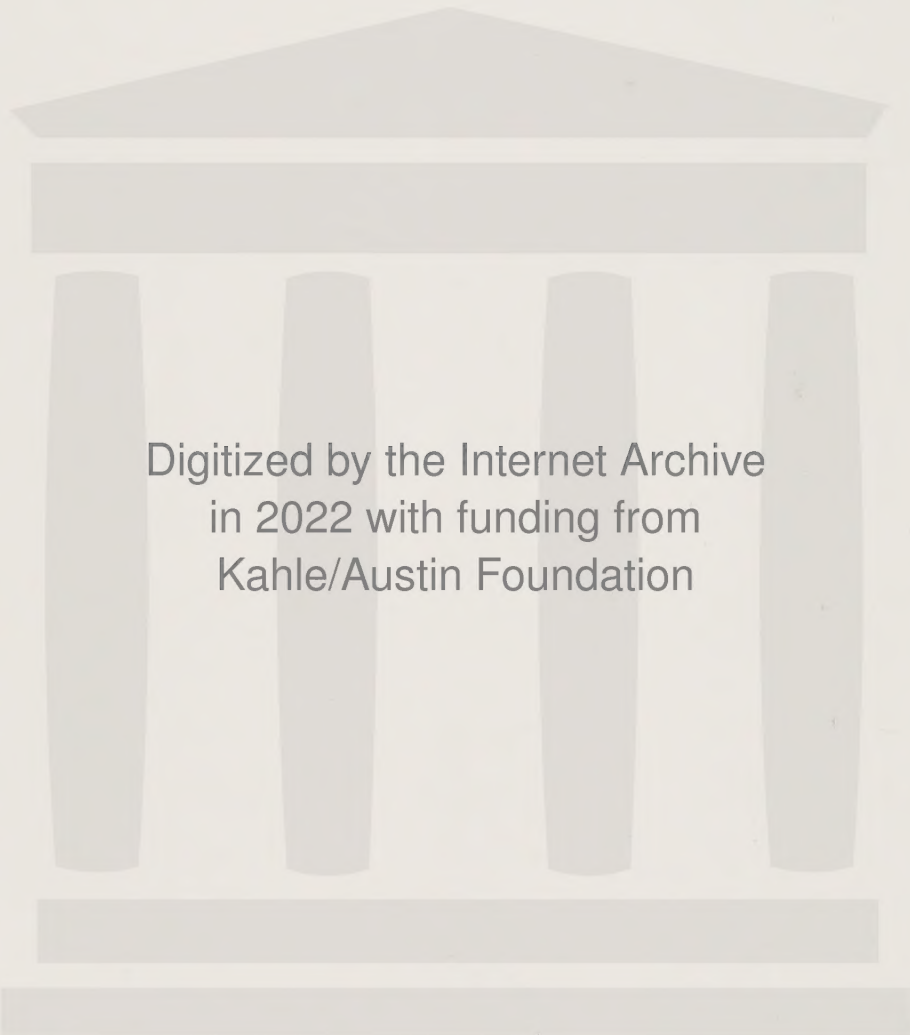


GROUPWARE *for* REAL-TIME DRAWING

A Designer's Guide

Saul Greenberg Stephen Hayne Roy Rada



Digitized by the Internet Archive
in 2022 with funding from
Kahle/Austin Foundation

<https://archive.org/details/groupwareforreal0000unse>

GROUPWARE FOR REAL-TIME DRAWING

GROUPWARE FOR REAL-TIME DRAWING A Designer's Guide

Saul Greenberg
Stephen Hayne
Roy Rada

McGRAW-HILL BOOK COMPANY

London • New York • St Louis • San Francisco • Auckland
Bogotá • Caracas • Lisbon • Madrid • Mexico
Milan • Montreal • New Delhi • Panama • Paris • San Juan
São Paulo • Singapore • Sydney • Tokyo • Toronto

Published by
McGRAW-HILL Book Company Europe
Shoppenhangers Road, Maidenhead, Berkshire, SL6 2QL, England
Telephone 01628 23432
Fax 01628 770224

British Library Cataloguing in Publication Data

Groupware for Real-Time Drawing

I. Rada, R.

620.00420285

ISBN 0-07-707899-3

Library of Congress Cataloging-in-Publication Data

Groupware for real-time drawing / [edited by] Saul

Greenberg, Stephen Hayne, Roy Rada.

p. cm.

Includes index.

ISBN 0-07-707899-3 (pbk.)

1. Real-time data processing.
2. Computer software--Development.
3. Computer graphics.
4. Work groups--Data processing.

I. Greenberg, Saul. II. Hayne, Stephen. III. Rada, R. (Roy)

QA76.54.D47 1995

651.7--dc20

95-12573

CIP

Copyright © 1995 McGraw-Hill International (UK) Limited. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of McGraw-Hill International (UK) Limited.

Typeset at Alden Press Limited, Oxford and Northampton, Great Britain.

12345 BP 9876

Printed and bound in Great Britain by the Bath Press, Avon

Printed on permanent paper in compliance with ISO Standard 9706

CONTENTS

Contributors	vii
Preface	xi
SECTION 1: DESIGNING A COLLABORATIVE WORKSPACE	1
1 Designing at a distance via real-time designer-to-designer interaction Stephen A. R. Scrivener, David Harris, Sean M. Clark, Todd Rockoff and Michael Smyth	5
2 Some design principles for sharing in Tivoli, a whiteboard meeting support tool Thomas P. Moran, Kim McCall, Bill van Melle, Elin Rønby Pedersen and Frank G. Halasz	24
3 Human and technical factors of distributed group drawing tools Saul Greenberg, Mark Roseman, Dave Webster and Ralph Bohnet	37
4 Gesturing through cursors: implementing multiple pointers in group support systems Stephen Hayne, Mark Pendergast and Saul Greenberg	63
5 Using a shared pen-based tool for meeting support Catherine G. Wolf, James R. Rhyne and Laura K. Briggs	81
6 Integration of interpersonal space and shared workspace: ClearBoard design and experiments Hiroshi Ishii, Minoru Kobayashi and Jonathan Grudin	96
SECTION 2: SYSTEMS, ARCHITECTURES AND TOOLKITS FOR GROUPWARE	127
7 Wscrawl 2.0: a shared whiteboard based on X-Windows Brian Wilson	130

8	GroupKit: a groupware toolkit for building real-time conferencing applications	
	Mark Roseman and Saul Greenberg	143
9	SLICE: a logical model for shared editors	
	Alain Karsenty and Michel Beaudouin-Lafon	157
SECTION 3: APPLICATIONS OF COLLABORATIVE TOOLS		175
10	SMART 2000 conferencing system: what we learned from developing the system	
	David A. Martin	179
11	Commercializing a real-time collaborative toolkit	
	Dorab Patel and Scott D. Kalter	198
12	GroupGraphics: prototype to product	
	Mark O. Pendergast	209
13	TelePICTIVE: computer-supported collaborative GUI design for designers with diverse expertise	
	David S. Miller, John G. Smith and Michael J. Muller	228
	Index	242

CONTRIBUTORS

Michel Beaudouin-Lafon, Laboratoire de Recherche en Informatique (CNRS URA 410),
Université de Paris-Sud - Bâtiment 490, 91405 Orsay Cedex, France
Email: mbl@lri.fr

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

Laura K. Briggs, Department of Psychology, Columbia University, New York, NY 10598,
USA

Sean M. Clark, LUTCHI Research Centre, Loughborough University, Loughborough,
Leicestershire, UK
Email: S.M.Clark@uk.ac.lut

Saul Greenberg, Department of Computer Science, University of Calgary, Calgary,
Alberta, Canada T2N 1N4
Email: saul@cpsc.ucalgary.ca

Jonathan Grudin, Department of Information and Computer Science, University of
California, Irvine, CA 92717, USA
Email: grudin@ics.uci.edu

Frank G. Halasz, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto,
CA 94304, USA
Email: Halasz@parc.xerox.com

David Harris, Sensor Science Engineering Group, University of South Australia,
Adelaide, South Australia, Australia
Email: Harris@Levels.UniSA.edu.au

Stephen C. Hayne, School of Management, Arizona State University-West, Phoenix, AZ,
85069-7100, USA
Email: hayne@asu.edu

Hiroshi Ishii, NTT Human Interface Laboratories, 1-2356 Take, Yokosuka-Shi, Kanagawa, 283-03 Japan
Email: ishii.chi@xerox.com

Scott D. Kalter, Twin Sun Inc., 360 N. Sepulveda Blvd., Suite 2055, El Segundo, CA 90245, USA

Alain Karsenty, Laboratoire de Recherche en Informatique (CNRS URA 410), Université de Paris-Sud - Bâtiment 490, 91405 Orsay Cedex, France
Email: ak@lri.fr

Minoru Kobayashi, NTT Human Interface Laboratories, 1-2356 Take, Yokosuka-Shi, Kanagawa, 283-03 Japan
Email: minoru@aether.ntt.jp

Kim McCall, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA
Email: mccall@parc.xerox.com

David A. Martin, SMART Technologies Inc., #599, 240 11th Avenue S. W., Calgary, Alberta, Canada T2R 0C3

Bill van Melle, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA
Email: vanmelle@parc.xerox.com

David S. Miller, Bellcore, 444 Hoes Lane, Piscataway, NJ 08854, USA
Email: dsm@ctt.bellcore.com

Thomas P. Moran, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA
Email: moran@parc.xerox.com

Michael J. Muller, US WEST Advanced Technologies, 4001 Discovery Drive, Suite 270/ Boulder, CO 80303, USA
Email: michael@advtech.uswest.com or muller.chi@xerox.com

Dorab Patel, Twin Sun Inc., 360 N. Sepulveda Blvd., Suite 2055, El Segundo, CA 90245, USA

Elin Rønby Pedersen, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA (current address: Department of Computer Science, Roskilde University, DK-4000, Denmark)
Email: elin@dat.ruc.dk

Mark Pendergast, Management Information Systems Department, College of Business and Public Administration, University of Arizona, Tucson, AZ 85721, USA
Email: pendergast@bpa.arizona.edu

Roy F. Rada, Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, England
Email: rada@csc.liv.ac.uk

James R. Rhyne, IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598, USA

Email: jrhyne@watson.ibm.com

Todd Rockoff, Joint Research Centre, Flinders University, Adelaide, South Australia, Australia

Email: rockoff@jrc.flinders.edu.au

Mark Roseman, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Email: roseman@cpsc.ucalgary.ca

Stephen A. R. Scrivener, Derby School of Art and Design, Derby College of Higher Education, Derby, UK/LUTCHI Research Centre, Loughborough University, Loughborough, Leicestershire, UK

Email: S.A.Scrivener@uk.ac.lut

John G. Smith, Bellcore, 444 Hoes Lane, Piscataway, NJ 08854, USA

Email: hammy@ctt.bellcore.com

Michael Smyth, LUTCHI Research Centre, Loughborough University, Loughborough, Leicestershire, UK

Email: M.G.Smyth@uk.ac.lut

Dave Webster, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Brian Wilson, Apple Computer, Inc., 10500 North De Anza Boulevard, MS: 35-VK, Cupertino, CA 95014, USA

Email: brianw@apple.com

Catherine G. Wolf, IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA

Email: cwolf@watson.ibm.com

PREFACE

In the late 1980s, the near ubiquity of networked computers in organizations created a paradigm shift in the way we view human–computer interaction. Instead of creating applications that allowed a person to pursue only individual work, researchers and developers started thinking about systems that let people work together. The discipline of ‘Computer Supported Cooperative Work’ (CSCW) arose, collecting researchers interested in the study and theory of how people work together, and how the computer and related technologies affect group behaviour. There was a strong applied component of CSCW as well, and ‘groupware’ systems were designed and built to support and augment group work.

Drawing programs, very popular as single-user applications, became the natural candidates for consideration as groupware for several reasons. First, groupware for drawing could be a real benefit to the community. These systems were seen as a potentially valuable media for people to share on-line work with each other – notes, plans, slides and drawings – in both face-to-face meeting rooms and in geographically distributed environments. They could also provide a common workspace that would allow participants to annotate, draw, brainstorm, record and convey ideas during the meeting’s progress. Second, the area is a rich research domain. Researchers in psychology, sociology, management information systems and ethnography could study how people draw together in non-computer settings, and extract principles that could be applied to groupware design. They could also investigate how groups react to novel facilities that groupware could offer over its paper counterparts. Third, groupware is full of challenging technical problems and innovative potential. Computer scientists and developers could grapple with many issues, such as distributed systems, multimedia, communications and protocols, real time responsiveness, design of novel interface widgets and toolkit development. Finally, groupware for drawing is attractive because of its commercial possibilities. If marketers could attract people already using single-user versions of these tools into the fold, then the potential market of users could be enormous.

CONTENT OF THIS BOOK

Initial technological breakthroughs of building computer-based group drawing spaces were made several years ago, and many researchers are now constructing their own group tools as well as pursuing human factors issues. While new ideas are plentiful, there is also a great deal of unnecessary overlap and replication of effort. It is time to proceed beyond mere replication of systems and move instead towards empiricism, where lessons could be drawn from the communal research experiences and empirical design rules formulated. This book documents this process for collaborative drawing.

As seen by the sections and articles, the emphasis of this book was placed on:

- the human factors research and requirements behind the design of shared drawing tools
- the technical factors and innovations involved in building these systems
- specific applications of these tools
- empirical validation of experiences with people using shared work spaces
- the factors involved in the adoption of these tools by groups and organizations
- the factors involved in making such groupware a commercial product

The book begins with ‘Designing a collaborative workspace’ (Section 1). Since the dawn of humanity, we have used drawings to communicate. Drawings have ranged from scratches in the sand about where a friend could find food to virtual reality tours of architectural designs. It is natural that people progress from working together face-to-face over physical drawings to working together through a computer display. But sharing a drawing is far more than sharing marks on a screen. For example, observation of a group’s drawing process reveals the prevalence of gesturing, where a person’s hand motion and gaze direction act as important cues necessary for mutual understanding and participant interaction. The impact of this and other findings is that systems must be constructed to support quite subtle human protocols; designs based on naive views of the drawing process will not suffice.

Section 2 concerns ‘Systems, architectures and toolkits for groupware’. Conventional software is difficult to create, and groupware is even harder. The problem is that a computer’s operating system is largely designed to handle single-user applications and provides minimal support for the requirements of groupware. The consequence is that developers have to go to great extents to write even conceptually simple groupware systems. While we are still a long way off from good groupware development platforms, problems can be mitigated by having practitioners share their experiences in building specific groupware applications, and explaining strategies for architectures and toolkits for groupware development.

The final section is concerned with ‘Applications of collaborative tools’. The technological breakthroughs necessary to build computer-based shared drawing spaces are almost complete. However, what works well under controlled circumstances in the lab may not perform as well in the field. The transition from research principles and prototypes to robust commercial systems is a difficult one. Extensive experience shows that the bottom line is that technology must fit into the work flow. Furthermore, groupwork must be well managed for a group to succeed, and this is true whether or not technology is used. But if technology is used, then the management of the group to benefit from this technology is perhaps as important as any particular feature of the technology itself. Technology should fit unobtrusively into the workplace.

HISTORY OF THIS BOOK

On 31 October 1992, Saul Greenberg and Stephen Hayne organized a workshop on 'Real Time Group Drawing and Writing Tools', and brought together 25 active researchers in the area and system builders. Hosted as part of the ACM 1992 Conference on Computer Supported Cooperative Work, the workshop had an international focus, with members from Australia, Canada, England, France, Germany, Japan and the USA. Many communities were represented as well, including computer scientists, psychologists, management and information systems people, sociologists and commercial developers.

The workshop focused on human factors and technologies that provided face-to-face or geographically distributed groups with real-time access to shared drawing or writing spaces. The goals of the workshop were threefold:

- to bring together active practitioners to discuss the human and technical issues surrounding shared drawing and writing tools
- to create a community aware of its innovations and collective experiences
- to consider a book that collects together the relevant literature of the community as a whole

Potential participants at the workshop submitted a position paper and, or a full paper (new or previously published) that best described their work. Papers were reviewed by the organizers, and participants were accepted based upon the appropriateness, quality and novelty of their research. The fruits of the workshop were then collected into a limited edition proceedings.

The workshop was quite exciting, with an exhausting schedule of presentation and discussion. At its end, we decided that a book collecting the 'state of the art' knowledge of group drawing tools would be a valuable contribution to the community, and participants undertook the challenge of substantially revising the proceedings into this book. At this point, Roy Rada joined the editorial team to assist with the onerous tasks of finding a publisher, coordinating authors, collecting papers and producing copy for typesetting.

After the workshop, the articles were reviewed again for their pertinence as a book contribution, and a round of revisions performed. Some of the previously published papers were reprinted in their entirety; others were revised to incorporate new work. Participants who had submitted working papers and position papers then delivered a finished product. Additional papers were solicited. The collection of papers that we then presented to our publisher, McGraw-Hill, was larger than had been expected, and the editors were forced to trim the book substantially by focusing only on collaborative drawing.

We thank all the participants at the workshop for their enthusiasm and willingness to stick to schedules. Graham Bird and Claude Ghaoui were instrumental in doing much of the legwork and seeing that the book got to the publishers in proper form.

Saul Greenberg
Stephen Hayne
Roy Rada

SECTION ONE

Designing a collaborative workspace

Drawings are a cornerstone of communication. They are so much a part of our everyday world that it would be impossible to envisage life without them. As society embraces groupware, it is only natural that people progress from working together face to face over a physical drawing medium to working together over distance through a computerized drawing surface. This section examines collaborative workspaces that are used for group drawing. The papers illustrate what existing systems look like, and recount the design decisions made.

