locks. Related approaches are called *certification* by Bernstein and Goodman [Bernstein81] and *validation* by Kung and Robinson [Kung81] and Sarin [Sarin84].

Although the database management facilities in Colab may eventually have to adopt provably reliable techniques, probably involving some form of two-phase locking and time-stamping, in Colab's special domain — where all users of the database are in constant verbal, visual, and computational contact with each other — it is reasonable to consider user intervention in occasional cases of synchronization failure.

The ideal database model would run acceptably fast and guarantee data consistency across machines. Work on more efficient locking schemes, data dependencies, and consistency checking is in progress. The best model now appears to involve consistency checking, dependency detection, and roving-keys. The ideal model will also take advantage of the social-process lessons learned from the cooperative approach. This is discussed further in chapter seven.

Summary

This chapter has described the object-oriented implementation of Colab and Cognoter. The system is flexible and open-ended, suiting the experimental goals of the project. There is no claim being made that the implementation described has converged to an ideal state. The implementation works within the design constraints and desired features presented. The next (fifth?) overhaul of Colab is currently in the planning stages. Cognoter continues to be in a evolutionary state.

Particular lessons learned from the implementation include a confirmation of the benefits of object-based systems and the layering of system abstractions. The ColabExec is the high-level interface to the system. Conversations are encapsulations of machines, tools, and participants. Cognoter consists of special active windows, items, links, and group items. After several trials, the database issues for cooperative work systems are now better understood. The interim scheme of choice is the cooperative system that relies on social coordination.

The next chapter describes observations of early use of Cognoter and Colab. These informal and semi-formal observations led to a series of experiments to test the effectiveness of the design and features of Cognoter and Colab.

Note:

[1] According to Garth Gibson, a former employee of a telephone company that shall remain nameless.

6

The endless praises of the choirs of angels had begun to grow wearisome; for, after all, did he not deserve their praise? Had he not given them endless joy? Would it not be more amusing to obtain undeserved praise, to be worshipped by beings whom he tortured? He smiled inwardly, and resolved that the great drama should be performed.

— The history of creation, as told to Dr. Faustus in his study by Mephistopheles

---

Experiments with Cognoter

During the later stages of its development Cognoter was used informally by various groups as an idea organization tool. Observation of these early meetings provided an opportunity to reconsider several assumptions and test some guiding hypotheses. These first observations also revealed some surprises. The observation of initial uses of Cognoter led to a series of experiments to explore some of the early notions of the design of Cognoter and its effect on user behavior and task effectiveness. The observations of early users and the results of the experiments provided the background for a deeper understanding of what Cognoter does for groups that use it and the formulation of more precise research questions.

Introduction

Earlier chapters have made occasional allusions to "early uses of Cognoter" and "Cognoter experiments" for on-the-fly justification or comment. This chapter presents some evidence and plausibility arguments for the general statements and unsubstantiated claims that have appeared in previous chapters. Much of the evidence presented here is anecdotal. The appropriate technology for observing and measuring activity in the Colab did not exist during this dissertation work. It is just now beginning to be incorporated into the Colab environment and will be used for the next phase of the project. The main evaluation strategy used for the present work was to collect reports from observers who have travelled to the collaborative system front and returned to tell the tale.

The "early observations" come in two flavors: informal observations made from meetings using Cognoter in the Colab meeting room at Xerox PARC during the last year or so, and more formal observations made during a small set of controlled Cognoter experiments done at Berkeley in the spring of 1986.

This chapter organizes these observations by considering them under a few general headings: *Organizing Ideas*, *Phasing the Process*, *Equal Opportunity*, and *Group Focus*. The observations are discussed in relation to assumptions and hypotheses engendered during the design phase of Cognoter. There is also an attempt to understand the corroborations, contradictions, and surprises that occurred during design, early use, and experimentation. Throughout this chapter are headline paragraphs like:

> *Assumption: An interesting and workable domain to consider for computer-based collaboration is face-to-face meetings where each participant has a workstation, and the workstations are connected by a network.*

The assumptions, hypotheses, and surprises headlined in this way serve as a framework for considering facets of the early observations and experiments. Assumptions generally have implications and the headlining of an assumption means that observed implications will follow. Most of the following hypotheses could be recast in negative language and the corroboration that usually follows them could be viewed as refutation. There were a few instructive surprises. An exploration of the surprising activity and processes at work follows a headlined surprise. Many of the following assumptions and hypotheses concerning Cognoter apply generally to Colab as well. The Reader has been spared reading the phrase "Cognoter in particular and Colab in general" over and over.

It is important to emphasize that in the complicated and broad–based task domain chosen, the method of evaluation was to gather information from Cognoter session observers and participants. First a small digression to make things a bit more specific.

A Little History

During the design and implementation of Cognoter, a few brave souls attempted to use it for idea organization and the planning of presentations. The tool design evolved by taking into account the difficulties these explorers encountered and the desires they expressed for new features (creeping featurism...). Observations made during these early sessions led to the design of a simple set of controlled experiments.

The set of experiments was designed to get a better indication of the effectiveness of Cognoter and the problem–solving structures it supplies (such as the phasing of the organization process), as well as to test some to the basic ideas behind Colab (i.e., "Can people work concurrently on a shared problem?"). It was expected that any observed improvement in task performance by Cognoter over traditional problem–solving techniques would be significant since no claims were being made concerning the optimality of the Cognoter and since users have much greater familiarity with paper or blackboard than they have with Colab and Cognoter.

These experiments involved *dyads*, pairs of participants. Each group (dyad) performed two tasks. One task was to generate an outline for an article on a randomly chosen topic using whatever traditional technology they wished (usually paper and pencil, or blackboard and chalk equivalent). The other task was to generate an outline for an article on a different randomly chosen topic using Cognoter. The order of the two tasks was determined randomly. Each dyad functioned as its own control group since they performed both the target task and the control task in randomly determined order.

This is only a brief introduction to the experiment; a more detailed account appears later in this chapter and in appendix D. In the following, when referring to the actions of the dyads during the experiments, numbers and percentages are not used since the sample size was too small (five groups) to be statistically compelling. The experiments were, however, instructive and serve as an initial test of Cognoter's design and effectiveness and a guide for enhancements.

Organizing Ideas

Computers can be used to support idea organization and writing processes in many different ways. Cognoter, as implemented, expresses one set of assumptions about idea processing. It embodies a process that is intended to more–or–less monotonically move the group from undeveloped and unstructured ideas through small–scale organization to the final generation of a presentation path or outline on the desired topic.

Observations of the use of Cognoter show that in many cases its design dovetails with what people were trying to do. Brainstorming, for instance, is a natural beginning. The dyads participating in the experiments usually started with at least a short brainstorming session to get some ideas on paper (or the board). During the Cognoter subtask they generally spent more time brainstorming, though there are several explanations for this — including simple time pressure and lack of understanding about how to use the tool.

The ease of adding ideas to the group pool of ideas was particularly appreciated by most early users of Cognoter.

In some other cases, the design of Cognoter caused friction and subtle difficulties. For example, the main Cognoter brainstorming window during the brainstorming phase is designed to be an unstructured area for participants to put their ideas without having to decide on an idea's place in the overall scheme of things. Early sessions with Cognoter, however, showed that group members used the physical proximity of idea nodes in the brainstorming window to establish connectedness and order between ideas before moving on to the ordering phase where the group members are supposed to consider the relationships between ideas. Later, in the ordering phase, even after items have been explicitly linked, the spatial proximity continues to imply relationships between them. This is all very good, except that the spatial cues (the closeness of items) sometimes overpowered actual links in *perceived* closeness — items displayed close together were often implicitly the most strongly linked than actually linked items in the participants minds — so the participants may have a conceptual image of their idea structure that is significantly different from the idea organization contained in Cognoter's data structures.

> *Assumption: Small-scale local decisions are easier to make than global decisions.*
>
> *Hypothesis: Incremental small-scale decisions can yield a coherent large-scale organization.*

The incremental approach encouraged by the ordering phase of Cognoter is surprisingly effective. In the early Cognoter meetings, as hoped, many small decisions about the ordering of ideas generally yielded a globally cohesive organization. Flower and Hayes [Flower80] describe the act of writing as an "act of juggling a number of simultaneous constraints" and stress the necessity of "effective strategies for handling this large number of constraints."

Incremental decision-making coupled with the computer's flexibility and memory seemed to reduce the "cognitive strain" of paper design.

Small-scale decisions about the ordering of items are usually simple to make and can lead to large-scale organization, even when large-scale decisions appeared to be difficult. It's usually easier to see that **A** comes before **B** and that **C** comes before **A** than to figure out, all at once, where **A** fits into the grand scheme of things.

In early uses of Cognoter there was a general impression reported among the users that the ease of placing idea nodes and the flexibility of the ordering process made overall organization easier. In both instances the low commitment to placing an idea or linking two ideas together made the decision to place or link much easier — it could be moved with little effort. However, in interviews after the experiments, there was no specific mention of the incremental actions helping to make large-scale decisions.

Phasing the Process

Hypothesis: Partitioning the problem-solving process into brainstorming, ordering, and evaluation phases with distinct operations and goals is helpful.

In theory, the process of ordering and evaluation proceeds more effectively when brainstorming is complete and all the ideas are available. The process of ordering can also expose under-brainstormed areas (subtopics that need to be developed by generating more ideas).

There is some evidence counter to the hypothesis headlined above, at least in terms of the specific phasing and operations offered by Cognoter. Several of the early users of Cognoter said that they felt unproductively constrained by the brainstorming phase. The usual complaint was the lack of linking and grouping capability during that phase. Delayed deletion and evaluation of the nodes was generally accepted and appreciated in early use and experiments — the protection from criticism by the phases and the rules of the game were generally seen as favorable, but the desirability of separating brainstorming and ordering was less clear. Groups often wanted to link as they brainstormed. Based on early observation, the question of

whether or not a tool should enforce the rules of the game by phasing and partitioning operations is still open.

