GROUPW

WARE

Groupware reflects a change in emphasis from using the computer to solve problems to using the computer to facilitate human interaction. This article describes categories and examples of groupware and discusses some underlying research and development issues. GROVE, a novel group editor, is explained in some detail as a salient groupware example.

SOME ISSUES AND EXPERIENCES

C.A. Ellis, S.J. Gibbs, and G.L. Rein

Society acquires much of its character from the ways in which people interact. Although the computer in the home or office is now commonplace, our interaction with one another is more or less the same now as it was a decade ago. As the technologies of computers and other forms of electronic communication continue to converge, however, people will continue to interact in new and different ways.

One probable outcome of this technological marriage is the electronic workplace—an organization-wide system that integrates information processing and communication activities. The study of such systems is part of a new multidisciplinary field: *Computer-Supported Cooperative Work* (CSCW) [29]. Drawing on the expertise and collaboration of many specialists, including social scientists and computer scientists, CSCW looks at how groups work and seeks to discover how technology (especially computers) can help them work.

Commercial CSCW products, such as *The Coordinator*™ [24] and other PC-based software [67], are often referred to as examples of *groupware*. This term is frequently used almost synonymously with CSCW technology (see [8] or [44] for general descriptions of, and strong motivation for groupware). Others define groupware as software for small or narrowly focused groups, not organization-wide support [30]. We propose a somewhat broader view, suggesting that groupware be viewed as the class of applications, for small groups and for organizations, arising from the merging of computers and large information bases and communications technology. These applications may or may not specifically support cooperation.

This article explores groupware in this larger sense and delineates classes of design issues facing groupware developers. It is divided into five main sections. First, the **Overview** defines groupware in terms of a group's common task and its need for a shared environment. Since our definition of groupware covers a range of systems, the second section provides a **Taxonomy of Groupware Systems.** The third describes the widely ranging **Perspectives** of those who build these systems. The fourth section, **Concepts and Example,** introduces some common groupware concepts, and applies these to GROVE, one example of a groupware system. The fifth section contains a discussion of some Design Issues facing groupware designers and developers. Our emphasis in this section is upon system-level issues within real-time groupware. In our conclusion to this article we both issue a note of caution concerning the difficulty of developing successful groupware due to social and organizational effects, and in-

Illustration: Eva K. Sutton, Untitled.

dicate that there is much interesting work remaining to be done in this field.

Overview

Most software systems only support the interaction between a user and the system. Whether preparing a document, querying a database, or even playing a video game, the user interacts solely with the computer. Even systems designed for multiuser applications, such as office information systems, provide minimal support for user-to-user interaction. This type of support is clearly needed, since a significant portion of a person's activities occur in a group, rather than an individual, context. As we begin to focus on how to support this group interaction, we must attend to three key areas: communication, collaboration, and coordination.

The Importance of Communication, Collaboration, and Coordination

Computer-based or computer-mediated communication, such as electronic mail, is not fully integrated with other forms of communication. The primarily asynchronous, text-based world of electronic mail and bulletin boards exists separately from the synchronous world of telephone and face-to-face conversations. While applications such as voice mail or talk programs blur this distinction somewhat, there are still gaps between the asynchronous and the synchronous worlds. One cannot transfer a document between two arbitrary phone numbers, for example, and it is uncommon to originate a telephone conversation from a workstation. Integrating telecommunications and computer processing technologies will help bridge these gaps.

Similar to communication, collaboration is a cornerstone of group activity. Effective collaboration demands that people share information. Unfortunately, current information systems—database systems in particular—go to great lengths to insulate users from each other. As an example, consider two designers working with a CAD database. Seldom are they able to simultaneously modify different parts of the same object and be aware of each other's changes; rather, they must check the object in and out and tell each other what they have done. Many tasks require an even finer granularity of sharing. What is needed are shared environments that unobtrusively offer up-to-date group context and explicit notification of each user's actions when appropriate.

The effectiveness of communication and collaboration can be enhanced if a group's activities are coordinated. Without coordination, for example, a team of programmers or writers will often engage in conflicting or repetitive actions. Coordination can be viewed as an activity in itself, as a necessary overhead when several parties are performing a task [62]. While current database applications contribute somewhat to the coordination of groups—by providing multiple access to shared objects—most software tools offer only a single-user perspective and thus do little to assist this important function.

A Definition of Groupware

The goal of groupware is to assist groups in communicating, in collaborating, and in coordinating their activities. Specifically, we define groupware as:

> *computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.*

The notions of a *common task* and a *shared environment* are crucial to this definition. This excludes multiuser systems, such as time-sharing systems, whose users may not share a common task. Note also that the definition does not specify that the users be active simultaneously. Groupware that specifically supports simultaneous activity is called *real-time groupware;* otherwise, it is *non-real-time groupware.* The emphasis of this article is real-time groupware and system-level issues.

The term groupware was first defined by Johnson-Lenz [46] to refer to a computer-based system plus the social group processes. In his book on groupware [44], Johansen restricts his definition to the computer-based system. Our definition follows the line of reasoning of Johansen since this article is primarily concerned with system-level issues. All of the authors mentioned agree with us that the system and the group are intimately interacting entities. Successful technological augmentation of a task or process depends upon a delicate balance between good social processes and procedures with appropriately structured technology.

The Groupware Spectrum

There is no rigid dividing line between systems that are considered groupware and those that are not. Since systems support common tasks and shared environments to varying degrees, it is appropriate to think of a groupware spectrum with different systems at different points on the spectrum. Of course, this spectrum is multidimensional; two dimensions are illustrated in Figure 1. Following are two examples of systems described according to our definition's common task dimension:

1. A conventional timesharing system supports many users concurrently performing their separate and independent tasks. Since they are not working in a tightly coupled mode on a common task, this system is usually low on the groupware spectrum.
2. In contrast, consider a software review system that electronically allows a group of designers to evaluate a software module during a real-time interaction. This system assists people who are focusing on the same specific task at the same time, and who are closely interacting. It is high on the groupware spectrum.

Other systems, such as those described in the following examples,

can be placed on the groupware spectrum according to how they fit the shared environment part of our definition. In other words, to what extent do they provide information about the participants, the current state of the project, and the social atmosphere?

1 The typical electronic mail system transmits messages, but it provides few environmental cues. Therefore it is rather low on the groupware spectrum.
2 In contrast, the "electronic classroom" system [74] uses multiple windows to post information about the subject being taught, and about the environment. Emulating a traditional classroom, this system allows an instructor to present an on-line lecture to students at remote personal workstations. In addition to the blackboard controlled by the teacher, windows display the attendance list, students' questions and comments, and the classroom status. Many commands facilitate lecture delivery and class interaction. This system is high on the groupware spectrum.

Over time, systems can migrate to higher points on the groupware spectrum. For example, Engelbart's pioneering work on augmenting the intellect in the 1960s demonstrated multiuser systems with groupware capabilities similar to some of today's research prototypes. Engelbart's On-Line System [NLS] [21], an early hypertext system, contained advanced features such as filters for selectively viewing information, and support for on-line conferencing. Today's improved technology and enhanced user interfaces have boosted this type of system higher on the groupware spectrum. Additionally, the technological infrastructure required for groupware's wide use—an infrastructure missing in the 1960s—is now emerging.

Taxonomy of Groupware Systems

This section presents two taxonomies useful for viewing the variety of groupware. The first taxonomy is based upon notions of time and space; the second on application-level functionality.

Time Space Taxonomy

Groupware can be conceived to help a face-to-face group, or a group that is distributed over many locations. Furthermore a groupware system can be conceived to enhance communication and collaboration within a real-time interaction, or an asynchronous, non-real-time interaction. These time and space considerations suggest the four categories of groupware represented by the 2x2 matrix shown in Figure 2. Meeting room technology would be within the upper left cell; a real-time document editor within the lower left cell; a physical bulletin board within the upper right cell; and an electronic mail system within the lower right cell.

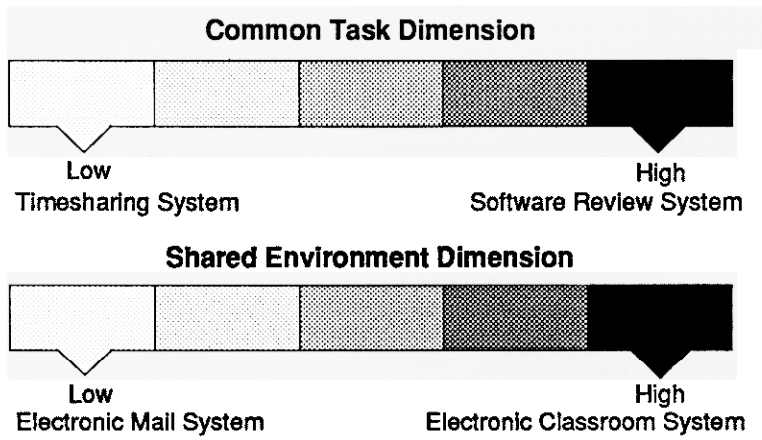
A comprehensive groupware system might best serve the needs of all of the quadrants. For example, it would be quite helpful to have the same base functionality, and user interface look and feel (a) while I am using a computer to edit a document in real-time with a group (same time/same place or same time/different place) and (b) while I am alone editing in my office or home (different time). Of course, there are other dimensions, such as group size, that can be added to this simple 2x2 matrix. Further details of this taxonomy are presented by Johansen [45].

Application-Level Taxonomy

The second taxonomy presented in

FIGURE 1. Two Dimensions of the Groupware Spectrum.

FIGURE 2. Groupware Time Space Matrix.



	Same Time	Different Times
Same Place	face-to-face interaction	asynchronous interaction
Different Places	synchronous distributed interaction	asynchronous distributed interaction

this section is based on application-level functionality and is not meant to be comprehensive; furthermore, many of the defined categories overlap. This taxonomy is intended primarily to give a general idea of the breadth of the groupware domain.

