Edgar H. Sibley
Panel Editor

Although individual use of computers is fairly widespread, in meetings we tend to leave them behind. At Xerox PARC, an experimental meeting room called the Colab has been created to study computer support of collaborative problem solving in face-to-face meetings. The long-term goal is to understand how to build computer tools to make meetings more effective.

BEYOND THE CHALKBOARD: COMPUTER SUPPORT FOR COLLABORATION AND PROBLEM SOLVING IN MEETINGS

MARK STEFIK, GREGG FOSTER, DANIEL G. BOBROW, KENNETH KAHN, STAN LANNING, and LUCY SUCHMAN

Meetings are used for virtually any intellectual task that requires the coordination or agreement of several people. Statistical studies suggest that office workers spend as much as 30–70 percent of their time in meetings [26]. Paradoxically, even with the widespread distribution of computers, most computer systems in use aid the work of separate individuals rather than their work in groups. In meetings, computers are typically left behind in favor of more passive media like chalkboards[1] and flip charts.

Media influence the course of a meeting because they interact strongly with participants' resources for communication and memory. Chalkboards, for example, provide a shared and focused memory for a meeting, allowing flexible placement of text and figures, which complements our human capabilities for manipulating spatial memories. However, space is limited and items disappear when that space is needed for something else, and rearranging items is inconvenient when they must be manually redrawn and then erased. Handwriting on a chalkboard can be illegible. Chalkboards are also unreliable for information storage: They are used in rooms shared by many groups, and text and figures created in one meeting may be erased during the next. If an issue requires several meetings, some other means must be found to save information in the interim.

Many of the functions that are awkward or impossible with chalkboards are implemented easily with computers. Window systems and drawing aids, for example, provide flexibility for rearranging text and figures, and text can be displayed in fonts that are crisp and reproducible. File systems make it possible to retrieve information generated from previous meetings, to revisit old arguments, to show the history of a series of arguments, and to resume discussions. Independent workstations allow meeting participants to share views, point to objects under discussion, and work on different aspects of a problem simultaneously, with the result that participation can feel less like being a member of a committee, and more like acting as a collaborator at a barn raising.

To explore these ideas, an experimental meeting room known as the Colab has been set up at Xerox PARC. In the Colab, computers support collaborative processes in face-to-face meetings. The Colab is de-

[1] The term *chalkboard* in this article refers to any of the wall-mounted erasable writing surfaces commonly used in meeting rooms, whether they are white, black, or some other color and whether the marks are made with chalk, crayon, or ink. We use this term to avoid misunderstandings about the word *blackboard*, which, among other things, can mean a commercially available teleconferencing product, or a programming organization for artificial-intelligence systems. We also avoid the term *whiteboard*, which can mean a white metal writing surface on which colored pens are used, or a specific graphical database tool developed at Xerox PARC [9].

signed for small working groups of two to six persons using personal computers connected over a local-area network (Figure 1). In our design, we have drawn on familiar elements from conventional meeting rooms. The focus of the Colab project is to make our own meetings among computer scientists more effective and to provide an opportunity for conducting more general research on how computer tools affect meeting processes.[2]

Much prior research has focused on the use of computer and communication technology to support teleconferencing [18, 19] and what is known as computer conferencing [16, 17], which emphasizes the use of computers to support asynchronous communication and discussion over a computer network. The Colab, on the other hand, focuses on problem solving in face-to-face meetings—the most common kind of meeting in our research group and our starting point.

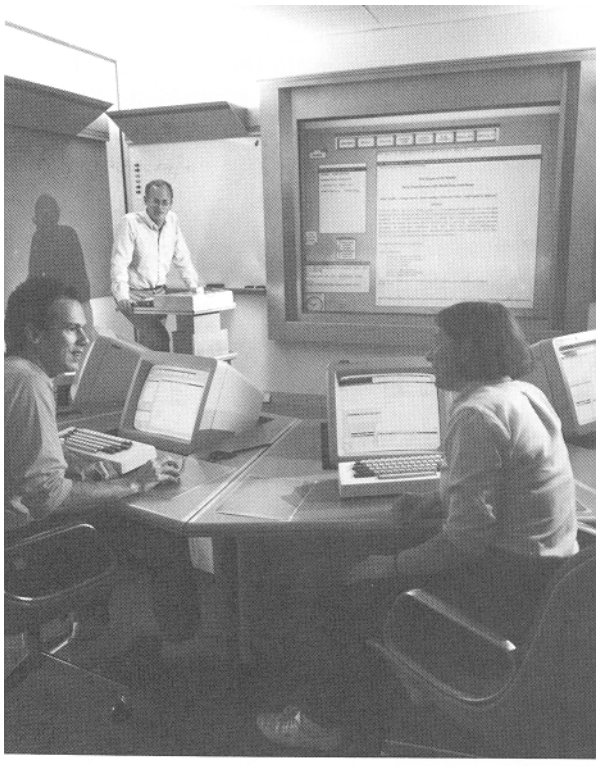
In this article, we describe the meeting tools we have built so far as well as the computational underpinnings and language support we have developed for creating distributed software. Finally, we present some preliminary observations from our first Colab meetings and some of the research questions we are now pursuing.

TOOLS FOR COLLABORATION

An office worker using a computer will choose different programs to achieve different purposes. Completing a single project may involve the use of several different tools: a spreadsheet program, a text editor, and a sketching program. In a similar vein, activities arise in the course of a meeting that require different supporting programs. In this article, we use the term *meeting tools* to refer to programs that support group interaction and problem solving in meetings, and the term *Colab tools* to refer to meeting tools developed specifically for use in the Colab.

A fundamental requirement for meeting tools is that they provide a coordinated interface for all participants. Such a *multiuser* interface is intended to let meeting participants interact with each other easily and immediately through a computer medium.

The term *WYSIWYG* (what you see is what you get) is generally used to describe text editors in which text appears the same during editing as it will during printing. To describe an important abstraction for meeting tools, we have defined an analogous term: *WYSIWIS* (what you see is what I see—pronounced "whizzy whiz"), which refers to the presentation of consistent images of shared information to all participants. A meeting tool is *strictly* WYSIWIS if all meeting participants see exactly the same thing and where the others are pointing.



The Colab is an experimental meeting room designed for typical use by two to six persons. Each person has a workstation connected to a personal computer. The computers are linked together over a local-area network (ethernet) that supports a distributed database. Besides the workstations, the room is equipped with a large touch-sensitive screen and a stand-up keyboard.

FIGURE 1. A View of the Colab

WYSIWIS creates the impression that members of a group are interacting with shared and tangible objects. It extends to a group conversation the kind of shared access to information that is experienced by two people sitting together over a sketch. WYSIWIS is the critical idea that makes possible the sense of teamwork illustrated 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: to "see where their hands are." With meeting tools, this visual cue can be approximated by providing pointers to work in progress and by graying out objects that are being worked on.

Although *strict* WYSIWIS would give everyone the same image on their displays, in practice we have

[2] Lucy Suchman, the last author of the present article, is an anthropologist for whom the Colab represents part of a larger study of face-to-face collaboration and its technology.

found this too limiting and instead use relaxed versions of WYSIWIS [32]. For example, it can be useful to differentiate between *public* interactive windows that are accessible to the entire group, and private windows with limited access (e.g., for personal electronic mail). Private windows violate the concept of strict WYSIWIS, as does relaxation of pointer displays. Although pointing is an efficient way to refer to things in conversation, displaying the cursors of all active participants is usually too distracting. Making pointers visible only on request becomes an effective compromise. Another WYSIWIS relaxation permits public windows to appear at different places on different screens so that public pointers can be translated into window-relative coordinates. This sacrifices some ability to refer to things by screen position, but it does permit personalized screen layouts.

Meetings, like other processes, can be more efficient when several things are done at once. Since Colab tools support simultaneous action, a key issue in tool design is recognizing and supporting those activities that can be decomposed for parallel action. For parallel action, a task must be broken up into appropriately sized operations that can be executed more or less independently by different members of the group. If the operations are too small, they will be too interdependent, and interference will preclude any substantial parallelism. For example, to create a shared text, interactions should not be at the level of individual keystrokes. On the other hand, if operations are needlessly large, opportunities for synergy are lost.

