Interacting with Computers vol 4 no 3 (1992) 364–392

Special issue on CSCW: Part 1

Human and technical factors of distributed group drawing tools

Saul Greenberg, Mark Roseman, Dave Webster and Ralph Bohnet*

Groupware designers are now developing multi-user equivalents of popular paint and draw applications. Their job is not an easy one. First, human factors issues peculiar to group interaction appear that, if ignored, seriously limit the usability of the group tool. Second, implementation is fraught with considerable technical hurdles. This paper describes the human and technical factors that have been met and handled by researchers and implementors of group drawing tools. We emphasize our own experiences building four systems supporting remote real time group interaction: GroupSketch and XGroupSketch, both multi-user sketchpads; GroupDraw, a prototype object-based multi-user drawing package, and GroupKit, a groupware toolkit. On the human factors side, we summarize empirically-derived design principles that we believe are critical to building useful and usable collaborative drawing tools. On the implementation side, we describe our experiences with replicated versus centralized architectures, schemes for participant registration, multiple cursors, network requirements, and the structure of the drawing primitives. A brief survey of other approaches to group drawing is also included.

Keywords: human–computer interaction, group drawing, shared workspace, real-time remote conferencing, computer-supported co-operative work, groupware

Most research efforts in geographically distributed conferencing have been in the field of *tele-presence* – a way of giving distributed participants a feeling that they are in the same meeting room (Egido, 1988; Johansen and Bullen, 1984; MIT, 1983). The goal of tele-presence is to transmit both the explicit and subtle dynamics that occur between participants. These include body language, hand gestures, eye contact, meta-level communication cues, knowing who is speaking and who is listening, voice cues, focusing attention, and so on. Tele-presence facilitates effective management and orchestration of remote meetings by the natural and practised techniques used in face to face meetings. *Tele-data,*

Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4.
*MPR TelTech Ltd, 8999 Nelson Way, Vancouver, British Colombia, Canada

0953-5438/92/030364-29 © 1992 Butterworth-Heinemann Ltd

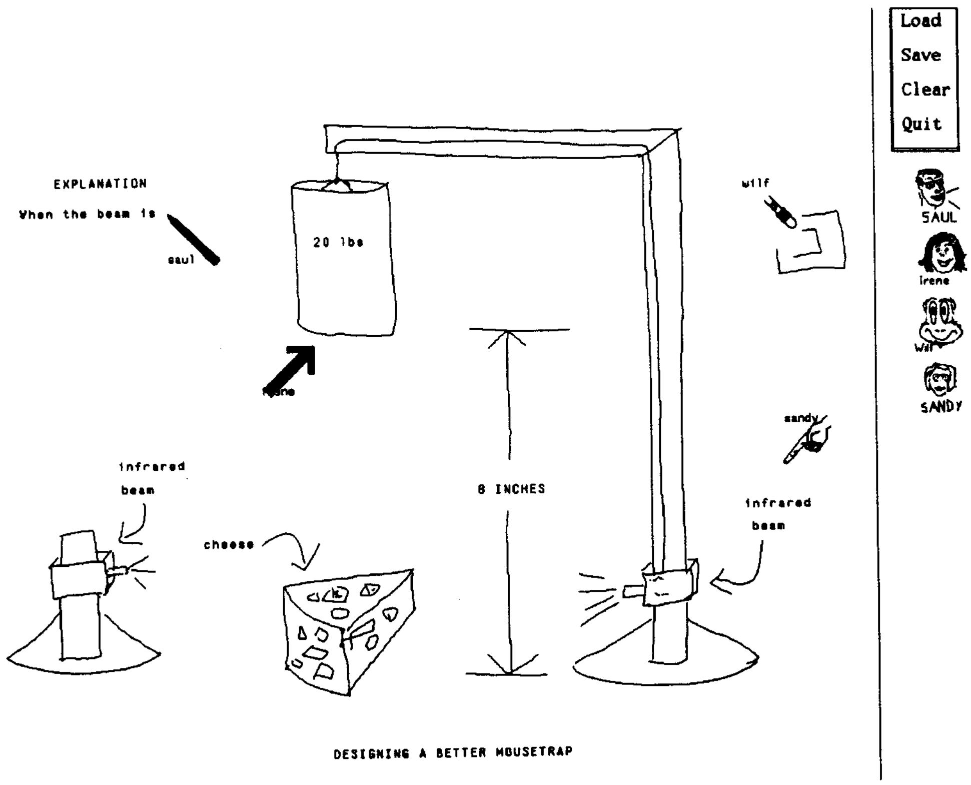
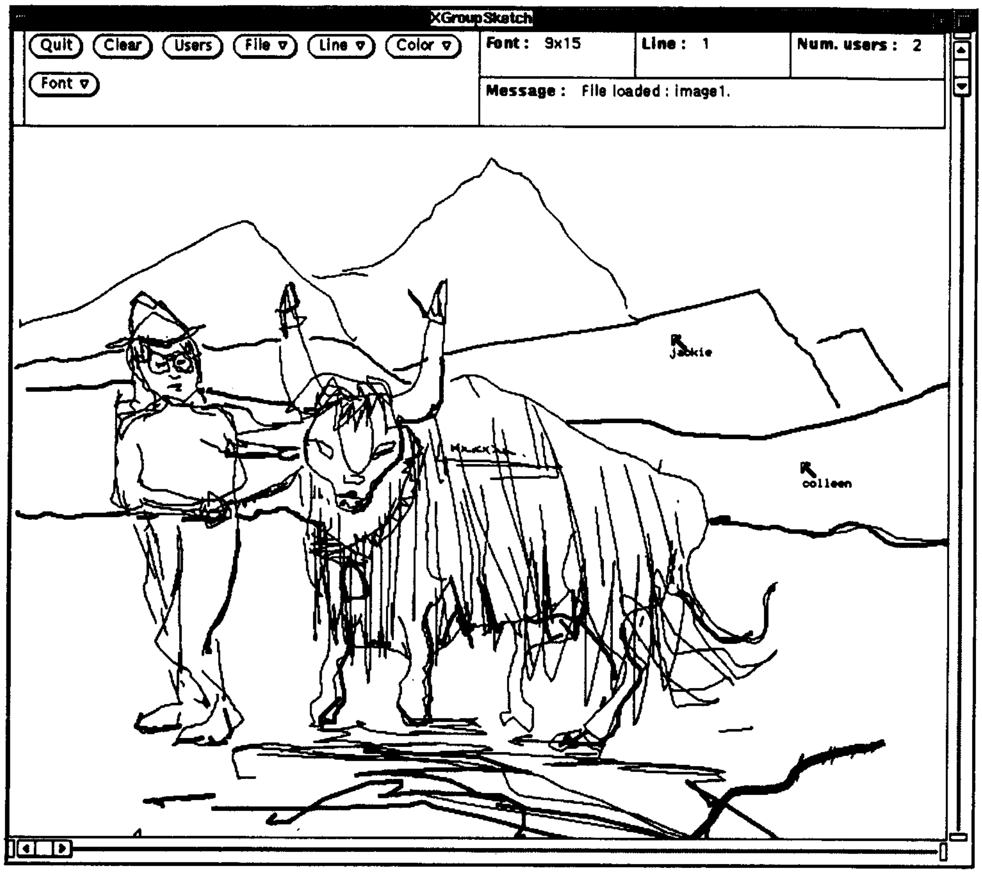


Figure 1. Sample GroupSketch session

on the other hand, allows participants at a meeting to present or access physical materials that would normally be inaccessible to the distributed group (Greenberg and Chang, 1989). These include notes, documents, plans and drawings, as well as some common work surface that allows each person to annotate, draw, brainstorm, record, and convey ideas during the meeting's progress. Given that an individual's work is commonly centred around a computer workstation, the networked computer can become a valuable medium for people to share on-line work with each other.

In this paper, we will focus on tele-data that provides small groups (2 to ~8 people) with real-time access to a shared drawing space via multi-user equivalents of the now-common paint and draw programs (Greenberg *et al.*, 1992a). We decribe both human and technical factors behind our design and implementation of four systems supporting real time group interactions.

- **GroupSketch** is a minimalist multi-user sketchpad that takes over the entire computer display (see Figure 1) (Greenberg and Bohnet, 1991). Its users all see exactly the same thing, and anyone can draw, type, erase, move their cursors around the drawing, and save or restore the image. It is a robust program that we have made available to the research community via anonymous FTP.