Message Systems

The most familiar example of groupware is the computer-based message system, which supports the asynchronous exchange of textual messages between groups of users. Examples include electronic mail and computer conferencing or bulletin board systems. The proliferation of such systems has led to the "information overload" phenomenon [37]. Some recent message systems help manage information overload by easing the user's processing burden. "Intelligence" is sometimes added to the message delivery system; for example, the Information Lens [63] lets users specify rules that automatically file or reroute incoming messages based on their content. Other systems add intelligence to the messages themselves; the Imail system [38], for example, has a language for attaching scripts to messages. Scripts are sender-specified programs that execute in the receiver's environment and that can, for example, query the receiver, report back to the sender, or cause the message to be rerouted.

Multiuser Editors

Members of a group can use multiuser editors to jointly compose and edit a document. Some of these editors, such as ForComment™ [67], are for asynchronous use, and conveniently separate the text supplied by the author from the comments of various reviewers. Real-time group editors allow a group of people to edit the same object at the same time. The object being edited is usually divided into logical segments; for example, a document could be split into sections or a program into procedures or modules. Typically, a multiuser editor allows concurrent read access to any segment, but only to one writer per segment. The editor transparently manages locking and synchronization, and users edit the shared object as they would a private object. Examples include the Collaborative Editing System (CES) [28], Shared Book [58], and Quilt [22, 57].

Some multiuser editors provide explicit notification of other users' actions. For example, Mercury [47], an editor intended for programming teams, informs users when their code needs to be changed because of program modifications made by others. The DistEdit system [49] tries to provide a toolkit for building and supporting multiple group editors.

Group Decision Support Systems and Electronic Meeting Rooms

Group Decision Support Systems (GDSSs) provide computer-based facilities for the exploration of unstructured problems in a group setting (see [51] or [16] for recent surveys). The goal is to improve the productivity of decision-making meetings, either by speeding up the decision-making process or by improving the quality of the resulting decisions [51]. There are GDSS aids for decision structuring, such as alternative ranking and voting tools, and for idea generation [2] or issue analysis [11].

Many GDSSs are implemented as electronic meeting rooms that contain several networked workstations, large computer-controlled public displays, and audio/video equipment (examples are discussed in [2, 12, 16, 64, 77 and 78]). Some of these facilities require a specially trained operator; others assume operational competence among the group members.

A well-known example is the PlexCenter Planning and Decision Support Laboratory at the University of Arizona [2]. The facility provides a large U-shaped conference table with eight personal workstations; a workstation in each of four break-out rooms; a video disk; and a large-screen projection system that can display screens of individual workstations or a compilation of screens. The conference table workstations are recessed to enhance the participants' line of sight and to encourage interaction. They communicate over a local area network and run software tools for electronic brainstorming, stakeholder identification and analysis, and issue analysis.

Recent work at the University of Arizona has concentrated on the support of larger groups. The current large group facility has 24 workstations designed to support up to 48 people. The support of large groups presents unique challenges and opportunities.

Computer Conferencing

The computer serves as a communications medium in a variety of ways. In particular, it has provided three new approaches in the way people carry out conferences: real-time computer conferencing, computer teleconferencing, and desktop conferencing.

Real-Time Computer Conferencing

Real-time computer conferencing allows a group of users, who are either gathered in an electronic meeting room or physically dispersed, to interact synchronously through their workstations or terminals. When a group is physically dispersed, an audio link, such as a conference call, is often established.

There are two basic approaches to implementing real-time computer conferencing software [73]. The first embeds an unmodified single-user application in a *conferencing environment* that multiplexes the application's output to each participant's display [42]. Input comes from one user at a time, and a *floor passing* protocol (determining who has the floor) exchanges input control among users [56]. Examples include *terminal linking* (a service found in some time-sharing systems) and *replicated windows* (typically implemented by a window server that drives a set of displays in

tandem). The second approach is to design the application specifically to account for the presence of multiple users. Some examples are Real Time Calendar [RTCAL] [73], a meeting scheduling system, and Cognoter [78], a real-time group note-taking system.

Each approach has its advantages and disadvantages. While the first allows existing applications to be used, each user has an identical view of the application—there is no per-user context. The second approach offers the possibility of a richer interface, but the application must be built from the ground up or with considerable additional effort.

Computer Teleconferencing

Telecommunication support for group interaction is referred to as teleconferencing [43]. The most familiar examples of teleconferencing are conference calls and video conferencing. Teleconferencing tends to be awkward, requiring special rooms and sometimes trained operators. Newer systems provide workstation-based interfaces to a conference and make the process more accessible. Xerox, for example, established an audio/video link for use by a project team split between Portland and Palo Alto [26]. Most video interactions occurred between large Commons areas at each site, but project members could also access video channels through their office workstations. A similar system, CRUISER [72], lets users electronically roam the hallways by browsing video channels.

Desktop Conferencing

Teleconferencing is not only relatively inaccessible, but it also has the disadvantage of not letting participants share text and graphics (see [18] for a discussion of the failure of video conferencing). Real-time computer conferencing does not offer video capabilities. A third type of computer-supported conferencing combines the advantages of teleconferencing and real-time conferencing while mitigating their drawbacks. Dubbed *desktop conferencing*, this method still uses the workstation as the conference interface, but it also runs applications shared by the participants. Modern desktop conferencing systems support multiple video windows per workstation. This allows display of dynamic views of information, and dynamic video images of participants [80].

An example of desktop conferencing is the MMConf system [14]. MMConf provides a shared display of a multimedia document, as well as communications channels for voice and shared pointers. Another example is the Rapport multimedia conferencing system [1]. Rapport is designed for workstations connected by a multimedia network (a network capable of transmitting data, voice, and video). The system supports various forms of interaction, from simple telephone-like conversations to multiparty shared-display interaction.

Intelligent Agents

Not all the participants in an electronic meeting are people. Multiplayer computer games, for example, might automatically generate participants if the number of people is too low for a challenging game. Such nonhuman participants are a special case of intelligent agents (a similar concept is "surrogates" [44]). In general, intelligent agents are responsible for a specific set of tasks, and the user interface makes their actions resemble those of other users.

As a specific example, we have developed a groupware toolkit that includes an agent named Liza [25]. One of the tools in the toolkit displays the pictures and locations of all session participants. When Liza joins a session, a picture of an intelligent-looking android is also displayed, indicating to the group that Liza is *participating*. Liza's participation means that a set of rules owned by Liza become active; these rules monitor session activity and result in Liza suggesting changes of content or form.

Coordination Systems

The coordination problem is the "integration and harmonious adjustment of individual work efforts toward the accomplishment of a larger goal" [76]. Coordination systems address this problem in a variety of ways. Typically these systems allow individuals to view their actions, as well as the relevant actions of others, within the context of the overall goal. Systems may also trigger users' actions by informing users of the states of their actions and their wait conditions, or by generating automatic reminders and alerts. Coordination systems can be categorized by one of the four types of models they embrace: form, procedure, conversation, or communication-structure oriented.

Form-oriented models typically focus on the routing of documents (forms) in organizational procedures. These systems address coordination by explicitly modeling organizational activity as fixed processes [59, 83]. In some of the more recent systems there is an effort to make process support more flexible. For example, in Electronic Circulation Folders [ECF] [48] exception handling is addressed through migration specifications that describe all the possible task migration routes in terms of the steps to be carried out in processing organizational documents.

Procedure-oriented models view organizational procedures as programmable processes; hence the phrase "process programming" [3, 68, 69]. This approach was first applied to coordination problems in the software process domain and takes the view that software process descriptions should be thought of and implemented as software. The development of process programs is itself a rigorous process consisting of specification, design, implementation, and testing/verification phases [69].

Conversation-oriented models are based on the observation that

people coordinate their activities via their conversation [15, 24, 65, 81]. The underlying theoretical basis for many systems embracing the conversation model is speech act theory [75]. For example, The Coordinator [24] is based on a set of speech acts (i.e., requests, promises, etc.) and contains a model of legal conversational moves (e.g., a request has to be issued before a promise can be made). As users make conversational moves, typically through electronic mail, the system tracks their requests and commitments.

Communication structure-oriented models describe organizational activities in terms of role relationships [10, 39, 77]. For example, in the ITT approach [39, 40], a person's electronic work environment is composed of a set of centers, where each center represents a function for which the person is responsible. Within centers are roles that perform the work and objects that form the work materials for carrying out the function of that center. Centers and roles have connections to other centers and roles, and the behavior of the connections is governed by the role scripts of the interacting roles.

Summary

As mentioned, overlap exists in these categories. As the demand for integrated systems increases, we see more merging of these functionalities. Intelligent message systems can and have been used for coordination. Desktop conferencing systems can and have been used for group editing. Nevertheless, many systems can be categorized according to their primary emphasis and intent. This, in turn, may depend upon the perspectives of the system designers.

Perspectives

As the preceding section's taxonomy suggests, groupware relies on the approaches and contributions of many disciplines. In particular, there are at least five key disciplines or perspectives for successful groupware: distributed systems, communications, human-computer interaction, artificial intelligence (AI), and social theory. It is important to note that the relationship between groupware and these five domains of study is a mutually beneficial one. Not only does each discipline advance our understanding of the theory and practice of groupware, but groupware presents challenging topics of research for all five domains—topics that without groupware might never be explored.

Of equal importance is the notion that a given groupware system usually combines the perspectives of two or more of these disciplines. We can see the desktop conferencing paradigm, for example, as having been derived in either of two ways:

1. by starting with communications technology and enhancing this with further computing power and display devices at the phone receiver, or
2. by starting with the personal workstation (distributed systems perspective) and integrating communications capabilities.

Distributed Systems Perspective

Because their users are often distributed in time and/or space, many multiuser systems are naturally considered to be *distributed* systems. The distributed systems perspective explores and emphasizes this decentralization of data and control. Essentially, this type of system infers global system properties and maintains consistency of the global state by observing and manipulating local parameters.