The ability to act in parallel on shared objects also brings with it potential for conflict. Conflict resolution strategies will become necessary in some cases, but often we can rely on social constraints. A conflict detection system or "busy signal" graphically warns users that someone else is already editing or otherwise using an item; a busy item is grayed out on all screens.

Our initial goal was to create tools to support the kinds of meetings that our group has, which range from the informal to the formal. One of the informal meeting tools we have developed, Boardnoter, closely imitates the functionality of a chalkboard (Figure 2). It is intended for informal meetings that rely heavily on informal freestyle sketching. To draw with Boardnoter, one uses the "chalk," to erase one uses the "eraser," to type one uses the miniature "typewriter," and to point one uses the "pointer." To sketch a square with Boardnoter, one simply "picks up the chalk" and makes four strokes. A subsequent version of Boardnoter will go beyond the chalkboard by adding capabilities for copying, moving, resizing, linking with rubber band lines, grouping, and smoothing (neatening), and for using and scaling selections from a set of predrawn images.

Other Colab tools are based on much more formal models of the meeting process. In this article, we focus our attention on two such tools: *Cognoter*, a tool for organizing ideas to plan a presentation; and *Argnoter*, a tool for considering and evaluating alternate proposals. Although both tools are intended to bring appropriate computational support to structured meeting processes, the contrast between the two processes will highlight the range of opportunities that exist for applying computer technology in this medium.

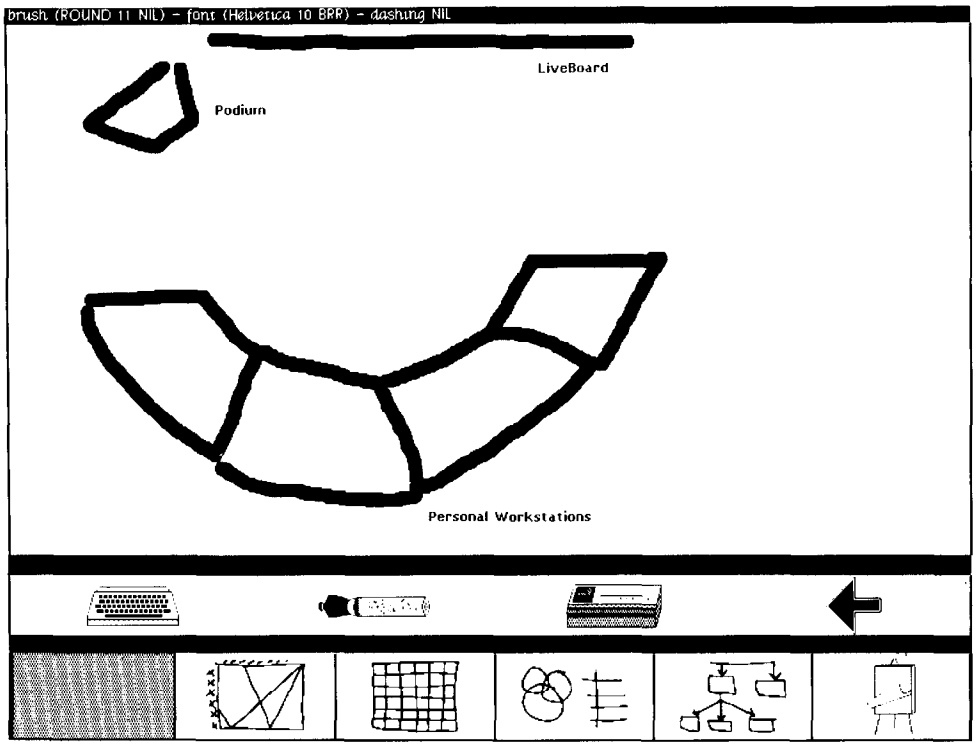
ORGANIZING IDEAS FOR A PRESENTATION USING COGNOTER

Cognoter[3] is a Colab tool used to prepare presentations collectively. Its output is an annotated outline of ideas and associated text. We have used Cognoter to prepare outlines for talks and papers, including this one. In some ways, it is similar to the ThinkTank, Freestyle [25], and NoteCards [34] programs. All are used to organize ideas, but Cognoter is unique in that it is intended for collective use by a group of people.

The Cognoter process imitates a meeting style for collaborative writing that we have used at Xerox PARC without computational support for several years. Usually, we begin with a clear slate: The ideas are in our heads and nothing is written down. The problem at this point is how to get started: It is not very helpful to begin by asking, "Well, we need an outline. What should we put in I.A.1?" Rather, planning a presentation requires that the group decide what the ideas are, which ideas go together, which ideas come first, the order of presentation, and, finally, which ideas warrant elimination.

Cognoter organizes a meeting into three distinct phases—*brainstorming*, *organizing*, and *evaluation*—each of which emphasizes a different set of activities. As the group advances through the respective phases, the set of possible actions is expanded: For instance, brainstorming, which is emphasized in the first phase, is still possible in the last phase. Groups that find the rigid enforcement of phases too prescriptive can skip immediately to the last phase where all the operations are possible. Our intention is to experiment with methods for encouraging particular meeting processes and styles of behavior without making the tools too inflexible and prescriptive.

[3] The name *Cognoter* comes from both *cog-noter* (a cognition noter) or *co-gno-ter* (knowing together).



Note: The actual screen colors for the current version of Colab are black on white; in Figures 2–6, the color green has been added for editorial emphasis—*Ed.*

The Boardnoter meeting tool in the Colab is operational but still in the early stages of development. A key feature is that it provides a large area for freestyle sketching. Below the writing area is a "chalk tray" containing several implements: a piece of chalk, an eraser, a miniature typewriter, and a pointer. To draw on the board, one picks up the chalk by clicking the mouse or pen over the chalk icon; to erase one picks up the eraser; to point one picks up the pointer. Since more than one boardful of information may be needed in the course of a meeting, the "stampsheet" of shrunken stamp-sized boards at the bottom makes it possible to obtain a fresh board or to switch back to a board created earlier.

FIGURE 2. Screen Image of Boardnoter

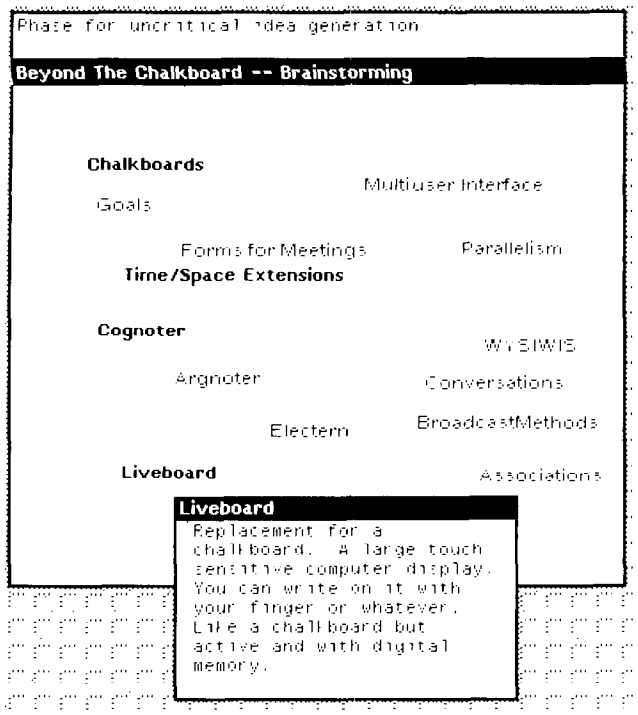
Brainstorming

Since the brainstorming phase involves the initial generation of ideas used in the presentation, it is important to encourage synergy in group interactions and to not interfere with or inhibit the flow of ideas [10]. In Cognoter, therefore, ideas are not evaluated or eliminated in this phase, and little attention is given to their organization (see Figure 3, next page). Instead, there is one basic operation: A participant selects a free space in a public window and types in a catchword or catchphrase characterizing an idea. Participants may act simultaneously, adding idea items and supporting text at any time, but may not delete an item (even their own), although they can move them around. Supporting text is used to clarify the meaning of an item and to establish terminology for the presentation. Once entered, it can be publicly displayed or further edited by any participant. As the window fills up to encompass what appears to be a jumble of ideas on different levels, begging for organization, pressure to move on to the next phase begins to mount.