In Chapter 1, Stephen Scrivener, Sean Clark, Michael Smyth, David Harris and Todd Rockoff explore the usefulness of a shared workspace when two designers are separated by a huge distance. The ROCOCO project provides pairs of designers with an eye-to-eye video link, a high-quality audio link and a shared sketching surface using basic rate ISDN and telephone lines. In this particular study, the pair were separated by over 16 000 kilometres, one in England and one in Australia. Users had to overcome social distance, cultural differences, time zones and heterogeneous workstations. This project is unique in its aggressive pursuit of the groupwork envelope. The team reports that the technical implementation was sound as well as usable and performance acceptable. It appears that as the designers engaged in their tasks, they became unaware of the medium and the distance between them, and cooperated in a natural way. However, only when the limitations they discovered are removed and the connection is as simple to make as a telephone call will we be able to assess and exploit the full potential of cooperative design.

Tivoli, an electronic whiteboard meeting support system, was designed to assist small groups, both during same time, same place and same time different place meetings. This advanced project by Tom Moran, Kim McCall, Bill van Melle, Elin Pedersen and Frank Halasz (Chapter 2) features a fully replicated architecture, object-based drawing and immutable objects. In this paper, they document the design principles they used for developing the system. They report that these are not complete, i.e. the complexities of an 'undo' feature are as yet unresolved in a group context. However, these principles provide a deep insight into the implementation issues that need to be addressed for high-quality shared drawing tools. As this goes to press, we understand that this prototype has been turned into a product and is being marketed by Xerox.

Saul Greenberg, Mark Roseman, Dave Webster and Ralph Bohnet (Chapter 3) build upon Tang's six design criteria for shared work surfaces. They built several prototypes and observed user behaviour. Most notably, participants converse and interact as they would over a shared piece of paper, yet it is not identical to a face-to-face meeting. Users are more task-focused as they try to gather as much tele-presence as they can. But, we must also be concerned with activities when the work surface is not central to the discussion! When conversation shifted to a meta-level topic not requiring the work surface, the authors notice that many users turned away from the computer display to look at each other. This will not be possible when groups are geographically dispersed. The work by Ishii *et al.* later in Chapter 6 addresses this problem for two-person groups (dyads). This raises the issue of group size: the authors report that most of the design criteria scale, except for screen clutter and real-estate. The factor that larger groups seem to take longer to develop a social protocol may mean that protocols should be embedded in the system for more efficient group interaction.

Developers often disdain the personal computer environment for group work due to its low power, deciding to implement their prototypes on high-powered workstations using a variation of UNIX. Sometimes tasks are too complex and the extra power is necessary. In

Chapter 4, Stephen Hayne, Mark Pendegast and Saul Greenberg explore the limitations of the PC and show that the concept of multiple telepointers can be implemented in this environment. There is consensus that gesturing is an integral part of computer collaboration, and the authors put forth a design framework for guiding support for this part of tele-presence. Issues such as cursor motion, group size and dispersion and relaxed 'What You See Is What I See' (WYSIWIS) all affect gesture designs. They go on to examine in detail the performance benchmarks on multiple MS-DOS and Microsoft Windows personal computers. Small unreliable broadcast datagrams at a minimum of 10 per second appear suitable for smooth gesturing. Major performance bottlenecks exist within the Windows kernel and the video display adapters, yet the system is still usable for small groups. Many relevant suggestions for how to handle different group tasks and configurations are made in this paper, thus making a sizable contribution to practical knowledge.

Catherine Wolf and James Rhyne (Chapter 5) have built a pen-based group bitmap drawing system (We-Met) designed for small, informal workgroups. We-Met is quite unusual because it records all meeting events and enables users to identify 'viewpoints' in a scene for later review. This allows the group to pursue multiple design paths. They also studied several groups using We-Met for a design task and report that the pen-based interface works well due to minimal mode switching between drawing, writing or gesturing. Users had some difficulty with WYSIWIS and scrolling the group document because they would lose other participants, i.e. others would not understand where they had been scrolled to. Searching the meeting event space also presents a challenge, since it appears that users have multiple strategies, yet most strategies could be augmented with computer support.

Initial observations and findings gained through the experimental use of prototype dyadic drawing systems are the focus of Hiroshi Ishii, Minoru Kobayashi and Jonathan Grundin (Chapter 6). The concept of 'gaze awareness' is explored with informal and formal experiments by seamlessly integrating a shared workspace and interpersonal space. The ClearBoard systems are based on the metaphor of 'looking through' and 'drawing on' a glass board, implemented through shared video technology. These systems are the most exceptional and current in their use of technology of those presented in this section. The role of eye contact appears to be especially important when instructing other people. How extensive its use is and when eye contact might be disruptive are open topics.

DESIGNING AT A DISTANCE VIA REAL-TIME DESIGNER-TO-DESIGNER INTERACTION

Stephen A. R. Scrivener, David Harris, Sean M. Clark,
Todd Rockoff and Michael Smyth

ABSTRACT

This paper describes an experiment in which teams of two designers used a shared computer 'sketchpad' to generate design concepts for products while geographically separated by a distance of over 16 300km. The aim of the separation, which included a large time-zone difference, was to examine critically the capability of the sketchpad to provide useful support for real design work. It was also used to examine potential problems due to the extreme geographic separation of team members and to uncover others which may not have been apparent from localized laboratory studies. The results indicate that the tool is usable and useful for design work at a distance. The paper concludes by pointing to issues that will need to be addressed in developing computer-supported cooperative design (CSCD) systems for the whole design process.

1.1 INTRODUCTION

While much computer support for design has been developed primarily to aid the individual designer, the industrial reality is that designers work in teams. This fact has, of course, been widely recognized. The philosophies of concurrent design and concurrent engineering reflect the multi-skilled input required in design teams, while much of scientific management methodology is aimed at optimum use of the human resources in team activity.

The term computer-supported cooperative work (CSCW) denotes the recently established field of study concerned with the development of systems that support multiple individuals working together with computer systems. The field is maturing rapidly and has seen a series of dedicated conferences both in the United States and Europe (ACM, 1986;

ACM, 1988; CSC, 1989; ACM, 1990; Bannon *et al.*, 1991). Throughout the 1980s, a number of technological developments occurred which established the preconditions for the emergence of CSCW and its subsequent development. The wide-scale introduction of personal computing was followed closely by the trend to network these systems together, allowing users access to a variety of services. Developments in open systems and Integrated Services Digital Networks (ISDN) have made these resources more generally accessible. Finally, the information that can be handled and shared by users has been enriched by improved display and multimedia technology, bringing IT media on a par with more conventional media.

The technologies, concerns and ideas of CSCW are not new, and have been around for many years. Rather, the emergence of CSCW as a subject of study reflects a shift of viewpoint that combines an understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques (Wilson, 1990).

This chapter is concerned with CSCW applied to the domain of design; in other words with computer-supported cooperative design (CSCD). It focuses on the development of a 'shared drawing system' and its use in design work accomplished by pairs of designers working together at the same time on a shared task, but located in different continents (one in England, the other in Australia).

1.2 SUPPORTING DESIGN TEAMS

The fundamental method for implementing design teamwork is the meeting – or more accurately, a chain of meetings – interspersed with individual work. This process is illustrated in Figure 1.1. The first and subsequent meetings are fundamentally different. At the first meeting there is no product database and no one reports on work done to date. There is no plan and therefore no reporting of progress. Subsequent meetings require reporting against the plan developed at the first meeting, and the presentation of the evolving design, usually in the form of drawings.

This difference raises an important point in providing computer support for the process because the tools used by each specialist will, in general, not be readily usable by the others. A member of the team may want to see the effect of moving a hot component but may not know how to use the heat flow engineer's CAD system to do this. The marketer

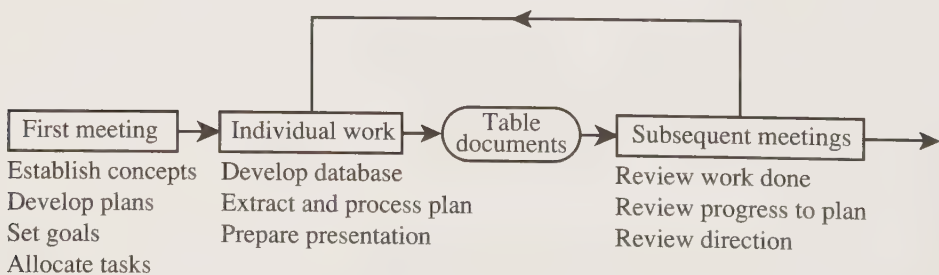


Figure 1.1 Iterations of individual and group work in a team design project.

may wish to see the cost effect of various options but may be unable to manipulate the cost accountant's data to do so. Each specialist, however, is required to report to the meeting on work done, and decisions made, and to present this in a way which the other team members can understand and discuss. Thus, for CSCD it is necessary to have facilities that support both specialist and group interactions with the product database.

In looking at CSCW tools to support the team design process the terminology changes to reflect the possible dispersal of team members (see Table 1.1, and Rodden, 1991; Scivener and Clark, 1992).

Table 1.1 Relationship between conventional work practice modes and CSCW terms

Conventional working	CSCW	Characteristic
Meetings	Synchronous work	Work in the same space/time
Individual work	Asynchronous work	Work in different space/time

The term 'in the same space/time' implies that team members are working simultaneously, are generally working on the same thing, are in continuous, real-time communication and are aware of each other's activities. However, they do not need to be in the same physical location. Instead they can share a computer workspace in which the work of the entire group is presented to each team member with continuous real-time update. Typically, such systems are developed using the 'What You See Is What I See' concept, known by the acronym WYSIWIS (Stefik *et al.*, 1986), where each group member sees the same view of the data on their local workstation. The work described in this chapter explores the use of such a system in meetings where the design concept for a product was developed from an initial brief.

Asynchronous work, on the other hand, implies that team members are generally not working together in the same place at the same time on the same task, are not in continuous real-time communication, and are not aware of each other's activities. However, the notion of a team implies that they are nevertheless working towards a shared goal, and are in this sense engaged in groupwork. Although not concerned with asynchronous work, the study described in this chapter did elicit a number of factors having implications for the design of computer systems that support such work.

1.3 DISPERSED WORK TEAMS

Face-to-face meetings using familiar tools (pencil and paper, overhead projector, etc.) provide a very effective communication methodology to support concurrent design. Do dispersed teams offer any benefits? They appear to offer at least four:

1. A time-efficient method, where team members must otherwise travel long distances to meet, as is frequently the case with large companies.
2. Access to the best available expertise, where it may otherwise be logistically impractical or expensive.

3. Access to a specific technological environment, for example, the technical team of a specialist supplier, where local knowledge of availability and practicality is key.
4. Access to a rich information environment, where additional specialist information may be obtained as the need arises during a meeting (i.e. the requirement that we take with us what we think we will need to a face-to-face meeting can constrain its progression).

However, the use of computer tools and the geographical isolation of team members bring additional difficulties including:

1. The practicality of the computer systems involved: do they need a team of computer scientists at each node of the network to make them work?
2. The ease of use of CSCW tools: can any CSCW tool be as practical, flexible and intuitive as pencil and paper?
3. The presence of cultural differences: these can be those of personal cultures involving everything from status to language difficulties. The technological culture may also be different, where technology easily available to one team member may not be to another.
4. The presence of time zone differences: these may restrict the possibilities for synchronous work to small and fixed time windows.

It is perhaps worth noting that CSCW tools may also be useful to non-dispersed teams. Meetings between two or more persons occur widely in most organizations. However, although the individual use of computers is fairly widespread, in meetings we tend to leave them behind (Stefik *et al.*, 1986). Computer supported meeting environments attempt to reduce this particular distinction between individual and group working by giving meeting attendees specialist computer tools. Such tools include computerized chalkboards, brainstorming/idea generation software, voting/prioritizing software and multi-user access to external databases. A tool used to show a CAD model to remote team members may be just as useful to a team working around a table.

1.4 THE LOUGHBOROUGH/ADELAIDE EXPERIMENT

1.4.1 The background to the experiment: the ROCOCO project

The long-term aim of the ROCOCO (RemOte COoperation and COmmunication) project is the identification of the communication requirements of remotely sited designers working on a shared problem. The result will be a set of system design requirements which will facilitate the design of usable CSCW systems of the future.

A basic premise of the research is that an increased understanding of the communication channel usage of designers working proximally is a prerequisite to the introduction of technologically based communication systems. Hence, the first phase of the ROCOCO project involved a study of face-to-face working. This study investigated pairs of designers engaged in predefined tasks regarded as being characteristic of the early stages of design. In these proximal studies, the designers sat opposite each other across a table. On the table between them was a pad of A1 plain white paper on which they could both draw using their own pens. The seating arrangement is shown in Figure 1.2. Six one-hour design sessions were performed in this environment. Each session was video and audio recorded for later analysis.



Figure 1.2 The seating of the designers in the face-to-face studies.

Observations and analysis (Scrivener *et al.*, 1992) have led to the formulation of hypothesis about the communication requirements of the setting studied. In the second phase of the ROCOCO project the communication requirements of group design are being investigated in conditions where, typically, communication is impoverished. These experimental conditions are achieved by manipulating an electronic workspace and communication environment called the ROCOCO Station. The environment provides pairs of geographically separated designers with an eye-to-eye video link, a high-quality audio link and a shared drawing surface known as the ROCOCO Sketchpad.

1.4.2 The ROCOCO Station

When using the ROCOCO Station in a remote design session, designers sit at their own terminals. The shared drawing surface is displayed on a large computer workstation screen in front of each of them. All interaction with it is via a digitizer and stylus. To one side of each computer workstation is a 'VideoTunnel' video link containing a video monitor and camera. Displayed on the monitor is a head and shoulders image of the remote partner. The VideoTunnel arrangement, developed by Smith *et al.* (1989), uses half-silvered glass and mirrors to allow eye-contact to be made over the video link. The designers wear lightweight headsets in order to communicate verbally. A designer in position at the ROCOCO Station is shown in Figure 1.3. The person's image, as displayed to the remote partner via the VideoTunnel, is shown in Figure 1.4. Our initial results have shown that working with the ROCOCO Station is very similar to working face-to-face.

In the current phase of the ROCOCO project's studies, four different configurations of the ROCOCO Station are being investigated. Three of these remove certain communication channels from use in a design session. The permutations of the communication channels for each configuration are shown in Table 1.2. During an experiment, the ROCOCO Station terminals are located in different rooms in the same building.



Figure 1.3 A designer in position at the ROCOCO Station.



Figure 1.4 The image displayed to the remote designer.

Table 1.2 The ROCOCO Station experimental configurations

Configuration	Sketchpad	VideoTunnel	Audio link
1	On	On	On
2	On	On	Off
3	On	Off	On
4	On	Off	Off

The long-distance design session between Loughborough and Adelaide replicated the third of these configurations. Designers communicated verbally via an audio link and visually via the ROCOCO Sketchpad shared drawing surface. The aims of the experimenters were to demonstrate the usability and usefulness of the system using existing commercially available communications technology over a much greater distance than investigated in the ROCOCO project, and to investigate any problems arising from the extreme geographical isolation of team members.

1.4.3 The ROCOCO Sketchpad

A central part of the ROCOCO Station is the ROCOCO Sketchpad shared drawing surface. The Sketchpad allows two or more designers, sitting at different computer workstations, to share a drawing surface and to engage in many of the interactions available in face-to-face working. It takes the form of a large 'shared window' which is displayed on each designer's workstation screen. These are selected from a menu displayed at the top of the window. All marks drawn on the surface are made visible to all users almost instantly. As well as being able to sketch, a designer is able to direct remote colleagues' attention to a particular area on the surface via a continuously displayed

'telepointer'. The menu also allows the designers to select a blank page to draw on or return to a previous page of drawings.

The functionality of the ROCOCO Sketchpad was defined by the drawing behaviour observed in the ROCOCO project's face-to-face studies. In these studies it was found that, despite having a large variety of drawing implements available, the subjects used only a few pen types. Additionally, rulers and templates were very rarely used. Hence it was felt that the Sketchpad need only offer four pen colours and three pen types.

Resulting from an ongoing research effort at Xerox PARC into shared drawing activity and shared drawing surfaces (see Bly, 1988; Tang and Leifer, 1988; Bly and Minneman, 1990; Tang and Minneman, 1990), four requirements for systems which attempt to support shared drawing activity have been identified (Bly and Minneman, 1990). These are:

1. Marks and gestures should be made visible to all participants without significant delay.
2. Rapid switching between drawing, writing and gesturing should be possible.
3. Users should be able to mark, erase and gesture in the same space simultaneously.
4. Familiar mechanisms for drawing space activity should be maintained.

The ROCOCO Sketchpad was designed to satisfy these requirements. Additionally, since the guiding principle was that the Sketchpad should appear similar to pen-and-paper, it also aims to be directly usable; to allow its users to make use of their existing skills; to require the learning of few new skills; and to involve the minimal loss of skills (Scrivener *et al.*, 1991). In other words, when using the ROCOCO Sketchpad there is a feeling of *familiarity*. The ROCOCO Sketchpad application was written using the X-Windows System (Scheifler and Gettys, 1986).

1.4.4 The equipment used in the experiment

The experimental work was set up between Loughborough University of Technology, in the UK, and the Flinders University in Adelaide, South Australia. Each location is researching in the field of CSCW but works in different areas and uses different equipment. While some additional equipment was used at both sites to establish the experimental link, this was kept to a minimum because one of the aims of the project was to investigate the effect of *heterogeneous* computer facilities.

ROCOCO Station terminals were established at both sites. A basic-rate ISDN communication channel was used to exchange the drawing surface data and headset telephones were used to create an audio link. The link between the local area networks (Ethernet) at each end and the ISDN line was via a 3-COM bridge-router.

The most visible heterogeneous components of the experimental system were the input devices used. The designer located in the UK interacted with the ROCOCO Sketchpad, as in the ROCOCO project, via a standard digitizing tablet and stylus. However, the designer in Australia used a large stand-up 'Liveboard'. This device was effectively a large digitizer and stylus on to which an image of the ROCOCO Sketchpad running on a Macintosh computer was projected via an overhead projection panel. Hence, any drawings made with the stylus appeared on the Liveboard at the point of entry. The UK and Australian configurations of the ROCOCO Station are shown in Figures 1.5 and 1.6.

