Issues and Experiences Designing and Implementing Two Group Drawing Tools

Saul Greenberg
Mark Roseman
David Webster
Ralph Bohnet

1992

Cite as:
Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992) "Issues and experiences designing and implementing two group drawing tools." In *Proceedings of Hawaii International Conference on System Sciences*, **4**, pp. 138-150, Kuwaii, Hawaii, January, IEEE Press. Reprinted in Baecker, R. (ed) (1993) *Readings in Computer Supported Cooperative Work*, Morgan-Kaufmann.

An earlier version was published as Research report 91/438/22, Department of Computer Science, University of Calgary, Calgary, Alberta.

Issues and Experiences Designing and Implementing Two Group Drawing Tools

Saul Greenberg
Mark Roseman
Dave Webster
Department of Computer Science, University of Calgary, Calgary, Canada T2N 1N4

Ralph Bohnet
MPR TelTech Ltd, 8999 Nelson Way, Vancouver, Canada

Abstract

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 hurdles. This paper describes the issues and experiences we have met and handled in the design of two systems supporting remote real time group interaction: GroupSketch, a multi-user sketchpad; and GroupDraw, an object-based multi-user draw package. 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.

Keywords: *shared workspace, real time remote conferencing, computer supported cooperative work.*

1: Introduction

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 & 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*, 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 & 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 centered around a computer workstation, the networked computer can become a valuable medium for people to share on-line work with each other.

In this document, we will focus on tele-data that provides small groups (2 to ~4 people) with real-time access to a shared drawing space via multi-user equivalents of the now-common paint and draw programs. We describe the issues encountered and experiences gained in the design of two systems supporting real time group interaction: *GroupSketch*, a multi-user sketchpad; and *GroupDraw*, a prototype object-based multi-user drawing program. The intent is to highlight human factors issues critical to the design of real time collaborative drawing tools, and to pass on our own experiences building such systems.

We begin the paper with a literature summary of research 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 design, and the early experiences people had using the systems. The next section then describes our implementation experiences, concentrating on our 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.

2: Designing for the human factors of small group design meetings

Almost every group process begins with a set of initial design meetings, where participants express, discuss, and develop ideas. It is a creative forum where people are encouraged to present their thoughts to the group, to build upon the ideas presented by fellow members, and to problem-solve. 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 the human factors knowledge of face to face design meetings to the design of workstation conferencing tools supporting remote work surfaces. This section will describe how people use conventional work surfaces, the implications for the design of a workstation-based work surface, and how our systems instantiated the design recommendations.

2.1: Understand collaboration.

In order to design a software-based work surface, we must have an adequate understanding of how traditional ones are used in 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 disproves this naive view (Bly 1988). 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; Tang 1991; Tang & Leifer 1988). 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. When participants were huddled in close proximity around the table-mounted paper, the sketchpad played a key role in mediating the conversation, and simultaneous access to the work surface was a normal occurrence (45–68% of all activity). This role was lessened in the whiteboard situation where people were seated several feet away.

Tang 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 (Tang 1989).

<u>*Actions:*</u>

- *listing* produces alpha-numeric 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 eg pointing to an existing drawing.

<u>*Functions:*</u>

- *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 ~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 (~35% of all actions). For example, participants enacted ideas using gestures to express them. Gestures were used to signal turn-taking and to focus the attention of the group. Information can be cognitively chunked and preserved through gestures.

2.2: 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 emphasises 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 the reasons why each is offered are listed in the first two columns of Table 1 (condensed from Tang, 1989). These form the foundation for the rest of this paper.

2.3: The *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) (Greenberg & Bohnet 1991). It is designed around the criteria listed in Table 1. Its main features are:

- a 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; any user can do anything 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 *GroupSketch*'s user interface features and how they were designed around Tang's 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 (eg speaker phones).

Four action modes are supported: gesturing through cursors, drawing, textual listing, 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 freestyle, 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 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 privately save an image, 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. Loading an image or clearing the work surface affects all participant's screens.

We have performed limited usability studies on *GroupSketch* under relatively informal conditions (Greenberg & Bohnet 1991). 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 a drawing) and cooperative (multiple people working together 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 primitive functionality, is reasonably effective as a distributed work surface.