Organizing

In the organizing phase, the group attempts to establish an order for the ideas generated in the brainstorming phase. With Cognoter, the order of ideas can be established incrementally by using two basic operations: linking ideas into presentation order and grouping ideas into subgroups. In addition, the item-moving operation allows these operations to be discussed prior to actually executing them by moving items near each other before clustering or linking.

The basic operation is to simply assert that one idea should come before another. Linking is usually accompanied by some verbal discussion: For example, a participant may say, "I'm putting *Colab tools* before *open issues* because you need to understand



In the brainstorming phase, participants may add ideas and supporting text. Criticism or deletion of ideas is discouraged. Ideas are entered into the window by clicking the mouse in the background of the window and typing in a short title or phrase that stands for the idea. Text explaining the ideas in more detail is entered by selecting the item with a mouse and then using a text editor in a separate window.

FIGURE 3. Brainstorming with Cognoter

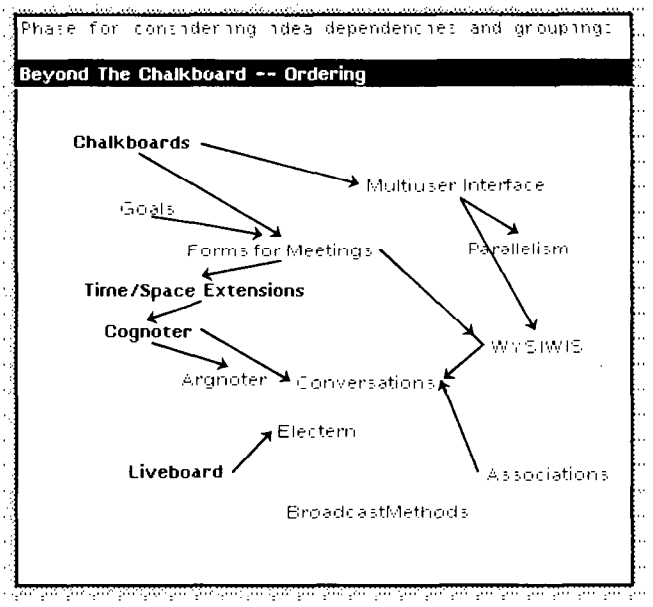
what we have done before you can understand what comes next." The ordering is indicated visually by directed links between items as shown in Figure 4. The meaning of the links is transitive, meaning that, if *X* comes before *Y*, and *Y* comes before *Z*, then *X* must come before *Z*. The links are used collectively to determine a complete order of presentation. Items can also be clustered into groups and moved to their own windows as shown in Figure 5. When a group is formed, a bracketed item standing for the whole group is displayed in the window; the grouped items themselves are displayed in an associated window. Links are distributive across groups; a link to or from a bracketed item is treated like a link to or from the whole group. By these transitive and distributive operations, a small number of explicit links can highly constrain the total order of ideas.

Evaluation

The third phase, evaluation, determines the final form of the presentation. Participants review the overall structure to reorganize ideas, fill in missing details, and eliminate peripheral and irrelevant ideas.

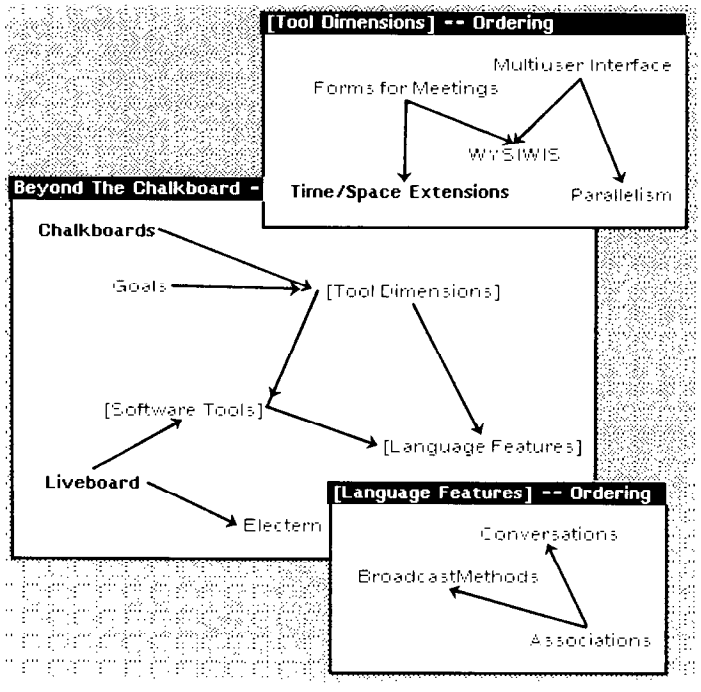
In Cognoter, the various decision-making processes are separate and distinct operations. Delaying deletion until the last phase, for example, provides a more visible basis for argument in the sense that an argument for deleting an idea because it is not relevant may be more convincing when that idea is not visibly linked with any others; or arguing the unimportance of an idea may be more convincing when the competing ideas are available for comparison. In the same sense, an argument that there is an excess of material may be more compelling when all the material can be seen, or a charge that an idea is vague may be more convincing in the presence of other ideas that are more fully substantiated.

Delaying deletion also has some beneficial effects on group dynamics: Deleting an idea during the brainstorming phase could easily be interpreted as criticism and might either inhibit certain participants or provoke tangential argument, whereas arguing that an idea does not fit or is insubstantial in the



In Cognoter, the order of ideas is established incrementally. The basic operation is determining that one idea should come before another, which is indicated visually by directed links between items. The meaning is transitive, meaning that, if *X* comes before *Y*, and *Y* comes before *Z*, then *X* must come before *Z*. Collectively, the links determine the order of idea presentation. Links are added or removed by clicking the mouse on the desired items. Items will usually have one or more links to other items.

FIGURE 4. Establishing the Order of Ideas



Items can be clustered into groups representing ideas that will be worked on together. Each group has an associated window for displaying its items. A group is named when it is formed, and that name appears as a bracketed item in the original window.

FIGURE 5. Grouping Items

evaluation phase may have the beneficial effect of prodding other group members to clarify or extend the idea.

Other operations besides deletion are also appropriately delayed until the evaluation phase. For example, arguing that an idea is misplaced is more compelling when alternate places to put it are visible; this is a good time to consider the reordering of ideas. Since the linking operation that takes place in the organizing phase is usually based on considerations local to two ideas, seeing the entire presentation, with most of the links in place, allows the user to appraise the overall structure and consider more global concerns, such as balance.

Cognoter provides a systematic process for answering the question, "What should we put in I.A.1?" Starting points for a presentation can be identified systematically: These are the items with no incoming links. Cognoter then helps in the final ordering of ideas by preparing an outline and indicating which ideas are ordered arbitrarily. By traversing the item graph, an outline is generated, with or without the attached text.

In many respects, Cognoter supports a process that is quite different from that underlying tools like ThinkTank. Beyond the most obvious difference, which is that Cognoter is designed for simultaneous use by multiple participants (although the process it embodies is also useful for single users), Cognoter also divides the thinking process into smaller and different kinds of steps that are incremental and efficient. In ThinkTank, ideas are always organized in an outline—there is no other place to put them—whereas Cognoter separates the tasks of idea generation and ordering. Cognoter also provides for incremental ordering through a link-forming operation whereby a partial ordering of ideas is refined stepwise toward a complete ordering. Transitivity and grouping operations make it possible to organize the ideas efficiently with a small number of links.

Some important parts of the presentation planning process are not explicit in Cognoter: For example, Cognoter does not inquire as to the audience, the appropriate technical level, the goals of the paper, or arguments for deleting or ordering ideas. Modifications to Cognoter could make such questions explicit, but they are now outside the scope of the current tool.

Cognoter is the first useful Colab tool developed and is still evolving. We are now experimenting

with various relaxations of the WYSIWIS concept. In the current version of Cognoter, for example, windows showing links and items are public, but outline and item editing windows are private. The absence of visual cues indicating which are public and which private can be confusing for the first-time users. With several months experience using Cognoter's multiuser interface, we are actively exploring trade-offs in the design of the next generation of the tool [32].

AN ARGUMENTATION SPREADSHEET FOR PROPOSALS (ARGNOTER)

Argnoter,[4] the Colab tool being developed for presenting and evaluating proposals, is now in the early stages of design and implementation and is presented here chiefly as a contrast to Cognoter. Implementing and experimenting with Argnoter are now major focuses of the Colab project. As with Cognoter, the basic meeting process supported by Argnoter has been used by our group without computational aid for several years.