Font: 9x15

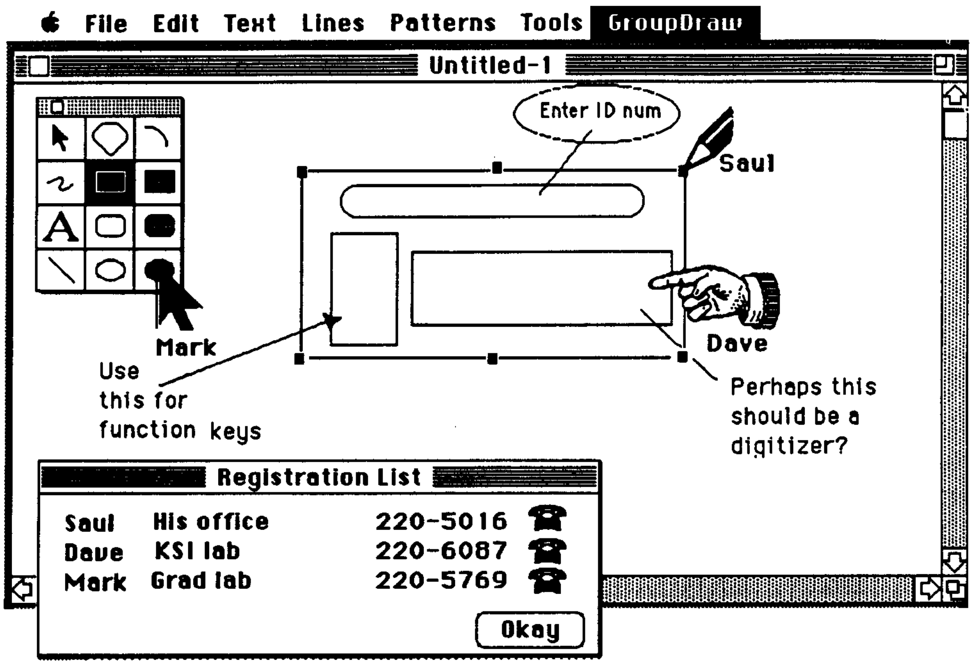
Line: 1

Num. users: 2

Message: File loaded: image1.

Figure 2. Artists Jacqui MacFarland and Colleen Campbell produce a joint drawing with XGroupSketch. The system runs within an X-window; the upper panel shows controls for font selection, line thickness, colour choice, and so on

- **XGroupSketch** is a re-implementation of GroupSketch (see Figure 2). It differs in that it runs within a standard re-sizeable X-window, has a scrollable drawing surface, and includes features such as different pen sizes and colors, font types and font sizes, and the ability to save and restore multiple screen images. It too is a moderately robust program.
- **GroupDraw** is an object-oriented drawing program with features similar to those of most structured drawing packages (see Figure 3) (Greenberg *et al.* 1992b). unlike the sketchpads, users create objects (such as lines and squares) that can then be manipulated. GroupDraw is a prototype developed as a throw-away experimental platform; it still contains bugs and is fragile, and will not be released to the community.
- **GroupKit** is a toolkit for building distributed real time groupware (Roseman and Greenberg, 1992a, 1992b). Even though GroupKit applications can go far beyond drawing, the toolkit design is based upon many of the lessons we have learnt from the applications described above. GroupKit is


File Edit Text Lines Patterns Tools GroupDraw
Untitled-1
Enter ID num
Saul
Mark
Dave
Use this for function keys
Perhaps this should be a digitizer?
Registration List
Okay

Saul	His office	220-5016	☎
Dave	KSI lab	220-6087	☎
Mark	Grad lab	220-5769	☎

Figure 3. Sample of GroupDraw session, showing work surface and registration window

now under development, and will be released to the community on completion.

These implementations are not unique. The technological breakthrough of building computational group drawing spaces was reported many years ago (Sarin and Greif, 1985), and other researchers are busy constructing their own group drawing packages and toolkits (see final section). However, we believe the time is ripe to proceed beyond mere replication and move towards empiricism, where lessons are drawn from experience and formulated as empirical design rules (Gaines, 1991). Our intent in this paper is to use our systems, concentrating on GroupSketch and GroupDraw, to highlight human factors issues critical to the design of real-time collaborative drawing tools, and to pass on the important technical factors we have encountered building such systems.

We begin the paper with a literature summary of human factors studies of face to face design teams and the resulting design principles generated from them. We then explain how these principles were incorporated into our groupware designs, and the early experiences people had using these systems. The following section describes the technical factors encountered during implementation, concentrating on the choice of a replicated over a centralized architecture, handling of participant registration, displaying of multiple cursors, the network requirements, and the underlying structure of the drawing primitives. We conclude with a brief survey and perspective of other group drawing systems.

Designing for the human factors of small group meetings

Almost every group meeting includes a process where participants express, discuss, and develop ideas. It is a creative forum where people are encouraged to present their thoughts to the group for feedback, to build upon the ideas presented by fellow members, and to solve problems. Participants typically use some large communal work surface – a group drawing area – to facilitate their interactions. Typical media now used include whiteboards, flipcharts, large sheets of paper, as well as a variety of coloured pens for drawing.

Our aim is to apply existing human factors knowledge of face-to-face meetings to the design of workstation conferencing tools supporting remote work surfaces. This section will describe how people use conventional work surfaces, the implications of the design of a workstation-based work surface, and how our systems instantiated the design recommendations. While most of the human factors knowledge is derived from studies of small group design teams brainstorming through initial design ideas, we are convinced that the findings are generally applicable to most uses of a shared work surface.

Understand collaboration

In order to design a software-based distributed work surface, we must have an adequate understanding of how traditional ones are used in face-to-face group meetings. Indeed, Grudin has identified a lack of understanding of group behaviour to be one of the reasons why groupware has not been generally successful (Grudin, 1989). He asserts that designers rely too heavily upon their own intuition, which is often based upon experiences that may not be applicable to the target group as a whole.

For example, an intuitive 'conventional' view of the communal work surface would consider it merely as a medium for creating and storing a drawing artifact (Tang, 1989). Bly (1988) disproves this naïve view. She studied two designers communicating through three different media offering different access to a drawing surface: face-to-face including a shared sketchpad; over a video link that included a view of the other person and their personal drawing surface; and over the telephone. From her observations, she asserts that the drawing process – the actions, uses, and interactions on the drawing surface – are as important to the effectiveness of the collaboration as the final artifact produced. Bly also noticed that allowing designers to share drawing space activities increases their attention and involvement in the design task. When interaction over the drawing surface is reduced, the quality of the collaboration decreases.

Tang refined Bly's findings even further through his ethnographic study of eight short small-team design sessions (Tang 1989, 1991; Tang and Leifer, 1988; also see Garner *et al.*, 1991 for further work in this area). Each team used large sheets of paper as a shared work surface and were given problems to solve, Some teams placed the paper on a table, others tacked it to a whiteboard. Even this simple difference had a profound effect on how the group used the shared work surface. Several important observations resulted:

- **Orientation:** when people sat around the table, drawings made on the table-mounted paper were oriented in different directions. Although people had greater difficulty drawing and perceiving the images, orientation proved a resource for facilitating the meeting. Because drawings faced a particular person, a context and an audience was established. Marks made by participants that were aligned to an image conveyed support and focus. People working on their own image used orientation as a 'privacy' boundary until they were ready to call in the group's attention. The group using whiteboard mounted paper did not exhibit these behaviours.
- **Proximity:** Tang noticed that when participants were huddled around the table-mounted paper, the sketchpad played a key role in mediating the conversation. This role was lessened in the whiteboard situation where people were seated several feet away.
- **Simultaneous access:** given good proximity, a high percentage (45–68%) of work surface activity around the table-top involved simultaneous access to the space by more than one person.

Tang (1989) built a descriptive framework to help organize the study of work surface activity, where every user activity was categorized according to what action and function it accomplished, as listed below.

Actions:

- *listing* produces alphanumeric notes that are spatially independent of the drawing;
- *drawing* produces graphical objects, typically a 2-dimensional sketch with textual annotations that are attached to the graphic;
- *gesturing* is a purposeful body movement that communicates specific information, e.g. pointing to an existing drawing.

Functions:

- *storing information* refers to preserving group information in some form for later recall;
- *expressing ideas* involves interactively creating representations of ideas in some tangible form, usually to encourage a group response;
- *mediating interaction* facilitates the collaboration of the group, and includes turn-taking and focusing attention.

Tang's classification of small group activities within this framework revealed that the 'conventional' view of work surface activity – storing information by listing and drawing – constitutes only around 25% of all work surface activities. Expressing ideas and mediating interaction comprised the additional ~50% and ~25% respectively. Gesturing, which is often overlooked as a work surface activity, played a prominent role in all work surface actions (around 35% of all actions). For example. participants enacted ideas by using gestures to express them, and gestures were used to signal turn-taking and to focus the attention of the group.

Implications for design of a work surface

Tang's observations led him to derive six design criteria that shared work surface tools should support. He stresses the importance of allowing people to gesture to each other over the work surface, and emphasizes that the process of creating a drawing is in itself a gesture that must be shown to all participants through continuous, fine-grained feedback. Another key point is that the tool must not only support simultaneous activity, but also encourage it by giving participants a common view of the work surface. The six design criteria plus a summary of their rationale are listed in detail in the first two columns of Table 1 (condensed from Tang 1989), and deserve special attention of the reader. These criteria form the foundation for the rest of this paper.

GroupSketch interface

GroupSketch is a simple group sketching tool that allows an arbitrary number of people to draw on a virtual piece of paper (the screen) using a mouse (Greenberg and Bohnet, 1991). It is designed around the criteria listed in Table 1. Its main features are:

- it uses a strict 'what you see is what I see' (WYSIWIS) display;
- multiple, active cursors that identify their owners are always visible on all displays;
- simultaneous interaction is fully supported, and any user can perform any action at any time;
- any user action (cursor movement or drawing), no matter how small, is immediately visible on all screens;
- drawing, gesturing and listing are as modeless as possible.

Column 3 of Table 1 provides detail of the GroupSketch user interface features and how they were designed around Tang's six criteria.