2.4: The GroupDraw interface

While *GroupSketch* is a simple paint program where one 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* is currently under development (a working prototype now exists). While its design is based upon our experiences with *GroupSketch,* we are using it as an experimental platform to study different interface and architecture features. *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, WYSIWIS has been relaxed to provide a scrollable work surface; this is a concession to the limited screen real estate available, and also provides users with room to work on private drawings. Figure 2 illustrates a *GroupDraw* screen with three users working on a design. The smaller registration window lists conference participants, their physical location, their phone numbers, and buttons that will dial that number.

The effect of having objects that can be selected and modified by different people raises a surprising number of issues in terms of interface design. The most obvious is what to do when several people try to manipulate a single object. The simplest strategy is to let only one person at a time acquire a particular 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 the current implementation, 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 so fast that the selection/acquisition process occurs almost instantly—a rejected acquisition appears as a momentary flicker. We are also experimenting with other visual cues, such as grey-scale coloring, to indicate acquisition status during this transition period.

Another issue relates to Tang's fourth design criteria (Table 1) recommending a seamless intermixing of workspace actions and functions. The fact that users must now select from a variety of object types and go into a particular drawing mode makes this recommendation difficult to fulfil. Selecting an object from a palette or a menu can detract from the fluidity of group interaction. We do not yet

Observation	Details
GroupSketch is very easy to learn.	People with even limited computer experience learnt *GroupSketch* in moments (ie 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 a large improvement.
Increasing the number of participants in an open floor policy increases parallel activity but also decreases focused attention.	We observed much simultaneous activity. As noted by Tang, this comes at the price of reduced group attention (Tang 1989). 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 communal 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 to 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.
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.

Table 2. Observation of *GroupSketch* use.

have an acceptable solution. Rather, we are using *GroupDraw* as a platform to test methods for minimal impact mode-switching. These include palettes, menus, hot keys, and gesture recognition.

Another issue is 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). Second, an object's "coupling status" can be specified (Dewan & Choudhary 1991). Here, a user can indicate whether an object can be manipulated by all others, can only be viewed by others, or can be private. One interface issue here is how to indicate the object status to the group. Although an obvious scheme includes identifying object status by colors, it means that it will no longer be available as a drawing resource.

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.

3: Implementation experiences

GroupSketch is implemented on Sun workstations running Unix connected together via Ethernet. *GroupDraw* is built upon the Macintosh/AppleTalk platform. The design of both 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, both 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.

3.1: Replicated vs Centralized Architecture

Two architectural alternatives for constructing distributed groupware are the *centralized* and *replicated* approach (Ahuja, Ensor & Lucco 1990; Lauwers, Joseph, Lantz & Romanow 1990; Lauwers & Lantz 1990). In the centralized approach, a single program called the *central agent* mediates all distributed work surfaces. Each person's workstation runs a *participant process* that just collects user input and passes it to the central agent. After processing this information, the agent tells each participant process what to display on their screens. In effect, the central agent acts as one large program managing users. An example is *WScrawl*, a public domain group drawing program that runs on the X window system. The single *WScrawl* program acts as the central agent that decides what to do with user events and where to display the output. In *WScrawl*, the participant process is simply the X window server. 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 be a network bottleneck as all activity must be channelled through it.

In the replicated approach, there is no central agent. Instead, the participant processes replicated on every machine are totally responsible for maintaining the integrity of the drawing surface. Rather than passing information to a central agent, the participant processes communicate directly with each other. The advantages are that network 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 catch 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 conflicting user requests. All other activities are performed within and between participant processes.

Both *GroupSketch* and *GroupDraw* use fully replicated architectures, with the participant process running as a single process on every workstation. Taking *GroupSketch* as an example, participant processes communicate via Unix stream sockets using only eight primitive events, as listed and explained in Table 3. Since the only actions that can be done by a process are to either draw/erase on a bit mapped surface or to move the cursor, there is no need to synchronize user activities[1].

Event	Information passed
Registering a new user	host name, port number, name of participant, caricature
Unregistering a user	Id of participant
Moving cursors	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

Table 3. Communication protocol between processes

The interaction between replicated participant 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 to not merely view, but to interact with the entities in the shared workspace in a structured way. However, we must ensure that the object's behavior 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, by their very nature, can easily resolve this problem. 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* conference[2], and each process may be the owner of more than one object. The responsibility of the owner is to maintain object

[1] 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 bitmap could look different depending upon the order in which these events arrive at each participant process. In practise, this is not a problem due to the scarcity of this occurring, and the minimal visual disruption to the drawing.