The investigation of efficient algorithms for distributed operating systems and distributed databases is a major research area in distributed systems theory. Some of these research results are applicable to groupware systems. For example, implementing electronic mail systems evokes complex distributed-systems issues related to robustness: recipients should be able to receive messages even when the mail server is unavailable. One solution is to replicate message storage on multiple server machines [6]. Discovering and implementing the required algorithms—algorithms that will keep these servers consistent and maintain a distributed name lookup facility—is a challenging task.

Communications Perspective

This perspective emphasizes the exchange of information between remote agents. Primary concerns include increasing connectivity and bandwidth, and protocols for the exchange of many types of information—text, graphics, voice and video.

One of the commonly posed challenges of groupware to communications technology is how to make distributed interactions as effective as face-to-face interactions. Perhaps the correct view of this challenge is that a remote interaction, supported by appropriate technology, presents an alternative medium. While this will not replace face-to-face communication, it may actually be preferable in some situations for some groups because certain difficulties, inconveniences, and breakdowns can be eliminated or minimized. For example, distributed interactions allow participants to access other relevant information, either via the computer or in a book on the shelf, without interrupting the interaction flow. This is analogous to findings on the use of telephone, electronic mail, and other technologies. While none of these replace face-to-face interaction, each has a niche where it is a unique and useful mode of communication. The challenge, then, is to apply appropriate technological combinations to the classes of interactions that will benefit the most from the new medium.

Human-Computer Interaction Perspective

This perspective emphasizes the importance of the user interface in computer systems. Human-computer interaction is itself a mul-

tidisciplinary field, relying on the diverse skills of graphics and industrial designers, computer graphics experts (who study display technologies, input devices, and interaction techniques), and cognitive scientists (who study human cognitive, perceptual, and motor skills).

Until recently, most user interface research has focused on single-user systems. Groupware challenges researchers to broaden this perspective, to address the issues of human-computer interaction within the context of multiuser or *group* interfaces. Since these interfaces are sensitive to such factors as group dynamics and organizational structure—factors not normally considered relevant to user interface design—it is vital that social scientists and end users play a role in the development of group interfaces.

Artificial Intelligence Perspective

With an emphasis on theories of intelligent behavior, this perspective seeks to develop techniques and technologies for imbuing machines with human-like attributes. The artificial intelligence (AI) approach is usually heuristic or augmentative, allowing information to accrue through user-machine interaction rather than being initially complete and structured.

This approach blends well with groupware's requirements. For example, groupware designed for use by different groups must be flexible and accommodate a variety of team behaviors and tasks: research suggests that two different teams performing the same task use group technology in very different ways [71]. Similarly, the same team performing two separate tasks uses the technology differently for each task.

AI may, in the long run, provide one of the most significant contributions to groupware. This technology could transform machines from passive agents that process and present information to active agents that enhance interactions. The challenge is to ensure that the system's activity enhances interaction in a way that is procedurally and socially desirable to the participants.

Social Theory Perspective

This perspective emphasizes social theory, or sociology, in the design of groupware systems. Systems designed from this perspective embody the principles and explanations derived from sociological research. The developers of Quilt [22], for example, conducted systematic research on the social aspects of writing, and from this research they derived the requirements for their collaborative editing environment. As a result, Quilt assigns document access rights according to interactions between users' social roles, the nature of the information, and the stage of the writing project.

Systems such as this ask people to develop a new or different awareness, one that can be difficult to maintain until it is internalized. For example, Quilt users must be aware when their working styles—which are often based on informal agreements—change, so that the system can be reconfigured to provide appropriate access controls. With The Coordinator [24], users need to learn about the language implications of requests and promises, because the system makes these speech acts explicit by automatically recording them in a group calendar. Both examples suggest the need for coaching. Perhaps the systems themselves could coach users, both by encouraging and teaching users the theories on which the systems are based.

Real-Time Groupware Concepts and Example

The vocabulary and ideas embodied in groupware are still evolving. In this section, we list some important terms useful for explanation and comparison of groupware systems, followed by an illustrative real-time groupware system. Our emphasis throughout the remainder of this paper is on real-time groupware. Functionality, design issues, and usage experience of GROVE, a real-time group text editor allowing simultaneous editing of private, shared, and public views of a document will also be explained.

- *shared context.* A shared context is a set of objects where the objects and the actions performed on the objects are visible to a set of users. Examples include document objects within coauthoring systems and class notes within electronic classrooms. This notion of shared context is a subset of the larger, more elusive concept of a *shared environment* discussed earlier.
- *group window.* A group window is a collection of windows whose instances appear on different dis-

The artificial intelligence (AI) approach is usually heuristic or augmentative, allowing information to accrue through user-machine interaction rather than being initially complete and structured.

play surfaces. The instances are connected. For example, drawing a circle in one instance makes a circle appear in the other instances, or scrolling one instance makes the others scroll.

- *telepointer.* A telepointer is a cursor that appears on more than one display and that can be moved by different users. When it is moved on one display, it moves on all displays.
- *view.* A view is a visual, or multimedia representation of some portion of a shared context. Different views may contain the same information but differ in their presentation (for instance, an array of numbers can be presented as a table or as a graph), or they can use the same presentation but refer to different portions of the shared context.
- *synchronous and asynchronous interaction.* In synchronous interactions, such as spoken conversations, people interact in real time. Asynchronous interactions are those in which people interact over an extended period of time such as in postal correspondence. Most groupware systems support only one of these interaction modes.
- *session.* A session is a period of synchronous interaction supported by a groupware system. Examples include formal meetings and informal work group discussions.
- *role.* A role is a set of privileges and responsibilities attributed to a person, or sometimes to a system module. Roles can be formally or informally attributed. For example, the person who happens to like to talk and visit with many people may informally take on the role of information gatekeeper. The head of a group may officially have the role of manager [37].

GROVE: A Groupware Example

The *GR*oup *O*utline *V*iewing *E*ditor (GROVE), [20], is an example of real-time groupware that illustrates some of the concepts just introduced. GROVE, implemented at MCC, is a simple text editor designed for use by a group of people simultaneously editing an outline during a work session.

Within a GROVE *session,* each user has his or her own workstation and bitmap display. Thus each user can see and manipulate one or more *views* of the text being worked on in multiple overlapping windows on his or her screen. GROVE separates the concept of a view from the concept of a viewer. A *view* is a subset of the items in an outline determined by read access privileges. A *viewer* is a group window for seeing a contiguous subset of a view. GROVE views and viewers are categorized as private, shared, and public. A *private view* contains items which only a particular user can read, a *shared view* contains items readable by an enumerated set of users, and a *public view* contains items readable by all users.

Figure 3 shows a GROVE group window—group windows provide the shared viewers for synchronous interactions among users.

In addition to displaying views, group windows indicate who is able to use the window and who is actually participating in the session at any given time. This information is provided by displaying images of the people who are members of the view (or simply printing their names if their images are not available) along the bottom border of the window. Thus as users enter or leave the session, their pictures appear and disappear in all appropriate group windows. The window in Figure 3 appears on the workstations of the three users shown along the bottom border, and each user knows that the others have joined the session. Users can modify the underlying outline by performing standard editing operations (insert, delete, cut, paste, and so on) in a group window. When this is done, all three of the users immediately see the modification. Outline items which are grey (like the last item, in Figure 3) rather than black on a particular user's screen cannot be modified by that user. Users can also open and close parts of the outline (by mousing on the small buttons on the left-hand side) or change the read and write permissions of outline items.

Participants can enter and leave a GROVE session at any time. When users enter (or reenter) a session, they receive an up-to-date document unless they choose to retrieve a previously stored version. The current context, is maintained even though changes may have occurred during their absence from the session. A session terminates when there are no remaining participants.

Design Issues and Rationale

GROVE was built as an experimental prototype to explore systems implementation issues, and to gain usage experience. We chose to build this system from scratch rather than beginning with the code of an existing editor because we wanted to understand, control, and modularize the code in particular ways. We were especially concerned with the user interface, and wanted to carefully architect the system's features and its look and feel. In keeping with the experimental nature of this tool, we chose to minimize the functionality and coding time spent on the standard editing features, and to concentrate on its groupware features. These features include the private, shared, and public group window support; the shared context present in the user interface; and the replicated architecture to allow fine-grained (keystroke level) concurrent editing and notification.

The architecture uses a local editor and replicated document at each user's workstation, and a centralized coordinator that serializes the operations of the various editors. This forced us to immediately face problems of response times, concurrent actions, and data inconsistencies. These are problems that plague real-time groupware systems in general. We have investigated this further, and using some

concepts from the distributed systems literature, have devised an algorithm for distributed concurrency control. This eliminates the need for centralized coordination as will be shown in the later section on concurrency control.

GROVE proposes an alternative style of interaction. It is designed to encourage and assist in tightly coupled interaction as opposed to the majority of systems for editing documents or doing multiuser computing. The default in GROVE is a mode where everyone can see and edit everything, and there is absolutely *no locking* while editing. New users ask "Isn't it chaotic to all edit in the same document, even the same paragraph, at the same time?" and "Why would a group ever want to edit in the same line of text at the same time?" Indeed, this editor is at the opposite extreme from most CASE systems which force a group of software engineers to lock modules and work in a very isolated and serial manner. The answer to the above questions are related to groups learning to work in new and original ways. Part of the answer is that after a learning period, it is not chaotic, but rather surprisingly useful, because social protocol mediates. The above questions imply that we can learn a lot by observing teams using this editor for *real work*. In the next subsection, we report on our observation and reflection on some of this usage.

Usage Experience

Groupware developers need to be conscious of the potential effects of technology on people, their work and interactions. A sensitivity to this dimension can make the difference between a groupware system which is accepted and used regularly within an organization, and one that is rejected [32]. Issues of user friendliness, flexibility, and technological control must be considered during design and implementation. Much can be learned from ongoing observation and empirical study of groupware systems.