Figure 1 displays a typical GroupSketch screen with four participants engaged in a design session. On the left is the shared work surface where people draw, enter text, or gesture. Every person has a cursor labelled with their name. All participants see the same work surface on their display, and every movement of the cursor and change in the drawing is immediately visible on all displays. Each participant is represented by a unique labelled caricature located outside the work surface on the right of the screen. While audio is not directly supported, we expect a full duplex audio channel to be available by other means (e.g., conference calls).

Four action modes are supported: gesturing through cursors, drawing, typing, and erasing (Figure 1). With no mouse buttons or keyboard keys pressed, the cursor portrays the image of a pointing hand (Sandy's cursor). To draw free-style, the user depresses the left mouse button of a three-button mouse, changing the cursor from a hand to a pen (Saul's cursor). The

Table 1. **Six criteria for designing a communal work surface (condensed from Tang, 1989), and how they were met in GroupSketch**

Design criteria	Reasons	How criteria were met in GroupSketch
1. Provide ways of conveying and supporting gestural communication. Gestures should be clearly visible, and should maintain their relation with objects within the work surface and with voice communication.	• Gestures are a prominent action. • Gestures are typically made in relation to objects on the work surface. • Gestures must be seen if they are to be useful. • Gestures are often accompanied by verbal explanation.	• Physical gestures are conveyed via specialized multiple cursors synchronized with voice. • As gestures must be seen if they are to convey information, all cursors within a work surface are always visible to all participants. Cursors are also made prominent on the display by being larger than normal size. • Cursors change their shape to reflect a natural action. Four gesture modes are supported (pointing, writing, erasing, and directing attention) by distinct cursor shapes (Figure 1). The default cursor shape is the pointing hand, while the large arrow allows users to point at and direct the group's attention with greater emphasis than the normal hand. • Cursors are unique, each identifying the person it belongs to by labelling it with the user's name. In addition (and more subtly) each cursor is orientated at different angles (see Figure 1). • Cursor movements appear with no apparent delay on all displays, which means that they remain synchronized with verbal communication. • Cursors always maintain their same relative location on every display so that they retain their relation to the work surface objects.
2. Minimize the overhead encountered when storing information.	• Only one person usually records information. • Other participants should not be blocked from continuing private or group work while information is being stored.	• Any person may store a snapshot of the current work surface into their own private directories at any time. While that particular workstation display will 'freeze' for a short period, other workstations remain unaffected. Any person may restore their private images back to the public work surface at will.
3. Convey the process of creating artifacts to express ideas.	• The process of creation is in itself a gesture that communicates information. • Speech is closely synchronized with the creation process. • Artifacts in themselves are often meaningless.	• In GroupSketch, any work surface action, no matter how small, is visible with no apparent delay on all participants' screens. Every movement of the cursor, every pixel that is drawn, and every letter typed is immediately broadcast to other screens and are therefore immediately visible. • As with cursors, all details of artifact creation and manipulation are transmitted in real time and remain synchronized with accompanying speech.

continued

Table 1. *continued*

Design criteria	Reasons	How criteria were met in GroupSketch
4. Allow seamless intermixing of work surface actions and functions.	• A single action often combines aspects of listing, drawing and gesturing. • Writing and drawing alternates rapidly. • Actions often address several functions.	• The simplicity of GroupSketch allows it to have a nearly modeless interface. When no mouse buttons are depressed, the cursor is in the pointing gestural state. Drawing occurs as long as the left button is depressed, which also turns the cursor into a pen. Similarly, the right button will invoke the erase function, and the middle button the large arrow for focusing attention. Typing immediately inserts text at the current cursor location and the cursor image changes to the pen, automatically reverting back to the hand-shaped cursor after a reasonable pause in typing is detected.
5. Enable all participants to share a common view of the work surface while providing simultaneous access and a sense of close proximity to it.	• People do not see the same things when orientation differs. • Simultaneous activity is prevalent. • Close proximity to the work surface encourages simultaneous activity.	• Advantages of a common view were considered more beneficial than the lesser advantages of differing orientations. By using a WYSIWIS display and by having all cursors present, we believe GroupSketch promotes a close sense of proximity. As participants track other cursors, they naturally associate actions in the work surface with people who are executing those actions. • Simultaneity is fully supported. All participants have free and equal access to the work surface. Any one can do anything at any time.
6. Facilitate the participants' natural abilities to coordinate their collaborations.	• People are skilled at co-ordination communication. • We do not understand the co-ordinating process well enough to mechanize it.	• As GroupSketch does not enforce any style of social protocol and as all participants are in direct control of their actions, the group is free to use whatever coordination method suits them (an argument favouring this approach is presented by Dykstra and Carasik (1991)).

pen-shaped cursor also appears automatically when typing. Pressing the middle mouse button changes the cursor into a large arrow to draw participants' attention (Irene's cursor). Users can erase graphics or text in the work surface by holding down the right mouse button, which changes the shape of the cursor into an eraser (Wilf's cursor).

The menu on the right of Figure 1 allows a person to save an image privately, retrieve a previously stored image to the group display, clear the public work surface, or leave the collaboration (leaving other participants in the meeting). Menu selections and cursor movements outside the work surface are private and are not broadcast to other workstations (that is the only part of GroupSketch that is not WYSIWIS). Loading an image or clearing the work surface affects all participants' screens.

XGroupSketch is similar to GroupSketch with several important differences. First, it runs within a standard X-window (see Figure 2), and users can manipulate this window in all the usual ways. Second, it is functionally richer. Through a control panel with pull-down menus, users can choose different line styles, line colours, and fonts. Multiple images can be saved as pages in files through the file menu. A user can raise a list of all XGroupSketch participants, shown in a separate window, through the 'Users' button. Feedback is located on the top right, with a panel specifying the font, line width, the number of users, and arbitrary system messages. Third, the window is a viewport into a large drawing area, and users can scroll independently to, and work on, any part of the drawing they wish.

We have performed limited usability studies on GroupSketch under relatively informal conditions (Greenberg and Bohnet, 1991)[1]. In a typical GroupSketch scenario, participants converse and interact as they would over a shared piece of paper. Yet it is not identical to a face-to-face meeting. People tend to concentrate intently on the group work surface (they cannot see each other), not only for tele-data but for the limited sense of tele-presence provided via gestures. People focus attention to objects in the display by pointing at them or by circling objects with the cursor. Drawing and listing are both independent (one person responsible for drawing) and co-operative (multiple people working on a drawing). People can – and often do – work simultaneously on any part of the display, and anyone can be actively gesturing, creating, or editing the drawing artifact. The specific observations we had made about GroupSketch are summarized in Table 2. We feel it fair to say that GroupSketch, in spite of its minimalist functionality, is reasonably effective as a distributed work surface.

GroupDraw interface

While GroupSketch is a simple paint program where users can only make and erase marks on a bit-map surface, GroupDraw is an object-oriented drawing program with features approaching those of most structured drawing and drafting packages (such as Claris MacDraw). GroupDraw was developed as a throw-away experimental platform, and the implementation still contains bugs and is quite fragile. Still, it helped us study and select interface and architecture features for GroupKit, our toolkit for building groupware. GroupDraw supports the multiple active cursors and simultaneous activity that we have seen in GroupSketch. However, users can now create, move, resize, and delete drawing objects (rectangles, lines, circles, text, etc). Also (and similar to XGroupSketch), WYSIWIS has been relaxed to provide a scrollable and resizable work surface. This is a concession to the limited screen area available, and also provides users with room to work on private drawings. Figure 3 illustrates a GroupDraw screen with three users working on a design. The smaller registration window lists conference participants, their physical location, and their phone numbers.

The effect of having objects that can be selected and modified by different people raises a surprising number of interface design issues. The most obvious

[1]While XGroupSketch use is similar to GroupSketch, some problems accompany its added features. These will be discussed later in the GroupDraw section.

Table 2. Observation of GroupSketch use

Observation	Details
GroupSketch is very easy to learn.	People with even limited computer experience learnt GroupSketch in moments (less than a minute). We attribute this ease to its direct analogy to the paper sketchpad, the modeless nature of the system, and its simple syntax.
GroupSketch is effective.	In spite of its simplicity, GroupSketch worked. Participants were able to pursue their tasks effectively, using strategies analogous to those observed in face-to-face design meetings.
The worst part of GroupSketch is trying to draw with a mouse.	People expressed frustration when drawing with a mouse. A stylus would have been an improvement.
Increasing the number of participants in an open floor policy increases parallel activity but also decreases focused attention.	We observed much simultanous activity. As noted by Tang (1989), this comes at the price of reduced group attention. For example, when four participants were collaborating, one person commented that she found it difficult to listen to another participant when others were actively writing or drawing in the communual work area. We expect this problem to be exacerbated as group size increases. Yet most participants agreed that restricting access to the work surface or introducing turn-taking would be unacceptable.
Movement of the cursor synchronized with a participant's voice provides the greatest sense of tele-presence.	The presence of even idle cursors in the work surface was considered important by participants. People did not have serious problems distinguishing who was doing what. Still, the quality of presence did not match that of a face to face meeting. For example, we observed two occasions when visually separated but co-located participants involved in an intense discussion left their computers to speak face-to-face.
The shared work surface captured participants' attention and focused interaction.	There is a strong focus of attention on the work surface. Participants' eyes remained fixed on the shared area for long periods of time, as if they did not want to miss any of the actions occurring in the work surface. The ease of drawing and talking simultaneously around artifacts seemed to provide a focused interaction.
Participants desired greater functionality.	People familiar with computer systems wanted functionality greater than a simple sketchpad could provide. These included object-oriented drawing tools over free-hand bit-mapped sketching, editable text fields, and other features commonly available in single-user graphical packages. This finding was our main motivation for designing GroupDraw.
Intermixing listing and drawing (text and graphics) occurred frequently and naturally.	Resulting artifacts contained a good mixture of graphics and textual lists of points.
Vertical orientation of the work surface removed the physical limitations of the table top.	Users had no problem recognizing objects on the display. As people could literally draw on top of one another, we observed people working together on objects in quite close proximity (examples include multiple people erasing different parts of a single line and cooperative construction of a drawing artifact).
Saving only one image is not enough.	GroupSketch only allowed any one user to save one image at a time. This was not enough to allow rapid switching between drawings.