[2] 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.

consistency across all sessions. 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 object, while processC will receive a message denying permission.

Under this approach, object ownership is distributed through all participant processes; it is only the *GroupDraw* session as whole that maintains full state information for its objects. The result is a fairly robust system. If a participant process leaves (eg 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.

3.2: 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 going of other participants?

In *GroupSketch*, a central *registrar* process 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 connects to the registrar, opens its own communication port for other connections, and sends the port address along with the participant's name and caricature data to the registrar (see Table 3).
2. The registrar acknowledges the newcomer and informs other participants' *GroupSketch* process of the newcomer's address in the network.
3. Each *GroupSketch* process connects to the newcomer, with the nearest sending it the current state of the work surface image. 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 reusable in other groupware applications requiring registration. In fact, we were able to reuse in part a registrar that we had originally designed for a completely different application (Greenberg 1990; Greenberg 1991).

3.3: Multiple Cursors

Multiple cursors present a significant problem, for current window systems only support single cursors. As a result, multiple cursors are usually implemented independently from the system-supplied cursors.

In *GroupSketch* we eschewed window systems completely in favour of a graphical library that allowed us to manipulate the bit-map display directly (we used Sun's Pixrect library). Large multiple cursors are implemented directly by exclusively OR'ing bitmaps. The general algorithm for handling multiple cursors follows. After initial variable setup 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. A participant's activity is detected through keyboard events for character insertion; and mouse events for specifying cursor gesture movement, drawing and erasing. Modifications are then broadcast to the remote processes where it appears to them as an "other participant event"; 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 the coordinates of the new position broadcast to userB. UserB's participant process receives an 'other participant interrupt' event, and reads the new mouse location. From an internal table, 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 (eg to manage cursors, menus, simple drawing). We also made a design error in our choice of handling local events differently from remote events. For example, *GroupSketch* now has a bug that will cause it to leave a spurious mark on the screen if one cursor is erasing on top of another person's cursor. While we understand why the problem occurs, it has proven difficult to fix due to the different ways local and remote cursor events are handled; a complete overhaul of the code would be required.

In contrast, *GroupDraw* 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 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 works well in practice—tracking is fairly smooth and occurs in real time.

3.4: 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 (eg Ahuja, Ensor & Lucco 1990; Lauwers, Joseph, Lantz *et al* 1990; Lauwers & Lantz 1990).

Our experiences with *GroupSketch* and *GroupDraw* shows that communication throughput is not a problem over typical local area networks (Ethernet runs at 10Mbps, Appletalk at ~250Kbps). Our communication requirements were modest. Both *GroupSketch* and *GroupDraw* use a standard stream-oriented connection that guarantees in-order, error free delivery (Unix Stream Sockets and Appletalk Data Stream Protocol respectively). 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 per 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. Another thing we noted is that each user's network demands are unequal, for 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 information 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".

One final point worth mentioning is the relation of bandwidth to the number of participants. In the current setup, network demands will increase factorially (worst case) with every new participant, for each participant process must broadcast the same message to all others[1]. If a multi-cast network were available instead, then the communication demands would be linear to the number of participants, for only a single message need be broadcast by the sender.

3.5: 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 are shareable 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 3 provides some detail of the instance variables and methods it contains. Each graphical object has an *ownerProcess*, which is by default the creator of the object. The ownerProcess serves to arbitrate contention in manipulating the object. Objects also have a

[1] The factorial relation only occurs if every person is actively doing something on the work surface. In practise, some people will be idle.

GroupGraphicalObject, the root object of all graphics primitives

instance variables	
int couplingStatus	*-indicates if object is private, public, or shareable*
int ownerProcess	*-indicates who the owner of the process is*
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 complete description of the object suitable for transmission over the network*
sendDescription()	*-send the description to a requestor*
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 sub-class of GroupGraphicalObject

instance variables	
point startPoint	*-the start point of the line*
point endPoint	*-the end point of the line*
methods	
initializeGroupObject()	*-initializes attributes specific to a line, then calls the super-class' initializeGroupObject*
makeDescription()	*-specialized to include line descriptions, then calls the inherited method*
saveOriginal()	*-specialization line-aware form of the inherited method*
restoreOriginal()	*-specialization line-aware form of the inherited method*
doDrag()	*-actually drags the object*
draw()	*-actually draw the object*