Hypothesis: Outline generation will be the final operation in a presentation planning session.

During the design of Cognoter, the outlining utility was seen as something that would only be used as an output format. In practice the users of Cognoter were much more resourceful. In addition to using the outlining feature to display the final result, they also used it to view work in progress — they found the familiar outline form useful for looking at intermediate states of the emerging structure and highlighting under–constrained nodes and under–developed subtopics.

The preceding disjuncts between the phasing assumptions of the tool design and the expectations and actual use of Cognoter show that "natural" organization techniques and assumed partitioning of task phasing is not quite right. The difficulties that some users suffered suggests different phase boundaries (if any) and operations than were originally assumed. Future tool designs will be judged by how well or poorly its design assumptions fit the reality of what people want and are able to use.

Equal Opportunity

Assumption: Sequential access to group problem–solving technology limits each individual's ability to contribute to a collaborative effort.

Hypothesis: The ability to simultaneously act in the system encourages equal participation and parallel activity.

A motivation for the design of Cognoter's multi–user interface is the notion that each participant in a Cognoter session should have equal access to the changing database of the group's ideas and to the tools to alter the database. Cognoter is an attempt to circumvent the usual turn–taking by allowing users to act simultaneously. Participants at personal workstations may compose new ideas and add them to the group database at virtually the

same time. Participants can act on ideas at the moment of inception. This is not to say that all participants are assumed to have equal contributions to make to a given situation. People have their individual strengths and weaknesses and will, presumably, contribute according to their abilities given the opportunity. Cognoter strives to provide the opportunity.

In the experiments, where participants had a chance to directly compare the free-wheeling process of Cognoter with traditional serial techniques, they reported feeling constrained in the serial interaction. The participants wanted to feel they had freedom of expression — even in situations where they didn't need it or didn't use it.

For cooperative problem-solving in face-to-face real-time situations, the kind Cognoter is designed to support, collaboration seems ideally not to involve any prescribed division of labor among participants. Cognoter's multi-user interface is intended to enhance a single non-synchronous and flexible role. By equalizing the access of all participants to views on the database and to the shared data itself, Cognoter encourages wide participation and makes it difficult for any single person to dominate the group activity.

It may well be that for some groups and for some kinds of problem-solving activities, role specialization, participants taking on fixed roles or roles prescribed by a problem-solving technique, will be more effective than the egalitarian approach encouraged by Cognoter. Role specialization, if deemed desirable, can be used with Cognoter software as it exists by changing the rules of the game. For example, Cognoter could be used by a group of people each of whom has specialized expertise with a single discussion leader partitioning the work and making assignments.

Group Focus

Observation of problem-solving sessions showed that the serial access to writing tools enforced by traditional technology isn't all bad — one thing good about it is that it helps to maintain a shared focus. People working

together on a problem using traditional technology, say a blackboard, maintain group focus by watching the person writing on the blackboard. The apparent bottleneck of having to get all ideas through the person writing at the blackboard has the beneficial side effect of slowing things down enough to allow all participants access to all ideas as they are entered on the blackboard. Any changes to the group database (the marks on the blackboard) are made by the single database manager (the person with the chalk) in full view of the other participants. In serial access situations like this, everyone paying attention can stay current.

> *Hypothesis: A shared database and synchronized displays enable shared focus among participants in a computer-based meeting.*

Shared access to objects contributes to a common group focus. This applies to computer-based tools such as Cognoter as well as it does to blackboards. A group working together needs as much common experience relating to the problem at hand as possible. This is especially relevant when seeking to assess possible improvements brought about by working in parallel. Participants need to be able to easily refer to common objects by pointing at them or by verbally specifying their location. The WYSIWIS ideal described in chapters three and four is a recognition that efficient reference depends on the maintenance of a common view.

Giving a group the ability to work in parallel makes it difficult for each member to keep in touch with what the others are doing. Unfortunately, to make effective contributions to a group effort, it is usually necessary know what the rest of the group is up to. If the usual requirements of taking turns is relaxed by allowing parallel activities, simultaneous entry of new items to the group's blackboard, for instance, then some way of accomplishing what turn-taking accomplishes — orderly transitions, incremental development of the group memory, and group focus — must be found. Two basic techniques were discovered (both involve periods of serial interaction): summarization and task partitioning.

Surprise: The divergence/convergence tension created meetings that oscillated between episodes of parallel activity and episodes of summarization to bring the group back together and planning to hold it together longer.

In Cognoter sessions the need to maintain a shared focus is expressed in a characteristic pattern of activity. There are three distinct stages: joint planning, parallel individual activity, and summarization — these stages are interestingly similar to the method partitioning described in chapter three (user–actions, semantic–actions, and display–actions). In the first stage, users interact verbally (conversationally and serially) for a few minutes, discussing goals and plans of action. In the second stage, the group members settle into portions of the task they have taken on and a period of intense individual interaction with the system takes place. Gradually, after a few minutes of simultaneous idea generation and editing, the group tends to lose track of what the others are doing. When a majority of participants begin feeling lost, the group slows down, stops interacting with the system, and returns to serial verbal interchange. In the third stage the participants summarize the current state and explain what they have been doing, and return to goal discussion and planning for another round of activity.

Cognoter sessions routinely consist of several such cycles of divergence and convergence: joint planning, parallel individual action gradually diverging, and summarization to converge back again.

Surprise: The video multiplexer in the Colab meeting room provided a desired access to smaller scale writing processes, like text editing.

In early Cognoter sessions it was frequently observed that one participant would want to watch another editing. Occasionally two or three participants would want to edit together — usually with a "driver" and a "navigator" or two. In Cognoter's current (pragmatically dictated) design, editing is done in private windows with the result being distributed publicly. Sharing information at the character level is not provided for: the person editing has to explicitly "send" edited data. It so happened, however, that in

the Colab meeting room at PARC each workstation had a video switch connected to it that allows the screen image of any other station to be slaved and displayed there. The switch was originally installed so the meeting room would be more useful for ordinary (i.e., non–Colab) demonstrations. The video switch was frequently used by participants to watch the editing process of others. In the tests at Berkeley there was no video switch, but on several occasions one participant would slide over to watch editing taking place "privately" on another participant's screen. This unexpected desire of users to watch the editing process underscores the importance of maintaining a shared focus by shared participation in group processes over a range of grainsizes.

The Experiments

This section presents a description of the Cognoter experiments done at Berkeley and their results. The effectiveness of a group working with a blackboard is taken as the standard by which the effectiveness of a collaborative system can measured.

The first experiments consisted of five dyads (pairs) performing two subtasks: a Paper[1] task and a Cognoter task. The ten participants were all computer science graduate students and all were experienced computer users. The experiments were run in an ordinary office with two Xerox Lisp machines, some table space, and a whiteboard. This experiment did not use the Colab meeting room at PARC. The various response forms, reference cards, faked telegrams, etc. used in the trials are reproduced in appendix D.

The Protocol. Each group generated an outline for each of two different topics: one outline was done using the support technology of their choice (usually paper and pencil, or whiteboard and pens), and the other one was done using Cognoter (Figure 6.1 delineates the basic experimental protocol).

Introduction

Basic Information Form

Coin Flip for Support Technology

(Cognoter Instruction — if necessary)

Random choice of First Topic

First Fake Telegram

15 Minutes to Generate Outline

Break

(Cognoter Instruction — if necessary)

Random choice of Second Topic

Second Fake Telegram

15 Minutes to Generate Outline

Debriefing Form

Figure 6.1. *The Experimental Protocol.* The experiment is self–controlled — each group used both support technologies (Cognoter and Paper) in random order.

The group was gathered and offered refreshment while they responded to a few questions on a short demographic form. This form was purely informational; it concerned the computer expertise of the volunteers, their familiarity with Cognoter, and the extent to which the group had previously worked together. A coin–flip determined which organizational technology (Paper or Cognoter) was to be used first.

The first topic was randomly chosen from the four possible topics (see table 6.1). The participants were given a fake telegram purporting to be from

a national magazine (*Atlantic Monthly*) begging for their help in designing an article on the chosen topic. (Telegrams addressed to all groups for all topics were made up in advance.) The Participants were told that in fifteen minutes a special courier from the magazine would arrive to rush the outline to the desperate editors. A watch was started and the group began their work. The participants were reminded of the time left at five minutes and two minutes.

The Topics
Games and Sports in Society (GSS)
The Strategic Defense Initiative (SDI)
National Parks and Wildlife Preserves (NPWP)
Affirmative Action (AA)

Table 6.1. *The Topics used for the experiments.* Each subtask in the experiment involved generating an outline for a magazine article on one of these topics. Topics were chosen for their generality and open-endedness. It was expected that some topics would be more fruitful and thought-provoking than others.

After the group had completed the first task as best they could in the limited time and taken a short refreshment break, a second topic was randomly chosen and the participants were given another fake telegram: this one purporting to be from the editorial staff of another national magazine (*Harper's*) that had heard of their success with the previous magazine article. This magazine also needed an outline — for the second topic — ASAP. The group again was given a deadline of fifteen minutes before a courier would arrive to snatch up their outline and rush it to the anxiously waiting editors.

Immediately before the Cognoter subtask was undertaken the group was seated at workstations running Colab and Cognoter, and given about

fifteen minutes of Cognoter instruction. The instruction consisted of a guided tour through generation of an outline for a dummy topic and a general explanation of the main features of Cognoter.

After both sessions were complete the participants were given a short debriefing form to capture their reactions to the experiment and any other comments about it that they wished to make. A summary of these comments appears below in the Results subsection.