continued

Table 2. *continued*

Observation	Details
The work surface is too small.	The work surface quickly becomes cluttered during long design sessions, especially with larger group sizes. Larger displays, windowing strategies, or better storage and retrieval facilities are required.

is what to do when several people try to manipulate an object. The simplest strategy is to let only one person at a time acquire the object – others are prevented from accessing it. Even so, feedback must be supplied to the user differentiating the act of selection from actually acquiring the object, for there could be a time lag between these two states. In GroupDraw, users can actively grab an object and start manipulating it – the system assumes optimistically that there will be no conflict. If an acquisition conflict does occur and permission is denied to manipulate the object, the object will snap back to its original position. In practice, system response time is rapid enough that the selection/acquisition process occurs almost instantly – a rejected acquisition appears as a momentary flicker. Still, other visual cues may be necessary to indicate acquisition status during this transition period, such as grey-scale colouring of the object's handles[2].

Another issue relates to Tang's fourth design criteria (see Table 1) recommending a seamless intermixing of workspace actions and functions. In GroupSketch this was accomplished by designing a relatively modeless interface. The fact that GroupDraw users must now select from a variety of object types and go into a particular drawing mode makes this recommendation difficult to fulfil. While the time taken to switch modes is not critical in single-user systems, selecting an object from a palette or menu does detract from the fluidity of a group interaction (this problem is less serious in XGroupSketch because mode changes are infrequent). We do not yet have an acceptable solution, and consider this an area for further research.

A further concern arises from private drawings, which Tang had observed as a positive resource. These drawings are often worked on and then presented to the group at a later time. GroupDraw implements privacy in two ways. The first is by providing a scrollable drawing surface; a user can scroll and work in their own area of the screen, and then move the image to the main view (a split screen may work well here, but see below). Second, an object's 'coupling status' can be specified (Dewan and Choudhary, 1991). Here, a user can indicate whether an object can be manipulated by all others ('all of us can see and touch it'), can only be viewed by others ('I can touch it but you cannot'), or can be private ('only I can see it'). While attaching coupling status to drawing artifacts is a nice technical feature, its actual value (and usability) to real group drawing situations requires further exploration.

Scrolling deserves special attention, for it can completely disrupt the WYSI-

[2]GroupDraw actually allows multiple access to a single object. For example, two people can grab different endpoints of a line and move it. The locking described above is done on the handle level e.g. when two people try to grab the same endpoint of a line.

WIS principle. In GroupDraw and in XGroupSketch, views of different participants can now coincide, overlap partially, or be completely disjoint. The result is that a participant is no longer certain of what others can see, and comments such as "Can you see this part of the drawing?" and "Where are you?" occur in the verbal dialogue. Strategies exist to mitigate this problem. The first is to supply each user with a separate 'radar' window that shows a miniaturized animated view of the complete drawing space and the location of all participants' viewports on it (Smith *et al.*, 1989). The cost is the extra screen area occupied by the radar window and the cognitive distraction of having to refer to it. A second strategy is *view-slaving*, where the view-port of one participant (called a follower) can be enslaved to the view-port of another participant (called a leader) (Pendergast and Hayne, 1991). Whenever a leader scrolls the view, the corresponding views of the followers are scrolled as well. What makes view-slaving interesting is that all users can choose to be leaders and followers at the same time, so that anyone can scroll and WYSIWIS is maintained. Of course, 'scroll wars' become possible, where people can fight over what part of the image the group should see. However, we expect social protocol will minimize these occurrences.

In summary, our early experience with the GroupDraw interface raises more questions than it answers. We know for certain that some of the interaction techniques now found in conventional drawing programming will not transfer well to a multi-user domain.

Other human factors concerns

Three other human factors concerns are critical to the design of group drawing systems. These are:

- the need for a better sense of tele-presence;
- the ability to have a seamless melding of shared work on the computer and the desktop;
- consideration of the size of the group.

Does the tele-presence supplied by GroupSketch and GroupDraw's shared cursors, activity space, and voice channel suffice for group work? As noted in Table 2, GroupSketch users did have a strong focus of attention on the work surface, and we feel that tele-presence was adequately supported as long as people were directly interested in the drawing surface. Yet what about the times when the group work surface was not central to the discussion? We conducted several GroupSketch sessions where all users were co-located in a terminal room. When the conversation shifted to a meta-level topic not requiring the drawing surface, we noticed that many of the participants turned away from the computer screen to look at each other instead, even though the layout of this particular room made eye contact awkward. Clearly, people prefered face-to-face contact in these situations.

Smith *et al.* (1989) made a similar observation in a slightly different groupware environment, where two geographically distributed people each had two monitors. One monitor showed the computerized work surface, the other provided a 'video tunnel' – a real-time video link that allows eye contact. When the conversation shifted to meta-level discourse, the participants would turn away from the workstation and towards the video monitor.

Several other researchers are now examining what happens when the graphical work surface and the video image of the participant's faces are merged via transparent overlays into a single display (eg ClearBoard-1: Ishii and Arita, 1991; Rococco: Scrivener *et al.*, 1991). They argue that these systems promote not only eye contact between participants, but also (and perhaps more importantly) 'gaze awareness', where a participant can easily recognize what their partner is looking at on the work surface (Ishii and Kobayashi, 1992).

A second human factor concern is the barrier that exists between the work we do on our computers and the work we do on our physical desks. We have not yet reached the age where all work is computerized, and many physical items that we wish to share with our geographically-distributed colleagues cannot be easily converted into computer medium in a fast-paced meeting. One simple approach towards a seamless work environment is to allow participants to import static images of documents and other artifcats via paper scanner and video camera/frame grabber; the images can then be manipulated by the groupware (IIS Technologies, date unknown). Another approach fuses two or more live-video images of computer screens and physical desktop surfaces (Ishii, 1990). Here, participants can combine computer artifacts (eg a single-user drawing program) with desktop artifacts (e.g., an existing drawing). Participants can gesture around and interact with these artifacts using their hands and pencils (by placing them under the video camera and drawing on the surface), or through the standard computer cursor.

Finally, the size of the group will affect how the group uses a drawing surface. The studies of work surface activities mentioned in this paper examined group interactions of two to four people. Yet many face-to-face meeting involve far larger groups, and much of the technology in the group decision support system domain accommodates 10 to 40 or more people (e.g., GroupSystems, Nunamaker *et al.*, 1991). Will group drawing tools such as GroupSketch scale up? In one two-hour GroupSketch session, we had eight people working on a design task, a brainstorming task, and an ideas organization task. While we do not have any hard data on the session, the general feeling was that the tool remained useful. However, several issues were noticed. First, eight large cursors cluttered the screen. As a result, we modified GroupSketch so that the cursor size would shrink as the number of group members increased, although any explicit activity (such as drawing or erasing), would restore the active cursor to its full size for the duration of the action. With eight people, the smaller cursors[3] sufficed to provide an uncluttered screen. Participants maintained a group sense of awareness, and temporarily restoring busy cursors to their full size focused the group's attention to the active work. A second issue concerning group size is screen size – large groups need far more working space in view than contemporary bit-mapped displays will provide. Finally, larger groups

[3]Cursor sizes here were 8x8 pixels. With larger group, cursors could be shrunk to occupy only a single pixel on the screen.

appeared to spend more time on meta-level discussion. As mentioned previously, this means that the work surface alone will not suffice, and that a visual channel will probably be required as well.

Implementation experiences

GroupSketch, XGroupSketch, and GroupKit are implemented on Sun workstations running Unix and networked via Ethernet. GroupDraw is built upon the Macintosh/AppleTalk platform. The design of all systems contain two features unusual in single-user interface design. First, they are distributed programs – resulting issues are the tradeoffs between a replicated or centralized architecture; the network communication demands; and how users can dynamically register with an existing electronic meeting. Second, all systems support multiple cursors and simultaneous activity – issues here are how multiple cursors are implemented, and how the graphics primitives are structured to support multiple synchronized access.