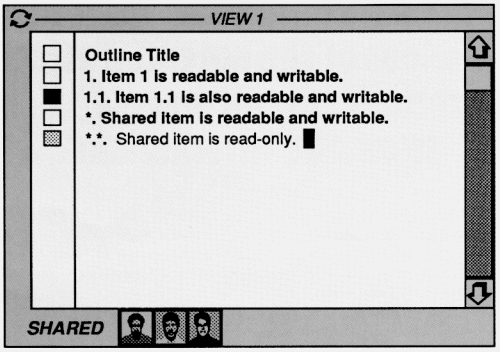


FIGURE 3. A GROVE Group Window.

GROVE has been used by several groups for a variety of design activities, from planning joint papers and presentations to brainstorming. In general, sessions can be divided into three types:

1. **face-to-face sessions** in the electronic meeting room at our lab where there are three Sun workstations and an electronic blackboard,
2. **distributed sessions** where the participants work from machines in their offices and use a conference call on speaker phones for voice communication, and
3. **mixed-mode sessions** where some of the participants are face-to-face and others are distributed.

Table 1 lists the session type, group size, and task for fifteen GROVE sessions. The early sessions were mostly face-to-face sessions where we (the GROVE creators) used the tool and fine-tuned it. More recent sessions have primarily been distributed or mixed-mode sessions

TABLE 1. *Summary of GROVE Sessions*

Session Type	Number of Users	Task
distributed	3	Identify issues in a project description.
face-to-face	3	Refine list of issues in project description.
face-to-face	3	Outline a technical report.
distributed	3	Plan a managerial presentation.
face-to-face	3	Continue planning a managerial presentation.
face-to-face	2	Plan a tutorial.
face-to-face	3	Discuss project plans.
face-to-face	3	Discuss software enhancements for a system.
face-to-face	3	Continue to discuss project plans.
face-to-face	3	Continue to discuss project plans.
mixed-mode	5	Identify similarities/differences of two projects.
distributed	3	Remote session test.
distributed	5	Brainstorm on two related topics.
distributed	5	Outline a paper.
mixed-mode	6	Outline a paper.

Groupware developers need to be conscious of the potential effects of technology on people, their work and interactions.

across thousands of miles, and have included participants at remote locations at the MCC Human Interface Program, from the University of Michigan, and from the Arthur Andersen Consulting Company. Distributed and mixed-mode sessions frequently involve as many as five or six people.

From the user's perspective, distributed editing sessions are distinctly different experiences from face-to-face editing sessions. Here are some pro and con observations regarding distributed sessions:

Increases information access. Participants in distributed sessions who reside in their offices have access to their local books and files. This sometimes allows easy access to important information that would not otherwise be available during the session. People have commented positively on the convenience, comfort, and familiarity associated with remaining in their offices.

Encourages parallel work within the group. People often divide into subgroups to work on different parts of the task by using a social protocol and shared views. Then their work is merged with the rest of the group's work by changing the access rights on the shared items to public items. This is also done in face-to-face sessions, but not as frequently as in distributed sessions (perhaps because there are more participants in a typical distributed session).

It is easy for distributed members to drop out for a while, do something else (such as work on some code in another window or get a drink), then return. This is not socially acceptable in most face-to-face situations, but is accepted in distributed sessions.

Makes discussion more difficult. Distributed sessions have a noticeably different communication pattern from face-to-face sessions. Because our phones are not full-duplex, only one person's voice is transmitted at a time. Consequently, people tend to take turns and are unusually polite—if they are impolite or uncooperative, remarks get cut off and the discussion is incomprehensible.

Makes group focus more difficult, requiring more concentration. People have commented that in general, face-to-face sessions feel shorter, seem to accomplish more in less time, and are frequently more exhilarating. In contrast, distributed and mixed-mode sessions seem to require more concentration and are more tiring. Since discussion is more difficult when some of the group members are distributed, people appear to work harder (i.e., they make a conscious effort) to get and give feedback.

Cuts down on social interaction. Distributed sessions tend to be more serious. Since there is less interchange about nontask-related topics, people tend to focus on the task immediately. The effect is a possible efficiency gain from time saved and a possible loss from social needs.

Most of the face-to-face sessions seem to have more intense, richer interactions, but we think the reasons are deeper than simply the ability to look directly at other participants. Group members rarely look directly at each other during face-to-face sessions, but being in the same room seems to increase the awareness of other members' activities to the point where highly cooperative work can be done. Most of the GROVE cooperative usage techniques have emerged in the face-to-face sessions, then have been used again in the distributed sessions because they were successful in the face-to-face environment.

In addition to comparing distributed with face-to-face sessions, it is interesting to compare group editing (in the synchronous or real-time sense) with single-user editing. Our observations regarding group editing are:

Can be confusing, unfocused, and chaotic. Many things can be going on at once. Several people may be busy in different parts of the outline. At times someone starts wordsmithing a public item while another is still working on it. Since GROVE does not provide a telepointer or other explicit turn-taking mechanisms, actions on the public view (such as scrolling or opening and closing items) are generally disruptive unless accompanied by some verbal explanation. Without verbal explanations, such as "Let's scroll to the next page" or "I'm opening line 2," one wonders "Who is doing this?" and "Why is this being changed?"

Collisions are surprisingly infrequent. Awareness of others' activities is frequently at a subconscious level. As one user expressed it, "During the brainstorming phase, I remember feeling that I was totally occupied with entering my own thoughts as fast as I could. I didn't feel at the time that I was paying much attention to what others were doing—but I know I was . . . First of all, there was very little duplication (most of the items were fresh material), so I must have been reading others' contributions without being aware of it. Secondly, there

were very few collisions with people working in the same item at the same time—I was aware of where others were working and steered clear of their space."

Can be efficient. Group editing provides many opportunities for parallel work. The most interesting cooperation patterns also involve an agreed-upon social protocol for using the tool. For example, GROVE does not have an easy way to move a subtree: one group's protocol was that one person should create new empty items where he or she wanted to move the existing lines, then each person took responsibility for cutting and pasting certain agreed-upon lines to new locations in the outline. The group accomplished the subtree move in less time than if one person had done it alone.

Can help prevent information loss, leading to a tangible group product. All the groups observed have produced significant outlines at the end of their GROVE sessions. These outlines are *group compositions that emerge out of the contributions of individuals*. The mechanism for generating the outline is a fascinating process which can consist of any of the following actions:

- **independent entry**—a user enters information while paying little attention to what is already there or what is being discussed,
- **reflective entry**—a user comments on, appends to, or modifies what has already been entered (perhaps by other users),
- **consensus entry**—as the result of discussion the group decides on an appropriate entry or modification,
- **partitioned entry**—the group assigns particular members to refine or reorganize particular parts of the outline, and
- **recorded entry**—a user paraphrases what is being discussed verbally.

This variety of contribution styles has two effects. First, there is little information loss (as compared with having a single person enter information), and consequently all groups have a significant, tangible product at the end of their sessions. The production of tangible output leads to interactions with high satisfaction/productivity ratings. Second, different groups tend to use the tool in different ways, perhaps adapting it to how they already work or experimenting with new formats.

Can make learning a natural aspect of tool use. Since people are using the same tool at the same time for a shared purpose, when one has a question, friendly help is right at hand. The shared context makes the exchange between requester and provider efficient and relevant.

An unexpected finding is that GROVE users say they now find using single-user tools frustrating. Once one has experienced the flexibility and support provided by a groupware tool, one wants groupware features in all tools. For example, one group had a distributed session in which they used a document-processing system to review slides for a joint talk. This system was basically a single-user tool, despite its shared desktop feature. People could not edit slides on the spot and effect a shared view of the slide. They were constantly saving and closing-and-reopening document files. There was no support for multiple writers—whoever saved last was what the system remembered. Although this system had powerful graphics and formatting capabilities, it was not adequate for the task at hand and users missed GROVE's collaborative editing features.

Design Issues

Groupware systems of the future will probably incorporate contributions from most, if not all, of the five disciplines of study previously outlined. Furthermore, the groupware designer will increasingly be called on to grapple with several important issues that bear directly on a system's success. Researchers are currently exploring methods and techniques for resolving these issues, but many key research problems remain to be solved. This section focuses on groupware research, describing the problems that continue to face groupware designers and developers. The emphasis of this section is on real-time groupware designed for use by small- to medium-sized groups. We focus on this form of groupware since we feel it is here that technical challenges faced by groupware designers are most apparent.

Group Interfaces

Group interfaces differ from single-user interfaces in that they depict group activity and are controlled by multiple users rather than a single user. One example of a group interface is the GROVE group window illustrated in Figure 3. Other examples include interfaces to real-time computer conferencing systems and to multiplayer games.

Group interfaces introduce design problems not presented by single-user interfaces. A basic problem is how to manage complexity: multiple users can produce a higher level of activity and a greater degree of concurrency than single users, and the interface must support this complex behavior.

Other important questions are: What single-user interface techniques and concepts are useful for constructing group interfaces? Where do they fail, pointing to the need for new concepts? For example, is something like a scrollbar useful when it can be manipulated by more than one person, or is it simply too distracting?

WYSIWIS Issues

One approach to constructing group interfaces is known as WYSIWIS [78]. This acronym stands for "What You See Is What I See" and denotes interfaces in which the shared context is guaranteed to appear the same to all par-

ticipants. The advantages of WYSIWIS are a strong sense of shared context (e.g., people can refer to something by position) and simple implementation. Its major disadvantage is that it can be inflexible.

Experience has shown that users often want independent control over such details as window placement and size, and may require customized information within the window. The contents of the GROVE window in Figure 3, for example, vary among users in that color indicates user-specific write permissions (i.e., black text is read/write, gray text is read-only). This is an example of *relaxed* as opposed to *strict* WYSIWIS. Stefik et al. [78] have suggested that WYSIWIS can be relaxed along four key dimensions: display space (the display objects to which WYSIWIS is applied), time of display (when displays are synchronized), subgroup population (the set of participants involved or affected), and congruence of view (the visual congruence of displayed information).