The four topics chosen for the experiments (see table 6.1) were chosen for their generality and open–endedness. Some topics were intended to be controversial (SDI and AA). Some were less controversial but still thought provoking (GSS and NPWP). There is no claim that the topics are equivalently deep or broad or exciting. They were intended to be topics about which everyone has some knowledge or opinions. It was expected that some would yield more ideas than others. This variation in topic interest can be factored out by viewing each topic as a partition of the experiment, though this expected variation in topic yield turned out not to matter very much (see tables 6.2 and 6.3).

Comments on the Protocol. The experiment was designed to be self–controlling: each group performed the target task (Cognoter) and the control task (Paper). Functionally there were three groups: free Paper technique, Cognoter use, and Paper technique under the influence of the method embodied by Cognoter (where Cognoter was the first technique used or where already experienced Cognoter users entered the experiment). The *practice effect* (or the *familiarity effect*) was accounted for by sometimes running Cognoter first, sometimes running Paper first. It was expected that the second part of each session would yield higher quality results since participants were warmed up, individually and as a team. Fatigue and boredom were expected to be a less important counter–acting factor, since the sessions were short (approximately an hour all together) and reasonably entertaining.

Experimental Results

The goals of the experiments included seeing how Cognoter was used by people unfamiliar with it and gaining practical insight into what the tool does well and what the tool does poorly. The experiment was also trying to determine whether the *techniques* embodied by Cognoter or Cognoter itself, as a computer program and a collection of techniques, account for any improvements in performance. The simplest way to approach these goals was to compare the use of Cognoter to traditional ways of generating and organizing material. The experiments were too broad–based to provide incontrovertible evidence for or against any particular tool feature or problem–solving technique. It was impossible to separate the many processes going on during the test sessions. However, the experiments did supply a look at how Cognoter was actually used by groups trying to organize ideas.

Evaluation. Because of the subjectivity of the results, any sort of numerical evaluation would be difficult. Before the experiments were carried out it was thought that some crude indications of merit comparing work done during the tests would be possible. The total number of entries in each outline might give an indication of lines of argument and the total number of ideas generated. The number of complete sentences might measure the completeness of idea development. The number of words could be taken as a measure of overall length. The number of entries at each level of the outline might measure the degree of organization. The results appear below in Table 6.2. Since the participants did not know that such crude measurements were being considered, they may have some small validity.

Dyad Experimental Results

Cognoter					Paper				
topic	lines	words	sents	depth	topic	lines	words	sents	depth
NPWP	21	40	0	2.3	*SDI	17	58	2	1.8
AA	29	108	5	3.0	*SDI	23	67	3	1.8
GSS	23	59	2	1.7	*AA	9	28	0	1.6
*SDI	21	49	3	1.9	GSS	28	63	0	2.3
GSS	31	75	2	1.8	*NPWP	23	55	2	1.7
Ave:	25	66	2.1	2.1		20	54	1.4	1.8

Table 6.2. ***Results of the first experiment.*** Five groups performed pairs of tasks — one using Cognoter and one using their choice of support technology (called "Paper" here). Each row is a pair of tasks. "*" indicates the topic taken first. The topic acronyms are introduced in Table 6.1. Lines is the number of entries in the final outline. Words is the total number of words in the final outline. Sents is the number of complete sentences in the final outline. Depth is the average depth (indentation) of an entry in the outline.

The participants were more explorers than laboratory test subjects. Another criterion of merit is the perception by the groups themselves of what was going on. How satisfied were they with their own performance? Responses to the debriefing questionnaire appear in the following section. Future experiments will also measure quality by submitting the (standardly formatted) outlines to independent referees or panels for judgment and ranking.

Dyad Topic Comparison

GSS					SDI				
tech	lines	words	sents	depth	tech	lines	words	sents	depth
Paper	28	63	0	2.3	*Paper	17	58	2	1.8
Cogno	31	75	2	1.8	*Paper	23	67	3	1.8
Cogno	23	59	2	1.7	*Cogno	21	49	3	1.9
Ave:	27	66	1.3	1.9		20	58	2.7	1.8

NPWP					AA				
topic	lines	words	sents	depth	topic	lines	words	sents	depth
Cogno	21	40	0	2.3	Cogno	29	108	5	3.0
*Paper	23	55	2	1.7	*Paper	9	28	0	1.6
Ave:	22	48	1.0	2.0		19	68	2.5	2.3

Table 6.3. *Topic comparison in the first experiment*. Rows do not represent pairs of subtasks. "*" indicates that this subtask was first. Topic acronyms are introduced in Table 6.1. Lines is the number of entries in the final outline. Words is the total number of words in the final outline. Sents is the number of complete sentences in the final outline. Depth is the average depth (indentation) of an entry in the outline.

The data in Tables 6.2 and 6.3 is inconclusive. Performance was roughly the same across topics and across support technology. Cognoter's slightly better performance can be explained away by the practice effect since it was the second subtask for all but one group (compare GSS and SDI in Table 6.3, GSS was consistently a better topic than SDI — GSS was always the second subtask and SDI was always the first subtask). The fact that the results are essentially the same may be significant (in the long run) in favor of Cognoter, since most users spent the bulk of the 15 minutes for the Cognoter subtask learning how to use the system. Much of the work produced during the Cognoter subtasks took place in the last few minutes of each session.

Debriefing Questionnaire. This subsection presents and discusses a selection of responses to the debriefing questionnaire (the questionnaire itself appears in Appendix D). This questionnaire was designed to get the participants to subjectively evaluate their comfort, efficiency, and satisfaction. Most groups primarily used the whiteboard for the Paper[1] session, shifting to paper for the final form of the outline.

Question: Did Cognoter help or hinder?

- It helped me come up with ideas, but unfamiliarity with the tool hindered my organization.
- Learning curve dominated. Response time was slow — I can scribble faster than I can pop a window.
- It hindered the naive user from reorganizing a hierarchy.
- Some help in that one doesn't have to have one person do the writing. Some hindrance in that I couldn't get the windows to open to write something down.
- Helped organization. Hindered expression because I didn't know how to use the system.
- It hindered us in that we had to learn how to use it.
- Novices are going to spend more time fighting the system than working.
- It certainly makes entering data easy — we added a lot of ideas quickly.
- [The rest thought it generally helped]

The responses to this question can be summed up as: Cognoter was perceived as being useful but the difficulty of using the tool slowed users down. This clearly suggests that the users needed more instruction and/or a longer session.

Question: Did you feel that the phasing organized your approach? Or did it only get in the way?

- Yes, as long as the operations in earlier phases are available in later phases. You don't want to feel that changing phases is an irrevocable commitment.
- Got in the way a little. I like modeless operation (I would have liked linking sooner).

- I think brainstorming and ordering should be combined. Ordering is inherent in the way I think.
- I think one should be able to order in the brainstorming phase.
- Good if you can go back and forth.
- Too rigid. I would like to do them in different orders.
- I think you could be doing all at all times — the user should know enough to brainstorm first.
- At times I felt like I wanted to move on so I could use the added features of the next phase. I could see jumping right to the last phase.
- [The rest thought it helped]

There is no clear consensus here. Participants seemed largely in agreement that some linking and grouping should be allowed while brainstorming. Groups had little to say about the Evaluation phase — probably because they spent so little time there. People largely agreed that pure brainstorming was a good thing to do, but they didn't like being prevented from doing other operations during the brainstorming phase.

Question: Did your team work in parallel during either session?

- [Cognoter] Yes. By grouping things you can act in parallel. [Paper] Yes, but we didn't break things down and work independently as much.
- Much more parallelism during Cognoter session. But less fusion of ideas. During the paper session there was more fusion of ideas. We communicated verbally much more during this session.
- [Cognoter] Some, but I wanted to see what [my partner] was editing. [Paper] Yes we did, but I acted as secretary.
- [Cognoter] Somewhat, but it was hard to keep track of what was going on in parallel. [Paper] No.
- [Cognoter] Yes — independently, in fact. [Paper] No.
- [Cognoter] Yes — parallel and independent too. It takes a little while to get used to saying what you are doing so you can synchronize. [Paper] Yes — we had two talkers and one writer.
- [The rest said "yes" for Cognoter and split for Paper]

Members of the same Dyad didn't always agree on degree of parallelism in their group. All groups worked in parallel more of the time when using Cognoter than when using Paper.

Question: Which session did you like better?

- Cognoter makes generating a more complete outline easier.
- Cognoter — less sloppy — more *potential* power.
- Cognoter. It was much more interesting.
- I preferred paper, in part because of my experience with that tool.
- Cognoter is more fun, like a trip to the museum of science and industry.
- The [Cognoter session] was hilarious and fun for a hacker. The [paper session] was more satisfying in the sense that I was in control and not at the whim of a machine.
- In spite of slowness and unfamiliarity, I liked Colab [Cognoter].
- [The rest split or had no opinion]

Cognoter won the popularity prize, probably either because it was new or because the participants didn't want to hurt my feelings.

Question: Which session was more effective?

- Cognoter — mainly because it is easier to generate and organize ideas.
- Cognoter was more effective, but the learning curve for a new system got in the way.
- Time pressure in last phase of Cognoter got in the way. Paper gives a better feel for how much there is left to do. Cognoter should help this by showing items that are not yet linked or grouped[1].
- Paper. I'm much more used to it. Cognoter may be better with a little practice.
- We did a more complete job on paper because we could use the technology.
- [The rest said "Paper" or had no opinion]

Once again, comments concerning the users' lack of familiarity with Cognoter dominated here. Not surprisingly the people who managed to get into the swing of it felt more effective. People who felt Cognoter was more

effective may also have been giving it the benefit of the doubt because they perceived a potential; they imagined they could be more effective with such a tool. Participants were much better at time management during the Paper sessions. They were distracted by awkwardness with the system during the Cognoter sessions.