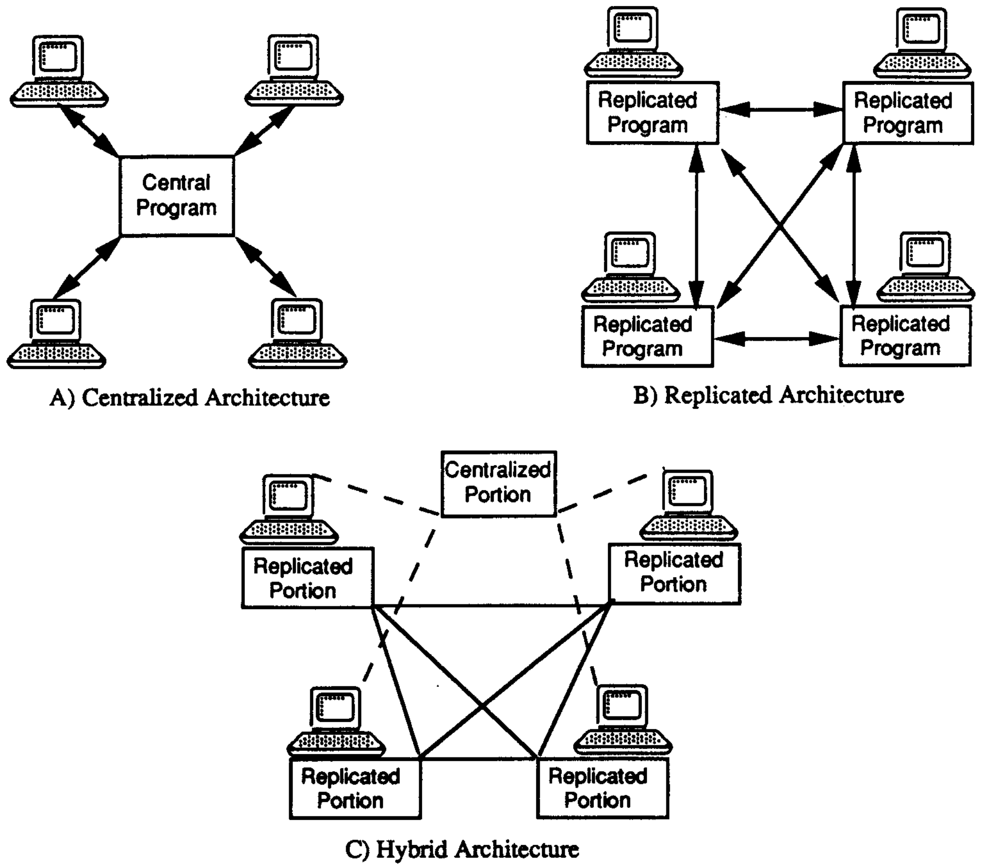
The internal architecture of GroupSketch and GroupDraw are briefly described in this section, indicating how the issues listed above were addressed. XGroupSketch and GroupKit are introduced when necessary.

Replicated versus centralized architecture

There are three architectural alternatives for constructing distributed software (see Figure 4). The *centralized* architecture contains a single central program that controls the distributed work of all users, while a *replicated* architecture executes a copy of the program at every workstation. A *hybrid* approach combines features of both. While these architectures have been contrasted previously in the design of 'collaboration-transparent' view-sharing systems (Ahuja *et al.*, 1990; Greenberg 1990; Lauwers *et al.*, 1990), many of the technical factors raised are similar to those we have seen in the collaboration-aware groupware discussed in this paper.

In the centralized approach, a single program, residing on one machine, controls all input and output to the distributed work surfaces (see Figure 4a). Server processes residing on each person's workstation are responsible only for passing user input events to the central program (e.g., mouse movements, key presses, window resizing), and for displaying output sent to it from the central program. The central program processes these events and then notifies all workstations of display updates. In effect, each workstation is nothing more than a graphical terminal/window server (e.g. the X-window server, Sheifler and Getties, 1986), while the central program is one large application program that decides what to do with all users' input and output. The advantage of a centralized scheme is that synchronization is easy, as state information is consistent since it is all located in one place. The disadvantages are that the complete system is now vulnerable to the failure (either machine or network) of the central agent, and that the central agent could become a network and processor bottleneck as all activity must be channelled through it.

In the replicated approach, there is no central program. Instead, each application program is replicated on every machine (see Figure 4(b)), and the



A) Centralized Architecture

B) Replicated Architecture

C) Hybrid Architecture

Figure 4. Schematic of a centralized, replicated and hybrid architecture

replicated programs stay synchronized by communicating directly with each other. Each replicated program is totally responsible for dealing with its local user's events, exchanging information with its counterparts on the network, and maintaining the integrity of the drawing surface. The advantages are that network and processor traffic is reduced because communication does not go through a central mediary, and that the system is more robust to network and machine failure. The drawback is that it becomes more difficult to keep the work surfaces and user requests synchronized.

Hybrid approaches are also possible. For example, the participant processes may use a central agent only for synchronization and for mediating potentially conflicting user requests. All other activities are performed within and between participant processes.

An example of a centralized architecture is WScrawl 1.0, a public domain group sketching program developed by Brian Wilson at Hewlett-Packard that runs within the X-window system. The single WScrawl process collects and decides what to do with all user events (which it collects from the X-window server) and where to display the output. Thus it has all the advantages and

Table 3. Communication protocol between GroupSketch processes

Event	Information passed
Registering a new user	Host name, port number, name of participant, caricature
Unregistering a user	ID of participant
Cursor moved	ID of participant, cursor shape, new coordinates
Drawing a line	ID of participant, start and end coordinates
Erasing a region	ID of participant, coordinates of region
Listing	ID of participant, string location, string, cursor shape, location of cursor
Clearing screen	–
Image transfer	Binary data of the work surface image

disadvantages of centralized architectures noted above. Another disadvantage arises when the system follows relaxed WYSIWIS. For example, several user actions in WScrawl 1.0, such as menu and control panel selections, are private actions that are visible only on that user's display. The central program must deal with these selections as special cases. Another example is independent scrolling (not implemented in WScrawl 1.0). If each user can scroll to different portions of the group drawing surface, the central manager cannot simply broadcast the same output directives to all participants. Rather, it must keep track of each user's view and decide what portion of that view (if any) requires updating. While the programming problems introduced by relaxed WYSIWIS are surmountable, it does add complexity to the code.

In contrast, both GroupSketch and GroupDraw use fully replicated architectures, where every workstation has its own local copy of the program. Taking GroupSketch as an example, the replicated GroupSketch processes communicate via Unix stream sockets, using only the eight primitive events listed in Table 3. On start-up, the new GroupSketch process asks the registrar demon (described in the next section) for the location of any other GroupSketch processes. If any exist, the closest process will send the new one a copy of the image, received as an 'image transfer' event. This is the only time screen output is ever sent over the communication channel. When any other local user event occurs, such as a cursor movement or line-drawing action, the process will update its own display and pass the event on to the other GroupSketch processes (see Table 3). When a remote event is received, it is interpreted and handled in essentially the same manner as a local event, e.g. remote cursors are moved, remote lines are drawn. Since the only actions that can be done by a GroupSketch process are either draw/erase on a bit-mapped surface or to move the cursor, there is no need to synchronize user activities[4].

The interaction between replicated processes in GroupDraw is more complex. As we have seen, one of the advantages of an object-based drawing system such as GroupDraw is the ability not merely to view, but to manipulate the entities in the shared workspace. However, we must ensure that the object's behaviour

[4]This is not quite true, for it is possible to get out of step. For example, if a user draws on a surface that is simultaneously being erased by someone else, the final appearance of the bit-map could look different depending upon the order in which these events arrive at each participant process. In practice, this is not a problem due to the scarcity of this occurring, and the minimal visual disruption to drawing.

is managed consistently between users so that (for instance) two people grabbing and dragging the same point on a line do not both succeed. This poses concurrency problems that are not encountered in the simpler GroupSketch situation. Central architectures can easily resolve this problem as all state information is in one place. However, there are effective approaches to dealing with this under a replicated architecture as well, which we employed in GroupDraw. We define an *owner* for each object drawn, who has final authority on all operations affecting the object. The owner of a single object is one of the replicated participant processes in the GroupDraw conferences[5], and each process may be the owner or more than one object. The responsibility of the owner is to maintain object consistency across all sections. This is best illustrated by an example.

Suppose three users: userA, userB and userC are part of a conference, each running processA, processB, and processC. Within the conference's workspace there is a line object owned by processA. UserB and userC simultaneously select the same end point of the line object – this is a situation where object consistency could be compromised as only one person should be allowed to select the object. Both processB and processC then send a message to processA requesting control of the end point. ProcessA then assigns permission on a first come, first served basis. If processB's request arrives first, processA will send a message to processB granting permission to grab the point, while processC will receive a message denying permission. The owner is then notified when the user has released the point, and the point again becomes available for selection.

Under this approach, object ownership is distributed across all participant processes; it is only the GroupDraw session as a whole that maintains full state information for its objects. The result is a fairly robust system. If a participant process leaves (e.g. when the participant leaves or their node fails), its objects are systematically transferred to other participant processes. When the last person leaves the conference, ownership need not be retained – it is relevant only during a single conference.

A variety of hybrid approaches are possible. In XGroupSketch, for example, we have minimized the need for replicated processes to track each other by moving the responsibility for registration, communication management, and even broadcasting into a centralized manager. As in GroupSketch, the replicated portion of XGroupSketch interprets all local and remote actions, and updates the display accordingly. The difference is that the replicated portion maintains only a single communication channel to the manager, which is used to send local events and to receive remote events. The manager acts as a communication hub, and is responsible for creating and maintaining links to all processes, and for broadcasting any events received.