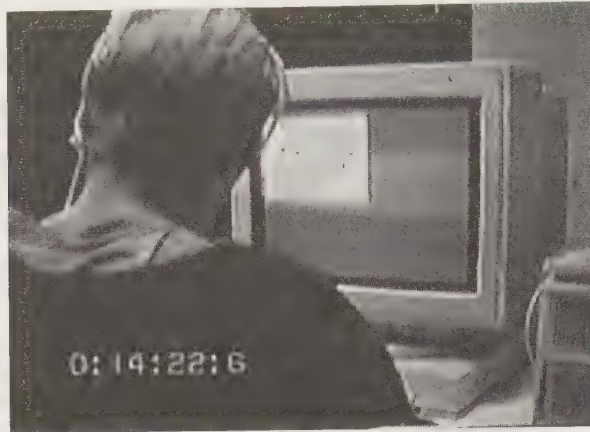


Figure 1.5 Designer located in the UK.

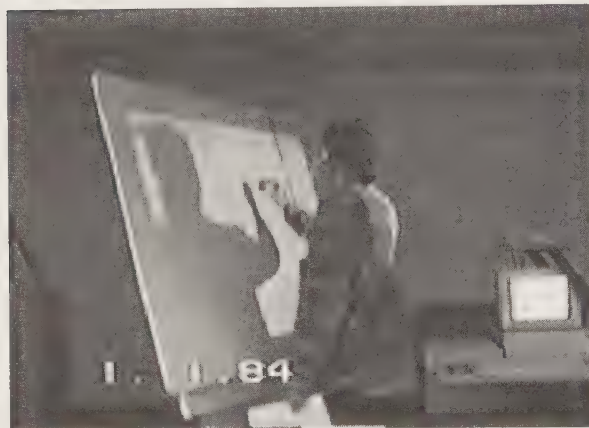


Figure 1.6 Designer located in Australia.

1.4.5 The design of the experiment

Four subjects were randomly selected, two from the final year of the B.Sc. in Design and Technology course at Loughborough University of Technology, and two newly graduated design students from the B.Des. Industrial Design course at the University of South Australia. Each subject was assigned to a test pair. The tasks used in the study were chosen from those used in the ROCOCO project; completion of the tasks did not require any prior market research, and each task incorporated a comparable level of difficulty. In the course of the study each subject pair completed the same three design tasks, resulting in each task being employed on two separate occasions. The design briefs, as presented to the subjects, are given in Appendix 1. Throughout each of the six studies audio and video recordings were made.

The six studies were carried out sequentially over a period of three days (i.e. two each day). The designers at Loughborough worked on similar tasks when taking part in the ROCOCO project's experiments; they had, therefore, around one hour's experience in the use of the system. The Australian designers were given an hour to familiarize themselves with the system. On arrival at each laboratory subjects were located at their respective collaborative terminals. At the outset of each study, prior to receiving a design brief, the subject pair was allowed to 'doodle' for a few minutes and, on indicating readiness to proceed, was read a standardized set of instructions. After 40 minutes of work on the task the subjects received a 10 minute warning, and after 50 minutes were asked to finish. At the end of the first and last session each designer was asked to complete a questionnaire. At the end of the series of three sessions per design pair subjects were de-briefed concerning the purpose of the study and asked for their permission to electronically store and manipulate the information collected. Finally, the subjects were thanked for their participation.

The subjects were thus placed in a situation forcing the possible emergence of problems due to:

1. Social distance: the designers had not met and may never meet. Their relationship was confined to that established and supported by the medium. Eye-to-eye contact was not available and their introduction to one another was perfunctory.
2. Cultural differences: these differences between English and Australian people are relatively small but they do exist. Both are native speakers of the same language (approximately!) and senses of humour are similar. They have different world views and operate within different technological environments.
3. Time zone differences: there was a 10.5 hour time difference (at the time of the experiments) between the UK and South Australia, i.e. 9:00 a.m. in the UK being 7:30 p.m. in Australia. This required that at least one group operate outside normal working hours.
4. Heterogeneous workstations: while both designers were using the ROCOCO sketchpad the hardware and its arrangement was different at each end of the collaborative link. In Loughborough the designer sat in front of a computer screen and used a stylus and tablet for drawing. In Adelaide the designer stood at a Liveboard.

1.4.6 Results

The first important result is that the link worked technically. At no point during around 13 hours of connection time did the technical system fail. This indicates that public service networks (such as ISDN) and local area networks (e.g. Ethernet) can be successfully combined to provide CSCD infrastructures, if at present in a nascent state (a point that will be returned to later).

The end-of-session questionnaire consisted of statements relating to the fitness for the task and the usability of the system. Some statements were positive about the system and others negative. Subjects were asked to rate their agreement with each statement on a five point scale. Figure 1.7 represents the responses for all subjects and all questionnaires in terms of four categories of statement: task-positive, task-negative, usability-positive and usability-negative. For a given category the frequency for each point on the rating scale is presented as a proportion of the total responses (i.e. the usability-positive category consisted of five statements rated eight times, yielding a total of 40 ratings). Considering

Figure 1.7 it is clear that a high proportion of subjects' responses show agreement with positive statements and disagreement with negative ones.

A small number of task-negative statements yielded a higher degree of agreement. Responses to these questions indicated that the technology made drawings harder to produce, resulting in fewer drawings, and that the system sometimes appeared inconsistent.

The questionnaire included a number of sections inviting open responses about the role of the technology, the design task, the subject's performance as an individual and as part of the design team, and finally about the experience in general (see Appendix 2). The designers expressed enthusiasm for the technology; they felt comfortable with it, found it easy and natural to use, and an enabler of good communication. One subject commented on the fact that the system allowed privacy while supporting interaction with others. In general the designers were positive and enthusiastic about the tasks. Other comments picked out the role of interaction in facilitating the integration of ideas, and the importance of the task in supporting interaction with a 'stranger'. There were mixed opinions about performance as individuals; the data revealed both overall satisfaction and specific dissatisfaction with the quality of the writing and drawing produced. However, interestingly, all user reports were positive about performance as part of a design pair, and this overall satisfaction with the whole experience is reflected in the general comments. Indeed, one designer reported, 'I couldn't have done better at this conceptual stage with someone in the same room (apart from the drawing quality)'.

The subjects were conscious of changes in their usual style of working and in the way they discussed the problem with their partner as a result of using the system. However, they felt that they contributed equally to the design solution and the discussion, and given the choice would not have chosen to work alone on the kind of tasks undertaken in the study.

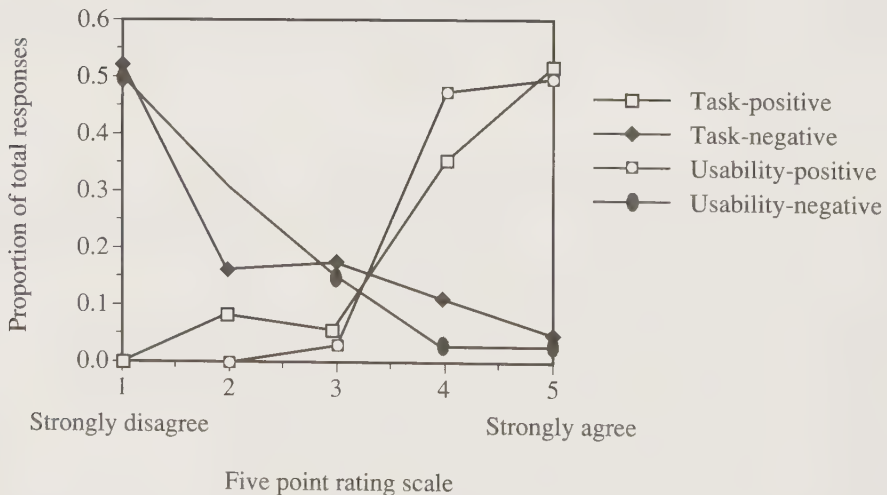


Figure 1.7 Questionnaire ratings per task and usability categories.

Cultural differences in terms of different manners and customs which could lead to irreconcilable misunderstandings were not seen to any significant extent. However, there were important differences in world view and cultural/technological milieu which did become apparent. This is important because, in design, references to ordinary everyday items is frequently used as a shorthand method for conveying an idea (see also Goldschmidt (1991) for an analysis of the use of analogical reasoning by individual designers). An English designer, wishing to convey the ideas that the object he has in mind is in the shape of a small toroid, may say 'it's like a Polo mint'. There is a confection of the same shape in Australia but it is known as a 'Lifesaver', so a communication breakdown occurs. Some interesting misunderstandings of this type which occurred included:

1. The concept of a garden leaf litter collector was seen differently. The Australian designer expected the leaves to be dry while the English designer expected them to be wet.
2. The English designer suggested a money box in the shape of a 'Mister Man' – a well-known children's book and television character. The Australian did not know of them.
3. The Australian suggested a concept like 'air hockey', a tabletop game related to ice hockey and using an air-suspended puck. This game was unknown to the English designer.

It was realized from the outset of the project that working between UK and Australia would necessitate one team working out of normal working hours. At 8:00 p.m. the Australians had finished a day's work, had a meal and were ready to relax. The English had just arrived at work, bright and fresh. Project workers experienced the normal irritations of working in an otherwise empty building, such as being locked out by security personnel, or having the air-conditioning turned off, effectively turning the laboratory into a sauna in the heat of the Australian summer. The subjects were all enthusiastic and the experiments only ran over three evenings, so these problems were only minor irritations. Nevertheless, it was expected that these environmental and temporal factors would have some impact on the Australian designers' performance. However, this expectation is not supported by the questionnaire data, which suggests that the English designers were less satisfied with their performance (see Figure 1.8).

Another environmental difference concerned the hardware configurations at each terminal of the link. As noted above, the cooperation terminals in Australia and England differed in important ways. The fact that the designers were able to achieve their tasks satisfactorily points to the generic usability of the ROCOCO Sketchpad tool. However, the hardware configurations in which this software resided were not ergonomically equivalent. The Liveboard used in Australia is an experimental device and suffered from a number of problems. In the first place, it proved difficult to calibrate the display cursor with the stylus tip. As a result the displayed mark was visually disconnected from the stylus tip. The Australian designers expressed dissatisfaction with this arrangement, although in practice they appeared to adapt to it. Another problem was that the image was front projected into the Liveboard. As a consequence the user had to work around his own shadow because his body, being located between the projector and the Liveboard surface, inevitably interfered with the projector beam. Again, the designers appeared to compensate for this by deft footwork and bodily contortions! One designer made considerable use of a sketchpad image visible on the Macintosh computer used to drive the overhead projector, which was perhaps a way of overcoming some of the Liveboard

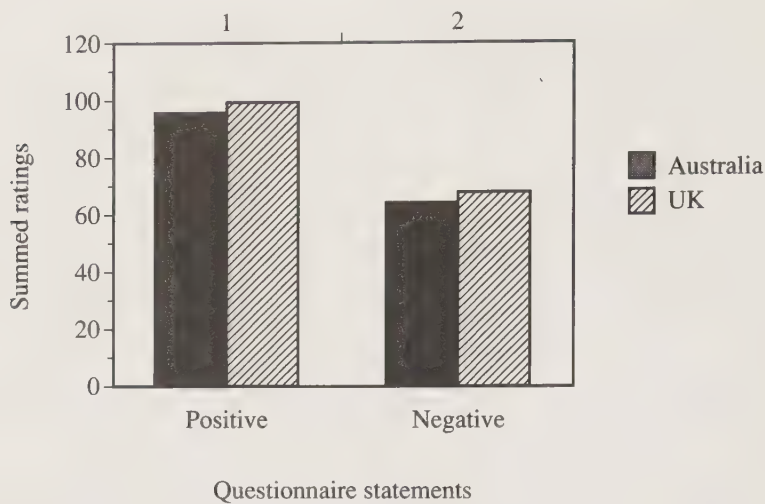


Figure 1.8 Summed ratings for responses to task statements.

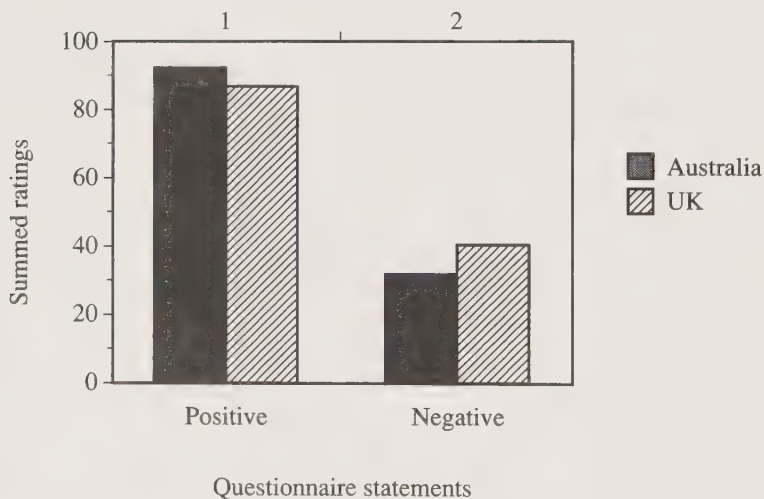


Figure 1.9 Summed ratings for responses to usability statements.

problems. Again, the questionnaire data does not corroborate these observations, suggesting to the contrary (see Figure 1.9) that the English designers were less happy than their Australian counterparts with the usability of the system.

1.5 DISCUSSION

Generally, the experiment was very successful, demonstrating the usability and usefulness of the ROCOCO Sketchpad as a tool for synchronous design at a distance. It appears that as the designers' engagement in the task developed they became unaware of the medium and the distance between them, and cooperated in a natural way. Indeed, this engagement in the task appears to have overcome the evident shortcomings of the technology. The designers rapidly developed a relationship which, while oriented towards the task, included social exchange about their own lives and experience. The fact that they were using a CSCD system to do this did not appear to be a hindrance to interaction.

The results concerning differences between the English and Australian designers' subjective assessment of the system in terms of task performance and usability is surprising. Four explanations can be postulated to account for this result. Firstly, the atmosphere in the laboratory in Australia was that of a special occasion. The sessions took place at night, in an empty building, in a room dimly lit by the light from the Liveboard projector, and with large windows that overlooked stunning views of Adelaide set against a brilliant sunset sky. These factors gave rise to feelings of elation and excitement. In these circumstances, it is possible that the subjects' enjoyment of the whole experience impaired their judgement, thus making them more compliant and agreeable than they might have been in a more familiar setting. A second explanation again relates to familiarity, but this time to the familiarity of the drawing surface. It may be that the stylus and tablet of the ROCOCO Station induced expectations of higher drawing quality in the English designers than did the rather clumsy Liveboard in the minds of the Australians, thus making the latter more tolerant of its inadequacies. The open questionnaire responses provide some corroboration for this hypotheses in that only the English designers complained about the quality of drawing and writing possible with the technology. Another possibility is that the English designers compared the transcontinental experience less favourably with their previous use of the laboratory version of the ROCOCO Station. Finally, the results may be taken at face value as indicating that environmental, temporal and technological differences had no effect on the designers' subjective views of the system. It may be that the results of a project concluded over a longer period of time would throw further light on this matter.

1.6 CONCLUSIONS

While the ROCOCO Sketchpad appears to be a useful representational and communicational tool for the first meeting of a design project, clearly its applicability for later meetings is limited. First, it does not include functions for developing and manipulating detailed drawings. Nor does it allow drawings produced earlier to be brought into the electronic workspace for manipulation and development during a meeting.

It is not technically difficult to provide many of these functions in a basic form. However, what is unclear is what functions should be provided and how they are to be provided in a usable manner. Designers are extremely adept in using rulers and templates and can rapidly produce quite detailed drawing. Such drawings, which might be produced as modifications or developments of existing drawings can be produced at speeds that allow a natural discourse to progress between those involved in a meeting. Computer

replication of sketching tools that exploit designers' skill in freehand drawing has proved relatively straightforward and worth while. Similarly, replicating the tools used to support high-speed detailing would seem worth while, but it is by no means obvious how this can be achieved.

Furthermore, the value of a 'replicationist' approach to providing detailing functionality is questionable. The value of producing electronic equivalents of pen and paper is that drawing, like language, is a skill that just about everyone possesses to some degree. However, the more specialized skills of high-quality design drafting are likely to be acquired using tools specific to a domain or even the preferences of the skilled individual, hence requiring the development of less generally familiar and therefore immediately applicable tools. A further argument against straightforward replication of traditional technology is that such technology can be used for drawing modifications to a design, but the drawing itself cannot be directly manipulated. In addition, three-dimensional visualization is time-consuming. In contrast, we now have available sophisticated computer drafting and modelling systems that significantly improve on traditional means in these areas.

However, the plethora of such systems (each of which may be enhanced for shared use in synchronous remote meetings) makes it quite likely that remotely located designers willing to meet and discuss the results produced by one of them using such a CAD system will not all be skilled users, and hence have equal control over the objects of discourse. Even given this unequal situation, we hypothesize that the fluency with which CAD objects can be created and manipulated in current systems will prove inadequate for effective discourse. The twin problems of democracy of control and interaction fluency are ones that must be solved in order to effectively support synchronous remote design meetings beyond sketch design.

As noted earlier, the successful completion of a design may require inputs from a variety of specialisms, where individual specialists may wish to bring to a synchronous remote meeting material produced on and displayed using their own personal computers. In this case, individuals are likely to be the controllers and only users of their applications, and the others mere viewers of the data. The issue is how to support group communication about views of objects generated with single-user applications.

The Australian and English people share much in common; however, differences between the two cultures did become evident during the studies. Resolution of such differences required discussion and negotiation in order to achieve a common understanding of differing states of mind, knowledge, manners, customs and social and physical environments. In general then, it will not be adequate to provide communication systems for remote synchronous work that only support communication and negotiation about the task.