Notes on the Paper Sessions from the Observer's Notebook. The following is a summary of notes taken for each group during the Paper session (most groups primarily used a whiteboard for organization and paper for the final outline). Bracketed comments were added after the session. The demographic survey revealed that none of the pairs had ever collaborated on a writing project before.

1. This dyad used a whiteboard. They started sequentially with "Ok, what's I.A?", but quickly decided to brainstorm [one member of this group had used Cognoter before]. The brainstorming decision notwithstanding they soon picked a few main headings and expanded them in a top down manner. Two minutes in: "What are the goals?" Seven minutes in: "But what are the goals?" [The general form of the outline was organized with links and groupings a la Cognoter.] A lot of things were said that were never written on the board or on the outline. They seemed to be condensing their material for the stated goal of an outline.

2. This dyad taped sheets of paper to a table and immediately began brainstorming in parallel — writing ideas down without consulting with each other. After five minutes they began to work together in a top down fashion. The last seven minutes were spent arguing details.

3. This dyad used the whiteboard. They first started a high-level discussion. "What's our thesis?" After three minutes, they began to write an outline but quickly returned to high-level discussion. "What's our thesis?". Five minutes in: a title was chosen. More high-level discussion. Seven minutes in: they began writing topics depth first. Eight minutes in they began a main topic outline: "What's our thesis?" With three minutes left they began

to seriously make an outline but ran out of time. [This group clearly lacked task focus.]

4. This dyad used the whiteboard. The first five minutes were spent in alternating parallel and sequential brainstorming [occasionally two writers]. Ten minutes in: they began to argue content and fine points of organization. Eleven minutes in: they began the sequential writing of the outline [one scribe and one kibitzer].

5. This dyad also used the whiteboard. General brainstorming (one scribe, one kibitzer) for three minutes. Three minutes in: one member took the lead — the scribe became the idea generator and the kibitzer became an agreer. With three minutes to go they began generating the outline: the ex-scribe read from the whiteboard while the ex-kibitzer took dictation on paper. The ex-kibitzer [now paper-scribe] did some re-arranging while the ex-scribe [now paper-kibitzer] looked on.

Notes on the Cognoter Sessions from the Observer's Notebook. The following is a summary of notes taken for each group during the Cognoter subtask. Bracketed comments were added after the session.

1. This group had difficulty with the system. They brainstormed for five minutes. Most of their work was done in the last few minutes [once they began to get the hang of Cognoter].

2. This group immediately began to work in parallel with no talking. One member preferred to use long idea labels instead of attaching text. This dyad began moving nodes into a outline-like arrangement during Brainstorming phase. After three minutes they began running low on ideas and started looking to each other [verbally] for inspiration. The rest of the time was spent expanding, linking, and grouping nodes. [They seemed to catch on to the spirit and techniques of Cognoter quickly.] Some time was spend figuring out how to get the grouping and linking to reflect the outline form they had arranged spatially during Brainstorming.

3. This dyad began brainstorming in parallel. Two minutes in: one member began watching the other since the other had started brainstorming inside a private editing window [that wasn't visible on the other screen]. Five minutes in: they began a high-level verbal discussion. Seven minutes in: they continued the discussion with one member building an outline in a private editing window. [This forced them to double up on one machine.] They decided to re-organize with five minutes to go. They entered the Ordering phase with two minutes left and had not finished organizing when time ran out.

4. This group began brainstorming in parallel. After two minutes they wanted to "clean up". They began moving the idea nodes to form a rough [spatial] outline. Five minutes in: they wanted to clean up again. This time they moved to the Ordering phase and began linking and grouping. [They moved to Evaluation at the last moment to generate the outline.]

5. In the first minute they couldn't remember what to do [they were immediately blocked by lack of knowledge about the system]. One minute in [after a few reminders from the observer]: they began brainstorming in parallel. Nine minutes in: there were (valid) complaints of system sluggishness [the network was very slow for a few minutes]. With five minutes to go they entered the ordering phase and began grouping and linking. [They moved to Evaluation at the last moment to generate the outline.]

Not much use was made of the Evaluation phase. No group spent more than two minutes in the Evaluation phase. Time pressures were usually such that little critical evaluation or deletion occurred. Three groups only moved to Evaluation at the last moment so they could generate the required outline.

Early users were confused by the difference between public windows and private windows. They didn't understand the underlying model of the conversation, yet were put in the position of having to deal with it.

All groups had difficulty estimating and budgeting their time using Cognoter. All groups ran into time trouble. This was partly due to the difficulty of learning how to use Cognoter. In most cases the bulk of the work

was done in the last few minutes. By contrast, the groups had little time trouble when using Paper, they usually finished their outline in about 14.5 minutes. This is hardly surprising since all users had vastly more experience using Paper than using Cognoter.

Future Experiments

These experiments were intended to guide intuition for future tool design, implementation, and evaluation. They lay the groundwork for a methodology for testing collaborative systems. To provide objective results the experimental protocol, while basically useful, would have to be changed in several ways. Clearly, the participants need more familiarity with the computer–based tool before a useful comparison with traditional techniques can be made. The tests showed that Cognoter was more complicated to use than anticipated. Another line of investigation might include tests of productivity/minute by allowing each group in each task an extra few minutes to improve their result. To provide quantitative results the experiments will have to be run over larger samples than these were. It will also be important to compare the performance of larger groups (three or more participants) to dyads and individuals. Informal observation suggests that groups of three or more are significantly different from groups of two.

The wiring for sound and video of the Colab meeting room (in progress at this writing) will enable more complete capture and analysis of meeting events. Future tests of Cognoter and other Colab tools will evaluate particular tool operations, features, and functions. What are the long– and short–term interactions among participants and which are the most important? At what stages of the process should diversity be encouraged and when is consensus desired? At what grainsize does different kinds of collaboration naturally take place? Should the collaborative process go through phases and to what extent should a tool prescribe or enforce phasing? To what extent do computer–based tools provide additional leverage over simply putting two or three motivated designers together at a desk?

The data collection sessions in this chapter suggest that while the phasing of activity is useful, some changes in the operations available in each phase will be necessary. It may be that the three phases should be combined into only two: essentially *brainstorming* and *evaluation* with the operations currently in *ordering* distributed between them.

The early uses of Cognoter were generally egalitarian and very task oriented. But all these sessions were relatively short — no more than a few hours in duration. What sort of culture and roles will develop in a group that uses Colab over a longer time? And once recognized, how does a tool enhance or enforce these things? This is difficult to predict and needs to be studied.

Summary

The most obvious lesson learned from these observations and experiments is that users must have a decent amount of familiarity with Cognoter before it can be fairly compared to anything. The amount of instruction for new users may be small (perhaps 30 minutes to an hour or two). It is encouraging that, in spite of persistent difficulties with using an unfamiliar system, participants in the experiments still managed to get appreciable work finished in 15 minutes. It became clear during the course of the experiments that the participants never got much beyond the "lag" portion of a sigmoidal learning curve.

There is some evidence that the small–scale incremental approach can help with larger decisions. The ease of adding and moving new items and making or breaking links was cited by several users as a useful feature. The correspondingly smaller commitment to placement may contribute as well — it's simpler to move an idea in a CognoterWindow than it is to erase and re–write it on a blackboard.

Another point is that people *do* work in parallel when using Cognoter. The "convergence/divergence" cycle — the necessity of maintaining a shared focus forced groups to stop working concurrently and, periodically, bring each

other up–to–date — was observed to a greater or lesser extent in all Cognoter sessions. Paper sessions did not exhibit this behavior and generally proceeded in a serial fashion (a few groups brainstormed on the whiteboard in parallel for a minute or two before settling into serial interaction). Related to this group focus issue was the observed desire of the participants to occasionally edit text as a group.

The three phases that Cognoter supplies to help users organize their ideas seem to be somewhat out of synch with what people want to do. Until people learn to work in new ways and use new techniques it will be necessary to provide tools that let them do what they want to do. This applies to the representation of idea organization as well. More than one group attempted to use the spatial relationship of idea nodes in Cognoter to design an outline. An active outline display that reflects back into the C-graph structure (and visa versa) appears to be a helpful, though possibly transitory, feature.

As an interesting sidelight, over the last several month Cognoter has beaten out the whiteboard and become the preferred tool for organizing talks and papers in the Knowledge Systems Area at Xerox PARC (the only place Cognoter is generally available at present).

In addition to evaluating the design of Cognoter, these early observations and "eye–witness reports from the frontier" are intended to generate questions for further research in collaborative systems (and computer–supported cooperative work). The next (and last) chapter sums up the work so far and speculates on the directions future work might take.

Note:

[1] Cognoter *does* show ambiguously ordered or unordered items, but no group used this feature — lack of familiarity once again.

7

There ain't any answer.
There ain't going to be any answer.
There never has been an answer.
That's the answer.
— Gertrude Stein

Conclusion and Future Work

In this chapter the goals and main concepts of this work are summarized and evaluated. The applicability and limitations of this work are discussed. Plans for extending the present work are described as well as some speculation on possible directions for future research.

Goals and Concepts

This dissertation addresses the design and implementation of *collaborative systems*, computer-based systems for supporting real-time cooperative work. Three questions have guided this work: Which facets and processes of cooperative work can computer systems effectively augment? What software tool features, underlying system organization, and interface presentations best support these facets and processes? What effects do these systems have on collaborators and the nature of collaboration?

An objective throughout this work has been to discover appropriate design dimensions for cooperative software tools and to determine their effect on multi-user interfaces and the structuring of group problem-solving processes, so that collaborative system tool designers may make enlightened choices when developing applications for a variety of cooperative activities. To illustrate and test general principles and concepts, a particular domain (face-to-face idea organization), collaborative system (Colab), and tool (Cognoter) have been developed and considered.