Group Focus and Distraction Issues

A good group interface should depict overall group activity and at the same time not be overly distracting. For example, when one user creates or scrolls a group window, opens or closes a group window, or modifies an object another person is viewing/working on, other users can be distracted.

This points up a fundamental difference between single-user and multiuser interfaces. With single-user interfaces, users usually have the mental context to interpret any display changes that result from their actions. As a result, the sudden disappearance of text at the touch of a button is acceptable; in fact, much effort goes toward increasing the system's responsiveness. By contrast, with group interfaces, users are generally not as aware of others' contexts and can less easily interpret sudden display changes resulting from others' actions.

What is needed are ways to provide contextual clues to the group's activity. A simple solution is for participants to audibly announce their intentions prior to taking action—suitable in some situations but often burdensome. A promising alternative is to use real-time animation to depict smoothly changing group activity. For example, text could materialize gradually or change in color as it is entered. This approach, however, introduces a new set of problems. First, animation is computationally expensive and requires specialized workstation hardware. Second, it is difficult to find visual metaphors that are suitable for animating operations, although work on artificial realities and responsive environments [54, 55] seems promising. Finally, any solution to this problem must take into account the dual needs for speed and continuity: the system's real-time responsiveness to the user making changes must not be sacrificed for the smooth, continuous notification to other users.

Issues Related to Group Dynamics

Group interfaces must match a group's usage patterns. Single-user text editors often rely on simple interfaces; characters appear and disappear as they are inserted and deleted. Multiuser text editors, must contend with a diversity of usage patterns as we observed with GROVE. The text was generated as independent, reflective, consensus, partitioned, and recorded entries and, therefore required much richer interfaces.

An experimental *cloudburst* model of multiuser text editing illustrates some needed group interface techniques. This model applies two techniques and is illustrated in Figure 4.

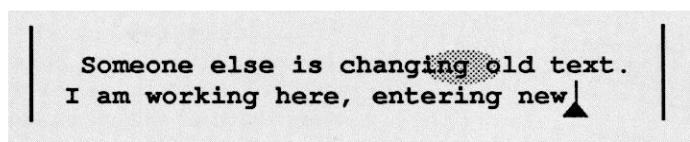
First, the text is aged so that recently entered text appears in bright blue and then gradually changes to black. Second, while textual modifications (insertions and deletions) are immediately visible to the person who initiates them, they are indicated on other users' displays by the appearance of clouds over the original text. The position and size of a cloud indicates the approximate location and extent of the modification. When a user has stopped typing for some time, the clouds on his or her display disappear and the new text is displayed, first in blue and gradually changing to black. The rationale for this interface is that an active user is only marginally interested in others' changes, which should therefore be indicated subtly and not disruptively. By the same token, when the changes are merged, everyone should be made aware of their contents.

Issues Related to Screen Space Management

Screen space is a limited resource in single-user applications, but it is even more of a problem with group interfaces in which each user can create windows that appear on other users' screens. Techniques for managing window proliferation are needed.

One approach is to aggregate windows into functional sets, or *rooms,* each of which corresponds to a particular task [9, 61]. Participants can move from room to room or be *teleported* by other users. When a room is entered, the windows associated with that room are opened.

FIGURE 4. Portion of an Editing Window Using the Cloudburst Model.



A second approach is to let one of the users bear some of the burden of maintaining window order. The LIZA system [25] provides a monitor tool, for example, which allows one user to open and close windows used by participants. This approach is particularly useful with inexperienced users.

Issues Related to Group Interface Toolkits

Single-user interface technology has matured significantly during the past decade. The advances can be attributed in part to the work on user interface management systems (see [60] for a summary) and in part to the proliferation of window systems and their interface toolkits.

Many of these single-user interface concepts can be generalized to multiuser interfaces. Group windows are one example, telepointers another. Several questions remain open, because there is little experience with these generalized techniques. Should there be group windows for subgroups? Should there be multiple telepointers for the multiple subgroups? What are the intuitive ways to share telepointers? Experience with showing all users' cursors on every screen suggests that groupware developers must be careful not to clutter the screen or overload the participants [78]. The point is that group interface toolkits must not simply be extensions of existing toolkits; rather, they must introduce new constructs that better accommodate shared usage.

Group Processes

Some well-defined tasks, such as code walk-throughs, require the participation of a set of users and are called *group processes*. Group processes offer increased synergy and parallelism, but the required coordination overhead can burden the group and dampen its effectiveness. Groupware technology seeks to enhance the benefits while minimizing the overhead.

Group Protocols

Protocols are mutually agreed upon ways of interacting. These protocols may be built into the hardware and software, called *technological protocols*, or left to the control of the participants, called *social protocols*. Examples of technological protocols are the floor control mechanisms in several conferencing systems [1, 27, 56]. These systems can only process one user's input requests at a time, imposing on participants a group process of turn-taking.

Alternatively, control of the group process can be left to the group's social etiquettes which are mutually understood and agreed upon, but not enforced by the groupware system. Social protocols include formal rules or policies, such as *Robert's Rules of Order*, and less formal practices, such as polite turn-taking or hand-raising. In GROVE, social protocols control the use of public windows. For example, anyone can scroll a public window at will, but a group quickly learns that this is disruptive unless accompanied by a verbal explanation along the lines of "Let's scroll to the next page."

Each approach to group processes has advantages and disadvantages. Leaving the processes to social protocols encourages collaboration: the group must develop its own protocols, and consequently the groupware itself is more adaptive. Social protocols (in particular, ad hoc protocols), however, can be unfair, distracting, or inefficient. In contrast, embedding a group process in software as a technological protocol ensures that the process is followed, provides more structure to the group's activity, and assists less experienced users. Technological protocols can be overly restrictive: a group's idiosyncratic working style may not be supported, and the system can constrain a group that needs to use different processes for different activities.

Group Operations

At times, it is appropriate and insightful to view the work of multiple people as a single operation. We call the resultant operations *group operations*. There are many cases of groups accomplishing a task with more speed and accuracy than would be possible by a single individual. Examples include basketball teams, and fire-fighting teams. In other cases the complex procedures carried out by a group are easier to understand if they are not divided into specific tasks performed by specific individuals.

Group operations occur in both synchronous and asynchronous situations. Office procedures present an asynchronous situation and have been studied extensively in the context of the office information systems [5, 13, 83]. Problems associated with supporting these procedures include the following: organizational knowledge, exceptions, coordination and unstructured activity. Knowledge of an organization's structure, history and goals, is useful when following office procedures [5], yet this knowledge is volatile and difficult to specify. Exceptions are frequent since offices are *open systems* [33]; in particular, they contain incomplete and partial information about their day-to-day activities, making it impossible to identify all the situations encountered by an office procedure. Office procedures consist of many parallel asynchronous tasks related by temporal constraints. There is a need for coordination—a mechanism for informing users of required tasks and reminding them of commitments. Finally, since office procedures are not entirely routine, unstructured activities, such as planning and problem solving, can occur at various points within an office procedure [70].

Synchronous group operations are one of the characteristics distinguishing groupware from other systems. The problems described above for asynchronous group operations also apply in the synchronous realm. This can be illustrated by considering a hypothetical vote tool intended for small groups. Suppose the tool functions as follows:

When a user activates the tool, a window containing a type-in area and "Start Vote" and "Stop Vote" buttons appears on that person's display. After this user enters the issue to be voted on and selects "Start Vote," a group window appears on all session participants' displays. The group window contains four buttons for voting ("Yes," "No," "Undecided," and "Uncast"), and a bar chart showing the tallies of the participants' votes.

The following paragraphs refer to this tool in discussions of the issues involved in supporting synchronous group operations.

Organizational and Social Factors. It is easy to build a tool with the above functionality; the difficulty lies in designing it to be useful in a number of different situations. The tool allows participants to change their votes, displays partial results, lets anyone pose an issue for voting, and provides anonymity (unless the users can see each others' actions). How closely this functionality matches a given group's needs depends on both organizational factors (e.g., whether it is a group of peers or a stratified, and perhaps less democratic, group) and social factors (e.g., how open or trusting the group is). In general, specializing a tool to meet a group's particular needs requires *group* knowledge (e.g., user and group profiles) as well as *organizational knowledge.*

Exceptions and Coordination. The voting tool example also points out the need for exception handling and coordination in synchronous group operations. Typical exceptions occur when a noncooperative user fails to complete his or her role in the operation, or when the group composition changes (a person unexpectedly leaves or enters during a vote). Coordination is necessary since group operations impose obligations on the participants and response times vary. A simple solution is to let the group resolve such difficulties using alternative communication channels, such as audio. The system should at least help detect problems, however, (e.g., by monitoring the progress of vote) and allow dynamic reconfiguration of the operation's parameters (e.g., changing role assignments or group size).

Integration of Activity Support. Asynchronous and synchronous operations are complementary subparts of larger tasks or activities. For example, system design projects include both high-level asynchronous tasks, such as requirements analysis, and synchronous activity, such as face-to-face meetings. A meeting proceeds in a largely unstructured way, but it can contain islands of structured synchronous operations—such as voting or brainstorming. This calls for integrating support for structured/unstructured activity on the one hand and for synchronous/asynchronous activity on the other. For instance, our voting tool should store vote results so that the group can use the results in the context of other tools and activities. In other words, the designer of group process support tools should look beyond the group and account for factors such as the group's goals and its place in the larger context of the organization or society.

Concurrency Control

Groupware systems need concurrency control to resolve conflicts between participants' simultaneous operations. With a group editor such as GROVE, for example, one person might delete a sentence while a second person inserts a word into the sentence. Groupware presents a unique set of concurrency problems, and many of the approaches to handling concurrency in database applications—such as explicit locking or transaction processing—are not only inappropriate for groupware but can actually hinder tightly coupled teamwork.

The following lists some of the concurrency-related issues facing groupware designers.