Interestingly, the act of resolving misunderstandings of a cultural nature can have a positive effect on the execution of the task. For example, in executing the leaf mulcher task, the fact that the designers inhabited different environments led to a broader discussion of the nature of the task; leaves in winter being wet in England and dry in Australia. Hence, possible requirements of the brief (e.g. the design should be suitable for both markets) can be made explicit through the resolution of different viewpoints. Of course, such conflicts are likely to occur where any group of designers is working on a task; however, it seems reasonable to postulate that their incidence is likely to be more frequent in culturally diverse project teams.

Australia has three time zones, and when one adds the complication of summer 'daylight saving' which applies in the UK and in some – but not all – Australian states, and that summer in Australia is six months earlier/later than summer in the UK, simply arranging appropriate synchronous work times becomes difficult. However, if this type of synchronous work is to become part of a normal work practice this problem will need to be recognized. In practice, synchronous remote working between Europe and Australia is likely to be infrequent and short in duration as a result of time differences, thus forcing greater reliance on asynchronous work. Given that uncertainties and ambiguities in understanding between groups members will not be resolvable, in general, by an immediate synchronous connection, the success of the asynchronous work would seem to depend crucially on the availability and range of knowledge about the design available to team members; that is to say a purely descriptive design database will be inadequate.

In conclusion, the work described in this paper demonstrates that a limited form of remote team design is now technically and practically feasible, and indeed enjoyable (at least in the short term). However, there are many factors which limit application of this way of working in real design projects. Research effort will need to be focused on the provision of facilities that support all stages of a design project, and all members of the project team, wherever they are located and whatever their skills, cultural backgrounds, organizational contexts and work preferences. Only when these limitations have been removed, and when connections are as simple to make as a telephone call will we be able to assess and exploit the full potential of computer-supported cooperative design.

ACKNOWLEDGEMENTS

Thanks to our colleagues at both ends of the link: Tony Clarke, John Connolly, Steve Garner, Hilary Palmen and Andre Schappo from the ROCOCO Project at Loughborough University; Arek Dadej of the University of South Australia. The ROCOCO project is supported by the AITP programme, reference numbers: IED3/7/1225, SERC GR/F/3514. Finally, thanks to the designers who took part in the experiments.

REFERENCES

- ACM Press (1986) *Proceedings of the First Conference on Computer Supported Cooperative Work*, ACM Press, 3–5 December, Austin, Texas.
- ACM Press (1988) *Proceedings of the Second Conference on Computer Supported Cooperative Work*, ACM Press, 26–28 September, Portland, Oregon.
- ACM Press (1990) *Proceedings of the Third Conference on Computer Supported Cooperative Work*, ACM Press, 7–10 October, Los Angeles, California.
- Bly, S. A. (1988) A use of drawing surfaces in different collaborative settings, in *Proceedings of the Second Conference on Computer Supported Cooperative Work*, ACM Press, 26–28 September, Portland, Oregon, pp. 26–28.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems*, ACM Press, 25–27 April, Boston, Texas, pp. 184–92.
- Bannon, L., Robinson, B. and Schmidt, K. (eds.) (1991) *Proceedings of the Second European Conference on Computer Supported Cooperative Work*, 24–27 September, Amsterdam, Netherlands, Kluwer Academic Publishers.

- CSC (1989) *Proceedings of the First European Conference on Computer Supported Cooperative Work*, 13–15 September, London, UK, Computer Sciences Company.
- Goldschmidt, G. (1991) The dialectics of sketching, *Creativity Research Journal*, 4(2), pp. 123–43.
- Rodden, T. (1991) CSCW and distributed systems, in *Proceedings of the Second European Conference on Computer Supported Cooperative Work*, 24–27 September, Amsterdam, Netherlands, Kluwer Academic Publishers, pp. 17–31.
- Scheifler, R. W. and Gettys, J. (1986) The X-Window system, *ACM Transactions on Graphics*, 5(2), pp. 79–109.
- Scrivener, S. A. R., Clark, S. M. and Keen, N. (1991) *The role of workspace-replication in the development of remote CSCW systems*, LUTCHI Research Centre, Loughborough University, Loughborough, Leicestershire, UK.
- Scrivener, S. A. R. and Clark, S. M. (1992) Computer-supported cooperative work in *Proceedings of the Seminar on Intelligent Buildings and Management*, 28–29 January, Heathrow, UK, Unicom Seminars Ltd, pp. 148–67.
- Scrivener, S. A. R., Clarke, A. A., Connolly, J. C., Garner, S. W., Clark, S. M., Palmen, H. and Smyth, M. (1992) The ROCOCO project face-to-face studies, LUTCHI Research Centre, Loughborough University, Loughborough, Leicestershire, UK.
- Smith, R. B., O'Shea, T., O'Malley, C., Scanlon, E. and Taylor, J. (1989) Experiments with a distributed multimedia problem solving environment, in *Proceedings of the First European Conference on Computer Supported Cooperative Work*, 13–15 September, London, UK, Computer Sciences Company, pp. 19–34.
- Stefik, M., Bobrow, D. G., Lanning, S., Tatar, D. and Foster, G. (1986) WYSIWIS revised: early experiences with multi-user interfaces, in *Proceedings of the First Conference on Computer Supported Cooperative Work*, 3–5 September, Austin, Texas, ACM Press, pp. 276–90.
- Tang, J. C. and Leifer, L. (1988) A framework for understanding the workspace activity of design teams, in *Proceedings of the Second Conference on Computer Supported Cooperative Work*, 26–28 September, Portland, Oregon, ACM Press, pp. 244–9.
- Tang, J. C. and Minneman, S. L. (1991) Video WhiteBoard: video shadows to support remote collaboration, in *Proceedings of the ACM/SIGCHI Conference on Human Factors in Computing*, April, New Orleans, pp. 315–22.
- Wilson, P. (1990) *An introduction to CSCW*, CCTA.

APPENDIX 1: DESIGN BRIEFS USED IN THE EXPERIMENTS

Design brief No. 1

Fallen leaves represent an excellent potential source of compost as leaf mould. However, the gathering and processing of leaves represents such a difficult procedure that many gardeners or smallholders tend to ignore this resource and purchase alternative products for soil improvements. The 'client' in this instance is a large manufacturer of hand and powered garden products with capabilities to form a wide range of materials.

You are asked to consider this area together with the implicit problems and to develop a response to it. You only have 55 minutes of time available for this exercise and the 'client' would like to see a fairly detailed proposal from your team by the deadline. You may find it necessary to consider the influence of materials, production, marketing or technological details in your development.

Design brief No. 2

Your client, a well known toy manufacturer, has come to an arrangement with a major building society in an attempt to get children to save more money. They have jointly decided that they will provide any investing family with a free money box. They have come

to your design team for ideas for the design of this money box and are particularly interested in exploiting the movement or noise associated with coins. They do not want mere animal shapes, cartoon characters, etc. They have not ruled out simple electronics if you think this is advantageous.

You are asked to consider this area together with the implicit problems and to develop a response to it. You only have 55 minutes of time available for this exercise and the 'client' would like to see a fairly detailed proposal from your team by the deadline. You may find it necessary to consider the influence of materials, production, marketing or technological details in your development.

Design brief No. 3

The directors of a company specializing in the manufacture of sheet metal products have become aware of the growing number of relatively cheap cast and sheet metal barbecues available in the home market which have been imported from the Far East. They believe that sufficient demand exists for an up-market barbecue offering a rotating-spit facility. A raised stand is seen as an important element as many users will be in the older age groups. Such a product would need to be folded or dismantled and transported – probably inside a caravan, as the largest purchasers are the holiday/camping market. Compact storage at home should also be a factor.

You are asked to consider this area together with the implicit problems and to develop a response to it. You only have 55 minutes of time available for this exercise and the 'client' would like to see a fairly detailed proposal from your team by the deadline. You may find it necessary to consider the influence of materials, production, marketing or technological details in your development.

APPENDIX 2: COLLATED RESPONSES TO OPEN SECTIONS OF QUESTIONNAIRE

1. The role of the technology?

- Very impressive, I can see many applications for this system.
- (En)Able the collaboration over long distance – very comfortable with this situation – enabling still a sense of privacy yet still able to interact with other individuals.
- Great to be involved in something of this nature. I appreciated a computer more than usual.
- Obviously enables long distance collaboration – would be nice to refer back to drawings and take part of these drawings and reuse. Overall amazing stuff.
- Made the whole thing possible, obviously. The inconsistency of the rate at which the drawings appeared on both screens was a problem.
- Made a design session possible that otherwise would not have been.
- Became less obtrusive as the session went on – this is obviously a good thing. Using it was easier and more natural.
- Pretty impressive!! Enabled good communication.

2. The design task?

- A thought-provoking, interesting task.
- Very enjoyable working with someone else with this task – having to immediately interact with stranger – design task made this interaction easier.
- Less challenging than the other two.
- Great fun – worked well with partner. Enjoyed (it).
- Worthwhile brief – got interested in it. Interactive working greatly facilitated integration of ideas – both using the same sheet of paper at the same time, the same way up.
- Good brief, although more time was needed to finish it.
- Enabled basic trans-global interactive drawings. I can see this sort of set-up being used extensively by design consultancies and industry worldwide.
- Fairly challenging for the time available but quite interesting.

3. Your performance as an individual?

- Not bad, some interesting ideas over a short time span.
- Successful – could have been more creative – more drawings – but was not used to the level of design knowledge of partner.
- I was pleased with my ideas.
- Very pleased.
- Found my drawings disappointing, writing was hard – once you accept the limitations of the system you know what you can and can't do.
- I think I worked well, sparking off good new ideas.
- Felt I had a basic sort of control over the equipment, improvements in stylus-tracking definition will improve the standard of writing and drawing.
- I was able to work well, and enjoyed the session.

4. Your performance as part of a design pair?

Worked well and listened to what each other had to say and built on these ideas.

- Thought went very well – felt very comfortable.
- I think we work very well together.
- Worked well – had fun – easy to communicate ideas.
- Did OK – once you get used to the speech delay and echo you can work together effectively and establish a rapport.
- I found it 'productive' to work with Nick and stimulating working together.
- Felt it went really well – once you get to 'know' the other guy you can predict his actions to a certain degree – gets more comfortable.
- I gave equally to the design problem as my partner and we worked well together, triggering off different ideas together.

5. General comments?

- An exciting project to be involved in.
- Very, very enjoyable – made a design task good fun – enjoyed whole concept. Felt very comfortable with system, task, partner.

- Great interest in using system – keeps private but also interact as well – great way of interacting with stranger.
- The main fault was the delay in the re-drawing of the images – this resulted in pictures being drawn on top of each other and one talking about an image that the other person could not see; otherwise impressive. (P.S. Work needs to be done on increasing the sensitivity of the stylus.)
- I couldn't have done any better at this conceptual stage with someone in the same room (apart from the drawing quality).
- Promises to be very useful commercially and not just for people who can draw – when video comes in as well make sure you get a patent on whatever you can!!
- Really good communication with the telephone link.

SOME DESIGN PRINCIPLES FOR SHARING IN TIVOLI, A WHITEBOARD MEETING-SUPPORT TOOL

Thomas P. Moran, Kim McCall, Bill van Melle, Elin Rønby Pedersen, and Frank G. Halasz

ABSTRACT

We have built a shared drawing system, called Tivoli (Pedersen *et al.*, 1993), for the Xerox Liveboard (Elrod *et al.*, 1992). It was designed to support small group meetings, both in a single room and between remotely connected sites. In building this system, it was necessary to make several basic design decisions about general system features, such as:

- a replicated system sharing architecture
- an object-based drawing representation
- immutable objects

As we have designed, implemented and begun to use the system, we have become more fully aware of the implications of these decisions. The purpose of this paper is to enumerate and discuss some of the design principles, issues and lessons we have identified from our experience so far. We begin by giving an overview of the Liveboard and Tivoli; both functionality and implementation architecture.

2.1 LIVEBOARD AND TIVOLI

The Xerox Liveboard is really a computer with a whiteboard-sized (46 x 32 inches), rear-projected display and with pen, as well as keyboard and mouse, input devices. The Tivoli software implements the whiteboard metaphor, so that users can approach the Liveboard as a whiteboard – just go up to it and scribble with the pen on the display surface.

The Liveboard and Tivoli exist in several versions. The prototype Liveboard, described in Elrod *et al.* (1992), was first created at Xerox PARC. The physical packaging was redesigned (see Figure 2.1), and this has become a Xerox product.¹ The research versions of Tivoli were created at PARC. The first version, described in Pedersen *et al.* (1993), did not implement the shared drawing facility; but the second version did. The product version of Tivoli is a result of porting and further development of the second research version. The issues discussed in this paper arose while creating the second research version. Thus, we



Figure 2.1 The product version of the Xerox Liveboard. The Liveboard has keyboard, mouse and pens for input. Tivoli is shown on the display, with the soft control buttons on the left and bottom of the display. A video window overlays the Tivoli window. The Tivoli page has an important image which is being annotated with Tivoli strokes.

¹ The product unit in Xerox is called Group Communications.

will describe the functionality of this version. Because this is one of the first shared drawing products, we will also mention a few considerations involved in the product version.

2.1.1 Tivoli functionality

The prototype Liveboard is powered by a Sun SparcStation, and Tivoli is based on the UNIX operating system and the X-Windows system. Tivoli creates a large drawing area covering most of the display, except for a few control buttons and icons along the left and bottom edges (see Figure 2.2).² The user can draw directly on the drawing area. Touching the display with the pen, dragging and lifting produces a stroke. While the user is dragging, 'ink' appears at the pen tip. Similarly, after acquiring the eraser cursor, the pen becomes an eraser and the user can erase strokes by sweeping over the display, each stroke disappearing as it is touched by the eraser. The user can also edit the drawing by means of gestures, which are 'meta-strokes', created like regular strokes after the user has acquired the gesture cursor. For example, drawing a gesture loop selects the strokes contained within the loop. The selected strokes can then be deleted or moved (by making delete or move gestures), coloured or made heavier (by touching the colour or line-thickness buttons), and so on.

Tivoli supports drawing on multiple *pages*, providing for the creation of new pages and rapid switching between pages (by caching the images of each page). The current pages can be printed and can be saved in a file. Pages can be retrieved from Tivoli files and can also be imported from raster formats or from a scanner.³

Tivoli was designed to support small meetings in front of a single Liveboard, and indeed this is mostly how it is used. But Tivoli was also designed to be a shared drawing tool. Each Tivoli program running on a machine is called a Tivoli *site*. At any time, a Tivoli site can connect to any other Tivoli site that is accessible over a network. Once connected, the sites exchange their current pages, so that each site has an identical set of pages (both its own original set of pages plus the pages received from the other site). Thereafter, any user-driven drawing or editing operations at either site are sent to the other site, thus providing a shared drawing functionality. The second research version of Tivoli allows only two sites to be connected.

Tivoli can run on any SparcStation, as well as on Liveboards, although drawing with a mouse at a workstation is awkward at best.⁴ Thus, a user at a workstation in an office can be connected to a group meeting at Liveboard. We intend to run Tivoli on pen-based notebook computers, so that multiple users in a meeting room can share notes from where they are sitting, but such computers have not yet proved to be powerful enough to run a Tivoli site. We have, however, experimented briefly with a wireless palm-sized input device to allow a user to interact in limited ways with the Tivoli on a Liveboard from anywhere in the room.

² Tivoli is implemented in C++. The control areas are implemented with the XView toolkit.

³ Usually, a raster image is placed in the 'background' of a page, where it is 'indelible', i.e. it cannot be edited. However, Tivoli has a facility to convert images into stroke objects, which can then be placed in the 'foreground' of a page, where they can be edited in the same way as hand-drawn strokes can.

⁴ Tivoli also allows keyboard input; see Section 2.2.1.

☒ (c) Copyright 1991-1993
 Xerox Corporation
 All rights reserved

Quit	Start fresh
Help	Share (open)

Index	Page 1 of 4	New Page
Page ops	Selection	Get
Undo	Save	Print
Misc ops		

stroke	width	color	

Page 1: **AGENDA**

Introductions

Staff report
New features

Issues in Shared Tivoli

Resolving conflicts

Network topology

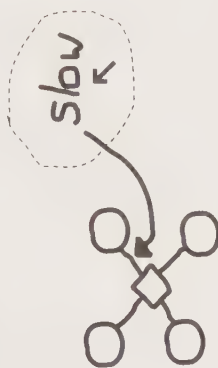


Figure 2.2 The Tivoli display for the third research version on a high resolution black and white display (on this display all the control buttons are on the left). The Tivoli page has a handwritten list. The first item is enriched by a selection loop (dashed strokes indicate Tivoli gestures made. A 'pigtail' gesture at the end of a dashed line means delete). This Tivoli is currently connected to another site, which is indicated by the 'Share' button (on the upper left next to the 'Help' button) being marked 'open'. The arrow cursor in the lower right of the display shows where the remote user is pointing. The lighter selection loop belongs to the remote user.

2.1.2 The Liveboard/Tivoli product

For the product version, the Liveboard was adapted to use a PC under Microsoft Windows with Pen Extensions, both because it is the most common platform for customers and because it takes advantage of a wide range of existing software packages. Tivoli was ported to the PC and Windows with Pen Extensions. The product supports the use of phone lines as well as networks for sharing between Tivoli sites. The initial product has only two-point connections.

The Liveboard/Tivoli product has several facets:

1. It is a device with an overall design that physically fits into the contemporary meeting-room setting.
2. It is a large-display computer serving as a presentation device for existing PC software.
3. It is a device that looks like and behaves like a whiteboard, i.e. it is simple to start using in familiar ways.
4. It is a tool that provides the value-added functionality of sharing materials in distributed meetings.

It is the sharing functionality that most customers use to justify the purchase of a Liveboard. Sharing includes both the dynamic shared drawing that Tivoli provides and the sharing of snapshots from other applications by importing them into Tivoli, which distributes them to other sites so they can be jointly annotated. Our experience with regular customer usage of sharing is still limited.

2.2 TIVOLI ARCHITECTURE

We first describe Tivoli's object-oriented graphics model and then describe the architecture for sharing this model.