Table 3. The GroupGraphicalObject and its LineObject sub-class.

couplingStatus (Dewan & Choudhary 1991) that indicates the extent to which graphical objects are shared. As mentioned in Section 2.4, *GroupDraw* defines three coupling levels: private, public, and shareable. 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 *initalizeGroupObject(),* which will set the instance variables mentioned above. Subclasses will specialize this method to initialize any extra properties it may have eg zeroing out the endpoints of a line (see *LineObject*, Table 3). For communication and storage purposes, each object must be able to construct and interpret a string representing itself (*make/send-Description()*); 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 approves 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 3 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 sub-class.

What must a sub-classed 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 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 state 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 sub-class.

4: Summary

This paper introduced some human factors issues and implementation experiences we have had designing two multi-user systems: *GroupSketch* and *GroupDraw*.

On the human factors side, it may appear that some of the design principles mentioned in Section 2 are self-evident eg 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, Bobrow, Foster, Lanning & Tatar 1987; Stefik, Foster, Bobrow, Kahn, Lanning *et al* 1987). While a single large tele-pointer could be seen by all, individual cursors were not. 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*, a group sketchpad in the public domain, does not show multiple cursors. 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 systems now under development that fail in the same manner to provide the basic necessities of a group drawing area.

On the positive side, there are several systems (including *GroupSketch* and *GroupDraw*) that do support the kinds of interactions 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. Two systems, for example, are video based: *VideoDraw* (Tang & Minneman 1990) and *TeamWorkStation* (Ishii 1990). Both 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 built independently but in parallel with *GroupSketch* (Bly & Minneman 1990; Minneman & Bly 1990 and 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.

We have also shared our implementation experiences, concentrating on where a multi-user drawing application would differ from its single-user counterparts. We found that while there are some tradeoffs between replicated versus centralized architecture, there is no compelling reason to choose one style of another. 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. We have also found that communication bandwidth on moderate speed local area networks 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. 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 sub-classing, it should be fairly straight forward to extend the library via conventional graphics procedures.

Note and acknowledgements. *GroupSketch* is available from the author at no cost through anonymous ftp. This research is supported by the National Science and Engineering Research Council of Canada.

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 *Proceedings of the Conference on Office Information Systems*, p238-248, Boston, April 25-27.

Bly, S. (1988) "A use of drawing surfaces in different collaborative settings." In *Proceedings of the*

Conference on Computer-Supported Cooperative Work (CSCW '88), p250-256, Portland, September 26-28, ACM Press.

Bly, S. A. and Minneman, S. L. (1990) "Commune: A shared drawing surface." In *Proceedings of the Conference on Office Information Systems*, p184-192, Boston, April 25-27.

Dewan, P. and & Choudhary, R. (1991) "Flexible user interface coupling in collaborative systems." In *ACM SIGCHI Conference on Human Factors in Computing Systems*, p41-48, New Orleans, April 28-May 2, ACM Press.

Dykstra, E. A. and Carasik, R. P. (1991) "Structure and support in cooperative environments: The Amsterdam Conversation Environment." In *Computer Supported Cooperative Work and Groupware,* S. Greenberg ed. Academic Press. Also in *Int J Man Machine Studies*, **34**(3), p419-434, March.

Egido, C. (1988) "Video conferencing as a technology to support group work: A review of its failures." In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88)*, p13-24, Portland, September 26-28, ACM Press.

Greenberg, S. (1990) "Sharing views and interactions with single-user applications." In *Proceedings of the ACM/IEEE Conference on Office Information Systems*, p227-237, Cambridge, Massachusets, April 25-27.

Greenberg, S. (1991) "Personalizable groupware: Accomodating individual roles and group differences." In *European Conference of Computer Supported Cooperative Work (ECSCW '91)*, Amsterdam, September 24-27, Klewar Press.

Greenberg, S. and Bohnet, R. (1991) "GroupSketch: A multi-user sketchpad for geographically-distributed small groups." In *Proceedings of Graphics Interface '91*, Calgary, Alberta, June 5-7.