Another objective — perhaps more properly called a *bias* — has been to keep in mind Joshua Lederberg's advice concerning the importance of people as well as machines when using the computer as a communication medium [Lederberg78]:

> *We do well to question our moral capability of enjoying the fruits of such cooperation* [between humans and machines]*; but this is not to damn ourselves in advance, especially if we acknowledge that anticipating the human problems is a task of equal priority to engineering the hardware.*.

In this dissertation the following areas were explored:

Desirability of Collaborative Systems. In the first chapter the utility and limitations of traditional meeting support technologies, such as blackboard and chalk, were discussed. Many of the shortcomings of traditional technologies — especially short- and long-term (space and time) memory limitations, difficulty of rearrangement, and passivity — can be

ameliorated by computer-based systems. The challenge is to retain the desirable qualities of traditional technologies — their informality, familiarity, and availability — in newly designed collaborative systems.

Issues in Multi-user Systems. As explained in chapter three, the design of a multi-user interface is much more complicated than the design of a single-user interface. Multi-user tools must, in addition to providing the task functionality and human-machine interface qualities of single-user tools, handle inter-machine communications and the attendant problems associated with multiple readers and writers on shared data. Multi-user interface presentation constraints complicate the otherwise attractive notions of run-time customization and individual control over workspace and display.

Collaborative systems like the Colab that support simultaneous actions must also be concerned with distributed display issues: how are the views of the shared objects updated and displayed? At one end of the display philosophy spectrum is strict WYSIWIS where every screen shows exactly the same thing to all users all the time. At the other end is total individual control (and responsibility) by each user for the views of any shared models on each screen. A compromise position is necessary. Each user must have ultimate control of his personal display screen, but control must sometimes be relinquished to permit updates from the outside to maintain the illusion that all users are operating on the same underlying aggregate model.

A closely related issue is the question of appropriate mix of public and private actions and objects. Obviously, some display items need to be private; for instance, the reading of personal mail. Just as obviously, shared objects need to be publicly active so the group can interact with them. The grainsize of collaboration and degree of public access often changes during a session.

Design of a Collaborative System Tool. Cognoter, a Colab tool that has been implemented and used for real (and contrived) tasks, was presented in chapter four. The design and implementation of this collaborative tool highlighted some important questions. To what extent should the tools

structure the problem-solving process? What operations should be offered to the users, and in what way?

Based on a theory of idea organization process structuring used successfully in traditional meetings, Cognoter partitions the problem-solving process into three phases: *brainstorming*, *ordering*, and *evaluation*. The phasing of the meeting process was intended to enhance the usefulness of *incremental actions* — local operations and decisions that can yield global results.

Cognoter also demonstrated that *social locking* and *busy signals* together constituted a viable, though imperfect, database consistency technique in a concurrent environment — at least when the actions of a few human beings are being synchronized. Social locking is an implied or verbal reservation of a shared object. Busy signals are visible changes in the way a shared object is displayed when it is being changed.

Organization and Implementation of a Collaborative System. The organization of system objects and functions in Colab and Cognoter is not the only way that a collaborative system might be implemented, but it does demonstrate a reasonable way to do it. Building the Colab environment brought to light many challenges in distributed computing and multi-user interfaces.

In the course of the implementation new ways of viewing software modules for collaborative and distributed systems were evinced. One is the idea of an *association*, where, conceptually, there is a single shared object, the association, with an *associate* at each node. Each node (or user) has the illusion of working on *the* object. Another idea is the *conversation*, a collection of tools and cooperating nodes that know about each other and work in concert. The "working set" of tools and users in a conversation can change freely.

A series of database consistency schemes for shared data and multi-user interfaces were explored. The key here was to provide several

participants to with the ability to act on and view shared objects simultaneously.

Difficulties in writing and debugging distributed programs led to the development of *BroadcastMethod*s that hide the underlying communication protocol from the programmer and to the U–S–D disciplined partitioning of tool methods into *user–actions*, *semantic–actions*, and *display–actions*. Semantic–actions encapsulate several data requests or changes into a single network transaction.

The state and history of a Cognoter session is maintained digitally. Thus, it is a simple matter to stop and start collaborative sessions, or to bring new–comers up to date. Session state can even be shipped to remote groups for further interaction (sessions have been shipped between Palo Alto and Berkeley).

Cognoter expresses one set of design decisions in the collaborative system tool design space. To present the evolving organization of ideas Cognoter uses the *C–graph* representation described in chapter five — a C–graph is a hierarchical directed graph of annotated nodes. This representation carries along more information than an ordinary outline. Cognoter provides capabilities for a dual presentation (C–graph and outline) of its single internal representation of items and their relationships.

Evaluation of a Collaborative System. It is difficult to quantitatively evaluate the task effectiveness or user satisfaction of something as complex as a collaborative system. In chapter six the prototype system developed for this dissertation was evaluated primarily by observation of its use and by soliciting reports from the users.

It became apparent that people can and do work in parallel given a collaborative system that supports simultaneous activity. The active views of a shared database as supported by Cognoter was generally reported to be useful.

When members of a group work simultaneously each tends to lose track of what the others are doing. The observed sessions showed that an important element of computer support of collaborative work is the maintenance of group focus. For instance, users frequently wanted to share access to a single editing process. This was not directly supported so they had to physically move to the terminal that was running the editor or used the video switch in the Colab meeting room.

Future Directions

Future versions of Colab will certainly want to incorporate more of the session interactions into the digital record. Graphics, video, voice, and other media should eventually be captured and made available to participants and the system for manipulation. Video multiplexed bitplanes that allow a select portion of a remote screen to be displayed will solve the "edit spying" problem where participants occasionally want to observe remote activities on a character by character basis.

Active Outline Representation. Although the unusual hierarchical "idea graph" (C–graph) organizational technique used in Cognoter was reported to be effective by the experimental users, observations suggested the utility of an additional "active outline" representation. Cognoter supports individual views of shared objects, but the problem of maintaining views on more than one active distributed representation of a single shared idea structure is an important extension that has implications for the design of future Colab tools.

Use Profile. Use profiles will be another line of enquiry. What kind of usage patterns are generated by a collaborative system? It is expected that the "load model" will be bursty: that is, periods of little activity (for instance, while people are talking) followed by periods of intense activity and net contention with several messages per second as the group simultaneously interacts with the multi–user interface. More experimentation and data gathering at the system level is needed to check these intuitions and informal observations of usage patterns.

Interactions between participants take place over widely varying time intervals — micro-seconds to minutes to weeks — for different processes. One underlying mechanism is probably insufficient to support all of these levels of interaction. What sort of layering will be necessary?

Distributed Database Consistency. The basics of Colab tool implementation is fairly well understood; the primary system implementation difficulties in the future will concern database consistency issues. The schemes currently in use are not robust and rely too much on user attention and intervention.

As shown in chapter five, a promising approach to the database consistency problem is the *dependency detection model.* This model fixes some of the shortcomings of the cooperative model by attaching author and time stamps to data broadcasts. This allows changes to the database to be checked against the state of the database to see if a conflict has occurred. Any detected conflicts can be resolved as they occur by appealing directly to the participants. This scheme needs to be implemented and the possibility of some conflicts being automatically handled explored.

Collaborative systems like Colab are a special case of the distributed database consistency problem since these applications are driven by real-time interaction among human beings as well human-machine and machine-machine interactions. These systems have a different set of performance requirements and failure profiles than do other database applications. On one hand, the intensely interactive nature of collaborative systems makes real-time response (in human terms) essential. While on the other hand, since most events occur on human time-scales and the number of humans involved is small, some of the usual constraints on distributed database transactions can be relaxed. For example, it may be worth sacrificing a guarantee of data consistency in rare cases for overall faster response time. In most database work, data consistency is essential — this is usually approached by serialization of transactions. Less work has been done on reliable, but imperfect, systems that converge to consistent states or save alternate versions in the case of conflict.

Language Features for Distributed Collaborative Computing. In an ideal world it will be a simple matter to build new cooperative tools for collaborative systems. Software engineering organizational techniques like the partitioning of methods described in chapter five can simplify and accelerate the programming effort for distributed applications like the Colab. This combined with other language features, like the BroadcastMethod encapsulation of network protocols, enables rapid prototyping and testing of new collaborative tools. The rapid development of tools will enable correspondingly fast exploration of tool features and their effects on cooperative work.

Future collaborative system architectures will likely require basic transactional mechanisms such as locks and timestamps. Language designers have begun to create languages with features such as these for writing distributed programs [Liskov85,86]. (This contrasts with the approach of using a programmatic interface to a database, an approach which appears to be both too heavy-handed and too inflexible for many applications.)

The Meeting Analyst's Workstation. Tools for collaborative systems can also be used as a magnifying glass to look at group problem-solving. Group processes and actions are difficult to observe and analyze because interesting events can happen simultaneously and focus can shift quickly. To aid in such observations the Colab meeting room is being equipped with microphones and computer synchronized video cameras to enable a more complete record of the activity that takes place during Colab sessions (Colab software already captures digitally the system semantic action history). Machine-machine, human-computer, and human-human interaction data from these sessions can be used to develop analytic models to test against previous observations. The Colab meeting record and analysis can provide the beginnings of an account of collaborative processes, and to clarify the effects of computer-based systems on collaboration.

Applications of Collaborative Systems. The Colab project focuses on the development of the "group computer" and understanding how it can be effectively used. While it is certain that collaborative systems will invade

unexpected communication niches — some of which don't even exist today — some areas already appear promising for collaborative assistance.

Expert and Knowledge–based Systems. The difficulty of acquiring and testing knowledge bases has slowed the development of knowledge–based systems. In expert systems the formalization, refinement, distribution, and application of knowledge is usually done by groups. A collaborative system could streamline this process by providing a sequence of tools for the creation and extension of knowledge bases: first, a tool like Cognoter for the initial gathering and organization of information, then an argumentation tool, perhaps like Argnoter (see chapter four), for considering inconsistencies in the data, disputes, and competing proposals, and finally tools to assist in formal knowledge representation and testing. The design of a such a suite of tools would provide insight into the creation and evolution of community knowledge bases. This approach would also help to uncover portions of the knowledge gathering and testing process that lend themselves to automation.