In GroupKit, we are considering (but have not yet implemented) a hybrid approach to the distributed ownership scheme described in GroupDraw. A central process (provided as a toolkit component) will mediate concurrency,

[5]Initially, the owner will be the process supporting the participant who creates the object, whether by drawing it, restoring it from a file, or copying it from another source. The ownership may change during the course of a conference or between conferences by various means.

while the replicated processes will handle all other actions. When a user performs a possibly contentious action, such as grabbing a line's endpoint, the replicated process will ask the central process for permission. If granted, the central process will disallow other users to access that point until it has been informed that the point has been released.

There is no real answer to the question of which scheme works best. Rather, it is a set of trade-offs that revolve around programming complexity, synchronization requirements, processor speed, the number of participants expected, communication capacity and cost, and so on.

Registration

We believe that any participant should be able to join and leave the conference at any time. Yet how do people 'register' with the shared drawing session, and how is this managed internally? How does each participant process know about and adjust to the comings and goings of other participants?

In GroupSketch, a central *registrar* demon performs dynamic registration functions. The following example indicates how the registrar incorporates new participants into a GroupSketch session:

(1) An incoming participant or late-comer starts a GroupSketch process, which connects to the registrar, opens a communication port ready for other connections, and sends the port address along with the participant's name and caricature data to the registrar (see Table 3 for the protocol).
(2) The registrar acknowledges the newcomer and informs all other participants' GroupSketch processes (if any) of the newcomer's address in the network.
(3) All other GroupSketch processes (if any) connect to the newcomer, with the nearest sending in the current state of the work surface image as an 'image transfer' event. The registrar is now out of the loop.

In contrast, GroupDraw uses a distributed registration scheme. Each workstation's registrar maintains an AppleTalk socket listener. Whenever a new GroupDraw participant enters a session, it announces its arrival over the network. The announcement is heard by every registrar, and connections are made directly. The new entrant then asks one of the other participant processes to send it a display list of the current work surface.

Both schemes work reasonably well. GroupDraw's method has the advantage that it is not tied to a single central registrar. But what is most important in both schemes is that the registrar is fairly independent of the underlying application. It is a high-level toolkit component that should be re-usable in other groupware applications requiring registration. In fact, for GroupSketch we were able to re-use, in part, a registrar that we had originally designed for a completely different application (Greenberg, 1990, 1991a). We have included a registrar as a fundamental toolkit component in GroupKit.

Multiple cursors

Multiple cursors – used for gesturing and tele-presence – present a significant problem, for current window systems only support single cursors. As a result, multiple cursors must be implemented independently from the system-supplied cursors.

In GroupSketch we avoided window systems completely in favour of a graphical library allowing us to manipulate the bit-map display directly (we used Sun Microsystem's Pixrect library). Large multiple cursors are implemented directly by exclusively OR'ing bitmaps. The general algorithm for handling multiple cursors follows. After initial set-up and participant registration, the participant process executes a main loop that acts on events arriving from four different sources: the keyboard, the mouse, the registrar, and from other participant processes (via Unix sockets). A participant's local activity is signalled by keyboard events for character insertion; and by mouse events for cursor movement, drawing, and erasing. These events are then broadcast to the remote processes (see Table 3); their workspaces are then updated.

As an example of managing cursor movements, assume two GroupSketch participants: userA and userB. When userA moves the cursor, the change is updated immediately on the local screen, and the cursor moved event along with the participant id and the co-ordinates of the new position are broadcast to userB. When userB's process receives the event, it looks up the old location of userA's cursor, erases it via an XOR operation, and then redraws it in its new position. The internal table is then updated.

Reading directly from the mouse device driver and writing to the screen provided efficient and fast cursor performance in GroupSketch. However, this approach is costly in terms of implementation effort, for we had to design a graphical interface from the ground-up (e.g., to manage cursors, menus and simple drawing). We also made a design error in our choice of handling local events differently from remote events. For example, GroupSketch initially had a bug that would cause it to leave a spurious mark on the screen if one cursor is erasing on top of another person's cursor. While we understood why the problem occurred, the bug took time to remove due to the different ways local and remote cursor events were handled.

In contrast, GroupDraw's multiple cursors are built in part on top of the Macintosh interface toolkit. While we use standard Macintosh events to determine where the mouse cursor is, we do not use the toolkit routines to display the cursor. Instead, the standard cursor is made invisible, and an XOR scheme similar to the one mentioned above is used in its place. The other difference is that all local and remote events are handled in exactly the same way – the same cursor redrawing function is notified when any cursor is moved (whether local or remote), and the display is updated accordingly. While the standard cursor could have been used to display the local cursor position, we believe it is not worth the effort. First, the local cursor has to be treated as a special case from the remote cursors, leading to more complex coding and debugging problems. Second, there are often limitations and system dependencies in application cursors that may conflict with design goals. For example, our cursors are larger than the cursors provided by some window systems. Finally, we have found that our scheme worked well in practice – tracking is fairly smooth and occurs in real time. The same approach was successfully employed in XGroupSketch.

GroupKit uses an entirely different approach by providing multiple cursors as transparent overlays to any underlying (programmer-defined) application (Roseman and Greenberg, 1992a; 1992b). GroupKit is built on top of the InterViews toolkit (Linton *et al.*, 1991), and its overlay facility relies upon the InterView page glyph. A page contains one background glyph, and one or more transparent foreground glyphs, each of which may be placed independently over the background glyph. The background glyph here represents some application specific graphical component (such as a sketchpad), while the foreground glyphs contain bitmaps representing cursors. As the system cursor is tracked, the foreground glyph for the local cursor is moved, and messages are sent to the other users via the standard mechanisms. The application programmer using GroupKit gets multiple cursors for free just by including the specialized cursor page glyph. Similar to cursor management in conventional window systems, the glyph handles the basic display and users' interactions with their multiple cursors, and then passes on cursor movement events to the application. These events can be ignored (which still gives users the ability to gesture around the display) or used by the application for special purposes.

Communications

People designing real time groupware systems are concerned that the communication channel will be the primary system bottleneck, and often go to great lengths to minimize the information transmitted over the channel. We believe it has been used as a reason **not** to implement multiple cursors, and for choosing some architectural styles over others (e.g., Ahuja *et al.*, 1990; Lauwers *et al.*, 1990; Lauwers and Lantz, 1990).

Our experiences with GroupSketch, XGroupSketch and GroupDraw show that communication throughput is not a problem over typical local area networks (Ethernet runs at 10Mbps, Appletalk at around 250Kbps). Our communication requirements were modest. All our implementations use a standard stream-oriented connection that guarantees in-order, error-free delivery (via Unix Stream Sockets or Appletalk Data Stream Protocol). When we ran profiling tools on our systems, we were surprised to find that it was the processor speed that was the performance bottleneck. While the network comfortably accommodated the events we had sent over the network, the slower processors had difficulty interpreting all of them in real time.

To get a feel for the network traffic, we traced the number of packets sent by three GroupSketch participants on several short but active design sessions. On average, a total of 15–25 packets/second were transmitted, with an average packet length (ignoring packet overhead) of 5–20 bytes long. This gives a network utilization ranging from 600 to 4000 bits/second. This rate is easily handled by LANs, but could be demanding for a low-speed telephone link. About 75% of the packets sent indicated a 'cursor moved' event. We also noted that each user's network demands are unequal – active users generated many more events than inactive users.

Because, in our situation, CPU processing, and not network bandwidth, seemed to be the limiting constraint, we adopted the following philosophy. Each participant process sends out as much information as possible to the other processes. For example, cursor movement is broadcast as often as possible. On the receiving end, decisions must be made about what information to process when backlogs in the event queue occur. A slower machine, for example, might ignore all cursor updates received from a particular process except for the last. While this can result in jerky cursor motion, it is far less disconcerting than waiting for the cursor to 'catch up'.

Of course, there will still be situations where the network is the bottleneck. Delays due to latency, for example, can seriously undermine performance. Also, network software may not be designed to handle the frequent short messages used to broadcast cursor events. For example, Microsoft Windows has an induced delay in packet transmission, probably due to the way its kernel only processes interrupts after a certain arbitrary number of clock ticks (Hayne *et al.*, 1992). For pragmatic reasons, transmitting all cursor events may not be feasible, and other cursor transmission and display strategies may be required. Hayne, Pendergast and Greenberg (1992) offer four low-bandwidth alternatives to full motion cursors:

- **Point** – when the user points and presses a mouse button, a single message is sent to all remote users. The gesturing cursor will jump to the new location.
- **Point and quiver** – in order to attract the viewer's attention, the above strategy is modified so that the receiving station 'jitters' the cursor back and forth in its new position.
- **Limited motion** – Only a certain number of gesture messages per second are allowed between stations; the fewer messages sent, the jerkier the motion (e.g. the SHDR group sketching system, by Paul Dourish, Rank Xerox EuroParc).
- **Limited motion with smoothing** – After receiving a new cursor position from the sender, the receiving station computes several points in between the previous and current location, and then animates the cursor through these intermediate points.