- **Responsiveness**—Interactions like group brainstorming and decision making are sometimes best carried out synchronously. Real-time systems supporting these activities must not hinder the group's cadence. To ensure this, two properties are required: a short *response time,* or the time it takes for a user's own interface to reflect his or her actions; and a short *notification time,* which is the time required for these actions to be propagated to everyone's interfaces.
- **Group Interface**—Group interfaces are based on techniques such as WYSIWIS and group windows, which require identical or near identical displays. If the concurrency control scheme is such that one user's actions are not immediately seen by others, then the effect on the group's dynamics must be considered and the scheme allowed only if it is not disruptive. A session's cohesiveness is lost, for instance, when each participant is viewing a slightly different or out-of-date version.
- **Wide-Area Distribution**—A primary benefit of groupware is that it allows people to work together, in real time, even when separated by great physical distances. With current communications technology, transmission times and rates for wide-area networks tend to be slower than for local area networks; the possible impact on response time must therefore be considered. In addition, communications failures are more likely, pointing out the need for resilient concurrency control algorithms.
- **Data Replication**—Because a real-time groupware system requires short response time, its data state may be replicated at each user's site. Many potentially expensive operations can be performed locally. Consider, for instance, a joint editing session be-

tween a user in Los Angeles and one in New York. Typically, each user would be working in a shared context with group windows. If the object being edited is not replicated, then even scrolling or repairing window damage could require communication between the two sites—leading to a potentially catastrophic degradation in response time.

- **Robustness**—Robustness refers to the recovery from unusual circumstances, such as component failures or unpredictable user actions. Recovery from a site crash or a communications link breakdown—typical instances of component failure—is a familiar concern in distributed systems and a major one in groupware. Groupware must also be concerned with recovery from user actions. For example, adding a new user to a set of users issuing database transactions is not normally problematic—but adding a participant to a groupware session can result in a major system reconfiguration. The system's concurrency control algorithm must adapt to such a reconfiguration, recovering easily from such unexpected user actions as abrupt session entries or departures.

We will now describe several concurrency control methods. Of particular interest are techniques useful to real-time groupware, because real-time systems exaggerate the concurrency problems we have just outlined. The discussion begins with traditional distributed systems techniques and ends with the newer groupware approaches, which strive for greater freedom and sharing.

Simple Locking

One solution to concurrency is simply to lock data before it is written. Deadlock can be prevented by the usual techniques, such as two-phase locking, or by methods more suited to interactive environments. For example, the system might visually indicate locked resources [58], decreasing the likelihood of requests for these resources.

Locking presents three problems. First, the overhead of requesting and obtaining the lock, including wait time if the data is already locked, causes a degradation in response time. Second, there is the question of granularity: for example, with text editing it is not clear what should be locked when a user moves the cursor to the middle of a line and inserts a character. Should the enclosing paragraph or sentence be locked, or just the word or character? Participants are less constrained as the locking granularity increases, but fine-grained locking adds system overhead. The third problem involves the timing of lock requests and releases. Should the lock in a text editor be requested when the cursor is moved, or when the key is struck? The system should not burden users with these decisions, but it is difficult to embed automatic locking in editor commands. If locks are released when the cursor is moved, then a user might copy text in one location, only to be prevented from pasting it back into the previous location. The system, in short, hinders the free flow of group activity.

More flexible locking mechanisms have been investigated and reported in the literature. Tickle locks [28] allow the lock to be released to another requester after an idle period; soft locks [17] allow locks to be broken by explicit override commands. Numerous other schemes notify users when locks are obtained or conflicting requests submitted.

Transaction Mechanisms

Transaction mechanisms have allowed for successful concurrency control in non-real-time groupware systems, such as CES [28] and Quilt [22, 57]. For real-time groupware, these mechanisms present several problems. Distributed concurrency control algorithms, based on transaction processing, are difficult to implement, incurring a cost in user response time. Transactions implemented by using locks lead to the problems described above. Other methods, such as timestamps, may cause the system to abort a user's actions. (Only user-requested aborts should be shown by the user interface.) Generally, long transactions are not well-suited to interactive use, because changes made during a transaction are not visible to other users until the transaction commits. Short (e.g., per-keystroke) transactions are too expensive.

These problems point to a basic philosophical difference between database and groupware systems. The former strive to give each user the illusion of being the system's only user, while groupware systems strive to make each user's actions visible to others. Shielding a user from seeing the intermediate states of others' transactions is in direct opposition to the goals of groupware. There has been some work on opening up transactions [4], but the emphasis of this work has been on coordinating nested transactions and not on allowing for interactive data sharing.

Turn-Taking Protocols

Turn-taking protocols, such as floor control, can be viewed as a concurrency control mechanism. The main problem with this approach is that it is limited to those situations in which a single active user fits the dynamics of the session. It is particularly ill-suited for sessions with high parallelism, inhibiting the free and natural flow of information. Additionally, leaving floor control to a social protocol can result in conflicting operations: users often err in following the protocol, or they simply refuse to follow it, and consequently, several people act as though they have the floor.

Centralized Controller

Another concurrency control solution is to introduce a centralized controller process Assume that data is replicated over all user workstations. The controller re-

ceives user requests for operations and broadcasts these requests to all users. Since the same operations are performed in the same order for all users, all copies of the data remain the same.

This solution introduces the usual problems associated with centralized components (e.g., a single point of failure, a bottleneck). Several other problems also arise. Since operations are performed when they come back from the controller rather than at the time they are requested, responsiveness is lost. The interface of a user issuing a request should be locked until the request has been processed; otherwise, a subsequent request referring to a particular data state might be performed when the data is in a different state.

Dependency-Detection

The dependency-detection model [79] is another approach to concurrency control in multiuser systems. Dependency detection uses operation timestamps to detect conflicting operations, which are then resolved manually. The great advantage of this method is that no synchronization is necessary: nonconflicting operations are performed immediately upon receipt, and response is very good. Mechanisms involving the user are generally valuable in groupware applications, however, any method that requires user intervention to assure data integrity is vulnerable to user error.

Reversible Execution

Reversible execution [73] is yet another approach to concurrency control in groupware systems. Operations are executed immediately, but information is retained so that the operations can be undone later if necessary. Many promising concurrency control mechanisms fall within this category. Such mechanisms define a global time ordering for the operations. When two or more interfering operations have been executed concurrently, one (or more) of these operations is undone and reexecuted in the correct order.

Similar to dependency-detection, this method is very responsive. The need to globally order operations is a disadvantage, however, as is the unpleasant possibility that an operation will appear on the user's screen and then, needing to be undone, disappear.

Operation Transformations

A final approach to groupware concurrency control is operation transformation. Used in GROVE, this technique can be viewed as a dependency-detection solution with automatic, rather than manual, conflict resolution.

Operation transformation allows for high responsiveness. Each user has his or her own copy of the GROVE editor, and when an operation is requested (a key is typed, for example), this copy locally performs the operation immediately. It then broadcasts the operation, along with a *state vector* indicating how many operations it has recently processed from other workstations. Each editor-copy has its own state vector, with which it compares incoming state vectors. If the received and local state vectors are equal, the broadcast operation is executed as requested; otherwise it is *transformed* before execution. The specific transformation is dependent on operation type (for example, an insert or a delete) and on a log of operations already performed [19].

Other System Issues

As this article has shown, groupware encompasses a wide range of systems—from relatively straightforward electronic mail systems to state-of-the-art, real-time, multiuser tools. Regardless of a system's place on the groupware spectrum, groupware designers face a common set of implementation issues. Some of these issues are described in this section.

Communication Protocols

Effective communication is vital to successful groupware. Unfortunately, current communications technology is not as fully capable of supporting groupware as one might hope.

First, fully integrated data communications and digitized audio/video is not universally available. Groupware developers need protocols that account for the differing requirements of the various media. With audio or video, for example, the occasional loss of data is not disastrous, but a short transmission time is crucial. Additionally, the telephone and the workstation need to be integrated at the system level. Existing prototypes, such as the Etherphone™ [82], are promising, but there is no single network and addressing scheme with an inclusive protocol suite that is accepted as a standard.

A second problem is inadequate support for multiparty communication [73]. Real-time computer conferences often require that messages be sent to a specific set of addresses; such restricted broadcasts are called *multicasts*. Current protocols, whether virtual circuit or datagram based, are better suited for communication between two parties than for general multicasts.

Finally, standardization of data exchange formats is essential if groupware systems are to be useful across organizational boundaries. The office document architecture [41] and other information exchange protocols are steps in this direction.

Access Control

Access control determines who can access what and in what manner. Effective access control is important for groupware systems, which tend to focus activity and to increase the likelihood of user-to-user interference. Theoretical and applied research on protection structures, such as capability lists, has dealt only with non-real-time multiuser systems where users are not tightly coupled [23]. These results need to be thought about in the context of groupware's requirements.

Groupware's access control re-

Effective access control is important for groupware systems, which tend to focus activity and to increase the likelihood of user-to-user interference.

quirements have been described in other literature [27]. For example, if a group task is viewed in terms of its participants' roles, access constraints are usefully specified in terms of roles rather than individuals. Access permissions are not static, but can be granted and revoked. A system can simplify the process of obtaining appropriate access rights by supporting negotiation between parties.

Groupware's requirements can lead to complex access models, a complexity that must be managed. Since access information changes frequently, there must be *lightweight* access control mechanisms that allow end-users to easily specify changes. User interfaces should smoothly mesh the access model with the user's conceptual model of the system. Changing an object's access permissions should, for example, be as easy as dragging the object from one container to another.

Notification

In a single-user environment, it is important to notify the user when constraints are being violated, or when automatic operations provoke triggers or alerters. Notification is even more vital in a multi-user environment, because users must know when other users make changes that affect their work. This points out the need for a *notification mechanism*—a way of alerting and modifying one user's interface in response to actions performed by someone at another interface.