Proposal meetings start when one or more members of the group have a proposal for something to be done, typically a design for a program or a plan for a course of research. The goal of the meeting then becomes to pick the best proposal. The proposals are at least partially worked out before the meeting, as opposed to Cognoter meetings, which begin with a blank slate. Since Argnoter participants have already invested some energy in the creation of these proposals, the meetings have a greater potential for dispute and disagreement. Discovering, understanding, and evaluating disagreement are therefore essential parts of informed decision making in these meetings.

In developing a design—which is essentially a dialectic between goals and possibilities—designers usually begin without knowing exactly what is wanted or what is possible. They explore parts of the design space as driven by their current goals, and sharpen their goals as they learn what is possible. In collaborative design tasks, this interaction and tension between goals and alternatives must play itself out in the communications among collaborators. At the beginning, design goals are not necessarily shared; the elaboration of a common set of goals is part of the collaborative process and includes the incremental development and selection of design alternatives.

The intuition guiding the Argnoter process is the recognition that much of the dispute and misunderstanding that arise in meetings about design proposals are due to three major causes: *owned positions*, that is, personal attachment to certain positions; *unstated assumptions*; and *unstated criteria*. Hence, a major theme of Argnoter design is that alternatives be made explicit: Proposals themselves are explicit, as are assumptions and evaluation criteria.

In essence, the Argnoter meeting comprises three distinct phases—*proposing*, *arguing*, and *evaluating*—which in some respects are similar to the respective phases in Cognoter, but different enough to warrant description.

Proposing

In the proposal phase, the proposals are stated explicitly: Each proposal is given a short text description, and perhaps a sketch, and is named according to its features or functions. In Argnoter, a proposal will be created in, and displayed by, a set of connected windows called proposal "forms," which can be either private or public. Public proposal forms are WYSIWIS, whereas a private form appears only on the machine of the participant who controls it. Private forms ensure that every participant can view or create a new proposal without having to share its use. Other windows will allow viewing any of the proposals under consideration in the meeting. New proposals are created by modifying an existing one or combining features from two or more different ones. A new proposal automatically inherits text, sketches, and statements from its parent proposals.

Even with the high-resolution, wide-format displays used in the Colab, space for windows is limited: A proposal displayed with its text, sketch, and arguments occupies about one-fourth of the screen. The default configuration allows enough viewing space for two public proposal forms, one private form, and a variety of other forms. However, displays of the kind available on most personal computers would be inadequate for viewing even a single proposal and would not work well for most Colab tools.

Arguing

The next phase consists of presenting reasons for choosing or not choosing individual proposals. Reasons must be written down. On the chalkboard, the reasons are written as statements underneath the respective proposals. Each statement is identified as either pro or con and consists of a short text description like "very expensive" or "can't be done in less than six months." The structure of Argnoter encourages participants to write pro and con statements about all proposals, not just pro statements for the ones they are in favor of and con statements for the rest. Since the pro and con statements are there for all to see and contemplate, participants tend to take the time to formulate them carefully. Insubstantial statements like "I just don't like proposal *X*" will carry less weight than ones that are specific and focused.

[4] The name *Argnoter* is intended to suggest *argument noter*, that is, a tool to help organize and evaluate arguments.

This shared use of a chalkboard to present proposals and arguments has been used habitually and successfully by other groups that we know about. The following anecdote about another laboratory illustrates this:

> On any given morning at the Laboratory of Molecular Biology in Cambridge, England, the blackboard of Francis Crick or Sidney Brenner will commonly be found covered with logical trees. On the top line will be the hot new result just up from the laboratory or just in by letter or rumor. On the next line will be two or three alternative explanations, or a little list of "what he did wrong". Underneath will be a series of suggested experiments or controls that can reduce the number of possibilities. And so on. The tree grows during the day as one man or another comes in and argues about why one of the experiments wouldn't work, or how it should be changed. [27]

For comparative purposes, it is possible in the argument phase to categorize pro or con statements across proposals in terms of categories like compatibility, cost, development time, efficiency, feasibility, simplicity, and utility. With computational support, it is possible to automatically create auxiliary tables that compare proposals on the basis of these categories.

In the argument stage, participants can add statements or modify existing proposals. This tends to foster a synergy among ideas, joint contributions to proposals and reasons, and the systematic development of parallel reasoning across proposals. According to Platt [27], this kind of group participation in the articulation of *multiple* proposals and arguments often leads to a very productive decision-making process:

> The conflict and exclusion of alternatives that is necessary for sharp inductive inference has been all too often a conflict between men, each with his single Ruling Theory. But whenever each man begins to have multiple working hypotheses, it becomes purely a conflict between ideas. . . . In fact, when there are multiple hypotheses which are not anyone's "personal property" and when there are crucial experiments to test them, the daily life in the laboratory takes on an interest and excitement it never had, and the students can hardly wait to get to work to see how the detective story will come out.

The articulation of multiple proposals and their arguments leads naturally into the next phase—evaluation—in the sense that proposals are being evaluated *indirectly* by analyzing the reasons behind them. Moreover, this articulation encourages a style of decision making that separates arguments about evaluation criteria from arguments about the proposals themselves.

Evaluating

First, the evaluation considers the assumptions behind individual arguments. Assumptions in Argnoter are expressed as statements about statements: For example, the statement "this assumes that labor costs can be ignored" could refer to the statement "this proposal is inexpensive." Whereas historically we might have written such assumptions on the chalkboard next to the corresponding arguments, with Argnoter, we will ultimately provide facilities for viewing the structure of arguments in terms of the connections between these statements.

Meeting participants often disagree about the validity of statements: One person might believe that "sixteen million bit memory chips will be readily available in six months" and another may not. In Argnoter, we will try to model these differences with explicit "belief sets," a belief set being a mapping of a set of statements into valid (believed) or invalid (not believed) categories. This kind of modeling is something that cannot effectively be done on chalkboards.

The act of making belief sets explicit enables Argnoter to act as a kind of *argumentation spreadsheet* where a proposal is viewed and evaluated in relation to a specified set of beliefs. The proposal display is generated by stepping through the arguments about the proposal, looking up the assumptions, and then displaying those arguments that are supported in the specified belief set. Multiple belief sets may coexist, and any participant is able to create (or specialize) belief sets. The belief sets are intended to characterize different generic points of view (e.g., liberal versus conservative, marketing versus development).

Just as a numerical spreadsheet program provides a way of exploring entailments of hypothetical numerical relationships, an argumentation spreadsheet like Argnoter provides a way of exploring belief entailments. A numerical spreadsheet program provides no in-depth understanding of the meanings of interest rate, tax rate, or monthly income, but it does compute the necessary sums and display changes in the derived values when the input values are changed. In the same way, Argnoter need not understand the meanings of design proposals: It need only differentiate between proposals, arguments, assumptions, and belief sets, and compute the relevant logical support relationships. One should be able to change a belief assignment and then immediately see the relevant changes in the proposal display. Differences in point of view can also be highlighted (e.g., by displaying a proposal under different belief sets). Other evaluations, like sensitivity analyses, can be done using the same information.

Next, evaluation criteria are selected and ranked.

The values of specific criteria are often ranked differently by different participants: Feasibility, for example, is usually considered important, but there may be disagreements about trade-offs between cost versus utility or space versus time.

Evaluation criteria and beliefs represent different dimensions of the evaluation process. Two participants may agree that cost is a primary criterion, but disagree about whether a specific proposal is expensive; conversely, they might agree on the costs of different proposals, but disagree about the significance of cost as a criterion. Using Argnoter, we can experiment with different ways of ranking criteria and provide mechanisms for viewing proposals according to these rankings.

A major working hypothesis behind the design of Argnoter is that making the structure of arguments explicit facilitates consensus by reducing disagreement that arises from uncommunicated differences. Since participants using Argnoter first agree on criteria and then systematically apply those criteria to proposals, experiments suggest themselves as to whether in fact such behavior actually speeds consensus and to what extent Argnoter actually encourages such behavior.