2.2.1 Tivoli object model

The intent of Tivoli is to look and feel like an informal scribbling medium (i.e. a whiteboard), akin to a painting program. But at the implementation level, Tivoli is more like a drawing program than a painting program. All data in the system are *Tivoli objects*. A Tivoli object is represented as a set of properties describing its x - y page location, its extent, its visual appearance, the device that created it, its time of creation etc. The most common Tivoli object is the *stroke*, which reifies the user's basic pen down-drag-up motion. It is represented as a path of line segments, a stroke width and a colour. There are two other kinds of Tivoli object. The system allows keyboard input and imported pixel images, each character or image being a Tivoli object. The graphical routines know how to measure each of these types of object (e.g. detecting intersections and inclusions) and how to render them into display buffers.

Three important properties of Tivoli objects are that they are independent, uniquely identifiable and immutable.

- *Independence*. A crucial property of the set of Tivoli objects is that they are independent. That is, there are no relationships between them, so that operations on any object have no effects on other objects. This is straightforward in stroke-based

drawing. For example, deleting one stroke has no effect on any overlapping or nearby strokes. Note that we also treat characters in the same way, which is in contrast to normal text editing. In text editing, the characters are structured into strings. Deleting a character, for example, causes the subsequent characters to move over. But in Tivoli there are no string structures; each character object is independent and occupies a fixed location. Thus, deleting a character does not cause characters on the right to shift over.

- *Unique identifiability.* Each Tivoli object is assigned a unique identifier (UID) upon creation, so that it can be unambiguously and independently referred to, both locally within a site and among different Tivoli sites.
- *Immutability.* Tivoli objects can only be deleted; they cannot be changed. To change an object (e.g. to move it), it must be deleted and a new object created that incorporates the change. The new object has a new UID.⁵ Each change operation is represented in the history list as an old–new object pair, which allows the undoing of the change.

2.2.2 Tivoli object-management architecture

Tivoli has several modules for managing operations on objects. The *Device Manager* interprets user actions as semantically meaningful operations on objects. The *Object Manager* maintains the current data. The *Display Manager* maintains the current view of the current data. The *Historian* maintains the history list of operations. At the centre of the traffic is the *Dispatcher* module, which receives interpreted operations from the Device Manager and passes them on to whichever other modules need to know about them. These modules (plus modules for reading, writing, importing, exporting, printing, scanning) are adequate to run a standalone Tivoli application. Another module is needed to enable shared-drawing functionality.

2.2.3 Tivoli shared-drawing architecture

Tivoli's shared-drawing functionality is built on a replicated architecture scheme. Independent Tivoli processes, usually running on separate processors, communicate via UNIX sockets. This design was motivated by the fact that Tivoli had to work most of the time as a standalone facility and that fast response to user actions was a prime goal. Our design philosophy is to respond to local user actions immediately and then notify other Tivoli sites, and then, if a conflict is detected, be prepared to deal with it in some way – resolve it by priority rules, warn the users and let them resolve it, rescind the actions, and so on.

The shared-drawing functionality is provided by a *Liaison* module in each Tivoli site for communicating with other Tivoli sites via a *Tivoli protocol*. The Liaison sends messages telling what operations are generated locally and receives messages telling what operations were generated remotely. The Liaison has full access to the Dispatcher. The Dispatcher passes all local operations to the Liaison. The Liaison passes all remote operations to the Dispatcher. As far as the Dispatcher is concerned, operations coming from the Liaison are treated the same as operations coming from the local Device Manager.

⁵ If we consider a more abstract object, then the UIDs can be seen to be labels for the *states* of the abstract object during its lifetime.

When Tivoli sites are initially connected, they exchange all their current Tivoli objects. User actions are blocked during this exchange process. After this, all sites have the same state (see section 2.1.1). It is the role of the Liaison modules to dynamically maintain the same state at all sites. Mostly this is done by broadcasting all operations as they are generated at each site.

Consider an example where a user at one site deletes some Tivoli objects by a loop gesture (to select them) and then a delete gesture. (Figure 2.2 pictures a selection loop plus the pigtail gesture for deletion.) The following operations are executed locally and sent to the other site:

1. Create the loop gesture stroke
(send the Create-Gesture operation)
2. Define the selection to be the list of UIDs of the objects enclosed in the loop
(send the list of UIDs with the Define-Selection operation)⁶
3. Create the delete gesture stroke
(send the Create-Gesture operation)
4. Delete the objects in the selection
(send the Delete-Selection operation)
5. Delete the delete gesture stroke
(send the Delete-Gesture operation)
6. Delete the loop gesture stroke
(send the Delete-Gesture operation)

Operations set and utilize *device state*. The Selection is a part of an input device's state, that is, each pen can have a different selection. Colour, thickness and current page are also part of a pen device's state. The Liaison for a given Tivoli site keeps other sites informed about the current state of its devices.

2.3 DESIGN PRINCIPLES FOR SHARING

We now lay out a set of design principles that have guided the design and implementation of the sharing functionality in Tivoli.⁷ The principles can be grouped by the kinds of issue they deal with:

- concurrent manipulation of objects and structures
- coordinating fine-grain user interactions
- providing both public and private workspaces

We articulate the principles below as organized responses to design problems: we first present a problem, then the principle, give examples and discuss further issues. Before we give the principles, let us be clear about the system criteria that the principles are trying to achieve.

⁶ Note that the selection is computed locally and the *results* are sent to other sites. It is done this way for consistency, in case the other sites momentarily had some different set of objects within the boundaries of the loop.

⁷ These principles are somewhat of a mixed bag – some are general principles, some specific heuristics, some techniques – but for purposes of this chapter we shall just call them all principles.

2.3.1 Criteria for designing a sharing architecture

- *Quick response to local user actions.* It is critical that a whiteboard system keep up with the user's pen actions, so that they seem to 'flow out of' the pen. This responsiveness must be preserved during sharing. This was the initial motivation for a replicated architecture, since we do not want every user action to have to take a round trip to a central server.
- *Low latency for remote user actions.* It is also important that user actions from other sites be received quickly, in order to maintain smooth interactions across the audio channel.⁸ This was another motivation for a replicated architecture, where user actions can be sent directly to other sites.
- *Operate with a low bandwidth across sites.* We want to be able to share data across low-bandwidth channels, such as phone lines or wireless connections.
- *Preserve data consistency among sites.* Data consistency means that the state of objects, structures and the history of operations are the same at all sites. What we mean by 'same' will be discussed below. When users act concurrently at different sites, the system must detect any potential inconsistencies⁹ and see that they are resolved.
- *Manage conflicting user intentions.* While the system resolves concurrent actions to preserve data consistency (according to the system's definition of consistency), the intentions of the users may be in conflict. There is no way the system can know about user intentions or how to resolve conflicting intentions, but it can make it easier for users themselves to manage the conflicts.
- *Avoid locking out user actions.* A common mechanism for preventing inconsistencies and conflicts is to lock users from each other's local area of workspace. However, empirical studies of shared drawing activity show that users often do want to work closely together on the workspace (Bly, 1988; Bly and Minneman, 1990).
- *Keep concurrency mechanisms simple.* Maintaining consistency in a replicated architecture can involve complex mechanisms (e.g. Ellis and Gibbs, 1989). We want to use the simplest concurrency mechanisms possible for the requirements of the Tivoli application. Fortunately, the particular properties of shared drawing allow us to use simpler mechanisms. Further, actual conflicts seem to be fairly rare events which can be limited to minor consequences if treated correctly.

2.3.2 Object-level principles

Problem: When there are two concurrent operations at different sites, they are executed locally and then exchanged. Thus, the operations are executed in reverse order at each site. This can result in data inconsistency if the operations are not additive. Thus:

PI: Maintain object independence and the additivity of operations.

In the domain of drawing, we observed that objects seem to be naturally independent and the operations on them additive. This makes it easier to maintain data consistency.

⁸ An audio or audio-video channel is obviously critical for remote collaboration to work, but this channel is not part of Tivoli.

⁹ An inconsistency is 'potential' in the sense that it would be a real inconsistency if it were not detected and resolved.

The design principle is to preserve this property as we extend the domain. This is why we treat text characters as independent objects and not as strings.

Example: User_a at site_a draws stroke₁ while user_b at site_b concurrently draws stroke₂ in such a way that stroke₁ and stroke₂ overlap. Although these strokes are created in the opposite order at each site, the end result is the same, because of the additivity of these operations.

Caveat: Consistency is not an absolute property. In the current implementation our data is consistent only if we ignore the 'Z-order' of objects (i.e. how objects overlay each other on the 2D drawing surface), since we do not control Z-order. In the example, stroke₁ would overlay stroke₂ at site_b, and stroke₂ would overlay stroke₁ at site_a. This is not a problem with monochrome displays, where the overlap is not visible. It is only a minor problem for colour displays as long as the overlap of scribbled strokes is not a significant feature, which of course it could be if strokes are used for 'scratching out' (i.e. overlaying other objects) or for highlighting (i.e. underlaying other objects with a light colour).

Problem: If two users concurrently act on the same object, the outcome can be inconsistent. The usual remedy is to prevent this by locking out one of the users. However, we can:

P2: Use the immutability of objects to prevent inconsistencies.

This works as follows: when an object is subjected to concurrent operations, each operation causes the creation of a new object from the original object, i.e. multiple objects are 'spawned' concurrently from the same object.

Example: Two users concurrently move stroke₁ to two different locations. The result is that two new strokes, stroke_{1a} and stroke_{1b}, appear, one at each location; the original stroke₁ is deleted.

Example: User_a deletes stroke₁ while user_b moves it. The result is that stroke_{1b} is created in the new location; stroke₁ is deleted both explicitly by user_a and implicitly by user_b (in changing from stroke₁ to stroke_{1b}).

When a system detects a potential inconsistency between two concurrent operations, different policies can be applied to prevent the inconsistency. The possibilities are:

- Do neither (i.e. retract them both).
- Choose to allow only one of them (by some choice rule applicable at both sites).
- Somehow combine them both (in the case of two moves, for example, do a 'vector addition'; but this policy is not general).
- Do both of them.

The last policy is the least restrictive, since both users 'get their way'. It is made possible by the immutability of objects¹⁰ (although it is not strictly necessary with this scheme).

The downside of having immutable objects is the space it takes to save all the objects created as users edit their drawings, which allows operations to be undone. For this reason (and because it does not really affect the consistency-maintenance policies), we are currently favouring a switch to mutable objects.¹¹

¹⁰ If objects were mutable, then an object could only be transformed into a single new state. Spawning new objects could only come about by duplicating objects. While this is possible, it is not an 'obvious' thing to do.

¹¹ The scheme in Rhyne and Wolf (1992) takes the extreme position in which there are no objects *per se*, just a history structure of operations that can be executed, 'unexecuted', and re-executed as needed to produce objects.

Problem: While any of the above policies result in consistent data, the initiating concurrent operations are probably a result of conflicting user intentions. Therefore:

P3: *Alert users to conflicting concurrent operations so that they can resolve them.*

Example: Moving the same object concurrently results in two objects, which should be a surprise to the users. However, when an object is concurrently deleted and moved, the user doing the move is not aware that a deletion took place.

The principle is clear, but we have not as yet refined the alerting interface beyond what was a natural side-effect of spawning operations. For example, we could use audio to signal potential conflicts when users concurrently operate on the same objects.

Another example: The Z-order problem again. If user_a changes the colour of a stroke to red concurrently with user_b changing it to green, then user_a will see the green stroke and not realize that there is a red stroke underneath it. User_b will have the inverse problem.

2.3.3 Structure-level principles

Problem: When objects have structural relationships between them (such as the relations defining a linear ordering structure), then the consistency of the structural relations must also be maintained. It is tempting to refer to objects in structures by their relationships to other objects (e.g. as the 'ith item in a list'¹²). But as structures change, these references are ambiguous, and will lead to inconsistencies. Therefore:

P4: *Use UUIDs to prevent ambiguous references.*

This would show very prominently with character string structures, which we do not support yet. One structural relationship that Tivoli must maintain is the linear sequence of pages.

Example: Two users concurrently delete different pages. Because the pages have UUIDs, the correct ones are deleted at both sides. However, if the pages were described by their sequential position, the description of the page later in the sequence would have a different referent at both sites.

Problem: Sometimes two operations on a structure are inherently inconsistent, according to the constraints of the structure. The problem is to be able to detect this situation locally (without having to compare the overall structures). Thus:

P5: *Use sufficient structural descriptions of operations to detect potential inconsistencies.*

A description of an operation is 'sufficient' if it includes the structural preconditions necessary to detect whether the operation is well-defined.

Example: Two users concurrently create a new page after the current final page (which has, say, UUID274). If the operations were specified as 'add new page after page UUID274', this would result in the new pages being in a different order at each site. The operations must be specified as 'add new page between page UUID274 and the end'. Each site can then detect that the operation received from the other site is inconsistent with the current local state, since page UUID274 is no longer at the end after the local operation is completed.

Problem: A detected inconsistency must be resolved quickly, which implies that each site should compute the resolution locally. The simplest way to do this is to:

¹² Schemes such as the one in Ellis and Gibbs (1989) are designed to cope with these kinds of reference.

P6: Resolve structural inconsistencies by giving sites a priority order.

The priority order among sites can be arbitrarily assigned, such as comparing machine identification numbers. It does not much matter that one site always ‘wins’, since such concurrency conflicts are rare. What is important is that all sites know the order and can compute the same result quickly.¹³

2.3.4 Interaction-level principles

Problem: Much user activity consists of ‘small-scale’ operations, such as handwriting words, which produces a series of Create-Stroke operations. These are transmitted at frequent intervals to remote sites, and so remote users can keep up with the activity. However, if a user performs a ‘large-scale’ operation, such as moving a cluster of words, then remote users can be surprised, if not confused or disturbed, when it occurs at their sites (e.g. Prakash and Knister, 1992). Of course, such ‘large-scale’ operations provide useful functionality for manipulating material, and so we must somehow ‘prepare’ remote users for these operations. Fortunately, the Tivoli user interface evokes such operations by use of gestures, which allow us to:

P7: Broadcast gestures to prepare remote users for actions.

Example: A user wants to move a set of strokes. First, the user selects them with a loop gesture, which is immediately sent to remote sites,¹⁴ thereby showing the remote users that the selected objects are about to be operated on¹⁵ (Figure 2.2 illustrates both local and remote selection loops). Then the user makes a move gesture, which again is sent to other sites, showing the remote users where the selected strokes are going to go. Finally, the selected strokes are actually moved.

Problem: Sometimes strokes are long, such as a box around some handwritten words or a long arrow between widely separated material. A long stroke takes a relatively long time to create, and if we wait till the end of the stroke, when it is finally created as a stroke object, to send it to remote sites, then it will appear suddenly to remote users. This can be annoying. But a real problem occurs when a user is making a stroke as a *conversational gesture*¹⁶ (Rhyne and Wolf, 1992), where drawing and talking are coordinated, such as circling some material to emphasize it while discussing it. Therefore, we must:

P8: Keep remote sites updated at frequent intervals on the partial creation of objects.

Tivoli accomplishes this by an ‘inner loop’ that transmits partially drawn strokes at regular time intervals.¹⁷ The Tivoli Liaison protocol has a Create-Partial-Stroke

¹³ A more sophisticated, dynamic scheme is to use time-stamps on the operations from each site. This will work to resolve conflicts and priorities ‘fairly’ assuming that the clocks at each site are ‘nearly’ synchronized.

¹⁴ A similar sequence of operations was described in section 2.2.3.

¹⁵ This raises the issue of whether the remote users should be locked out from operating on the selected objects. Tivoli does not lock users out, but we do not discuss this issue here.

¹⁶ This is totally different from the meta-stroke notion of gesture which is a command to the system. This is why the common use of the term ‘gestures’ in systems is inappropriate.

¹⁷ The appropriate time constant for the interval is a complex question. Perceptual continuity of a stroke would ideally require 35 ms intervals, but we find 250 ms satisfactory. These short times can easily be swamped by transmission delays, which are not under Tivoli’s control.

operation. The sending Liaison sends Create-Partial-Stroke operations, and the receiving Liaison tells its local Tivoli to ink them as they are received and to assemble the data from them. When a stroke completion is received, the receiving Liaison creates a stroke locally from the assembled data.

Problem: Often users do not touch the board when they are gesturing conversationally, so they are not creating strokes and remote users cannot see what they are doing. However, the Liveboard senses where its pen is pointing even when it is some distance from the display surface (Elrod *et al.*, 1992), and this location is displayed by the cursor. This allows us to:

P9: Keep remote sites updated at frequent intervals on where the user is pointing.

Tivoli uses a time-based inner loop to update remote sites on the cursor locations.¹⁸ This is of course just heuristic information for the remote users, since the pen is often inadvertently pointing to a location that has nothing to do with user activity. However, observations of initial use show that users take advantage of the updated cursor location in order to direct each other's attention to material in the shared workspace.

2.3.5 Independent work principle

Problem: The interaction-level principles apply when users are working on shared material. But users sometimes need to be able to work independently from the group, undisturbed and without being disturbing. The problem is to allow smooth shifting from shared working to independent working. One way is to:

P10: Provide 'private' workspaces where individual users can work by themselves.

Tivoli uses the multiple-page structure to provide a convenient device for both shared and private working. Tivoli users work on the same page for shared work. But any user can jump to a different page to work privately, and the default in Tivoli is to not execute transmitted Jump-to-Page operations. When user_a at site_a is working on a different page from user_b at site_b, user_a is still sending drawing operations to site_b; but user_b does not see them, because he or she is not displaying the page that user_a is working on. For the users to share the 'privately-created' material, all user_b has to do is jump to the page user_a was 'privately' working on.¹⁹ If the users want to share their work among all the pages, Tivoli provides options for the users to follow each other automatically from page to page (i.e. Jump-to-Page operations are executed at all sites).