In synchronous interactions, *real-time notification* is critical; in fact, notification and response times should be comparable. There are different granularities of notification; at the finest level, any user action—keystrokes, mouse motion—results in notification. For example, GROVE is based on keystroke-level notification: as one user types a character, this text becomes visible to the other users. Coarser levels of notification occur as user actions are chunked into larger aggregates. A text-editing system, for instance, could notify once a line or paragraph is completed. Factors such as performance, group size, and task are involved in choosing an appropriate level and style of notification. In general, however, we suggest that a fine-grained level of notification is useful for groups working in a tightly coupled manner, such as when reviewing a document or jointly operating a spreadsheet. As the focus shifts from group tasks to individual tasks—leading toward more asynchronous interaction—coarser notification becomes more appropriate.

Concluding Remarks

We have shown how the conceptual underpinning of groupware—the merging of computer and communications technology—applies to a broad range of systems. We have explored the technical problems associated with designing and building these systems, showing how groupware casts a new light on some traditional computer science issues. Information sharing in the groupware context leads, for example, to unexplored problems in distributed systems and user interface design that emphasize group interaction.

Although the prospects of groupware appear bright, we must take into account a history of expensive and repetitive failure [30]. Applications such as video conferencing and on-line calendars have largely been disappointments. These failures are not simply the result of poor technology, but can also be traced to designers' naive assumptions about the use of the technology [7].

Thus, an important area not covered in this article is the social and organizational aspects of groupware design—introduction, usage, and evolution. It should be noted that frequently a tool's effect on a group is not easily predicted or well understood [46]. As mentioned earlier, the system and the group are intimately interacting entities. A substantial literature explores the impact of computer technology on organizations and individuals [34,52,53,66]. Ultimately, groupware should be evaluated along many dimensions in terms of its utility to groups, organizations and societies.

Groupware research and development should proceed as an *inter*-disciplinary endeavor. We use the word interdisciplinary as opposed to multidisciplinary to stress that the contributions and approaches of the many disciplines, and of end users, must be *integrated,* and not simply considered. It is our belief that in groupware design, it is very difficult to separate technical issues from social concerns—and that the methods and theories of the social sciences will prove critical to groupware's success.

Acknowledgments.

The authors would like to thank Les Belady, Pete Cook and Bill Curtis for encouraging and supporting groupware research at MCC. Michael Begeman, Kim Fairchild, John Fehr, Mike Graf, Bill Janssen and Tom Smith provided many contributions to MCC's early groupware projects. For their many thought-provoking conversations,

we thank Jeff Conklin, Ira Forman, Jonathan Grudin, Nancy Pennington, Steve Poltrock and Baldev Singh. We are indebted to Peter Marks, Glenn Bruns, Nancy Gore, as well as numerous colleagues at other institutions, and anonymous referees for their constructive reviews of early drafts of this article. Finally we would like to express our appreciation to those people who provided us with excellent technical support at MCC. ◻

References

1. Ahuja, S.R., Ensor, J.R., and Horn, D.N. The Rapport multimedia conferencing system. In *Proceedings of the Conference on Office Information Systems* (Palo Alto, Calif., Mar. 23–25). ACM, New York, 1988, pp. 1–8.
2. Applegate, L.M., Konsynski, B.R., and Nunamaker, J.F. A group decision support system for idea generation and issue analysis in organization planning. In *Proceedings of the First Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec. 3–5). ACM, New York, 1986, pp. 16–34.
3. Balzer, R., Process programming: passing into a new phase. In *Proceedings of the Fourth International Software Process Workshop* (Devon, UK, May 11–13). *Softw. Eng. Not.*, ACM SIGSOFT *14*, 4 (June 1989), 43–45.
4. Bancilhon, F., Kim, W., and Korth, H. A model of CAD transactions. In *Proceedings of the Eleventh International Conference on Very Large Data Bases* (Stockholm, Sweden, Aug. 21–23). Very Large Data Base Endowment, Saratoga, Calif., 1985, pp. 25–33.
5. Barber, G. Supporting organizational problem solving with a work station. *ACM Trans. Off. Inf. Syst. 1*, 1 (Jan 1983), 45–67.
6. Birrel, A.D., Levin, R., Needham, R.M., and Schroeder, M.D. Grapevine: An exercise in distributed computing. *Commun. ACM 25*, 4 (Apr. 1982), 260–274.
7. Bodker, S., Knudsen, J.L., Kyng, M., Ehn, P., and Madsen, K.H. Computer support for cooperative design. In *Proceedings of Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 377–394.
8. *Byte*. December, 1988.
9. Card, S., Henderson, D.A. The use of multiple virtual workspaces to reduce space contention in a graphical user interface. *ACM Trans. Graphics*. ACM, New York, 1987.
10. Cashman, P.M., Stroll, D. Developing the management systems of the 1990s: The role of collaborative work. In *Technological Support for Work Group Collaboration*. M.H. Olson, Ed., Lawrence Erlbaum Associates, Publishers, Hillsdale, N.J., 1989, 129–146.
11. Conklin, J., and Begeman, M. gIBIS: A hypertext tool for exploratory policy discussion. In *Proceedings of Second Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 140–152.
12. Cook, P., Ellis, C., Graf, M., Rein, G., and Smith, T. Project Nick: Meetings augmentation and analysis. *ACM Trans. Off. Inf. Syst. 5*, 2 (Apr. 1987), 132–146.
13. Croft, B.W., and Lefkowitz, L.S. Task support in an office system. *ACM Trans. Off. Inf. Syst. 2*, 3 (July 1984), 197–212.
14. Crowley, T. et.al. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work* (Los Angeles, Calif., Oct. 8–10). ACM, New York, 1990.
15. DeCindio, F., DeMichelis, G., Simone, C., Vassallo, R., Zanaboni, A.M. CHAOS as coordination technology. In *Proceedings of the First Conference on Computer-Supported Cooperative Work* (Austin, Tex, Dec. 3–5), 1986, pp. 325–342.
16. Dennis, A.R., Joey, F.G., Jessup, L.M., Nunamaker, J.F., and Vogel, D.R. Information Technology to Support Electronic Meetings. *MIS Quarterly 12*, 4 (December 1988), pp. 591–619.
17. Ege, A., and Ellis, C.A. Design and implementation of GORDION, an object base management system. In *Proceedings of the International Conference on Data Engineering* (Los Angles, Calif., Feb. 3–5). IEEE, Washington, D.C., 1987, pp. 226–234.
18. Egido, C. Video conferencing as a technology to support group work: A review of its failures. In *Proceedings of the Second Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 23–25). ACM, New York, 1988, pp. 13–24.
19. Ellis, C.A., and Gibbs, S.J. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD '89 Conference on the Management of Data* (Seattle Wash., May 2–4 1989) ACM, New York.
20. Ellis, C.A., Gibbs, S.J., and Rein, G.L. Design and use of a group editor. In *Engineering for Human-Computer Interaction*. G. Cockton, Ed., North-Holland, Amsterdam, 1990, 13–25.
21. Engelbart, D.C., and English, W.K. A research center for augmenting human intellect. In *Proceedings of the Fall Joint Computer Conference* (San Francisco, Calif., Dec. 9–11). AFIPS, Reston, Va., 1968, pp. 395–410.
22. Fish, R., Kraut, R., Leland, M., and Cohen, M. Quilt: A collaborative tool for cooperative writing. In *Proceedings of the Conference on Office Information Systems* (Palo Alto, Calif. Mar. 23–25). ACM, New York, 1988, pp. 30–37.
23. Fites, P.E., Kratz, P.J., and Brebner, A.F. *Control and Security of Computer Information Systems*, Computer Science Press, Rockville, Md, 1989.
24. Flores, F., Graves, M., Hartfield, B., and Winograd, T. Computer systems and the design of organizational interaction. *ACM Trans. Off. Inf. Syst. 6*, 2 (Apr. 1988), 153–172.
25. Gibbs, S.J. LIZA: An extensible groupware toolkit. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Austin, Tex., April 30–May 4). ACM, New York, 1989.
26. Goodman, G.O., and Abel, M.J. Collaboration research in SCL. In *Proceedings of the First Conference on Computer-Supported Cooperative Work* (Austin, Tex. Dec. 3–5). ACM, New York, 1986, pp. 246–251.
27. Greif, I., and Sarin, S. Data sharing in group work. In *Proceedings of the First Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec. 3–5). ACM, New York, 1986, pp. 175–183.
28. Greif, I., Seliger, R., and Weihl, W. Atomic data abstractions in a distributed collaborative editing system. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages*. (St. Petersburg, Fla., Jan. 13–15). ACM, New York, 1986, pp. 160–172.