In the process of making particular kinds of statements explicit and leaving other kinds implicit, Colab tool designers may inadvertently bias the meeting processes. In both Cognoter and Argnoter, the lack of an explicit representation of goals for the meetings may prejudice the discussion at particular times. Designers of Colab tools are therefore necessarily creating more than just tools: They are also designing and enforcing meeting processes. We see the Colab as a working laboratory for increasing our understanding of meeting processes and examining the effects of computational support tools on these processes.

PROGRAMMING ISSUES AND CONCEPTS

Two primary assumptions about the Colab's computing architecture were made: (1) Each meeting participant was to have a personal computer, and all the computers were to be connected together through a local-area network; and (2) all the computers were to run the same software. In retrospect, this approach has been workable and appears reasonable: It is also open-ended to the extent that processors can be added to the network to carry out special functions, and special software can be added to some of the computers.

Programs distributed over several machines are notoriously difficult to write and debug. This, combined with our need to experiment and change code frequently, motivated us to develop programming tools to simplify developing, testing, and revising. In the balance of this section, we describe certain extensions to a programming language and environment we developed for the Colab, and how we have used them.

Colab tools enable people to share and jointly revise information presented in meeting situations. For this discussion, it is useful to think of this information as residing in a computer database that has certain properties. To this end, we leave the granularity of logical data items unspecified, although, in practice, grain size is determined by our intentions about the independence of the data items. The design of the database is a starting point for understanding the issues and programming techniques used in designing and executing Colab tools. The following goals arise naturally out of the Colab application and reflect as well general expectations about the use of personal computers:

- The delay in getting information should be very short. To generate complex information displays, the average retrieval time should be on the order of a few microseconds.
- The delay in changing information should be short. To avoid a feeling of sluggishness, it should usually be possible to change information in a fraction of a second.
- The database should converge quickly to a consistent state.
- The database should not be vulnerable to either the accidental actions of one participant or the failure in one participant's machine.

Maintaining the Database

To maintain the Colab database, we experimented with several control regimes: the centralized model, the centralized-lock model, the cooperative model, the dependency-detection model, and the roving-locks model.

Centralized Model. The straightforward centralized approach, which has been used successfully in other similar applications (e.g., [29, 30]), ensures that all participants use the same data: There is only one copy of the database, and concurrency control is straightforward. To coordinate simultaneous changes, database transaction mechanisms must be used. The centralized model was rejected out of hand for Colab purposes because we could not use these mechanisms to retrieve data fast enough to update our displays. Moreover, given our plan to use networked personal computers, we would incur the additional delay of network communication.

Centralized-Lock Model. The distributed model corrects the slow retrieval problem of the centralized model by caching data on each of the workstations.

Data for the visual display are always fetched directly from the cache and are updated whenever changes are received. Several variations on this approach are described in [2].

In this model, each computer has a copy of the database, but cannot make changes to an item until it obtains ownership of that item. To secure ownership of data means obtaining a lock from a centralized lock server. There can be one lock for the whole database, or separate locks for different parts of it. With multiple locks, changes to different parts of the database can proceed in parallel. When there is just one lock, only one participant at a time can make changes since any change requires locking the entire database. Another extreme is to have a separate lock for every datum.

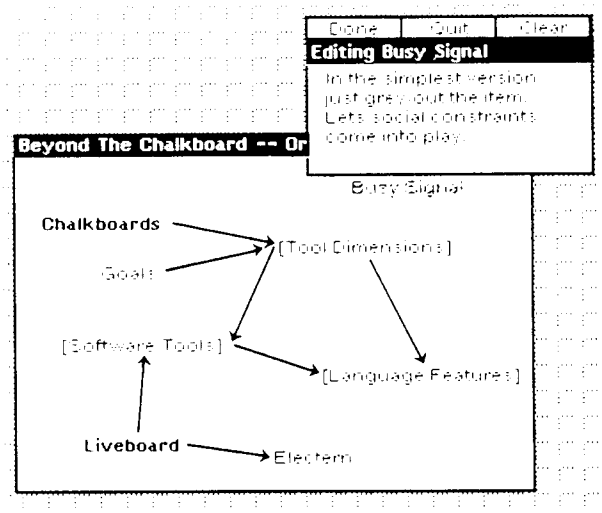
Locks provide mutual exclusion for processes that write data, and changes to the database are serialized by the numbered sequences of lock owners. This ensures that machines will converge to the same state. If some transactions require ownership of more than one lock, the usual cautions and techniques for avoiding deadlock apply [7, 15] (e.g., transactions that require multiple locks must acquire them all at once).

Unfortunately, our implementation of this model has so far yielded unacceptable delays for obtaining locks; these delays can be traced to a limitation of the process scheduler in our programming environment, namely, that it is not preemptive. There is no way to guarantee limits on delays in our system since processes are not prioritized and can run an arbitrary amount of time without yielding.

Cooperative Model. In the cooperative model, the approach we are currently using, each machine has a copy of the database, and changes are installed by broadcasting the change without any synchronization. By itself, this approach entails the following inherent race conditions: If two participants make changes to the same data simultaneously, there is a race to see which change will take effect first, and the results can be different on different machines.

Two factors mitigate against these apparent shortcomings. The first is that most sequences of changes to our databases yield results that are independent of the order in which they are done. Moreover, Colab participants are aware of the problem and use verbal cues ("voice locks") to coordinate their behavior; it is therefore rare that participants will change the same data at the same time.

The second factor is that, quite apart from the mechanisms for ensuring integrity of the distributed database, in the Colab we need to provide mechanisms that coordinate the activities of the participants and support the social mechanisms for both



When multiple users interact with a shared object, conflicts can occur. An early conflict detection system—a busy signal—quickly warns users that someone else is already editing an item, and brings social constraints into play; one way to indicate a busy item is to gray it out.

FIGURE 6. A Busy Item

partitioning work and reaching agreement that meetings by their nature rely on. One such mechanism is the busy signal described earlier. By graying out screen items that are in use, the signal warns other collaborators not to change them (see Figure 6). But, because there is an inherent delay between the moment that someone starts working on an item and the time that the busy signal is propagated to others, it is possible that a second participant will begin an incompatible revision. In this case, the busy signal ensures that two participants will *quickly discover* that they are working in a conflicting way.

In total, we are not satisfied with the properties of the cooperative model and are planning to investigate several alternatives presented in [2] for using two-phased locking and time stamps. Since the immediate users of our database are all people in visual and verbal contact, we are willing to consider the need for manual intervention in occasional cases of synchronization failure (i.e., to make some sacrifices to achieve the desired performance levels). Two specific approaches that we are considering are the last two discussed here: the dependency-detection model and the roving-locks model.

Dependency-Detection Model. The dependency-detection model corrects some of the shortcomings of the cooperative model by annotating data with a

stamp describing the author and time of the change. Every request to change data broadcasts several things: the new data, its stamp, and the stamp of the previous version of the data on the originating machine. When a machine receives a message requesting a change, it first checks whether the previous stamp in the request is the same as the stamp in its database. If they are different, a "dependency conflict" is signaled. The conflict is then resolved by a process that involves human intervention (at least to temporarily suspend activity), followed by propagation of the resolved values for data or the creation of multiple versions of the data.

The advantage of the dependency-detection approach is responsiveness. Changes to data do not first require serialization or the delay of obtaining a lock. The system assumes that a change can always be made, but it may have to fix things later if a conflict is detected.

Like the cooperative model, the dependency-detection model contains inherent race conditions, but it is able to detect them after the fact. If two participants change data at the same time, at least one of the machines will detect a dependency conflict as described above. However, it is possible to get "false alarms" if messages about changes to data from different sources arrive out of order; a dependency conflict would then be incorrectly signaled. Similarly, if two participants made a series of nearly simultaneous changes to a datum, multiple false alarms might be signaled. The ability to distinguish false alarms can be enhanced by keeping a longer history of changes. We do not yet have enough experience to decide whether the dependency-detection model (which is closely related to an approach called *certification* [2]) is necessary or practical.

Roving-Locks Model. The roving-locks model tries to reduce the delay in obtaining locks that is incurred with the centralized-lock model by distributing the lock-granting processes along with lock ownership. This is different than simply locating locks with the data; the intention here is to distribute control over specific data items to their last user, leading to a sort of "working set" [8] for locks. In this scenario, a participant's machine would tend to acquire the set of locks for that subset of the database on which it is actively working. Most lock requests would require no communication with other machines. After the first access, delay in getting a lock would be significant only in those cases where the lock is on a remote machine, that is, when two or more participants are actually competing for the same parts of the database.