Group Authoring. A similar sequence of tools would be appropriate for a community authoring system. After future Cognoter (and other tools) sessions have shed more light on the natural organization of the group writing process, a more finely targeted suite of tools can be developed. These tools would be used to see how people use the available tools to see the developing structure of their collective argument, and the relation of initial tool design assumptions to the actual uses that people make of the tools at hand. Once such a system is understood it may be extended to community authoring and used with other larger–scale tools for checking, librarianship, and usage charges or analysis.

Games and Pedagogy. It is easy to envision instructional uses of collaborative systems. Real–time multi–player computer games already exist and could be implemented using the underpinnings of Colab. Especially relevant and useful might be a combination of instruction and play: cooperative games for future training or entertainment applications. (Just calling something a game can make it appear to be more fun and interesting.)

Colab in its present form will be less appropriate for competitive situations since it is possible to maliciously misuse the system to the detriment of other participants — for instance, by intentionally editing or deleting an already busy node (or test score or spaceship).

Bootstrapping a Knowledge Medium. Books and other passive media can store knowledge. Expert systems, in the ideal, not only store, but also apply knowledge. Collaborative systems are somewhere in between, where knowledge processing is done mostly by people. There is a lot of space to explore between the printed page and Artificial Intelligence; and there are many opportunities to establish human–machine partnerships and for automating tasks incrementally [Stefik86].

Artificial and other Intelligence. It will likely prove desirable to extend the notion of a *session participant* to include non–human intelligent agents, like special–purpose programs. Initially these would be simple assistants, like automatic spelling checkers, attached to conversations. Later, more sophisticated participants, like argument structure analysts (that do things like detect conflicting assumptions) or other inferential subsystems, may be introduced.

Tele–Colab. A Tele–Colab, a version of Colab supporting non–face–to–face but still real–time interactions, would call for extensive redesign. While such an extension seems simple there are crucial differences. A Tele–Colab would need to support or simulate new concerns, such as "presence". The loss of face–to–face interaction would diminish the effectiveness of social locking and group planning (though both would still be largely possible with an ordinary voice link).

Contributions of this Thesis

This work is a beginning in the understanding of why collaborative problem–solving is organized as it is, how traditional practices relate to computer technology, and what the trade–offs are between supporting old procedures and supplying innovative techniques for collaborative systems.

In a nutshell the contributions of this thesis are:

- The exploration of the notion of a *multi-user interface*.
- The delineation of design principles and features for computer-based tools for cooperative work, including: support of simultaneous activity, busy signals, and WYSIWIS.
- The exploration of problem-solving structures and techniques for collaborative systems, including: process phasing, mutual protocols, and social coordination.
- The design and implementation of an illustrative collaborative system tool (Cognoter) and the organization of the system objects in a general purpose collaborative system (Colab).
- An initial experimental design for evaluation of collaborative system structures and tools.

Pragmatic techniques for programming of multi-user interfaces and complexity management were also discussed. In particular: *ActiveRegions*, a modularization of mouse sensitivity over screen areas; *Associations*, an abstraction of shared models expressed in replicated databases, *BroadcastMethods*, an abstraction of the broadcast communication protocols; and the disciplined partitioning of method function into *user-actions*, *semantic-actions*, and *display-actions*.

Appendix A

Glossary

Argnoter — A Colab tool for meetings in which proposals are prepared and evaluated.

Association — The set of representations on multiple machines that stand for the same object in a shared database.

Broadcast method — An object–oriented programming abstraction that extends the concept of method from a single machine to all the machines in a conversation. When a message invokes a broadcast method for an object on one machine, that method is automatically invoked on the associates of that object on other machines in the conversation.

Busy signal — A visual signal in a display that indicates when particular items are being modified by other participants.

Class — A template for object data–structures with attached data manipulating procedures.

C–graph — A decorated directed graph. A C–graph traversed in preorder, indenting at each level, produces an outline.

Cognoter — A Colab tool for meetings in which a presentation is prepared.

Collaborative system — A real–time computer–based cooperative work environment.

Conversation	A collection of machines, Colab tools, and participants working together on a problem.
Deixis	Referring to something. The reference can be done verbally ("e.g., the gray house across the street" or "that house") or by pointing.
Electern	A podium–like device in the Colab that provides a keyboard for a participant at the liveboard.
Group computer	See *collaborative system.*
Instance variable	A field in the object record or a slot in the data type.
Instantiation	An instance of a data structure described by a class: the actual data storage.
Liveboard	An electronic chalkboard. The Colab liveboard uses a high resolution video projector focused on the back side of a frosted–glass screen. It is also touch sensitive.
Message	An operation to do.
Method	An actual procedure.
Multi–user interface	A human–machine interface used by several people sharing information in a meeting.
Mutual protocol	The articulation, by members of a problem–solving group, of the problem–solving process they are engaged in.
Object	See *Instance*.
O–tree	A decorated tree. An O–tree traversed in preorder, indenting at each level, produces an outline. A subset of *C–graphs*.
Public window	An interactive window in a computer display that is accessible to the entire group in a meeting. Public windows usually adhere to some version of WYSIWIS.

Voice lock — A "social constraint" lock, where a user declares aloud that he is doing an operation on something and expects everyone else to leave it alone.

WYSIWIS — "What You See Is What I See". Strict WYSIWIS means that all meeting participants see exactly the same thing, and can see where the others are pointing.

WYSIWYG — "What You See Is What You Get". Used mainly to describe editors. It often refers to systems that let users see (a reasonable facsimile of) what their final output will look like.

Appendix B

Existing Multiple User Systems

In other — largely non–electronic and non–computer — domains there already exist many communication and distribution systems that are intended for multiple users. For instance, the world telephone system is a communication medium that provides an interface between two or more people. World postal services support data (document) broadcast to subsets of users and a package–switching network. There are many others.

Tables B.1 and B.2 compare several communication media along several dimensions. Most of the user interfaces described below support sequential actions, that is the users interact with the system or communication medium by turn–taking of some kind (explicit or implicit). Electronic mail is a good example of a sequential system even though more than one person may be active at any moment. There is often no exclusion of parallel action, as is the case with electronic mail, but no provision for it either.

Communication Dimensions

Media	Cycle Time	Participants	Simultaneously Active	Length of Interaction
Conference	sec/wks	10–1000	1–10	hr/days
Meeting	0	2–20	1 +	min/hrs
TeleConference	secs	2–200	1	min/hrs
InterOffice Memo	hr/days	2–20	1 +	days
Telephone	0	2	1	mins
Newspaper	days	1–Ms	1 +	wks
Electronic Mail	min/hrs	2–100	1 +	min/days
Postal Mail	days	2 +	0 +	wks
Blackboard	secs	1–30	1	mins
Voting Machine	hrs	100–Ms	10 +	mins
Television	0	any	0 +	min/hrs
QUBE–TV	sec/mins	10–Ks	1 +	min/hrs
StockExchange	sec/mins	10–Ks	10 +	min/hrs
Popular Magazine	wk/mos	Ks–Ms	1 +	mos
Professional Journal	mo/yrs	100–Ks	1 +	mo/yrs
CB Radio	secs	1–100	1	mins
Delphi	wks	5–50	0 +	wk/mos
NoteInBottle	mo/yrs	1–2	0 +	mo/yrs
Colab	secs	1–10	1 +	min/hrs

Table B.1. *Communication media compared on several dimensions.* *Cycle Time* is the time for a "round trip" interaction. *Participants* is the approximate number of participants that may be passively or actively involved in the interaction. *Simultaneously Active* is the number of participants that may be active at the same time. *Length of Interaction* is the total length of a single communication task.

More Communication Dimensions

Communication Media	Branching Ratio	Synchronicity	Setup Time	Transfer Media
Conference	100	both	mos	Verbal/Text/Personal
Meeting	10	seq	min/days	Verbal/Personal
TeleConference	10	seq	hr/days	Verbal
InterOffice Memo	10	async	min/hrs	Text
Telephone	1	seq	secs	Verbal
Newspaper	Ks	async	hr/days	Text/Graphics
Electronic Mail	10	seq	mins	Text
Postal Mail	1	seq	min/hrs	Text/Graphics
Blackboard	5	seq	sec/mins	Text/Drawings
Voting Machine	?	async	wks	Votes
Television	Ms	seq	sec/wks	Video
QUBE–TV	100	seq	min/days	Video/Votes
StockExchange	1000	both	min/wks	Text/Verbal/Money
Popular Magazine	Ms	seq	day/wks	Text/Graphics
Professional Journal	Ks	both	wks	Text/Graphics
CB Radio	10	seq	sec/mins	Verbal
Delphi	10	async	wks	Text/Graphics
NoteInBottle	1	seq	min/yrs	Text
Colab	5	both	mins	Text/Graphics/ Verbal/Personal

Table B.2. Various communication media compared on several more dimensions. ***Branching Ratio*** is the number of participants a single communicatione vent is likely to reach. ***Synchronicity*** is whether the medium is primarily sequential or asynchronous in character. ***Setup Time*** is the time to set up a communication session. ***Transfer Medium*** is the form in which most information is transferred.

Appendix C

A Cognoter Users' Guide

Cognoter is a Colab tool for creating a presentation path — an ordered hierarchical consensical outline for something like a talk or a paper. Cognoter is designed for use by a small group of collaborators each on a personal workstation. It can also be used by a single person.

Cognoter embodies a theory of effective problem solving. It encourages three phases in the presentation building process: Brainstorming, Ordering, and Evaluation. You can pretty much ignore the phases, but if you play the game we claim you will generate a higher quality outline with less effort.