Greenberg, S. and Chang, E. (1989) "Computer support for real time collaborative work." In *Proceedings of the Conference on Numerical Mathematics and Computing*, Winnipeg, Manitoba, September 28-30. Available in Congressus Numerantium vol 74 and 75.

Grudin, J. (1989) "Why groupware applications fail: Problems in design and evaluation." *Office: Technology and People*, **4**(3), p245-264.

Ishii, H. (1990) "TeamWorkStation: Towards a seamless shared space." In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, p13-26, Los Angeles, October 7-10, ACM Press.

Johansen, R. and Bullen, C. (1984) "Thinking ahead: What to expect from teleconferencing." *Harvard Business Review*, p4-10, March/April.

Lauwers, J. C., Joseph, T. A., Lantz, K. A. and Romanow, A. L. (1990) "Replicated architectures for shared window systems: A critique." In *Proceedings of the Conference on Office Information Systems*, p249-260, Boston, April 25-27.

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 *Proceedings of the ACM/SIGCHI Conference on Human factors in Computing*, Seattle Washington, April 1-5, ACM Press.

Lee, J. J. (1990) "Xsketch: A multi-user sketching tool for X11." In *Proceedings of the Conference on Office Information Systems*, p169-173, Boston, April 25-27.

Minneman, S. L. and Bly, S. A. (1990) "Experiences in the development of a multi-user drawing tool." In *The 3rd Guelph Syposium on Computer Mediated Communication*, p154-167, Guelph, Ontario, Canada, May 15-17, by University of Guelph Cont Education.

Minneman, S. L. and Bly, S. A. (1991) "Managing a trois: A study of a multi-user drawing tool in distributed design work." In *ACM SIGCHI Conference on Human Factors in Computing Systems*, p217-224, New Orleans, April 28-May 2, ACM Press.

MIT (1983) "Talking heads." In *Discursions*, Boston, Mass, Architecture Machine Group, MIT. Optical disc.

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tatar, D. (1987) "WYSIWIS revised: Early experiences with multiuser interfaces." *ACM Trans Office Information Systems*, **5**(2), p147-167, April.

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), p32-47.

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, California, April. Also available as research report SSL-89-3, Xerox Palo Alto Research Center, Palo Alto, California.

Tang, J. C. (1991) "Findings from observational studies of collaborative work." In *Computer Supported Cooperative Work and Groupware,* S. Greenberg ed. Academic Press. Also in *Int J Man Machine Studies*, **34**(2), p143-160, February.

Tang, J. C. and Leifer, L. J. (1988) "A framework for understanding the workspace activity of design teams." In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88)*, p244-249, Portland, Oregon, September 26-28, ACM Press.

Tang, J. C. and Minneman, S. L. (1990) "Videodraw: A video interface for collaborative drawing." In *ACM SIGCHI Conference on Human Factors in Computing Systems*, p313-320, Seattle, Washington, April 1-5, ACM Press.