Even if the working-set model is valid for locks, we suspect that the success of this model may depend on its having a preemptive scheduler to bound the delays in obtaining remote locks. More experience with the model is needed to determine whether roving locks is a practical solution.

Language Support

Colab software is built on Xerox Lisp Machines connected by an Ethernet [23]. The software is written in Loops [4], an object-oriented extension of Lisp [28] that resembles Smalltalk-80 [14] in that programs are organized in terms of objects that can hold data. Computation proceeds as objects send messages to each other. Loops supports the notion of permanent objects whose identity is specified by a unique identifier that is guaranteed to be unique across machines. Versions of these permanent objects can exist on several machines simultaneously. An *association* is a set of representations on multiple machines that stand for the same object; the individual representations are called *associates* and have the same unique identifier.

In the Colab, we use the term *conversation* to refer to the combination of a set of machines, Colab tools, and participants working together to solve a problem. When a new participant is added to a conversation, all participants find out about the newcomer, and the newcomer finds out about the other participants; the newcomer's machine gets copies of the object that represent the database.

In a conversation, communication is implemented by a combination of system facilities and programming abstractions and is supported over the Ethernet by several layers of protocols. Our implementation rests on a protocol for remote procedure calls [3]. On top of this, we have added a mechanism for sending messages to an object on a remote machine, and another for sending messages to all the associates of an object in a conversation.

Colab tools communicate via a programming abstraction that we call *broadcast methods*. Broadcast methods extend the object-oriented notion of methods from a single machine to multiple machines in a conversation. When a method is annotated as being a broadcast method, invoking it on one machine means that it will be run on all machines in the conversation. For example, if *Move* is a broadcast method in a Cognoter window for moving an item in the window, and *item37* receives a *Move* message on one of the machines, then *item37*'s associates on all the other machines will also receive the same message. All the details of queueing and transmitting the message to the relevant machines are handled automatically without further specification by the programmer.

Broadcast methods provide a simple abstraction for organizing communication, and a mechanism for

efficient communication about changes to the database. Colab tools assume that the software is loaded on the machines of all participants. In most cases, the bandwidth of network communication can be reduced by sending instructions rather than data.

Ideally, one should be able to take a program written for a single machine and change it into a distributed program by annotating some of the methods so that they will broadcast. In practice, this has worked out rather well. To support this facility, we have found it useful to establish a discipline for deciding which methods should be broadcast.

Methods are categorized roughly into three different sets that are treated differently with respect to conversion to broadcast methods: *user input*, *semantic actions*, and *display actions*. User-input methods control user interaction that specifies a change to be made to the database; they are run at the user's request (e.g., caused by mouse action) and are used to determine the nature and scope of a change. User-input methods are not made into broadcast methods because only the user initiating the change wants to engage in the interaction. The actual changes to the database are made by the semantic-action methods, which are broadcast so that the changes to the database will propagate to all machines containing the meeting database. Display-action methods update the displays and are not broadcast because the display is updated as a side effect of changing the database. If the image in more than one window depends on the value of a datum, then multiple display-action methods should be triggered by a single semantic-action method.

In some cases, the appropriate partitioning of methods into these categories can be subtle. For example, windows for displaying data can be parameterized (as in the case of proposal forms for Argnoter), thereby altering their display according to display parameters that specify belief sets or rankings of evaluation criteria. Maintaining WYSIWIS for these windows requires that changes to these parameters be considered part of the database and be broadcast as semantic actions; the subtlety arises to the extent that "display parameters" might be confused with display-action methods, which are not broadcast. Furthermore, when semantic actions can be derived from more primitive ones, only the primitive ones need be broadcast.

Support for Debugging

To make the debugging process more manageable, we have created tools for tracing and intercepting messages on the network. To monitor message transmission between machines, we use a conversation viewer. It works for all Colab tools, letting us monitor the broadcast queues and processes used to send messages between machines. The viewer shows when messages are queued, sent, and received, as well as the identity of the other machines. Using the viewer, we can often detect cases of unnecessary or incorrect message sending.

We have also developed tools for propagating program changes between machines. In debugging sessions, we have found it useful to make program changes on one machine and then to broadcast the changes to the other machines.

PRELIMINARY OBSERVATIONS AND RESEARCH QUESTIONS

If computers are to provide more effective meeting tools, we need a commensurately more adequate understanding of meeting processes. Although meetings are something that most of us know well, they come under the heading of those everyday activities that, because we know them so well, remain largely unexamined. Designing the Colab has required that we look again at the organization of meetings and meeting technology; at the same time, the Colab currently in place provides an experimental setting for pursuing these lines of research. In this section we present our preliminary observations about the Colab and describe the research issues that have been raised by these observations.

In their current form, Colab tools reflect our experience of, and ideas about, our own work processes, in particular those aimed at collaborative writing and argumentation. Our research strategy is to draw upon familiar practices first, and then to locate those practices within a wider range of face-to-face meetings in different settings and with different participants. The Colab was used early on to produce the present article, and even though the Colab was not yet fitted with audiovisual recording equipment or documenting software, these early sessions did provide a set of preliminary observations about the relationship between Cognoter tools and the writing process, and their relation to the process of collaborative writing.

The Structure of the Writing Process

The current Cognoter design reflects a set of conjectures regarding the writing process, from the early stage of idea generation and development through the generation of a path or outline for a final presentation. The actual use of Cognoter revealed not only the points of fit between design and process, but some subtle disjunctures as well.

For example, the design premise for Cognoter was that the brainstorming window be an unstructured repository for ideas. The availability of a public window, into which people could easily and spontaneously enter new text, would allow the group to

put a large number of ideas "onto the table" without a great deal of discussion or negotiation. Ideally, this initial brainstorming phase is followed by an organizing phase, in which group members elaborate the relationships between ideas and debate their cogency. However, in early sessions with Cognoter, we found that even before moving on to the organizing phase, members began using spatial grouping in the brainstorming window to display relationships between ideas. Even after items were explicitly linked, the spatial cues helped to display the relationships between items; these spatial cues, in turn, were important to the elaboration of meaning.

The process of organizing and evaluation made it easier to see whether or not the set of ideas generated during brainstorming was complete. Although our initial design assumption was that use of the outlining tool would follow completion of the evaluation phase, in practice, participants found the outlining tool useful for displaying intermediate states of the emerging structure as well. These observations suggest slightly different "joints" in the process than we had originally assumed. In future sessions, we will look carefully at the natural organization of the group writing process, the way people use the available tools to see the developing structure of their collective argument, and the relationship between the initial design assumptions and the actual uses people make of the tools.

Maintaining the Collaboration

The Colab's starting premise was that serial access to problem-solving technology obstructs the kind of equal participation that ideally characterizes collaboration, particularly for an activity like writing, where collaboration seems ideally not to involve any predetermined or fixed division of labor among participants. The multiuser interface was designed to overcome this obstacle by letting participants act simultaneously, write independently, and enter new text into a shared database—virtually at the same time. By equalizing access of all participants to displays and shared data, the Colab's interface enhances flexibility as to roles and discourages control over the activity by any one participant.

However, our early sessions demonstrated that the constraints imposed by current technologies are not just a limitation on collaboration but in some ways a resource as well. In particular, the fact that a writing technology allows only one person to enter text at a time enforces a kind of shared focus (i.e., a focus on that person's actions) that maintains a common context for the group. Where only one person at a time has access to the writing technology, roles are in a very real sense visible at a glance; moreover, what is being done to the text is transparent in the actions of whomever controls the writing technology. Many of the accompanying practices—rising to go to the chalkboard, taking over the keyboard—can also be viewed as resources for the participants in the sense of seeing what is going on and providing a basis for the smooth exchange of roles. The possibility of independent writing activity and simultaneous entry of new text brings new demands on participants to stay informed about what others are doing. Relaxing the requirements on turn taking by allowing parallel actions necessitates alternative ways of accomplishing what the turn-taking system accomplishes: namely, an orderly transition from one participant to the next, and an incremental, sequentially coherent development of the joint activity.