Getting Started:

Try to reserve a time slot in the Colab meeting room — it is a pleasant environment and is the only room with a liveboard. If this is impossible, try to get a machine for everyone in the same room. Cognoter is also usable by geographically separated people with a telephone link, but less effectively.

If anyone doesn't see the Colab Icon on their screen (it looks like the one at the start of this guide), then they probably don't have the proper files loaded and need to find somebody who knows how to load them. If the Icon appears on all screens then you're ready to go... almost.

First, decide on the topic and goal of the presentation you're trying to build.

OK, you know what the presentation is about and are ready to build it. One person should button the Colab Icon with the middle mouse button — this is quick–starting. A menu will appear offering the available Colab tools, pick Cognoter. In a moment another menu will appear giving you the chance to add participants to the conversation. When you add "other" machines, you can supply either the machine

name or the machine net number. The participant menu will keep reappearing until you button "done". In a few seconds the system will have everyone registered and ready to go.

1. Brainstorming:

Cognoter begins in the Brainstorming phase. This phase is intended for uncritical idea generation. The idea here is for everyone to put down as many ideas as possible, bad ones, goods ones, specific ones, vague ones. *Ideas* are referred to as ideas, idea nodes, nodes, and items.

Button functions:

There are more–or–less adhered to mouse button semantics: Any button on the window title gives you large scale operations, Right button affects the surrounding window, Middle button adds or changes things, and Left button lets you see things. Shift, Control, and Meta fine tune these operations.

Right button:

Anywhere in a window gives you a general window menu with functions like shaping the window, clearing the window, moving the window, etc.

Middle button:

On the window title bar gives you a menu of high–level Cognoter operations.

NextPhase moves the whole session into the next phase (don't do this for a while).

Display redisplays the contents of a window (in case it somehow got messed up).

Redisplay is the same as Display above.

RedisplayEveryone will redisplay the window on every machine participating in the conversation.

Shape reshapes the window across associations.

Spread increases the size of the window and proportionally spreads all the items.

Scrunch is like spread except everything is crammed into a smaller window.

ShapeToFit will shape the window to exactly fit around all the items without moving them.

Fonts changes the item fonts to your choice of:

BIGFONTS.

DefaultFonts.

littlefonts.

AddItem will prompt you to add a new item (Idea node) into the main window (there's a faster way to do this, see below).

AddNewRelation will prompt you to add a new relation item (Idea node) into the main window. At the moment, a relation item is simply an item in a lighter font.

Help may give you help.

Accelerator: On empty window background gives you the accelerated way to add items promised above. Just button over the window background and type in the label for your idea (end with *return*). If you mistakenly get a prompt for a label a *return* all by itself will tell Cognoter forget about this label.

On an item gives you a menu of ways to change the item.

Edit lets you edit some aspect of the item (Defaults to attached text) in a TEdit window.

EditLabel lets you edit the node label in a TEdit window.

EditText lets you add/edit text attached to the node (this text is not normally visible). Attached text is the place to amplify idea nodes.

Copy makes a copy of the item and prompts you for a place to put it.

Move does what you expect, it lets you move the item wherever you wish.

Left button:

On the window title bar gives you the same menu of high-level Cognoter operations described above.

On an Item gives you a menu with two options for publicly displaying the text attached to the item.

Show displays the attached text (it doesn't touch subgroup windows) and display the text attached to the item.

ShowFirst will close any existing public displays of attached text (it doesn't touch subgroup windows) and display the text attached to the item.

ShowAnother displays attached text in another public display window, leaving the others alone.

Accelerator: On an Item with the control key down at the same time locks the item to the cursor allowing you to move it until the button is released.

2. Ordering:

The second phase in Cognoter is the ordering phase. You should enter this phase when brainstorming seems to be drying up and there a consensus develops for adding more structure to the ideas. In this phase you are trying to group similar ideas together hierarchically ("these things are all really this") and add temporal links between items ("this should be persented before that").

Additional Button functions:

Menus and button functions in this phase are the same as in the Brainstorming phase with the following additional operations.

Middle button:

On the title bar you have all the menu options from the Brainstorming phase plus :

GroupItems groups items chosen by buttoning them. When you are done choosing items, click on the window background and you will be prompted for the name of the new group.

Ambiguities highlights ambiguous orderings among the items.

NoAmbiguities stops the ambiguity highlighting.

On an Item you have all the menu options from the Brainstorming phase plus :

Open shows the items contained in a group item.

Link establishes link between the item the menu came up on and the next item you choose with the Left Button.

Unlink undoes what ComesBefore does. It erases the dependency link between items.

Convert changes the type of an item (self explanatory).

ConvertToGroup

ConvertToItem

ConvertToRelation

Left button:

On the window title bar gives you the same menu described above.

On an item gives you the same menu described above.

3. Evaluating:

The third Cognoter phase is the Evaluation phase. You should enter this phase when items look to be essentially ordered and the work needs to be clarified and cleaned up. Only in this phase can ideas be deleted.

Button function changes:

Menus and button functions in this phase are almost the same as in the Ordering phase with the following changes and additions.

Middle button:

On the title bar the menu loses the NewPhase option and gains the Outline option. The submenu for Help also gains SeeDeletes as an option.

SeeDeletes will open the normally closed Deleted Items window so you can examine or rescue things previously deleted.

Outline will display the item graph you've built up as an outline in a separate window. Outline has a roll-out option.

TextToo displays the outline and any attached text.

On an Item you are given one more option:

DeleteItem lets you do what it says. Deleted items are not really completely gone — they're in the (usually) hidden Deleted Items window.

Left button:

On the window title bar gives you the same menu described above.

On an item gives you the same menu described above.

Adding new people:

New people can be added to an ongoing session by using the Left button on the Colab Icon and choosing **ChangeCollaborators**.

How to leave a Conversation:

Choose **Quit** from the Colab Icon Left button menu.

Saving the state of the Conversation:

First make sure you are connected to the directory you want to save the conversation in (the snapshotter isn't very smart at the moment). Having done that, choose

PutSnapShot from the ColabIcon Left button menu. You will be prompted for a file name. It will take a few minutes to save the snapshot if you've done very much work.

Restoring a saved Conversation:

First connect to the directory where the saved state is. Then choose **GetSnapShot** from the Colab Icon menu. Once the conversation is restored, you can add other people in using **ChangeCollaborators**.

Appendix D

Experimental Paraphernalia

(Example Fake Telegrams)

—Important Colab Telegram—

Ben Zorn and Margaret Butler
UC BERKELEY

BEN AND MARGARET:
HELP STOP NEED OUTLINE FOR ARTICLE ON **GAMES AND SPORTS IN SOCIETY** SOONEST STOP STAFF STRUCK STUPID STOP DOUBLE USUAL TERMS STOP

ATLANTIC MONTHLY EDITORS

—Important Colab Telegram—

Ben Zorn and Margaret Butler
UC BERKELEY

BEN AND MARGARET:
AFTER RESOUNDING ATLANTIC SUCCESS NEED OUTLINE FOR ARTICLE ON **THE STRATEGIC DEFENSE INITIATIVE** ASAP STOP STAFF IN AWE STOP TRIPLE USUAL TERMS STOP

EDITORS, HARPERS

(Reference Card Given to Participants)

Cognoter Reference

Select = LeftButton

Menu = MiddleButton

TitleBar menu for tool operations

Item menu for item operations

---

accelerators

New Item = MiddleButton (on window background)

Move Item = CTRL-LeftButton

(Background Form)

Group UID:

Number in group:

Previously worked together:

never some often

Computer literacy:

novice some expert

Previous experience with tool:

none some expert

First test:

tool:

topic:

Comments:

Second test:

tool:

topic:

Comments:

Overall Comments:

(Debriefing Form)

Debriefing

Did Cognoter help or hinder?

Did you feel that the phasing (Brainstorming, Ordering, Evaluating) organized your approach? Or did it only get in the way?

Did your team work in parallel during the Cognoter session?

During the Paper session?

Which session did you like better?

Which session was more effective?

References

[Axelrod81] Robert Axelrod, William N. Hamilton, The Evolution of Cooperation. *Science*, **211** 1390–96 (27 March 1981).

[Bernstein81] P.A. Bernstein, N. Goodman, Concurrency Control in Distributed Database Systems. *Computing Surveys*, **13:2**, (June 1981).

[Birrell83] A. D. Birrell and B. J. Nelson, *Implementing Remote Procedure Calls*, Technical Note CSL–83–7 Xerox Corporation (December 1983).

[Bobrow83] Daniel G. Bobrow and Mark J. Stefik, *The Loops Manual*, Xerox Corporation (1983).

[Bobrow86] Daniel G. Bobrow and Mark J. Stefik, Perspectives on Artificial Intelligence Programming. *Science*, **231** 951-956 (28 February 1986).

[Bobrow72] Bobrow, D.G., Burchfiel, J.D., Murphy D.L., Tomlinson, R.S., TENEX, a paged time sharing system for the PDP–10. *Communications of the ACM*, **15:3** (1972).

[Bolt80] R. A. Bolt, Put–That–There: Voice and Gesture at the Graphics Interface, *Computer Graphics, Proceedings of ACM SIGGRAPH '80*, **14:3** 262–270 (1980).

[Bransford85] Bransford, John D., Stein, Barry S., Arbitman–Smith, Ruth, Vye, Nancy J., Improving Thinking and Learning Skills: Analysis of Three Approaches, *Thinking and Learning Skills* (Judith W. Segal, Susan F. Chipman, and Robert Glaser, eds), Hillsdale, New Jersey: Lawrence Erlbaum Associates (1985).

[Brown83] John Seely Brown, Process versus Product. In *Report from the Learning Lab: Education in the Electronic Age* (S. Newman & E. Poor, Eds.). New York: WNET Educational Broadcasting Corporation.