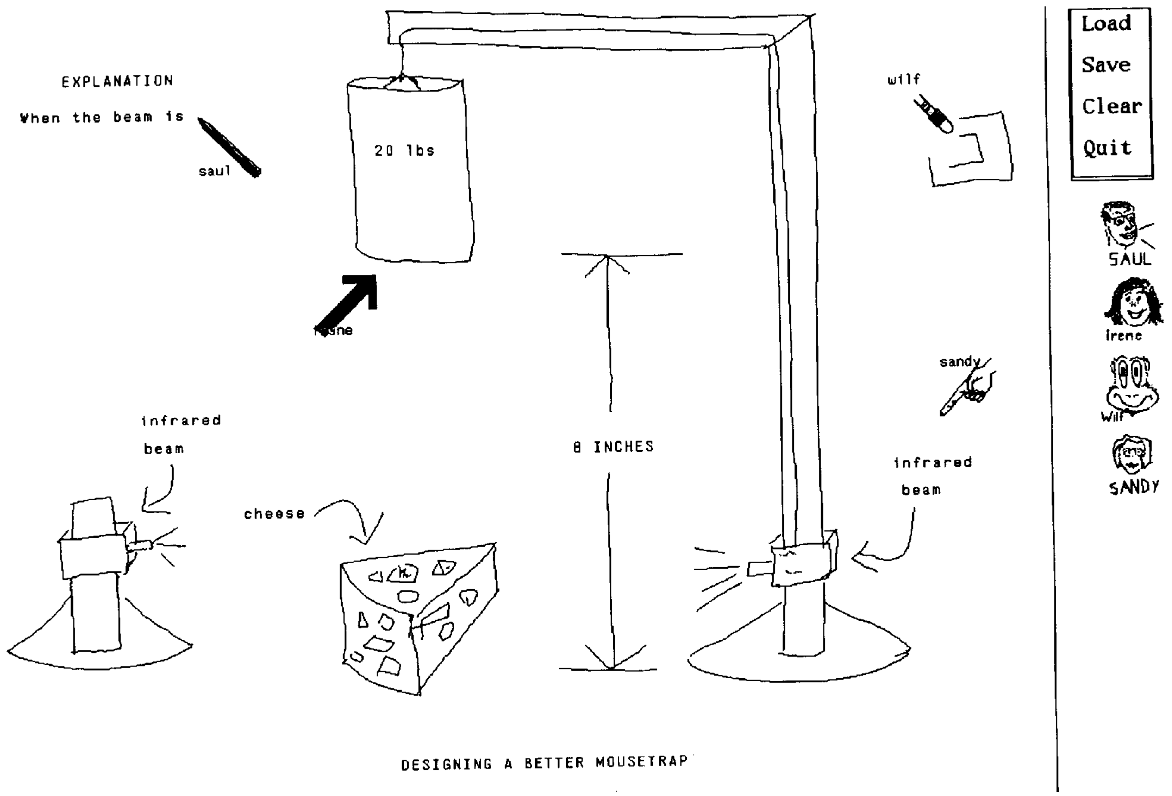

Figure 1: A sample *GroupSketch* session

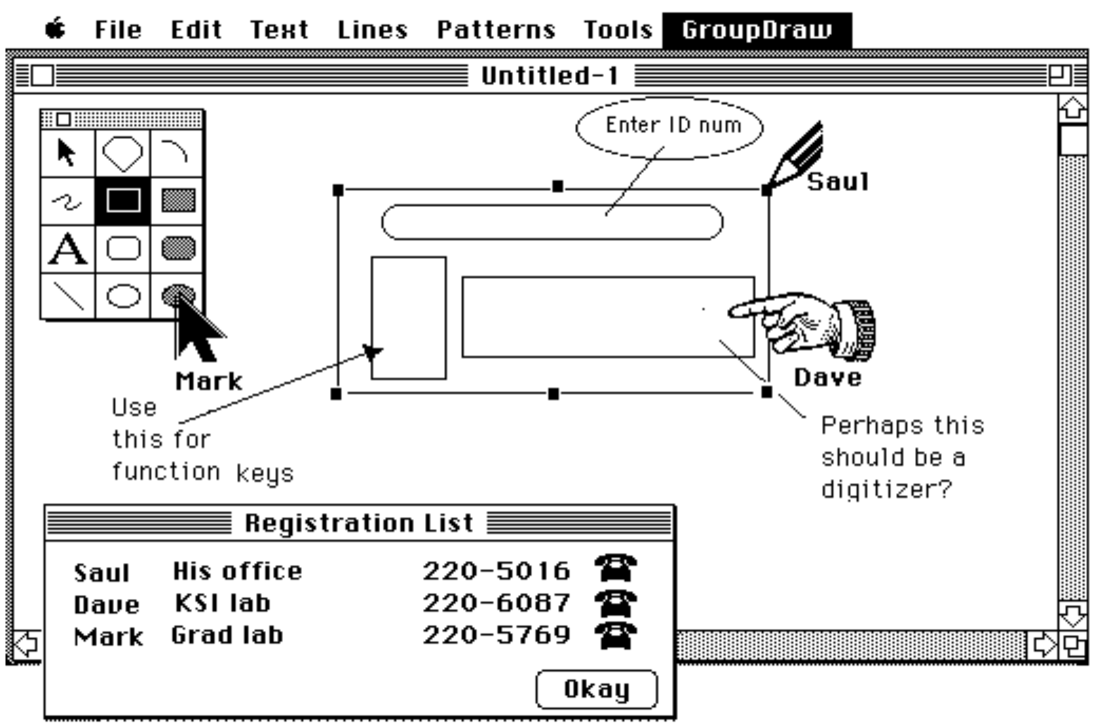

Figure 2: A sample *GroupDraw* session, showing the work surface and the registration window