One final point worth mentioning is the relation of bandwidth to n, the number of participants. In the current replicated architecture, the worst case network demands is a function of $n*(n-1)$, which occurs only when every participant is active on the work surface. In this case, each replicated process must broadcast its local events to all others (see Figure 4(b)). Of course, simultaneous activity from all members of larger groups is unlikely, and the average performance demands will be much smaller. It should be noted that if a multi-cast network were available, then the communication demands would be linear to the number of participants, for only a single message need be broadcast by the sender.

Drawing primitives

A structured drawing package provides its users with a set of drawing primitives: lines, circles, rectangles and so on. While these are familiar to most simple graphics packages, the property that they can be shared is still fairly novel. This section shows how object-oriented programming is used in GroupDraw to isolate most multi-user characteristics into a prototypical drawing object. Subclasses, which inherit these characteristics, need only specify the actual graphical properties of the drawable object. This prototypical object will be discussed in terms of its properties and operations. We will indicate how its subclasses can be implemented.

All drawable objects have certain properties which we isolate as much as possible into a root object called *GroupGraphicalObject*. Table 4 provides some detail of the instance variables and methods it contains. Each graphical object has an 'ownerProcess', which is the creator of the object by default. The ownerProcess serves to arbitrate contention in manipulating the object. Objects also have a *couplingStatus* (Dewan and Choudhary, 1991) that indicates the extent to which graphical objects are shared. As mentioned in the section on the GroupDraw interface, GroupDraw defines three coupling levels: private, public, and sharable. Each object is also referenced by a unique *id*, which is used in all network messages to identify the object being manipulated.

To create a new object, whether it be in response to a local or remote drawing action, we provide a standard initialization method: *initializeGroupObject()*, which will set the instance variables mentioned above. Subclasses will specialize this method to initialize any extra properties it may have e.g. zeroing out the endpoints of a line (see *LineObject*, in Table 4). For communication and storage purposes, each object must be able to construct and interpret a string representing itself (*make/sendDescription()*); this is used to save and restore images and to send update information to new users in the conference. The latter is implemented by requesting each object to tell the new user about itself.

Next, we look at operations dealing with changing the coupling status of an object. GroupDraw insists that the owner approve status changes; processes request permission by the *requestChangeStatus()* method. If the owner grants permission, the *doChangeStatus()* method will actually change the object status and broadcast the change to all participants. Table 4 lists several other methods that follow this request/do form of arbitration.

Because multiple users may select and start to drag an object asynchronously, the object may be in slightly different places on different screens. Yet each selected object will want to tell the other processes where it had been selected. Passing the precise pixel coordinates is often meaningless, since that point may not match its partner on the remote object. Instead, we define a 'logical' point for each object. For example, the logical points of a line will be the two endpoints (*start/endPoint*). Dragging a point within an object would then be described in relation to these logical points (*whereGrabbed*). Most of the group interaction algorithms are handled within the *GroupGraphicalObject*. The root methods handle arbitration for object selection (*requestToGrab(), doGrab(), drag(), endGrab()*) The *doDrag()* method, specialized for each subclass, actually does the dragging – it does not need to know how other users are manipulating the object. The actual drawing of the object by *draw()* will, of course, be specialized to each subclass.

What must a subclassed object such as *LineObject* be responsible for? First, it needs to create or interpret the description string it defines if it is to send a

Table 4. GroupDraw's *GroupGraphicalObject* and its *LineObject* subclass

GroupGraphicalObject, the root object of all graphics primitives	
Instance variables	
int couplingStatus	Indicates if object is private, public, or can be shared.
int ownerProcess	Indicates who is the owner of the process.
int id	A unique reference that identifies the object across the whole system.
boolean acquired	Indicates if the object is currently being manipulated by a participant.
point whereGrabbed	A logical point indicating the position where the object was acquired.
Methods for initializing and destroying an object, and for sending its description to others	
initializeGroupObject()	Initializes the object, filling in any defaults.
destroyObject()	Destroys the object and frees its space.
makeDescription()	Make a description of the object suitable for transmission over the network.
sendDescription()	Send the description to a requester.
Methods for changing object attributes	
requestChangeStatus()	Request the object owner to change the coupling status.
doChangeStatus()	Actually change the status, and broadcast change over network.
requestChangeOwner()	Request the object to change its owner.
doChangeOwner()	Actually change the owner, and broadcast change over network.
Methods for graphically manipulating and drawing the object	
draw()	A place holder for a routine that will draw the object on the screen.
requestToGrab()	Request the object for permission to acquire it.
doGrab()	Permission is granted or denied.
endGrab()	An acquired object is relinquished.
saveOriginal()	The original position of a selected object.
restoreOriginal()	Restore a moved object to its original position.
drag()	High-level drag-handler; checks permission, broadcasts changes, etc.
doDrag()	A place-holder; this routine will actually drag the object.
whereGrabbed()	A place-holder; checks to see where object was grabbed.

LineObject, a subclass of *GroupGraphicalObject*	
Instance variables	
point startPoint	The start point of the line
point endPoint	The end point of the line
Methods	
initializeGroupObject()	Initialize line-specific attributes, then call the super-class *initializeGroupObject*
makeDescription()	Specialized to include line descriptions, then call the inherited method.
saveOriginal()	Specialization line-aware form of the inherited model.
restoreOriginal()	Specialization line-aware form of the inherited model.
doDrag()	Actually drags the object.
draw()	Actually draw the object.

complete object description over the network. Second, given a physical point, it must determine the corresponding logical point. Third, it must handle the object-specific graphical activities, including methods to draw and erase itself, to drag itself around, and to save enough information to undo the dragging operation if the object's owner refuses dragging permission (*save/ restoreOriginal()*). Object status, ownership, contention, and other issues need not be dealt with by the subclass.

Conclusions

This paper introduced both human and technical factors we have experienced designing four multi-user systems: the GroupSketch and XGroupSketch sketchpads, the GroupDraw object-oriented drawing package, and the GroupKit groupware toolkit.

On the human factors side, it may appear that some of the design principles mentioned in the second section are self-evident, e.g., multiple cursors for gesturing, allowing simultaneous activities, and so on. Yet there are many examples of related groupware systems that have failed to live up to these seemingly self-evident criteria. Consider Xerox PARC's Boardnoter, a computerized whiteboard used to support face-to-face meetings (Stefik *et al.*, 1987a; 1987b). While a single large tele-pointer could be seen by all, individual cursors could not be seen by everyone. Neither did participants see each others' actions as they occurred, for actions were not broadcast until a complete graphical stroke was made or a complete text line entered. XSketch, a recent object-based group drawing package suffers a similar lack, as its objects are only transmitted after they are created (Lee, 1990). WScrawl 1.0 mentioned in the third section, does not show multiple cursors[6]. Group Technologies' Aspects does not necessarily show multiple cursors, nor can the process of creating or manipulating an object be seen by participants. We have also seen several other system now under development that fail to meet the basic human factors requirements of a group drawing tool.

On the positive side, there are several systems (including X/GroupSketch and GroupDraw) that do support the kinds of interactions most people expect from a group drawing surface. All have one thing in common: they were derived from Tang's design principles as listed in Table 1. While these systems are quite diverse, they all share a common 'feel', and observations of use are strikingly similar. Three systems, for example, are video-based and work by fusing video images: VideoDraw (Tang and Minneman, 1990), VideoWhiteboard (Tang and Minneman, 1991) and TeamWorkStation (Ishii, 1990). All are limited by scalability, for serious image deterioration results when too many video images are fused. In contrast, Commune is a workstation-based multi-user sketchpad build independently, but in parallel with GroupSketch (Bly and Minneman, 1990; Minneman and Bly, 1990; 1991). Although the interface to the two systems are remarkably similar, there are some minor differences. In Commune, people use a stylus to write directly on top of the horizontally-oriented monitor – the resulting artifacts are superior to the ones generated on our mouse-based system. Other workstation-based sketchpads are now coming available, for example, CaveDraw (Lu and Mantei, 1991); WScrawl 2.0, SHDR from Rank EuroPark, Rococo Sketchpad (Scrivner *et al.*, 1991); Muge (Pendergast and Hayne, 1991); GroupTeleconferencing System (IIS Technologies, date unknown). Some include video-links to promote face-to-face contact. We also believe that these guidelines can be generalized out beyond group drawing. For

[6]WScrawl 2.0 has just been released. It is a completely new design and implementation, and now has multiple cursors and excellent functionality. The creators cite the influence of Bly and Tang on their new work.

example, we know of one group brainstorming system (Cognoter) that has been redesigned to fit these criteria (Tatar *et al.*, 1991).

We have also described our experiences implementing a computational shared drawing surface, concentrating on where a multi-user drawing application would differ from its single-user counterparts. We found that there are trade-offs in the choice between replicated, centralized, and hybrid architecture, and that the reason to choose one style or another will often depend upon the physical requirements of the system. We recommended that conference registration be managed as independently as possible from the underlying application, and that it is best handled as a high-level toolkit component. Multiple cursors are considered fundamental to these systems; we recommended that future interface toolkits and window systems support these directly, as we do now in GroupKit. We have also found that communication bandwidth on moderate speed LANs is not a problem. While we recognize that slow-speed telephone lines are still a fact of life, we suggested that, in general, the underlying system functionality should not be compromised for communications problems that may not exist (especially as 14,400 bits/second modems are now available). Finally, we outlined how a multi-user graphics library can be created by having most of its collaborative-aware properties reside within a root prototypical graphics object. By subclassing, it should be fairly straightforward to extend the library via conventional graphics procedures.