29. Greif, I., Ed., *Computer-Supported Cooperative Work: A Book of Readings.* Morgan Kaufmann, San Mateo, Calif., 1988.
30. Grudin, J. Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *Proceedings of the Second Conference on Computer Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 85–93.
31. Grudin, J., Poltrock, S. Computer-supported cooperative work and groupware. Tutorial presented at the *ACM SIGCHI Conference on Human Factors in Computing Systems.* (Seattle, Wash., Apr. 2). ACM, New York, 1990.
32. Harper, R.R., Hughes, J.A., Shapiro, D.Z. Working in harmony: An examination of computer technology in air traffic control. In *Proceedings of the First European Conference on Computer-Supported Cooperative Work.* (Gatwick, London, UK, Sept. 13–15). 1989.
33. Hewitt, C. Offices are open systems. *ACM Trans. Off. Inf. Syst. 4,* 3 (July 1986), 271–287.
34. Hiltz, S.R. *Online Communities: A Case Study of the Office of the Future.* Ablex Press, 1984.
35. Hiltz, S.R., Turoff, M. *The Network Nation: Human Communication via Computer.* Addison Wesley, 1978.
36. Hiltz, S.R., and Turoff, M. The evolution of user behavior in a computerized conferencing system. *Commun. ACM 24,* 11 (Nov. 1981), 739–751.
37. Hiltz, S.R., and Turoff, M. Structuring computer-mediated communication systems to avoid information overload. *Commun. ACM 28,* 7 (July 1985), 680–689.
38. Hogg, J. Intelligent message systems. In *Office Automation,* D. Tsichritzis, Ed. Springer-Verlag, New York, 1985, pp. 113–133.
39. Holt A.W. Diplans: A new language for the study and implementation of coordination. *ACM Trans. Off. Inf. Syst. 6,* 2 (April 1988), 109–125.
40. Holt, A.W., Ramsey, H.R., and Grimes, J.D. Coordination system technology as the basis for a programming environment. *Electrical Commun. 57,* 4 (1983), 307–314.
41. Horak, W. Office document architecture and interchange formats: Current status of international standardization. *IEEE Comput. 18,* 10 (Oct. 1985), 50–60.
42. Ishii, H. Design of Team WorkStation: A realtime shared workspace fusing desktops and computer screens. In *Proceedings of the IFIP WG 8.4 Conference on multi-User Interfaces and Applications* (Heraklion, Greece, Sept. 24–26). IFIP, 1990.
43. Johansen, R. *Teleconferencing and Beyond: Communications in the Office of the Future.* McGraw-Hill, N. Y., 1984.
44. Johansen, R. *Groupware: Computer Support for Business Teams.* The Free Press, N. Y., 1988.
45. Johansen, R. *Leading Business Teams.* Addison-Wesley, Reading, Mass. (to be published 1991).
46. Johnson-Lentz, P. and Johnson-Lentz, T. Groupware: The process and impacts of design choices. In *Computer-Mediated Communication Systems: Status and Evaluation,* E.B. Kerr, and S.R. Hiltz, Academic Press, New York, N. Y., 1982.
47. Kaiser, G.E., Kaplan, S.M., and Micallef, J. Multiuser, distributed language-based environments. *IEEE Softw. 4,* 6 (Nov. 1987), 58–67.
48. Karbe, B. Ramsperger, N. Weiss, P. Support of cooperative work by electronic circulation folders. In *Proceedings of the Conference on Office Information Systems* (Cambridge, Mass., April 25–27). ACM, New York, 1990, pp. 109–117.
49. Knister, M.J., Prakash, A. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work* (Los Angeles, Calif., Oct. 8–10). ACM, New York, 1990.
50. Koszarek, J.L. et.al. A multi-user document review tool. In *Proceedings of the IFIP WG 8.4 Conference on Multi-User Interfaces and Applications* (Heraklion, Greece, Sept. 24–26). IFIP, 1990.
51. Kraemer, K.L., and King, J.L. Computer-based systems for cooperative work and group decision making. *ACM Comput. Surv. 20,* 2 (June 1988), 115–146.
52. Kraut, R.E. Social issues and white-collar technology: an overview. *Technology and the Transformation of White-Collar Work,* Erlbaum Associates, Hillsdale, Calif., 1987, 1–21.
53. Kraut, R., Egido, C., and Galegher, J. Patterns of contact and communication in scientific research collaboration. In *Proceedings of the Second Conference on Computer-Supported Cooperative Work* (Portland, Oreg, Sept. 26–28). ACM, New York, 1988, pp. 1–12.
54. Krueger, M.W. *Artificial Reality.* Addison-Wesley, Reading, Mass., 1983.
55. Krueger, M.W., Gionfriddo, T., and Hinrichsen, K. VIDEOPLACE: An artificial reality. In *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems* (San Francisco, Calif., April 14–18). ACM, New York, 1985, pp. 35–40.
56. Lantz, K. An experiment in integrated multimedia conferencing. In *Proceedings of the First Conference on Computer-Supported Cooperative Work* (Austin, Tex., Dec. 3–5) ACM, New York, 1986, pp. 267–275.
57. Leland, M.D.P., Fish, R.S., and Kraut, R.E. Collaborative document production using Quilt. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 206–215.
58. Lewis, B.T., and Hodges, J.D. Shared Books: Collaborative publication management for an office information system. In *Proceedings of the Conference on Office Information Systems* (Palo Alto, Calif., Mar. 23–25). ACM, New York, 1988, pp. 197–204.
59. Lochovsky, F.H., Hogg, J.S., Weiser, S.P., Mendelzon, A.O. OTM: Specifying office tasks. In *Proceedings of the Conference on Office Information Systems* (Palo Alto, Calif., March 23–25). ACM, New York, 1988, pp. 46–53.
60. Löwgren, J. History, state and future of user interface management systems. *SIGCHI Bulletin 20,* 1 (July 1988), 32–44.
61. Madsen, C.M. Approaching group communication by means of an office building metaphor. In *Proceedings of the First European Conference on Computer-Supported Cooperative Work* (Gatwick, London, UK, September 13–15). 1989.
62. Malone, T., and Crowston, K. What is coordination theory and how can it help design cooperative work systems? In *Proceedings of the Third Conference on Computer-Supported Cooperative Work* (Los Angeles, Calif., Oct. 8–10). ACM, New York, 1990, pp. 357–370.
63. Malone, T., Grant, K., Turbak, F., Brobst, S., and Cohen, M. Intelligent information-sharing systems

Commun. ACM 30, 5 (May 1987), 390–402.

64. Mantei, M. Capturing the capture lab concepts: A case study in the design of computer supported meeting environments. In *Proceedings of the Second Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 257–270.
65. von Martial, F. A conversation model for resolving conflicts among distributed office activities. In *Proceedings of the ACM Conference on Office Information Systems* (Cambridge, Mass., Apr. 25–27). ACM, New York, 1990, pp. 99–108
66. Olson, M.H., and Lucas, H.C. Jr., The impact of office automation on the organization: Some implications for research and practice. *Commun. ACM 25*, 11 (Nov. 1982), 838–847.
67. Opper, S. A groupware toolbox. *Byte* (December, 1988).
68. Osterweil, L. Software processes are software too. In *Proceedings of the 3d International Software Process Workshop* (Breckenridge, Colo., Nov. 17–19). Computer Society Press of the IEEE, Washington, D.C., 1986, pp. 79–80.
69. Osterweil, L. Automated support for the enactment of rigorously described software processes. In *Proceedings of the Fourth International Software Process Workshop* (Devon, UK, May 11–13, 1988). *Soft. Eng. Not*, ACM SIGSOFT *14*, 4 (June 1989), 122–125.
70. Panko, R.R. 38 offices: Analyzing needs in individual offices. *ACM Trans. Off. Inf. Syst. 2*, 3 (July 1984), 226–234.
71. Rein, G., and Ellis, C. The Nick experiment reinterpreted: implications for developers and evaluators of groupware. *Office: Tech. and People 5*, 1 (January 1990), 47–75.
72. Root, R.W. Design of a multi-media vehicle for social browsing. In *Proceedings of the Second Conference on Computer-Supported Cooperative Work* (Portland, Oreg., Sept. 26–28). ACM, New York, 1988, pp. 25–38.
73. Sarin, S., and Greif, I. Computer-based real-time conferencing systems. *IEEE Comput. 18*, 10 (Oct. 1985), 33–45.
74. Scigliano, J.A., Centini, B.A., and Joslyn, D.L. A Real-time Unix-based Electronic Classroom. In *Proceedings of the IEEE Southeastcon '87* (Tampa, Fla., April 5–8). IEEE, New York, 1987.
75. Searle, J.R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
76. Singh, B. Invited talk on coordination systems at the *Organizational Computing Conference* (November 13–14, 1989, Austin, Texas).
77. Sluizer, S., and Cashman P.M. XCP: An experimental tool for managing cooperative activity. In *Proceedings of the 1985 ACM Computer Science Conference*. ACM, New York, 1985, pp. 251–258.
78. Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tartar, D. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Trans. Off. Inf. Syst. 5*, 2 (Apr. 1987), 147–186.
79. Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., and Suchman, L. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Commun. ACM 30*, 1 (Jan. 1987), 32–47.
80. Watabe, K., et.al. A distributed multiparty desktop conferencing system and its architecture. In *Proceedings of the IEEE Phoenix Conference on Computers and Communications* (Phoenix, Ariz., Mar.). IEEE, New York, 1990, pp. 386–393.
81. Woo, C.C. SACT: a tool for automating semi-structured organizational communication. In *Proceedings of the Conference on Office Information Systems* (Cambridge, Mass., Apr. 25–27). ACM, New York, 1990, pp. 89–98.
82. Zelleger, P.T., Terry, D.B., and Swinehart, D.C. An overview of the Etherphone system and its applications. In *Proceedings of the Second IEEE Conference on Computer Workstations* (Santa Clara, Calif., Mar. 7–10). IEEE, Washington, D.C., 1988, pp. 160–168.
83. Zisman, M.D. Representation, specification, and automation of office procedures. Ph.D. dissertation, Wharton School, Univ. of Pennsylvania, Philadelphia, Pa., 1977.

Categories and Subject Descriptors: D.2.2 **[Software Engineering]:** Tools and Techniques—*user interfaces;* H.1.2 **[Models and Principles]:** User/Machine Systems—*human information processing;* H.4.3 **[Information Systems Applications]:** Communications Applications; K.4.0 **[Computers and Society]:** General

General Terms: Design, Human Factors

Additional Key Words and Phrases: Computer-Supported Cooperative Work, coordination, multiuser interfaces, organizational interfaces

About the Authors:

CLARENCE ELLIS is a senior member of the technical staff in the Software Technology Program at the Microelectronics and Computer Technology Corporation (MCC) and adjunct professor at the University of Texas. His research efforts have recently been in the areas of collaboration and coordination systems, office information systems, and distributed systems.

SIMON GIBBS is an assistant professor at the Centre Universitaire d'Informatique, University of Geneva, Switzerland. He is currently working on software information systems and multimedia programming. Author's Present Address: Centre Universitaire d'Informatique, University of Geneva, 12 Rue du Lac, Geneva 1207, Switzerland. simon@cuisun.unige.ch

GAIL REIN is a member of technical staff in the Software Technology Program at Microelectronics and Computer Technology Corporation (MCC). Her research interests are in multiuser interfaces, visual languages, distributed systems, group work dynamics, and technology transfer.

Authors' Present Address: Clarence Ellis and Gail Rein are with MCC, 3500 Balcones Center Drive, Austin, TX, 78759-6509. ellis@mcc.com, rein@mcc.com