2.4 CONCLUSION

We have tried to articulate some of the principles behind our efforts to design and implement the functionality for sharing a whiteboard drawing tool. Some of these

¹⁸ The cursor location is transmitted every 100 ms, which seems to provide a reasonable sense of the movement of the cursor.

¹⁹ This is not true privacy, in that any user can jump to any page and see what others are doing. We could provide locks on each page to keep them truly private.

principles were taken from the state of the CSCW art, and some emerged from our efforts to solve the particular problems that arose in our own design situation.

It should be clear that these principles are tentative, since they are based on our early experiences. The design is not complete, e.g. we have not implemented our sharing mechanisms to work for more than two sites. More significantly, we have not yet carried out much empirical observation beyond seeing that our current implementation actually works for at least a few users.

It should also be obvious that the set of principles is incomplete. The principles presented here do not cover the scope of issues we have already encountered; e.g. our initial explorations of an undo facility reveal that the semantics of undo are extremely complex in a shared context (see e.g. Prakash and Knister, 1992). We know of several areas where principles need to be developed: we need to define a notion of a 'session' (or meeting) and to provide facilities for managing sessions in a shared context. We need to deal with filing of materials from sessions; at present each site creates its own files of pages, making it confusing to keep track of who has the latest version. We may want a different kind of architecture for handling files than we use for sharing real-time editing operations. We want Tivoli to be able to interact with a variety of other kinds of machine and devices in a ubiquitous computational environment of heterogeneous devices (Stefik *et al.*, 1987). Liveboards, workstations, pen-based tablets at the tabletop, handheld devices and so on.

In spite of these limitations, we have found it worthwhile to attempt to rationalize our decisions and codify our understanding of them into a set of principles. We are not satisfied with the principles at this stage; they obviously need considerable refinement. This is a complex area of system design, and articulating an explicit set of principles for the area can help improve the state of the art. At minimum, they provide interesting foci for generating discussions that we can all learn from.

REFERENCES

- Bly, S. A. (1988) A use of drawing surfaces in different collaborative settings, in *Proceedings of CSCW '88, Conference on Computer-Supported Cooperative Work*, 26–28 September, Portland, Oregon, ACM Press, pp. 250–6.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface. *SIGOIS Bulletin*, pp. 184–92.
- Ellis, C. A. and Gibbs, S. J. (1989) Concurrency control in groupware systems, in *Proceedings of the International Conference on the Management of Data*, Seattle, Washington, ACM Press, pp. 399–407.
- Elrod, S., Bruce, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E. R., Pier, K., Tang, J. and Welch, W. (1992) Liveboard: a large interactive display supporting group meetings, presentations and remote collaboration, in *Proceedings of CHI '92, Conference on Human Factors in Computing Systems*, ACM Press, New York.
- Pedersen, E. R., McCall, K., Moran, T. P. and Halasz, F. (1993) Tivoli: an electronic whiteboard for informal workgroup meetings, in *Proceedings of InterCHI '93, Conference on Human Factors in Computer Systems*, ACM Press, New York.
- Prakash, A. and Knister, M. J. (1992) Undoing actions in collaborative work, in *Proceedings of CSCW '90, Conference on Computer-Supported Cooperative Work*, Toronto, Ontario, ACM Press, pp. 273–80.
- Rhine, J. R. and Wolf, C. G. (1992) Tools for supporting the collaborative process, in *Proceedings of UIST '92, Symposium on User Interface Software and Technology*, 15–18 November, Monterey, California, ACM Press, pp. 161–70.

HUMAN AND TECHNICAL FACTORS OF DISTRIBUTED GROUP DRAWING TOOLS

Saul Greenberg, Mark Roseman, Dave Webster and
Ralph Bohnet

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 technical hurdles. This chapter described 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.

3.1 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 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

The original version of this paper appeared in *Interacting with Computers*, Vol 4, No 1 (1992), pp. 364–92. Reprinted by permission of Butterworth-Heinemann.

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 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 chapter, 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 describe 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 3.1) (Greenberg and Bohnet, 1991). Its users all see exactly the same

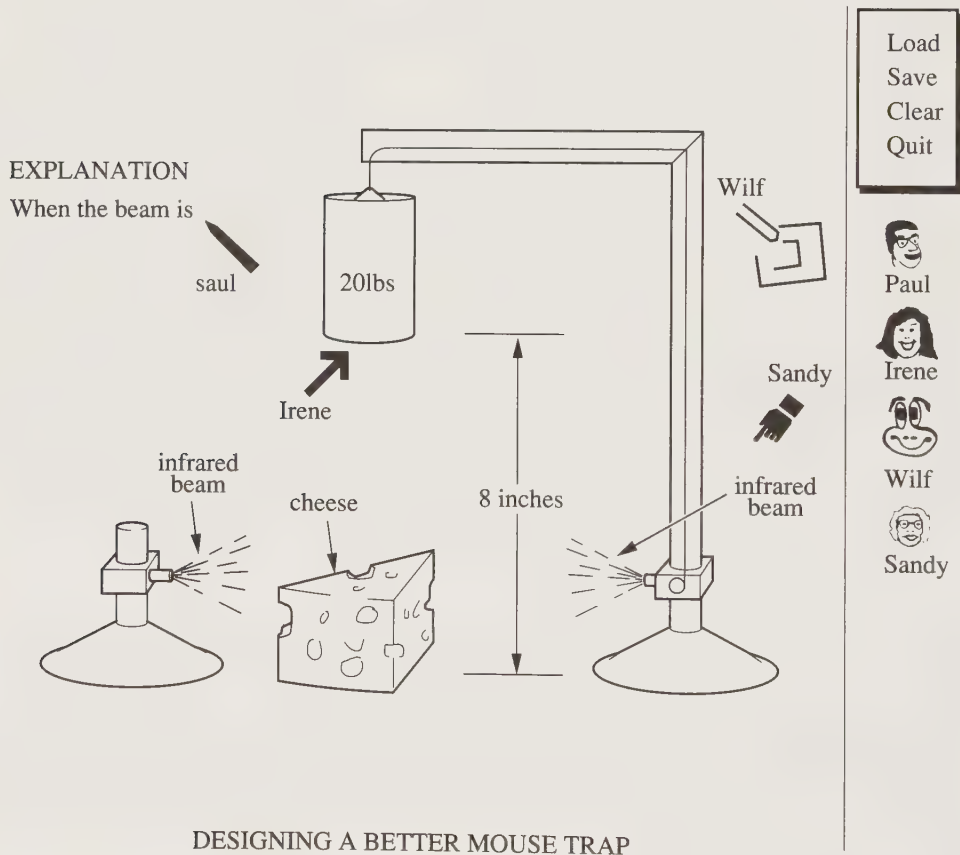


Figure 3.1 Sample GroupSketch session.

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.

- *XGroupSketch* is a re-implementation of *GroupSketch* (see Figure 3.2). It differs in that it runs within a standard resizable X-window, has a scrollable drawing surface, and includes features such as different pen sizes and colours, 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 3.3) (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.



Figure 3.2 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.

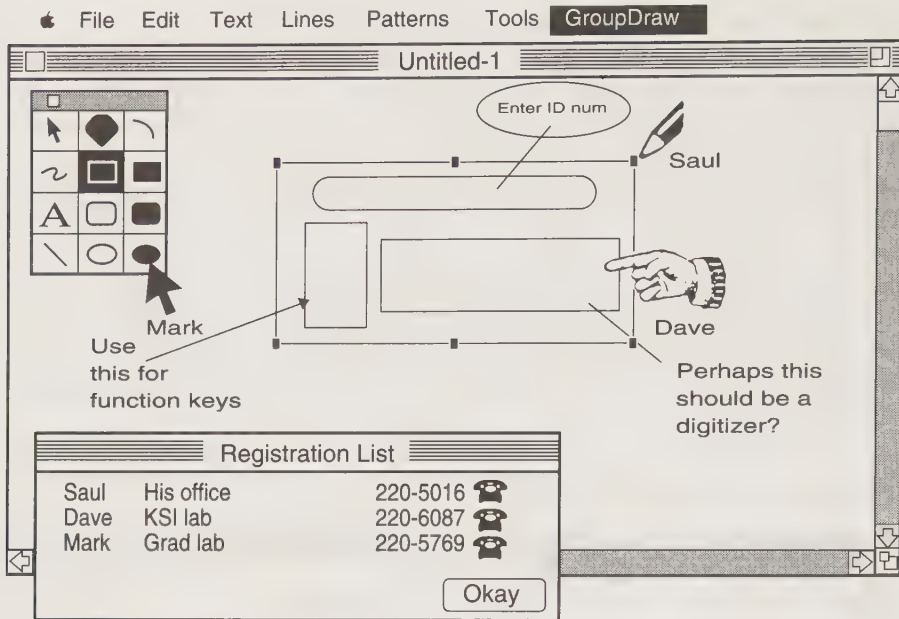


Figure 3.3 Sample of GroupDraw session, showing work surface and registration window.

- *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 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 Section 3.4). 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 chapter is to use our systems, concentrating on GroupSketch and GroupDraw, to highlight human factor 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.

3.2 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 and 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.

3.2.1 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 artefact (Tang, 1989). Bly (1988) disproves of this naive 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 artefact 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 provided 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 per cent) of work surface activity around the tabletop 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 two-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 per cent of all work surface activities. Expressing ideas and mediating interaction comprised the additional ~50 per cent and ~25 per cent respectively. Gesturing, which is often overlooked as a work surface activity, played a prominent role in all work surface actions (around 35 per cent 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.

3.2.2 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 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 3.1 (condensed from Tang 1989), and deserve the special attention of the reader. These criteria form the foundation for the rest of this paper.

Table 3.1 Six criteria for designing a communal work surface (condensed from Tang, 1989), and how they were met in GroupSketch

<i>Design criteria</i>	<i>Reasons</i>	<i>How criteria were met in GroupSketch</i>
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 3.1). 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 3.1). • 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 groupwork 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 artefacts to express ideas.	• The process of creation is in itself a gesture that communicates information. • Speech is closely synchronized with the creation process. • Artefacts 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 artefact creation and manipulation are transmitted in real-time and remain synchronized with accompanying speech.

continued

Table 3.1 *continued*

<i>Design criteria</i>	<i>Reasons</i>	<i>How criteria were met in GroupSketch</i>
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 coordination communication. ● We do not understand the coordinating 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)).

3.2.3 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 3.1. 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 3.1 provides details of GroupSketch user interface features and how they were designed around Tang's six criteria.

Figure 3.1 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 3.1). 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 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 3.1 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 3.2), 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 groupwork 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 on 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 cooperative (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 artefact. The specific observations we had made about GroupSketch are summarized in Table 3.2. We feel it fair to say the GroupSketch, in spite of its minimalist functionality, is reasonably effective as a distributed work surface.

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

Table 3.2 Observation of GroupSketch use

<i>Observation</i>	<i>Details</i>
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 simultaneous 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 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 artefacts 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 bitmapped 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 graphics) occurred frequently and naturally.	Resulting artefacts 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 (example include multiple people erasing different parts of a single line and cooperative construction of a drawing artefact).
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.

3.2.4 GroupDraw interface

While GroupSketch is a simple paint program where users can only make and erase marks on a bitmap 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 3.3 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 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 criterion (see Table 3.1) 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; users 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

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.

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 artefacts 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 WYSIWIS 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 viewport of one participant (called a follower) can be enslaved to the viewport of another participant (called a leader) (Pendegast 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.

3.2.5 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 3.2, 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 preferred 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 a 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 participants' faces are merged via transparent overlays into a single display (e.g. ClearBoard-1: Ishii and Arita, 1991; ROCOCO: 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 artefacts via paper scanner and video camera/frame grabber; the images can then be manipulated by the groupware (IIS Technologies). Another approach fuses two or more live-video images of computer screens and physical desktop surfaces (Ishii, 1990). Here, participants can combine computer artefacts (e.g. a single-user drawing program) with desktop artefacts (e.g. an existing drawing). Participants can gesture around and interact with these artefacts 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 effect 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 meetings 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 cursors³ 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 on the active work. A second issue concerning group size is screen size – large groups need far more working space in view than contemporary bitmapped displays will provide. Finally, larger groups 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.

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

3.3 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 trade-offs 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.

3.3.1 Replicated versus centralized architecture

There are three architectural alternatives for constructing distributed software (see Figure 3.4). 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 chapter.

In the centralized approach, a single program, residing on one machine, controls all input and output to the distributed work surfaces (see Figure 3.4(a)). 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 3.4(b)), and the 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 are reduced because communication does not go through a central intermediary, 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.

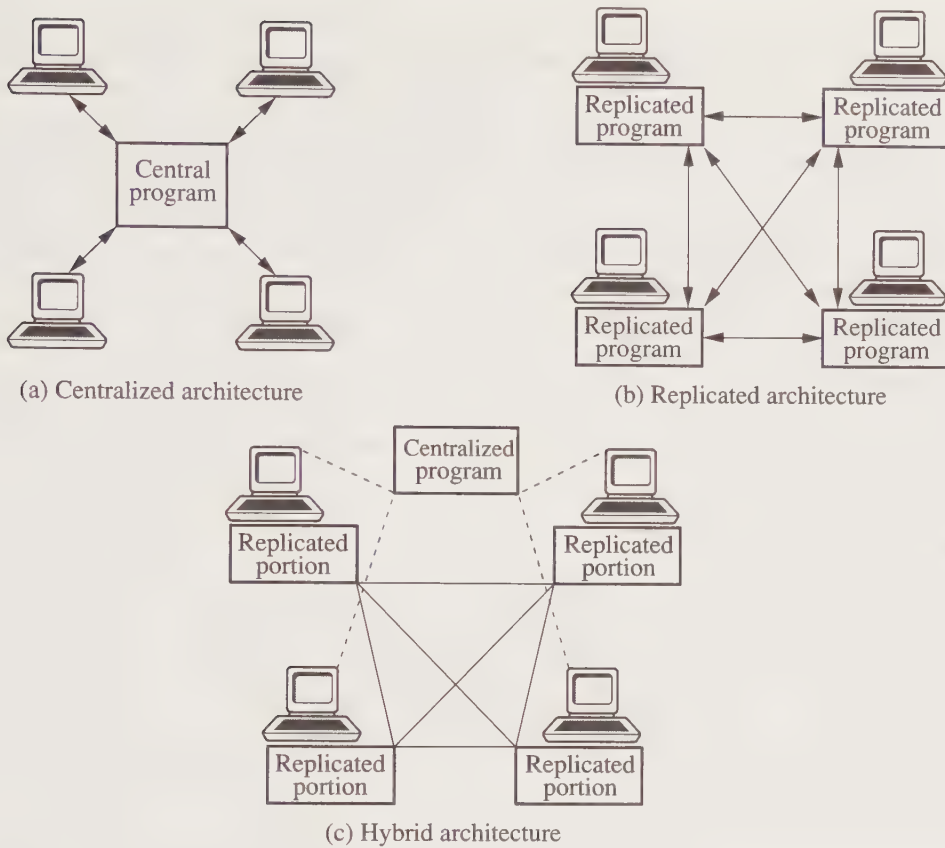


Figure 3.4 Schematic of a centralized, replicated and hybrid architecture.

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 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 3.3. 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 3.3). 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

Table 3.3 Communication protocol between GroupSketch processes

<i>Event</i>	<i>Information passed</i>
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

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 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 conferences⁵, and each process may be the owner of more than one object.

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 bitmap 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.

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.

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 endpoint 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, 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.

3.3.2 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 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 latecomer 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 3.3 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 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 reusable in other groupware applications requiring registration. In fact, for GroupSketch we were able to reuse, 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.

3.3.3 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 bitmap display directly (we used Sun Microsystems' Pixrect library). Large multiple cursors are implemented directly by exclusively ORing 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 3.3); 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 coordinates 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 the 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.

3.3.4 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 (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 10 Mbps, Appletalk at around 250 Kbps). 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. This gives a network utilization ranging from 600 to 4000 bps. This rate is easily handled by LANs, but could be demanding for a low-speed telephone link. About 75 per cent 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 networking 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.*, 1994). For pragmatic reasons, transmitting all cursor events may not be feasible, and other cursor transmission and display strategies may be required. Hayne *et al.* (1994) 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 current replicated architecture, the worst case network demands is a

function of $n \times (n - 1)$, 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 3.4(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.

3.3.5 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 from 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 3.4 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 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: *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 3.4). 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 3.4 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

Table 3.4 GroupDraw's *GroupGraphicalObject* and its *LineObject* subclass

<i>GroupGraphicalObject</i> , the root object of all graphics primitives	
Instance variables	
<i>int couplingStatus</i>	Indicates if object is private, public, or can be shared.
<i>int ownerProcess</i>	Indicates who is the owner of the process.
<i>int id</i>	A unique reference that identifies the object across the whole system.
<i>boolean acquired</i>	Indicates if the object is currently being manipulated by a participant.
<i>point whereGrabbed</i>	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	
<i>initializeGroupObject()</i>	Initializes the object, filling in any defaults.
<i>destroyObject()</i>	Destroys the object and frees its space.
<i>makeDescription()</i>	Make a description of the object suitable for transmission over the network.
<i>sendDescription()</i>	Send the description to a requester.
Methods for changing object attributes	
<i>requestChangeStatus()</i>	Request the object owner to change the coupling status.
<i>doChangeStatus()</i>	Actually change the status, and broadcast change over network.
<i>requestChangeOwner()</i>	Request the object to change its owner.
<i>doChangeOwner()</i>	Actually change the owner, and broadcast change over network.
Methods for graphically manipulating and drawing the object	
<i>draw()</i>	A place holder for a routine that will draw the object on the screen.
<i>requestToGrab()</i>	Request the object for permission to acquire it.
<i>doGrab()</i>	Permission is granted or denied.
<i>endGrab()</i>	An acquired object is relinquished.
<i>saveOriginal()</i>	The original position of a selected object is saved.
<i>restoreOriginal()</i>	Restore a moved object to its original position.
<i>drag()</i>	High-level drag-handler; checks permission, broadcasts changes etc.
<i>doDrag()</i>	A place-holder; this routine will actually drag the object.
<i>whereGrabbed()</i>	A place-holder; checks to see where the object was grabbed.
<i>LineObject</i> , a subclass of <i>GroupGraphicalObject</i>	
Instance variables	
<i>point startPoint</i>	The start point of the line.
<i>point endPoint</i>	The endpoint of the line.
Methods	
<i>initializeGroupObject()</i>	Initialize line-specific attributes, then call the super-class <i>initializeGroupObject</i>
<i>makeDescription()</i>	Specialized to include line descriptions, then call the inherited method.
<i>saveOriginal()</i>	Specialization line-aware form of the inherited model.
<i>restoreOriginal()</i>	Specialization line-aware form of the inherited model.
<i>doDrag()</i>	Actually drags the object.
<i>draw()</i>	Actually draw the object.