[Bush45] Vannevar Bush, As We May Think, *Atlantic Monthly*, **176:1** 101–108, (July 1945).

[Coffman71] Coffman, E.G. Elphick, M.J., Shoshani, A., System Deadlocks. *Computing Surveys*. **3:2** (June 1971).

[DeBono70] Edward DeBono, *Lateral Thinking*. New York: Harper & Row (1970).

[DeKoven86] Bernard DeKoven, Personal Communication (1 April 1986).

[Engelbart84] Engelbart, D.C., *Collaboration Support Provisions in AUGMENT*, OAC '84 Digest, Proc. of the 1984 AFIPS Office Automation Conf., Los Angeles, California, (February 1984).

[Engelbart68] Engelbart, D.C. and English, W.K., *Research Center for Augmenting Human Intellect*, Proc. Fall Joint Computing Cong., AFIPS press, 395–410, (December 1968).

[Flower80] Flower, Linda S., Hayes, John R., The Dynamics of Composing: Making Plans and Juggling Constraints, *Cognitive Processes in Writing* (Lee W. Gregg, Erwin R. Steinberg, eds), Hillsdale, New Jersey: Lawrence Erlbaum Associates (1980).

[Forsdick85] Harry C. Forsdick, Explorations into Real–time Multimedia Conferencing, *Proc. Second International Symposium on Computer Message Systems* (September 1985).

[Gerrold73] David Gerrold, *The World of Startrek*. New York: Ballantine Books (1973).

[Glegg69] G. L. Glegg, *The Design of Design*. Cambridge: Cambridge University Press (1969).

[Goldberg83] Goldberg, A. and Robson, D., *Smalltalk–80: The language and its implementation*. Reading, Mass: Addison–Wesley (1983) .

[Greif86] Irene Greif, Robert Seliger, William Weihl, Atomic Data Abstractions in a Distributed Collaborative Editing System, *Proceedings of POPL 86* (1986) .

[Halasz82] Frank Halasz and Thomas P. Moran, Analogy Considered Harmful, *Conference on Human Factors in Computer Systems* (March 1982).

[Halasz86] Frank Halasz, Randall Trigg, Tom Moran, *Notecards Release 1.2 Reference Manual*, XSIS, Pasadena, California (1986).

[Hansen73] Hansen, P.B., *Operating System Principles,* Englewood Cliffs, New Jersey: Prentice–Hall (1973).

[Hayes80] Hayes, John R., Flower, Linda S., Identifying the Organization of Writing Processes, *Cognitive Processes in Writing* (Lee W. Gregg, Erwin R. Steinberg, eds), Hillsdale, New Jersey: Lawrence Erlbaum Associates (1980).

[Henderson86] D. Austin Henderson, Jr. and Stuart K. Card, Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window–based Graphical User Interface, Xerox Palo Alto Research Center (1986).

[Hiltz81] Hiltz, S.R. and Turoff, M., The Evolution of User Behavior in a Computerized Conferencing System, *Communications of the ACM*, **24:11** (November 1981).

[Hiltz78] Hiltz, S.R., Turoff, M., *The Network Nation: human communication via computer.* London: Addison–Wesley (1978).

[Johansen84] Johansen, R., Teleconferencing and Beyond: communications in the office of the future. New York: McGraw–Hill (1984).

[Johansen79] Johansen, R., Vallee, J., Spangler, K., *Electronic Meetings: technical alternatives and social choices*. Reading, Massachusetts: Addison–Wesley (1979).

[Kerr82] E. Kerr and S. R. Hiltz, *Computer–Mediated Communication Systems*, New York: Academic Press (1982).

[Kraemer83] K. L. Kraemer, J. L. King, *Computer Supported Conference Rooms: final report of a state of the art study*. (unpublished draft, Department of Information and Computer Science at the University of California, Irvine.) (December 1983).

[Kung81] H. T. Kung and John T. Robinson, *On Optimistic Methods for Concurrency Control*. ACM Transactions on Database Systems, **6:2** 212-226 (June 1981).

[Lederberg78] Joshua Lederberg, Digital Communications and the Conduct of Science: The New Literacy, *Proceedings of the IEEE*, **66:11** (November 1978).

[Licklider60] J. C. R. Licklider, Man–Computer Symbiosis, *IRE Transactions on Human Factors in Electronics* (March 1960).

[Licklider68] J. C. R. Licklider, Robert Taylor, and Evan Herbert, The Computer as a Communication Device, Science & Technology, 21–31 (Apr 1968).

[Licklider78] J. C. Licklider and A. Vezza, Applications of Information Networks. *Proceedings of the IEEE*, **66:11** 1330-1346 (1978).

[Linstone75] Linstone, H.A. and Turoff, M., *The Delphi Method: techniques and applications*, Addison–Wesley, Reading, Mass. (1975)

[Liskov85] Liskov, B., The Argus Language and System, in *Distributed Systems: Methods and Tools for Specification; An Advanced Course, Lecture Notes in Computer Science*, Volume 190 [Alford et al, eds.], pp. 343–430, Springer–Verlag, Berlin (1985).

[Liskov86] Barbara Liskov, Maurice Herlihy, and Lucy Gilbert, Limitations of Synchronous Communication with Static Process Structure in Languages for Distributed Computing, *Proceedings of POPL 86* (1986).

[Malone82] Thomas W. Malone, Heuristics for Designing Enjoyable User Interfaces, *Conference on Human Factors in Computer Systems* (March 1982).

[Merriam69] G. & C. Merriam, *Webster's Seventh New Collegiate Dictionary*. Springfield, Mass.: G. & C. Merriam Company (1969).

[Metcalfe76] Metcalfe, R.M. and Boggs, D.R., Ethernet: distributed packet switching for local computer networks, *Communications of the ACM*, **19** 395–404 (July 1976).

[BNelson81] Bruce J. Nelson, *Remote Procedure Call*, CSL–81–9, Xerox Palo Alto Research Center (May 1981).

[TNelson81] Ted Nelson, *Literary Machines*, Swarthmore, Pennsylvania: Theodore Nelson (1981).

[O'Connor84] Rory J. O'Connor, Outline Processors Catch On, *InfoWorld*, 30–31 (2 July 1984).
A survey of ThinkTank, FreeStyle and other commercial "thought processors".

[Palme84] Jacob Palme, Survey of Computer–based Message Systems. *Interact '84: IFIP Conference on Human–Computer Interaction* (September 1984).

[Parnes77] Robert Parnes, Chris H. Hench, and Karl Zinn, Organizing a Computer–based Conference, *Transmat. Associations (J. Union of Intern. Associations)*, **10** 418-422, Univertity of Michigan (1977).

[Polya57] G. Polya, *How To Solve It, a New Aspect of Mathematical Method*. Garden City, New Jersey, Doubleday, Anchor Books (1957).

[Rice84] Ronald E. Rice and Associates, *The New Media: Communication, Research, and Technology*. Beverly Hills, California: Sage (1984).

[Sanella83] M. Sanella et al, *Interlisp Reference Manual*, Xerox Corporation (1983).

[Sarin84] Sunil K. Sarin, *Interactive On-Line Conferences*, PhD. thesis (MIT), MIT/LCS/TR-330 (December 1984).

[Sarin84a] Sunil K. Sarin and Irene Greif, Software for Interactive On-Line Conferences, *Proceedings ACM-SIGOA Conference on Office Information Systems*, Toronto, Canada, (25-27 June 1984).

[Sarin85] Sunil K. Sarin and Irene Greif, Computer-Based Real-Time Conferencing Systems, *Computer*, **18:10** 33-45 (October 1985).

[Sheil83] Beau Sheil, Power Tools for Programmers. *Datamation* (February 1983).

[Shneiderman83] B. Shneiderman, Direct Manipulation: A Step Beyond Programming. *IEEE Computer* (August 1983).

[Stefik84] Mark Stefik, How to Solve It, Collectively. Working paper (1984).

[Stefik86] Mark Stefik, The Next Knowledge Medium. *AI Magazine*, **7:1** 34-46 (Spring 1986).

[Stefik86a] Mark Stefik, Daniel G. Bobrow, Stanley Lanning, Deborah Tatar, Gregg Foster, WYSIWIS Revised: Early Experiences with Multi-user Interfaces. *Proceedings of the Conference on Computer-Supported Work*, Austin Texas (December 1986).

[Stefik87] Mark Stefik, Gregg Foster, Daniel G. Bobrow, Kenneth Kahn, Stanley Lanning, Lucy Suchman, Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. To appear in *Communications of the ACM* (January 1987).

[Thomas83] Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, George G. Robertson, Richard W. Schaaf, Raymond S. Tomlinson, Virginia M. Travers, Diamond: A Multimedia Message System Built Upon a Distributed Architecture. *Technical Report,* Bolt Beranek and Newman, Inc. (July 1983).

[Thompson84] Henry Thompson, RPC users guide. Manuscript, Xerox Palo Alto Research Center (1984).

[Trigg83] Randall Trigg, *Network Approach to Text Handling for the Online Scientific Community,* Ph.D. thesis, Department of Computer Science, University of Maryland, CSTR–1346 (November 1983).

[Turoff72] Murray Turoff, *Delphi Conferencing: computer–based conferencing with anonymity.* Technological Forecasting and Social Change, **3** 159–204 (1972).

[Vallee76] Jacques Vallee, Robert Johansen, H. Lipinski, T. Wilson. Pragmatics and Dynamics of Computer Conferencing: A Summary of Findings from the FORUM Project," *Proceedings of the 3rd International Conference on Computer Communication* (P. Verma, Ed.), 203-213 (1976).

[vonOech83] Roger von Oech, Ph. D., *A Whack on the Side of the Head.* Warner Books (1983).