Acknowledgement

This research is supported by the National Science and Engineering Research Council of Canada.

Software availabiilty

GroupSketch is now available via anonymous ftp from the University of Calgary, Department of Computer Science via *cpsc.ucalgary.ca*. Contact Saul Greenberg for further information on the availability of XGroupSketch and GroupKit.

References

Ahuja, S.R., Ensor, J.R. and Lucco, S.E. (1990) 'A comparison of applications sharing mechanisms in real-time desktop conferencing systems' in *Proc. Conf. Office Information Systems* (Boston, April 25–27) 238–248

Bly, S. (1988) 'A use of drawing surfaces in different collaborative settings' in *Proc. Conf. Computer-Supported Cooperative Work (CSCW'88)* ACM Press, 250–256

Bly. S.A. and Minneman, S.L.. (1990) 'Commune: A shared drawing surface' in *Proc. Conf. Office Information Systems* (Boston, April 25–27) 184–192

Dewan, P. and Choudhary, R. (1991) 'Flexible user interface coupling in collaborative systems' in *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems* ACM Press, 41–48

Dykstra, E.A. and Carasik, R.P. (1991) 'Structure and support in cooperative environments: The Amsterdam Conversation Environment' *Int. J. Man Machine Studies* **34**, 3, 419–434 Republished in Greenberg, 1991b)

Egido, C. (1988) 'Video conferencing as a technology to support group work: a review of its failures' in *Proc. Conf. Computer-Supported Cooperative Work (CSCW'88)* ACM Press, 13–24

Gaines, B.R. (1991) 'Modeling and forecasting the information sciences' *Inf. Sci.* 57–58, 3–22

Garner, S.W., Scrivner, S.A.R., Clarke, A.A., Clark, S., Connolly, J.H., Palmen, H., Schappo, A. and Smyth, M.G. (1991) 'The use of design activity for research into computer supported co-operative work (CSCW)' in *Proc. DATER'91* (Loughborough, UK, September) 84–96

Greenberg, S. (1990) 'Sharing views and interactions with single-user applications' in *Proc. ACM/IEEE Conf. Office Information Systems* (Cambridge, MA, April 25–27) 227–237

Greenberg, S. (1991a) 'Personalizable groupware: accommodating individual roles and group differences' in *Proc. Euro. Conf. Computer Supported Cooperative Work (ECSCW '91)* Kluwer Academic Press, 17–32

Greenberg, S. (1991b) *Computer Supported Cooperative Work and Groupware* Academic Press

Greenberg, S. and Bohnet, R. (1991) 'GroupSketch: a multi-user sketchpad for geographically-distributed small groups' in *Proc. Graphics Interface '91* (Calgary, Alberta, Canada, June 5–7) 207–215

Greenberg, S. and Chang, E. (1989) 'Computer support for real time collaborative work' in *Proc. Conf. Numerical Mathematics and Computing* (Winnipeg, Manitoba, September 28–30) Available in *Congressus Numerantium* vol 74 and 75

Greenberg, S. Bohnet, R., Roseman, M. and Webster, D. (1992a) 'GroupSketch' *SIGGRAPH Video Review* 87, ACM Press

Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992b) 'Issues and experiences designing and implementing two group drawing tools' in *Proc. Hawaii Int. Conf. System Sciences* **4** IEEE press

Grudin, J. (1989) 'Why groupware applications fail: problems in design and evaluation' *Office: Technology and People* **4**, 3, 245–264

Hayne, S.C., Pendergast, M.O. and Greenberg, S. (1992) 'Gesturing through cursors: implementing multiple points in group support systems', *Interacting with Computers* (in press)

IIS Technologies (date unknown) *Introduction to the Group TeleConferencing System* IIS Technologies Inc

Ishii, H. (1990) 'TeamWorkStation: towards a seamless shared space' in *Proc. Conf. Computer Supported Cooperative Work (CSCW '90)* ACM Press, 13–26

Ishii, H. and Arita, K. (1991) 'ClearFace: translucent multiuser interface for TeamWorkstation' in *Proc. Euro. Conf. Computer Supported Cooperative Work* Kluwer Academic Press

Ishii, H. and Kobayashi, M. (1992) 'ClearBoard: a seamless medium for shared drawing and conversation with eye contact' in *Proc. ACM Conf. Human Factors in Computing Systems* ACM Press, 525–532

Johansen, R. and Bullen, C. (1984) 'Thinking ahead: what to expect from teleconferencing' *Harvard Business Review* 4–10.

Lauwers, J.C., Joseph, T.A., Lantz, K.A. and Romanow, A.L. (1990) 'Replicated architectures for shared window systems: a critique' in *Proc. Conf. Office Information Systems* (Boston, April 25–27) 249–260

Lauwers, J.C. and Lantz, K.A. (1990) 'Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems' in *Proc. ACM/SIGCHI Conf. Human Factors in Computing* ACM Press

Lee, J.J. (1990) 'Xsketch: A multi-user sketching tool for X11' in *Proc. Conf. Office Information Systems* (Boston, April 25–27) 169–173

Linton, M.A., Calder, P.R., Interrante, J.A., Tang, S. and Vlissides, J.M. (1991) *InterViews Reference Manual* Stanford University, CA, USA

Lu, I. and Mantei, M. (1991) 'Idea management in a shared drawing tool' in *Proc. Euro. Conf. Computer Supported Cooperative Work (ECSCW'91)* Kluwer Academic Press, 97–112

Minneman, S.L. and Bly, S.A. (1990) 'Experiences in the development of a multi-user drawing tool' in *The 3rd Guelph Symposium on Computer Mediated Communication* (Guelph, Ontario, Canada, May 15–17) 154–167

Minneman, S.L. and Bly, S.A. (1991) 'Managing a trois: a study of a multi-user drawing tool in distributed design work' in *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems*, ACM Press, 217–224

MIT (1983) 'Talking heads' in *Discursions* Architecture Machine Group, MIT, Optical disc

Nunamaker, J.F., Dennis, A.R., Valacich, J.S., Vogel, D.R. and George, J.F. (1991) 'Electronic meeting systems to support group work' *Comm ACM* **34**, 7, 40–61

Pendergast, M.O. and Hayne, S.C. (1991) 'Assisting groups during the merging process' in *Decision Sciences Conf.*

Roseman, M. and Greenberg. S. (1992a) 'GroupKit: a groupware toolkit' in *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems: Posters and Short Talks* 43

Roseman, M. and Greenberg, S. (1992b) 'GroupKit: a groupware toolkit for building real-time conferencing applications' in *Proc. ACM Conf. Computer Supported Cooperative Work (CSCW'92)* ACM Press

Sarin, S. and Greif, I. (1985) 'Computer-based real-time conferencing systems' *IEEE Computer* **18**, 10, 33–45

Scheifler, R.W. and Gettys, J. (1986) 'The X window system' *ACM Trans. Graphics* **5**, 2, 79–109

Scrivner, S.A.R., Clark, S.M. and Keen, N.L. (1991) 'The role of workspace-replication in the development of remote CSCW systems' *Technical Report* LUTCHI Research Centre, Department of Computer Studies, Loughborough University, Leicestershire, UK

Smith, R.B., O'Shea, T., O'Malley, C., Scanlon, E. and Taylor, J. (1989) 'Preliminary experiences with a distributed, multi-media, problem environment' in *Proc. 1st Euro. Conf. Computer Supported Cooperative Work (EC-CSCW '89)* (Gatwick, U.K., September 13–15)

Stefik, M., Bobrow, D.G., Foster, G., Lanning, S. and Tatar, D. (1987) 'WYSIWIS revised: early experiences with multiuser interfaces' *ACM Trans. Office Information System* **5**, 2, 147–167

Stefik, M., Foster, G. Bobrow, D., Kahn, K., Lanning, S. and Suchman, L. (1987) 'Beyond the chalkboard: computer support for collaboration and problem solving in meetings' *Comm. ACM*, **30**, 1, 32–47

Tang, J.C. (1989) 'Listing, drawing, and gesturing in design: a study of the use of shared workspaces by design teams' *PhD thesis* Department of Mechanical Engineering, Stanford University, CA, USA (Also available as research report SSL-89-3, Xerox Palo Alto Research Center)

Tang, J.C. (1991) 'Findings from observational studies of collaborative work' *Int. J. Man Machine Studies* **34**, 2, 143–160 Republished in Greenberg, 1991b)

Tang, J.C. and Leifer, L.J. (1988) 'A framework for understanding the workspace activity of design teams' in *Proc. Conf. Computer-Supported Cooperative Work (CSCW'88)* ACM Press, 244–249

Tang. J.C. and Minneman, S.L. (1990) 'Videodraw: a video interface for collaborative drawing, in *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems* ACM Press, 313–320

Tang, J.C. and Minneman, S.L. (1991) 'VideoWhiteboard: video shadows to support remote collaboration' in *Proc. ACM SIGCHI Conf. Human Factors in Computing System* ACM Press, 315–322

Tatar, D.G., Foster, G. and Bobrow, D.G. (1991) 'Design for conversation: lessons from Cognoter' *Int. J. Man Machine Studies,* **34**, 2, 185–210 (Republished in Greenberg, 1991b)