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 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.

3.4 CONCLUSIONS

This chapter introduced both the 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 other's 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 Section 3.3.1, 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 systems 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 interaction 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 3.1. 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 built independently of, but in parallel with, GroupSketch (Bly and Minneman, 1990; Minneman and Bly, 1990; 1991). Although the interfaces 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 artefacts are superior to the ones generated on our mouse-based system. Other workstation-based

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

sketchpads are now coming available, for example, CaveDraw (Lu and Mantei, 1991); WScrawl 2.0, SHDR from Rank EuroPark, ROCOCO Sketchpad (Scrivener *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 beyond group drawing. For example, we know of one group brainstorming system (Cognoter) that has been redesigned to fit these criteria (Tartar *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 bps 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 AVAILABILITY

GroupSketch is now available via anonymous ftp from the University of Calgary, Department of Computer Science via `fsc.cpsc.ucalgary` in `pub` projects `grouplab`.software. 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 *Proceedings of the Conference on Office Information Systems*, 25–27 April, Boston, pp. 238–48.
- Bly, S. (1988) A use of drawing surfaces in different collaborative settings, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88)*, ACM Press, pp. 250–6.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems*, 25–27 April, Boston, pp. 184–92.

- Dewan, P. and Choudhary, R. (1991) Flexible user interface coupling in collaborative systems, in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 41–8.
- Dykstra, E. A. and Carasik, R. P. (1991) Structure and support in cooperative environments: The Amsterdam Conversation Environment, *International Journal of Man Machine Studies*, **34** (3) pp. 419–34; republished in Greenberg (1991b).
- 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)*, ACM Press, pp. 13–24.
- Gaines, B. R. (1991) Modelling and forecasting the information sciences, *Information Science*, **57–58**, pp. 3–22.
- Garner, S. W., Scrivener, 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 cooperative work (CSCW), in *Proceedings of DATER '91*, Loughborough, UK, September, pp. 84–96.
- Greenberg, S. (1990) Sharing views and interactions with single-user applications, in *Proceedings of the ACM/IEEE Conference on Office Information Systems*, 25–27 April, Cambridge, pp. 227–37.
- Greenberg, S. (1991a) Personalizable groupware: accommodating individual roles and group differences, in *Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW '91)*, Kluwer Academic Press, pp. 17–32.
- Greenberg, S. (1991b) *Computer Supported Cooperative Work and Groupware*, Academic Press, New York.
- Greenberg, S. and Bohnet, R. (1991) GroupSketch: a multi-user sketchpad for geographically-distributed small groups, in *Proceedings of the Graphics Interface '91*, 5–7 June, Calgary, Alberta, pp. 207–15.
- Greenberg, S. and Chang, E. (1989) Computer support for real time collaborative work, in *Proceedings of the Conference on Numerical Mathematics and Computing*, 28–30 September, Winnipeg, Manitoba, 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 *Proceedings of the Hawaii International Conference on System Sciences*, **4**, IEEE Press.
- Grudin, J. (1989) Why groupware applications fail: problems in design and evaluation, *Office: Technology and People*, **4** (3) pp. 245–64.
- Hayne, S. C., Pendergast, M. O. and Greenberg, S. (1994) Implementing gesturing with cursors in group support systems, *Journal of Management Information Systems*, **10** (3), pp. 43–61.
- IIS Technologies (date unknown) *Introduction to the Group TeleConferencing System*, IIS Technologies Inc.
- Ishii, H. (1990) TeamWorkStation: towards a seamless shared space in *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, ACM Press, pp. 13–26.
- Ishii, H. and Arita, K. (1991) ClearFace: translucent multiuser interface for TeamWorkStation, in *Proceedings of the European Conference on 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 *Proceedings of the ACM Conference on Human Factors in Computing Systems*, ACM Press, pp. 525–32.
- Johansen, R. and Bullen, C. (1984) Thinking ahead: what to expect from teleconferencing, *Havard Business Review*, pp. 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 *Proceedings of the Conference on Office Information Systems*, 25–27 April, Boston, pp. 249–60.
- 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*, ACM Press.
- Lee, J. J. (1990) XSketch: a multi-user sketching tool for X11, in *Proceedings of the Conference on Office Information Systems*, 25–27 April, Boston, pp. 169–73.
- Linton, M. A., Calder, P. R., Interrante, J. A., Tang, S. and Vlissides, J. M. (1991) *InterViews Reference Manual*, Stanford University, CA.
- Lu, I. and Mantei, M. (1991) Idea management in a shared drawing tool, in *Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW '91)*, Kluwer Academic Press, pp. 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*, 15–17 May, Guelph, Ontario, pp. 145–67.

- Minneman, S. L. and Bly, S. A. (1991) Managing a *trois*: a study of a multi-user drawing tool in distributed design work, in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 217–224.
- MIT (1983) Talking heads, in *Discussions*, 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, *Communications of the ACM*, **34** (7) pp. 40–61.
- Pendergast, M. and Hayne, S. (1991) Assisting groups during the information merging process, *Decision Sciences International Conference*, Miami, November, pp. 720–5.
- Roseman, M. and Greenberg, S. (1992a) GroupKit: a groupware toolkit, in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems: Posters and Short Talks*, p. 43.
- Roseman, M. and Greenberg, S. (1992b) GroupKit: a groupware toolkit for building real-time conferencing applications, in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '92)*, ACM Press.
- Sarin, S. and Greif, I. (1985) Computer-based real-time conferencing systems, *IEEE Computer*, **18**(10) pp. 33–45.
- Scheifler, R. W. and Gettys, J. (1986) The X-Window system, *ACM Transactions on Graphics*, **5**(2) pp. 79–109.
- Scrivener, 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 *Proceedings of the 1st European Conference on Computer Supported Cooperative Work (EC-CSCW '89)* 13–15 September, Gatwick, UK.
- Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tatar, D. (1987a) WYSIWIS revised: early experiences with multiuser interfaces, *ACM Transactions on Office Information Systems*, **5**(2) pp. 147–67.
- Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S. and Suchman, L. (1987b) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications ACM*, **30**(1) pp. 32–47.
- Tang, J. C. (1989) Listening, drawing, and gesturing in design: a study of the use of shared workspaces by design teams, *Ph.D. 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, *International Journal of Man Machine Studies*, **34** (2) pp. 143–60, republished in Greenberg (1991b).
- 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)*, ACM Press, pp. 244–9.
- Tang, J. C. and Minneman, S. L. (1990) Videodraw: a video interface for collaborative drawing, in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 313–20.
- Tang, J. C. and Minneman, S. L. (1991) VideoWhiteboard: video shadows to support remote collaboration in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 315–22.
- Tatar, D. G., Foster, G. and Bobrow, D. G. (1991) Design for conversation: lessons from Cognoter, *International Journal of Man Machine Studies*, **34** (2) pp. 185–210; Republished in Greenberg, (1991b).

GESTURING THROUGH CURSORS: IMPLEMENTING MULTIPLE POINTERS IN GROUP SUPPORT SYSTEMS

Stephen Hayne, Mark Pendergast and Saul Greenberg

ABSTRACT

The focus of this chapter is on the issues underlying the concept of gesturing. Gesturing from one human to another appears to span all cultural boundaries; one could possibly call it a universal means of communication. Studies of groupwork have shown that gesturing makes up over 35 per cent of all interactions. Participants gesture to enact ideas, to signal turn-taking, to focus the attention of the group and to reference objects on the work surface. Specifically, this chapter explores gesturing as applied to users of group support systems. The definition of gesturing is extended beyond simple telepointers to include (1) the use of motion as a means of expression, and (2) movement that expresses or emphasizes an idea, a sentiment or attitude. We address such issues as at what level of interaction should gesturing be supported, how large and what shape should gesture pointers be, how should they move, what are network and processor throughput requirements, and what effect group size may have. Our results show that full motion computer gesturing can be supported on PC-LAN systems for small groups. Gesturing for medium and large groups requires the use of special techniques, such as regulating transmission rates, motion smoothing, and point and quiver cursors. These techniques can also be applied to wide area network implementations to reduce network traffic and latency problems.

4.1 INTRODUCTION

Group support systems is an emerging area of research that spans computer-supported cooperative work (CSCW) and group decision support systems (GDSS). CSCW is defined as 'the study and theory of how people work together, and how the computer and related technologies affect group behaviour'. CSCW implementors build 'computer-based systems that support two or more users engaged in a common task (or goal) and that provide an

interface to a shared environment' (Ellis *et al.*, 1991). These systems usually facilitate communication between members of a small group and provide task-specific tools. The software rarely regulates the actual meeting process; the designers expect that normal social protocols between participants will suffice.

On the other hand, a GDSS is 'an interactive computer-based system that facilitates the solution of unstructured problems by a set of decision-makers working together as a group' (DeSanctis and Gallupe, 1987). The design goal is to increase the efficiency and effectiveness of meetings. The usual method is to 'manage' the interactions between the group members by enforcing rules of meeting protocol and structure (Nunamaker *et al.*, 1991). Existing GDSS tools facilitate both small (3–6 members) and large (7–30) groups through the various stages of the decision-making process (Dennis *et al.*, 1990; Valacich *et al.*, 1990).

We believe that meetings in most corporate situations lie somewhere in between CSCW and GDSS; not all meetings result in decisions nor do all group processes centre on a task. Sometimes structured processes are crucial at meetings; at others they are debilitating. Rather than split semantic hairs about where our research resides, we will use the label group support systems (GSS). This research entails understanding how groups interact; applying that understanding to designing systems that help people work together; and observing people using our systems. The last point often constrains our implementations to standard delivery platforms (such as IBM PCs), and much of our effort is devoted to overcoming the platform limitations. The focus of this paper is on *gesturing*: why it is required, how it can be implemented on a computer and what pragmatic decisions must be made during implementation.

4.2 GESTURING

gesture: 1) the use of motions of the body as a means of expression, 2) a movement of the body that expresses or emphasizes an idea, a sentiment or attitude.

Random House College Dictionary

Some sort of gesturing takes place throughout any interaction between human beings. Gesturing appears to span all cultural boundaries; one could possibly call it a universal means of communication. Gesturing not only occurs between people, but also between people and their artefacts. In an ethnographic study of eight short-term, small-group design sessions, Tang built a descriptive framework that categorized activities over a shared work surface (large sheets of paper tacked on a table or whiteboard) (Tang, 1989, 1991). Three primitive actions were observed, and were combined to mediate several essential functions.

Actions:

- *Listing* produces non-spatially located text or alphanumeric notes.
- *Drawing* produces graphical objects, typically a two-dimensional sketch with spatially located textual annotations.
- *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.
- *Mediating interaction* facilitates the collaboration of the group and includes turn-taking and focusing attention.

Hand gestures played a prominent role in all work surface activities (about 35 per cent of all actions). Participants gestured to enact ideas, to signal turn-taking, to focus the attention of the group and to reference objects on the work surface (Tang, 1989, 1991).

Clearly, gesturing is of paramount importance in group interaction. Tang demonstrated this for small design groups, and it is our belief that gesturing is essential for almost all group activities involving a shared work surface. We are exploring hand gesturing over displays in group support systems. Obviously, as soon as group work is moved from a 'manual' face-to-face environment to a dispersed 'computer-supported' environment, many questions arise about gesturing. How can hand gestures be tracked as input? How do we display a gesture on a screen? If there is to be a gesture icon, how large must it be? How should it move? What happens as group size increases (scalability)? What network bandwidth is needed? We will address these questions in Section 4.4, after first reviewing existing implementations of gesturing in Section 4.3. Our experiences implementing different gesturing strategies via multiple pointers on a LAN-supported IBM PC platform are described in Section 4.5, which is heavily illustrated by performance benchmarks.

4.3 PREVIOUS RESEARCH

We are concerned with enhancing a group's real-time interaction when using groupware tools in a computerized shared visual work surface, where one's actions are immediately visible to all (Ellis *et al.*, 1991). The displays are software equivalents to whiteboards and flipcharts; the tools could be text editors, hypertext, structured drawing tools and so on.

Many collaborative systems do not support gesturing. For example, we now see *teleconferencing* set-ups that combine voice conferencing with facsimile transmission, electronic whiteboards, and even slow-scan video. While all participants can see the same resulting information, gestures referring to an artefact are only visible through indirection or not at all. For example, if one points to a drawing on a document at one site, a person at the other site must find their equivalent document and determine where the reference is. Similarly, one rarely sees the fine-grained process of others creating and manipulating artefacts. While people can get their job done without gesturing, we argue that much of the conversation is used as a substitute to gesturing, e.g. 'I'm pointing to the second line in the third paragraph', and that breakdowns are frequent, e.g. 'which one do you mean?'.

One of the first systems to support gesturing explicitly was *BoardNoter*, a computerized whiteboard used to support face-to-face meetings (Stefik *et al.*, 1987a). Gesturing was through a single large 'telepointer'. One person at a time could grab and control this special cursor, which was then seen on every display. All other individual cursors remained invisible to the group. In addition to tracking the cursor, clicking the mouse button would cause a static image of the pointer to be deposited (and remain) on the board (Stefik *et al.*, 1987b). Several commercial *view-sharing systems*, which allow people to share standard

single-user applications through serial interaction (Greenberg, 1990), can also support single cursors. Unlike *BoardNoter*, there is no difference between the local cursor and telepointer, e.g. Farallon's *Timbuktu* (Farallon, 1988). The common cursor responds to each user's mouse movement. When several people move their mice, the result is 'cursor wars'.

The first serious works in gesturing, all motivated by Tang's work (described above), were implemented using two different technologies: video fusion and computational space. *VideoDraw* was the original video-based system (Tang and Minneman, 1990). Each person has a monitor that displays the image of a camera pointing to the other person's monitor. Participants draw directly onto the monitor, and the camera captures both the other person's hand and the drawing underneath. Feedback is eliminated through polarizing filters. The result is that the shared workspace contains (from one person's point of view) his or her physical hand, pen marks on top of the screen, and the other person's hand and marks in the image. *TeamWorkStation*, on the other hand, uses hardware to fuse video signals (Ishii and Miyake, 1991). The advantage here is that people can perform their activity on any work surface (such as a desktop), with a video camera recording and fusing its image with the work surface image of the other participant. Finally, *VideoWhiteboard* allows each user to see the drawings and a shadow of the gestures of collaborators at the remote site (Tang and Minneman, 1991). Here, people see their partners and movements as a silhouette appearing on the other side of a translucent whiteboard. These video systems are limited. First, participants cannot manipulate each others marks, since they only see them as a video image. Second, these systems are not scalable, since serious image deterioration results when too many images are fused.

At the same time, several computational systems were developed that implemented gestures through multiple cursors. *GroupSketch* is a computer-based group sketchpad where one or more people can simultaneously draw, type, and gesture around the display (Greenberg and Bohnet, 1991; Greenberg *et al.*, 1992a, 1992b). Its cursors were designed specifically for gesturing around the criteria listed below.

- Since gestures must be seen in order to convey information, all cursors within a work surface are always visible to all participants.
- Cursors must have enough prominence on a multi-cursor display to attract the attention of other participants. A large 64×64 bit cursor is used instead of the traditional 16×16 bit cursor.
- Cursors change their shape to reflect a natural action. Four gesture modes are supported (pointing, writing/drawing, erasing and directing attention) by distinct cursor shapes (a pointing hand, pen, eraser and large arrow respectively).
- Cursors are unique, each identifying the person it belongs to. While face-to-face gesturing has natural cues to help identify who is gesturing, cursors do not. *GroupSketch* labels each cursor with its owner's name and each new cursor is rotated 90° from the last (while the idea of rotation was first put in to reflect different seating orientations of participants around a drawing, its real value was that it allowed up to four cursors to touch the same pixel without overlapping each other).
- Cursor movements appear continuously and 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.

In parallel with *GroupSketch* came the very similar *Commune* (Bly and Minneman, 1990; Minneman and Bly, 1990, 1991). Instead of a mouse and a vertical display, *Commune* users were able to write directly on a flat screen with a pen. This provided them with improved control/display compatibility and fine motor coordination. Many second generation sketching packages now support gesturing, e.g. *XGroupSketch* (Greenberg *et al.*, 1992a, 1992b), a functionally richer window-based version of *GroupSketch*; *WScrawl* by Brian Wilson (Wilson, 1992), *GroupScratchPad* by Stephen Hayne (Hayne, 1992) and *ShDr* by Paul Dourish (Dourish, 1992). We are also seeing these ideas transferred to the domain of structured drawing: *GroupDraw* (Greenberg *et al.*, 1992a, 1992b), *MMM* (Brier and Freedman, 1991) and *MUGE* (Pendergast and Hayne, 1992).

Several computer tools for tasks other than drawing also support gesturing. In *ShrEdit*, a shared text editor, all insertion 'carets' of participants are visible, as well as the text that they type (Olsen *et al.*, 1990). In *rIBIS*, a hypertext system supporting argumentation, there are two modes of operation: loosely and tightly coupled. In the latter, a single group cursor available through turn-taking is visible to all, thus strengthening the feeling of participation (Rein and Ellis, 1991). Both the *Multi-User Graphical Editor* and the *Graphical Issue Analyzer* by Pendergast and Hayne (Pendergast and Hayne, 1992; Hayne and Pendergast, 1992) have implemented specific gesturing cursors (discussed in the next section).