In early Cognoter meetings, the work of maintaining a shared focus was evident in the ebb and flow of meeting activity. During the ordering phase particularly, where ideas are elaborated, participants tended to interact verbally for a few minutes, explaining immediate goals and making short-term plans of action, after which the group settled into their "assignments," typing intently for a while. After a few minutes of parallel editing, people would lose track of what the others were doing and, therefore, of what to do next. The group would then stop interacting with the system and again discuss where they were and what they should do. These transitions between parallel and convergent activity sometimes required negotiation. In particular, individuals engaged in different activities might not arrive at transition places simultaneously and might not be equally interruptable at any given time. The early Cognoter sessions encompassed several such cycles of regrouping, summarization, joint planning, and then parallel action.

Along with personal interaction, shared focus is achieved by means of reference to common objects. Cognoter's goal, as with a chalkboard, is to enable participants to refer to common objects through various kinds of efficient reference such as deixis[5] and pointing. Although the WYSIWIS idealization recognizes that efficient reference depends on a common view of the work at hand, a distinctive problem arises in computer-based environments in that the boundary between logical and physical objects is blurred. This represents a tremendous advantage, on one level, in that relaxations of WYSIWIS allow participants to tailor their individual display of the shared view to their own specifications. However, it also means that, although people may be referring to the "same" piece of text, the text may be in an entirely different location on their respective displays. With the use of windows that can be moved, re-

[5] *Deixis* means referring to something either verbally (e.g., "the gray house across the street") or by pointing.

shaped, and scrolled, conventions are required to avoid situations in which one person tries to see some text at the top of a long passage while another tries to see text at the bottom, or one member of the group puts up a very large public window, obscuring everyone else's view (situations that we have informally dubbed "Scroll Wars" and "Window Wars").

As well as confirming the usefulness of a single view of the public record, our early experience with Cognoter identified a more subtle element of shared focus. With a single display device (e.g., a chalkboard or workstation), it is common for one person to be assigned the task of actually entering new text into the record; typically, not only the new text, but the writing activity itself, is visible to the other participants. In the current design of Cognoter, however, the actual editing is done in private windows, with only the finished text broadcast to coparticipants. This design decision, while encouraging parallel activity, poses some interesting new problems for the collaborative process. In particular, participants in the early sessions expressed frustration at not being able to see what the others were doing; specifically, at not being able to watch when others were engaged in writing. To an important degree, it seems that participants need access not only to the product of each others' writing, but to the writing process itself. The unanticipated usefulness of the video switch, which allows one to switch between displays,[6] underscores the importance of a shared view for maintaining the joint focus. User frustrations in this regard reopen the question as to the ideal grain size at which individual and group transactions take place, and the relationship between private and public views.

In general, these early observations were confirmed by a small set of controlled experiments run at UC Berkeley. In the trials, several pairs of student collaborators unfamiliar with the Colab used either Cognoter or a chalkboard to plan article outlines. The outcomes showed that the interface of Cognoter is complicated enough to require practice to be used effectively [13]. More extensive trials with larger groups will await the completion of video recording and meeting analysis tools that are now being created.

Research Questions

Our guiding question has been, What are the processes of collaboration for which the computer is an appropriate tool, and what particular Colab tools could be designed to support these processes? As a first approximation, Cognoter and Argnoter have assumed two contrasting processes of collaborative writing and argumentation, both drawn from our own experience. Cognoter takes a joint presentation as its object and encourages consensus by supporting a single viewpoint, whereas Argnoter encourages competing proposals and delayed consensus by allowing the display and comparison of multiple views.

Having identified the collaborative processes and refined the associated tools, we need next to question the generality of our assumptions. To what extent do our work practices compare and contrast with other settings and other participants? Does a tool, by reifying a process and making it explicit, thereby also make it portable across groups? Or do we need a set of tools that can be customized to different users in different settings? Under what circumstances are explicit structures desirable, and under what circumstances do we want to minimize the amount of structure we build into our tools? These questions and others will be explored as we extend the design and experiment with its use.

RELATED WORK

The possibility that computers might be used to support group problem solving was appreciated by early visionaries long before it was practically feasible. In 1945, Bush presented a hypothetical system called a "Memex" that included an interactive database [6] by which associative "trails" of exploration could be saved to be recalled and retraced at a later time. Bush believed that a common encyclopedic database of information integrated from many areas of human activity would enhance the quality of societal problem solving.

In the 1960s, experimental systems like the NLS/AUGMENT [11, 12] began to use computers to support collaboration. The NLS/AUGMENT supported terminal linking, electronic mail, sharing of files, and "televiewing"—the ability to "pass the gavel" among several people working together at separate terminals. Englebart saw machines as providing an important medium for communication and was known for his development of novel user interfaces like the mouse. Englebart was also an early worker in hypertext, systems that organize fragments of text in annotated networks. This work has been pursued in several other systems including TEXTNET [33], Xanadu [24], NoteCards [34], and Annoland.

At a time when time-shared systems like TENEX [5] popularized electronic mail and shared files, some observers (e.g., Lederberg [21]) reported a qualitative difference in the ways they were interacting with colleagues. In the mid 1970s, researchers at the Stanford AI Lab built a video, audio, and keyboard

[6] The Colab video switch allows the content of any screen to be directed to another screen; it was originally designed to aid in debugging across multiple machines.

crossbar switch to allow users at multiple workstations to collaborate from separate workstations. At the same time, another line of work pursued the use of communications facilities to tie together people working at different locations. Known as *teleconferencing* [18, 19], this work eschewed much use of computers and has developed slowly, due largely to high communication costs for video images. Meanwhile, others have developed systems for remote conferencing that rely mostly on computers rather than video: Known as *computer conferencing*, these systems include electronic mail, editors, voting mechanisms, shared files, and archiving, but do not provide structure for the conferences based on any models of group problem-solving processes. In [16], Hiltz and Turoff review some of these systems and provide an extensive bibliography; prime examples are EIES [17] and some parts of NLS/AUGMENT [11].

Although computers have been used experimentally in meetings to support specialized problem-solving processes since at least 1972 [35], the impact has been much less dramatic than with other computer applications (see [20]). Most of these systems are organized around formal and mathematical models of decision making like multiattribute utility models and cost–benefit analyses. The Delphi method [22] and the Nominal Group method [20], for example, are techniques for structuring group problem solving that have been used with and without computer support. The Delphi model considered by Turoff [35] is designed for technological forecasting by a geographically dispersed group, while the Nominal Group represents a consensus-forming process for face-to-face meetings; both have been characterized as "rational but naive" [20]. Since we have little experience with them, we offer no independent assessment; however, we note that the meeting processes used in the Colab are similar to the meeting methods commonly taught in corporate training programs.

RTCAL/IOLC, a somewhat analogous system to the Colab that was developed at MIT by Sunil Sarin [30], allows a group of users to synchronously exchange information from personal calendar databases to schedule a future meeting. It differs from the Colab in particular trade-offs of computer communication (e.g., RTCAL has a centralized database management scheme) and the absence of process models for problem solving, but is similar in that it uses personal computers, works in real time, and maintains consistent views by message passing over a local network [29]. Another research project reported by Applegate, Konsynski, and Nunamaker [1] also resembles the Colab in that it provides personal computers to meeting participants around a conference table and uses a video projector to provide large public views; it also provides tools for brainstorming and analysis. However, unlike the Colab, it is oriented around decision support models for planning and quantitative analysis. Also, since it is built using microcomputers with very limited display space, there has been little opportunity to experiment with private and public windows or multiuser interfaces.

Kraemer and King [20] observe that there are very few successful computer conference rooms, if any, and that even these systems have been plagued by hardware difficulties. As the primary obstacles to success, they cite inaccessibility of computing resources, unreliable video projectors, and limited graphics capabilities. However, they quite rightly note that in recent years computing and projection technology have become much more reliable and also less expensive. We agree with them that most of the activity with computer-supported conferences over the next three to four years will center on research and development.

In terms of technology, there have been several advances that will enable this work to proceed at a much more rapid pace: among them, more powerful personal workstations, local-area networks, advanced programming environments [31], distributed programming, and interface technology. These advances will make it possible to develop prototype systems quite rapidly and thus to experiment readily with new tools.

CONCLUSIONS

Focusing on developing and understanding "team computers" (i.e., collaborative systems for group meetings), the Colab project has produced a usable meeting room and several operational tools. The liveboard is operational but not fully integrated with our software. As we begin to use the Colab on a regular basis, it will afford a laboratory for studying the effects of the tools on collaborative meetings. The Colab meeting room is now being fitted with the video equipment necessary to record working Colab sessions. We will use the Colab to try to understand why collaborative problem solving is organized as it is, the relationship of that organization to existing technology, and the trade-offs involved in displacing old practices with new technology.

Upon hearing about the Colab, a manager from a large American corporation whose job it is to introduce appropriate computing technology at the executive staff level told us an interesting story. After working diligently for several months to bring things up-to-date and to revitalize operations with tools like electronic mail, document processing, databases, and automatic spreadsheets, he remained unsure about the degree of success he had achieved. One

day, in a burst of frank evaluation, one of his charges told him that, despite the best intentions, he felt the computer was not making a difference and did not expect it to save him more than 30 minutes a day, even if he did learn how to use it. The reason was that this individual was not in his office for more than 30 minutes; he spent almost his entire day in meetings! *Moral*: *Office automation simply does not reach people who are away from their offices*, which brings us back to the premise of the Colab project: Meetings are important. They are at the core of the way most organizations do business. As such, tools like the Colab touch fundamentally the ways we meet and make decisions collectively.

Acknowledgments. This article has benefited greatly from the suggestions and criticisms of Agustin Araya, John Seely Brown, Richard Fateman, John Florentin, Mark D. Hill, Bernardo Huberman, Randy Katz, Mark Miller, Sanjay Mittal, Ted Selker, Jeff Shrager, and Mike Stonebraker.

Many thanks to Bill Volkers for creating the liveboard, and to Stu Card and Jeff Shrager for early ideas for the liveboard. We wish to acknowledge Ted Selker for his suggestions about many aspects of the Colab and for designing electronic chalk for the liveboard; Steve Osburn, Joan Osburn, Gene Hall, and Lee Anderson for creating the Colab physical setting; and Steven Levy for his contributions to the first implementations of Colab software.

Special thanks to John Seely Brown for his ideas, criticisms, and encouragement on the Colab project. Without his support, the project could never have been launched nor could the initial momentum have been sustained. Thanks also to Bill Spencer and George Pake for creating an environment at Xerox PARC that makes projects like this possible.

REFERENCES

1. Applegate, L.M., Konsynski, B.R., and Nunamaker, J.F. A group decision support system for idea generation and issue analysis in organizational planning. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec.). ACM, New York. To be published.
2. Bernstein, P.A., and Goodman, N. Concurrency control in distributed database systems. *ACM Comput. Surv. 13*, 2 (June 1981), 185–221.
3. Birrell, A.D., and Nelson, B.J. Implementing remote procedure calls. Tech. Note CSL-83-7, Xerox PARC, Palo Alto, Calif., Dec. 1983.
4. Bobrow, D.G., and Stefik, M.J. *The Loops Manual*. Xerox PARC, Palo Alto, Calif., 1983.
5. Bobrow, D.G., Burchfiel, J.D., Murphy, D.L., and Tomlinson, R.S. TENEX, a paged time-sharing system for the PDP-10. *Commun. ACM 15*, 3 (Mar. 1972), 135–143.
6. Bush, V. As we may think. *Atlantic Mon. 176*, 1 (June 1945), 101–108.
7. Coffman, E.G., Elphick, M.J., and Shoshani, A. System deadlocks. *ACM Comput. Surv. 3*, 2 (June 1971), 67–78.
8. Denning, P.J. Virtual memory. *ACM Comput. Surv. 2*, 3 (Sept. 1970), 153–189.
9. Donahue, J., and Widom, J. Whiteboards: A graphical database tool. *ACM Trans. Off. Inf. Syst. 4*, 1 (Jan. 1986), 24–41.
10. Doyle, M., and Straus, D. *How to Make Meetings Work*. Berkeley Publishing Group, New York, 1984.
11. Englebart, D.C. Collaboration support provisions in AUGMENT, OAC 84 digest. In *Proceedings of the 1984 AFIPS Office Automation Conference* (Los Angeles, Calif. Feb. 20–22). AFIPS, Reston, Va., 1984, pp. 51–58.
12. Englebart, D.C., and English, W.K. Research center for augmenting human intellect. In *Proceedings of the Fall Joint Computing Conference* (San Francisco, Calif., Dec. 9–11). AFIPS, Reston, Va., 1968, pp. 395–410.
13. Foster, G. Collaborative systems and multi-user interfaces: Computer-based tools for cooperative work. Doctoral dissertation, Computer Science Division, Univ. of California at Berkeley, Dec. 1986. To be published.
14. Goldberg, A., and Robson, D. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, Reading, Mass., 1983.
15. Hansen, P.B. *Operating System Principles*. Prentice-Hall, Englewood Cliffs, N.J., 1973.
16. Hiltz, S.R., and Turoff, M. *The Network Nation: Human Communication via Computer*. Addison-Wesley, Reading, Mass., 1978.
17. Hiltz, S.R., and Turoff, M. The evolution of user behavior in a computerized conferencing system. *Commun. ACM 24*, 11 (Nov. 1981), 739–752.
18. Johansen, R. *Teleconferencing and Beyond: Communications in the Office of the Future*. McGraw-Hill, New York, 1984.
19. Johansen, R., Vallee, J., and Spangler, K. *Electronic Meetings: Technical Alternatives and Social Choices*. Addison-Wesley, Reading, Mass., 1979.
20. Kraemer, K.L., and King, J.L. Computer supported conference rooms: Final report of a state of the art study. Dept. of Information and Computer Science, Univ. of California, Irvine, Dec. 1983.
21. Lederberg, J. Digital communications and the conduct of science: The new literacy. *Proc. IEEE 66*, 11 (Nov. 1978), 1313–1319.
22. Linstone, H.A., and Turoff, M. *The Delphi Method: Techniques and Applications*. Addison-Wesley, Reading, Mass., 1975.
23. Metcalfe, R.M., and Boggs, D.R. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM 19*, 7 (July 1976), 395–404.
24. Nelson, T. *Literary Machines*. Ted Nelson, Swarthmore, Pa., 1981.
25. O'Connor, R.J. Outline processors catch on. *InfoWorld* (July 2, 1984), 30–31.
26. Panko, R.R. Office work. *Off. Technol. People 2* (1964), 205–238.
27. Platt, J.R. Strong Inference. *Science 146*, 3642 (Oct. 1964), 347–353.
28. Sanella, M., et al. *Interlisp Reference Manual*. Xerox PARC, Palo Alto, Calif., 1983.
29. Sarin, S.K. Interactive on-line conferences. Ph.D. thesis MIT/LCS/TR-330, MIT, Cambridge, Mass., Dec. 1984.
30. Sarin, S., and Greif, I. Computer-based real-time conferencing systems. *Computer 18*, 10 (Oct. 1985), 33–45.
31. Sheil, B. Power tools for programmers. *Datamation* (Feb. 1983), 131–144.
32. Stefik, M., Foster, G., Lanning, S., and Tatar, D. The scope of WYSIWIS: Early experiences with multi-user interfaces. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec.). ACM, New York. To be published.
33. Trigg, R., and Weiser, M. TEXTNET: A network-based approach to text handling. *ACM Trans. Off. Inf. Syst. 4*, 1 (Jan. 1986), 1–23.
34. Trigg, R., Suchman, L., and Halasz, F. Supporting collaboration in NoteCards. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec.). ACM, New York. To be published.
35. Turoff, M. Delphi conferencing: Computer-based conferencing with anonymity. *Technol. Forecasting Soc. Change 3* (1972), 159–204.

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; *human information processing*; H.2.4 [**Database Management**]: Systems—*distributed systems*; H.4.m [**Information Systems Applications**]: Miscellaneous
General Terms: Design, Human Factors, Languages
Additional Key Words and Phrases: Computer-supported collaboration, computer-supported groups, computer-supported meetings, multi-user interfaces

Received 1/85; accepted 7/86

Authors' Present Addresses: Mark Stefik, Daniel G. Bobrow, Kenneth Kahn, Stan Lanning, and Lucy Suchman, Intelligent Systems Laboratory, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304; Gregg Foster, Computer Science Division, University of California, Berkeley, CA 94720.