Finally, several researchers are now investigating 'gaze-awareness', where a person can see where their partner is looking on the work surface (Ishii and Kobayashi, 1992). Although we do not pursue it further in this paper, a person's gaze is a gesture as well. By tracking another person's eye movement over the work surface, we see their focus of attention, their degree of interest, obtain cues for topic switching, and so on. The few computational attempts at supporting gaze-awareness rely on overlaying the work surface on top of a video image of a person's face, and arranging cameras and half-silvered mirrors in such a way that eye references to the work surface are consistent across sites. Again, this video configuration may not be scalable to larger groups.

In summary, existing systems support gesturing through a handful of strategies. These include a single static pointer that is placed on the screen but does not track mouse movement, a single dynamic pointer that follows the mouse, multiple pointers with fine-grained tracking, and video images of actual hands as well as exploring gaze awareness. Of course, there are also systems that do not support gestures at all, but we believe these provide an impoverished work surface for group members (Greenberg *et al.*, 1992a).

4.4 A DESIGN FRAMEWORK FOR GESTURING

The goal of our research is to focus on a computer-based mechanism for supporting telepresence in the form of gesturing. Telepresence attempts to give meeting participants the feeling that they are all in the same room (Egido, 1988) by sharing the explicit and implicit participant dynamics. Paramount among these actions are gestures and meta-level communications. As we are not targeting a multi-media environment, we have chosen to study the requirements for gesticulation as implemented by multiple cursors that can be displayed on every participant station in real time.

This section describes some of the design considerations we have encountered in our work on gesturing. It is presented as a loose framework revolving around four aspects: the

appearance of a gesture cursor; different ways cursors can express gesturing through motion; the effects of group size and distribution on cursors; and consequences of relaxed 'What You See Is What I See' (WYSIWIS) (Stefik *et al.*, 1987b). We assume that a voice channel is available to the group, and that multiple cursors are preferred over single cursors.

4.4.1 Cursor appearance

Single cursors are usually presented on a screen as a small (16×16 pixel) bitmap, usually a left-leaning arrow. A naive approach to multiple cursors is simply to copy the appearance of single cursors. However, several issues appear which demand a slightly more sophisticated approach.

Owner identification

Assuming non-anonymous interaction, remote cursors must be easily discernible from the local cursor, and participants should easily be able to distinguish who is doing what. In the first case, colour or slight shape changes can be useful. For example, the local cursor might be black (or solid) while the remote cursors are coloured (or outlined). In the second, the cursors can be uniquely labelled by a participant's name so that participants can quickly determine *who* is doing the gesture. In practice, this may be easy, since user actions are often tied to voice (Greenberg *et al.*, 1992a). Although labelling may be left off if some degree of anonymity is desired, participants will likely be able to identify a cursor's owner by other clues.

Cursor size

Cursors in most single-user graphical user interfaces are quite small. As screens get larger and the display busier, the cursor becomes difficult to find. Most people just jiggle their mouse (we can easily spot motion). Some window systems have tricks for focusing a user on the cursor location. Examples of the latter are 'Xeyes' (two graphical eyes track the cursor), and 'oneko' (where a kitten chases the cursor around the screen, eventually to sleep on top of it when idle). When multiple people are using a display, the cursor size becomes much more critical. Since we maintain that cursors are critical to gesturing, a larger size (say 64×64 pixels) would be more appropriate for small group interactions (Greenberg *et al.*, 1992a; Pendergast and Hayne, 1992). The trade-off is that cursors can quickly consume too much screen real estate as group size increases (described later). While we do not yet know what the optimum size is, we caution implementors against choosing the standard 16×16 pixel size just because they are accustomed to it.

Cursor shape

Since many different and natural actions are to be expressed, the cursor shape must change depending on the action – this will add to the quality of the gesture. Both *pointing* (e.g. finger) and *attracting attention* (e.g. flashing arrow) have been identified as crucial general actions (Greenberg and Bohnet, 1991; Greenberg *et al.*, 1992a; Pendergast and Hayne, 1992). If the application requires drawing or listing, the cursor should also reflect this by

perhaps using a pencil, a caret, or other shapes that indicate the action. Gratuitous cursor shapes, as implemented in the *Aspects* groupware system (Group Technologies, 1991) likely have no positive effect.

Cursor lifetime

While it can be useful to have cursors displayed at all times (especially to indicate who is present), inactive cursors can become passively intrusive in larger groups. To address this, remote cursor display can either be put under individual control or can have a decay function over activity and time. Two interesting approaches to decay are to have a gravity function which removes the inactive cursor to a corner of the screen called a *parking* area, or to have the cursor merely dissolve away (Hayne, 1992).

4.4.2 Gesturing through cursor motion

The presence of a cursor on the screen indicates who is active in the session. It is the motion of the cursor that indicates a pure gesture or a gesture-based activity. We believe that the best gesturing systems will support real-time motion on all screens. Unfortunately, some platforms may not allow full motion because of limited network bandwidth, latency and slow processor speed. The following strategies for supporting low to high-bandwidth cursor motions are presented in order of lowest to highest data and processing requirements. The implementation model used is that each workstation sends a message on its cursor's status to the other participants' workstations.

Point

When the user points and presses a mouse button, a single message is sent to all the remote users. The gesturing cursor leaps to the new location from its old one and remains there until its owner sends a new location message. The problem is that a user may not see the sudden transition, thus not noticing the gesture.

Point and quiver

In order to attract the viewers' attention, the above strategy is modified so that the receiving station jitters the cursor back and forth in its new position for a period of time.

Limited motion

Only a certain number of gesture messages per second are sent between stations. As the number is increased, the cursor motion moves from being stilted and jerky to a smoother animation. For example, ShDr (Dourish, 1992) allows the user to control the number of transmitted cursor movements through a slider in the window's control panel. The fewer messages sent, the less accurately the motion reflects the gesture. Also, the jerky cursor is visually unattractive.

Limited motion with smoothing

After receiving a new cursor position from the sender, the receiving station computes several points in between the previous and the current location, and then animates the

cursor through these intermediate points. While this eliminates jerkiness, it still does not guarantee that the path reflects the actual movement of the sender. Note that smoothing can also be applied to Point and to Point and Quiver. This technique is applicable to slow networks, but does not overcome slow processors.

Full motion

Every time the cursor is locally displayed in a new location, a gesture message is sent. Every movement of the cursor, no matter how small, is immediately broadcast. If the network and processor is fast enough, every receiver will display the cursor exactly as the sender does. For many years the motion picture industry used 16 frames per second as the standard for smooth animation. We feel that between 10–20 messages per second, will support full motion. While sending more messages may be possible, it is probably not necessary.

Which is the best choice? We believe that full motion gesturing is the preferred for most situations. However, there may be special occasions when a user would want to leave a pointer parked at a specific location, such as when long network latency (> 0.5 seconds) is combined with real-time voice. Full motion may be unacceptable when gestures do not match the verbal communication.

4.4.3 Group size and distribution

Clearly, cursors convey meta-level communication information, i.e. how many people are in the meeting and how their activity is balanced. As the group size increases, the number of cursors appearing on the screen could begin to intrude on the interaction. One approach to managing this problem is to reduce the size of the remote cursors using a step function. As examples:

1. For one to four people use the 'regular' size of 64×64 pixels.
2. For five to six people reduce the cursor size to 32×32 .
3. For seven or eight people reduce the size to 16×16 , and so on.
4. For larger groups, each user may be displayed as a single pixel on the screen – this will still provide participants with 'gestalt' of the activity level.

This step function was implemented in the latest versions of *GroupSketch* (as illustrated in Greenberg *et al.*, 1992b), MUGE and GroupSketchPad. Cursor size is altered by the software as people enter and leave the session. This notwithstanding, we feel that when groups become very large, the lifetime of cursors will need to be adjusted.

Decreasing the size of the cursor severely limits the amount of information it can display. Very small cursors do not have room for a label, so identification becomes problematic. Also, as cursor numbers increase, it will become difficult to tell who is doing something important, and whose voice is tied to which cursor. We believe that the cursor size should be altered to reflect important actions. For example, the cursor size in a medium-sized group of eight people can be quite small (4×4 pixels) (Greenberg *et al.*, 1992a), but users can press a mouse button and get a large labelled pointing arrow, especially designed to attract the group's attention. Similarly, whenever someone starts to draw or type, their cursor grows to 64×64 pixels. Gesturing can be used to get access to

the 'floor' or voice channel when individuals are dispersed geographically, particularly useful for sessions using conference calls. Perhaps a 'hand-raising' cursor is required?

4.4.4 Relaxed WYSIWIS

Much of our discussion assumes strict WYSIWIS, where everyone sees exactly the same view. Since we feel that most group software will eventually relax WYSIWIS along congruence of view, transformation of cursor coordinates between the sending and receiving views must be performed. In addition, screens have differing resolutions, and if a windowing environment is used, windows can be placed on different parts of the screen. This transformation is easily done by mapping the cursor location to a world coordinate system at the sender and (1) if that location is outside the receiver's view discard the message; and (2) map the world coordinate to the receiver's viewport coordinates. It is extremely important that when one user points at a place in their window or screen locally, the same logical place is 'gestured' at on the remote stations.

Of course, this leads to a variety of human factors issues. Consider Figure 4.1, which shows four users actively working on a shared canvas. From user A's perspective, B's view completely overlaps, C partially overlaps and D is disjoint. First, should A have peripheral awareness of the others in the virtual canvas, particularly those who are completely outside A's viewport (D)? In Shared ARK and MUGE for example, the display includes a miniature of the entire canvas that shows the boundaries of each person's area of view (Smith *et al.*, 1989; Pendergast and Hayne, 1992). This conveys some of the meta-level information described earlier. Second, if viewports overlap (A and B), what happens to the gesturing cursors when it moves across a view boundary? For example, if B moves the cursor to the lower right of the screen, would it suddenly disappear from A's view (as in XGroupSketch), or would it 'linger' in the last visible spot (as in GIA)? What consequence

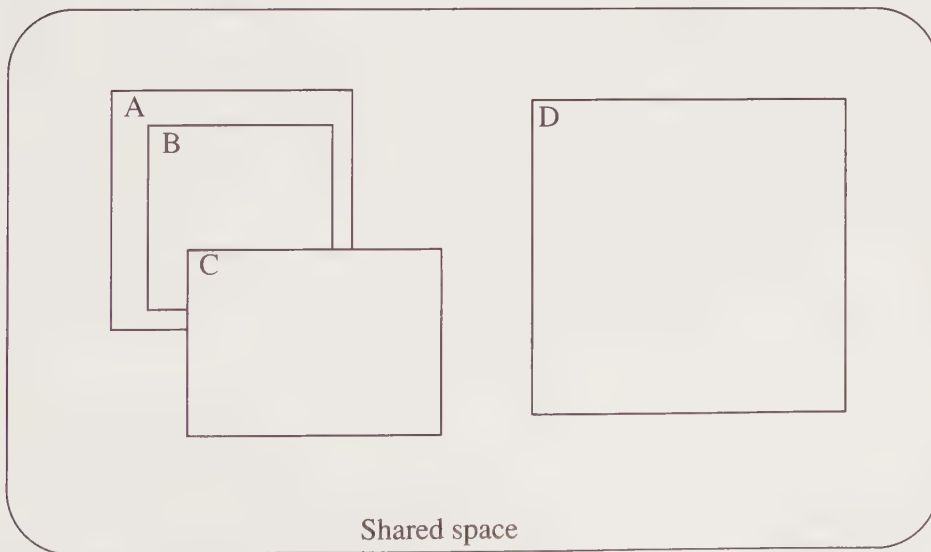


Figure 4.1 Translation between viewports A, B, C, D.

does this have on gesture interaction? Third, how should views be linked together? Assuming that users will move from a loosely coupled mode where they are working more or less independently to a tightly coupled mode throughout a session (*and back*), there should be ways to link views and gesture spaces together. For example, MUGE and GIA include the notion of leaders and followers, where users have the option of making their viewports and scrolling actions slaved with one another (Pendergast and Hayne, 1992). A user can be both a leader and a follower; leaders take followers along with them as they move about the work surface (zooming or scrolling).

4.5 PERFORMANCE BENCHMARKS

Successful implementation of full motion gesturing in GSS applications is dependent on the performance characteristics of the network and host operating system. Many of the previous systems have been implemented on UNIX hosts, but we feel that the more realistic corporate environment of IBM PCs and a graphical user interface would represent a lowest common denominator. Thus, we conducted our tests in the University of Calgary Norcen Meeting room, which has 34 IBM PS/2 Model 55 (386SX/16) PCs networked using both a 16 Mbps Token Ring LAN and a 10 Mbps Ethernet LAN supported by a Novell Netware (V3.21) file server.

Microsoft Windows 3.1 applications were used and the communication was managed by the Network Interface Object (NIO) as described by Pendergast and Hayne (1992). The NIO manages application layer communications (OSI) by providing a 'channel' object that facilitates the creation of GSS applications by providing virtual multicast connections that parallel the meeting room metaphor of communications. Multicast connections permit communication between two or more stations using a single network name or address, thus not requiring multiple separate session connections with each one. The NIO implements multicast channels using two transport level protocols: (1) reliable session connections and (2) broadcast datagrams. Depending on the nature of the data, the applications can dynamically select either the reliable (session connection) or unreliable (datagram) path when sending the message. Using the reliable channel requires the NIO to transmit the message to each station individually. Broadcast datagrams are sent just once, independent of the number of stations participating. The unreliable path has limited message size (approximately 500 bytes) and is not acknowledged, while the reliable path has no size limits and are guaranteed sent until a time-out occurs. The NIO is incorporated into several GSS applications, including MUGE, GIA, Graphical Brainstorming (Hayne, 1992), and GroupScratchPad (Hayne, 1992). Since all Windows (or MS-DOS) based applications will need to use a network layer (perhaps like the NIO) we felt it necessary to determine whether typical LANs and PCs could support gesturing.

4.5.1 Transmission time

This test determined the time required to transmit messages of various sizes over the unreliable (broadcast datagram) and reliable (multicast) communication channels. The test consisted of a station sending 100 messages as fast as possible and recording the elapsed time. The number of receiving stations was varied from 1 to 12 for message lengths

of 100, 1000 and 2000 bytes, thus requiring 1, 2 and 4 network packets respectively. Table 4.1 presents the elapsed time, in seconds, to transmit 100 messages. Figure 4.2 illustrates the performance differences between the multicast and broadcast datagrams channels.

Table 4.1 Transmission time in seconds for 100 messages

Receivers	Multicast			Datagram
	0.1 kbyte	1 kbyte	2 kbyte	0.1 kbyte
1	2.257	4.399	8.739	1.982
2	4.729	8.628	17.032	1.928
3	6.871	13.407	25.380	2.477
4	8.463	17.142	33.564	2.422
5	10.608	21.152	42.408	2.532
6	12.638	25.436	50.472	2.587
7	14.890	29.665	58.611	2.615
8	17.032	34.004	66.904	2.642
9	19.037	38.041	75.363	2.395
10	21.041	42.078	83.821	2.148
11	23.101	46.362	92.307	2.065
12	25.161	50.646	100.792	1.982

The transmission times for the datagram channel remained constant while the multicast channel's rate increased linearly with the number of receiving stations. This is as expected, since the multicast channel transmits a message to each receiver separately. The amount of time required to transmit (multicast) a 100 byte message to 12 stations is 0.25s and we

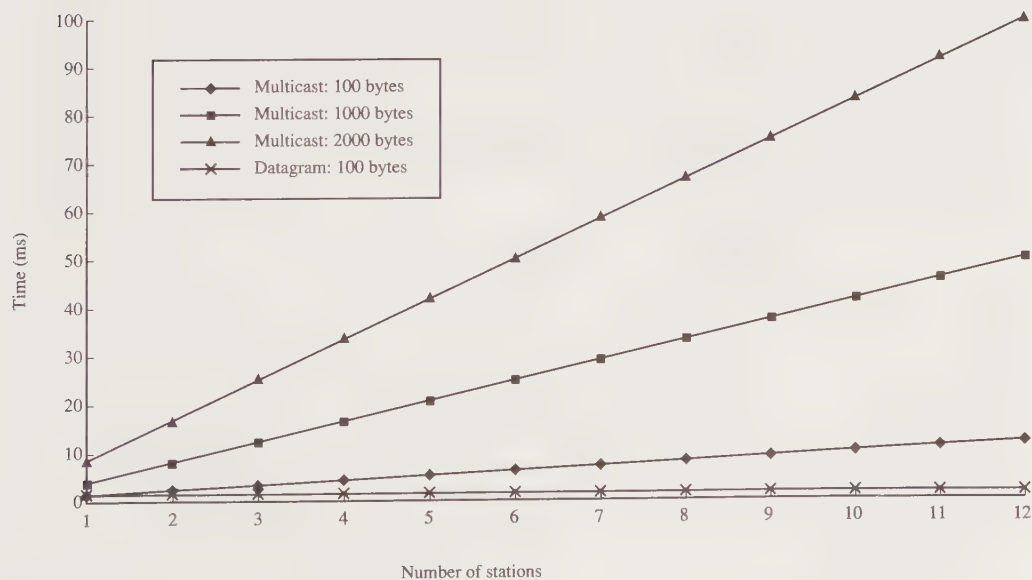


Figure 4.2 Performance differences between multicast and broadcast.

predict that for 24 stations it would be 0.50s. This makes a clear case for the use of unreliable datagram communications for gesture messages, even though some datagrams may be lost.

4.5.2 Datagram throughput

This test was conducted in order to determine the maximum rate that a station could transmit datagram messages of a size appropriate for gestures. A single station (IBM PS/2 55) transmitted 100 datagram messages of 100 bytes as fast as possible and recorded the elapsed time. Using the Novell Netware IPXSPX Windows interface, this time was measured at 2.314 s (average), or 23 ms per message (43 messages/second). To measure the overhead of the test program and high-level communications handler (NIO), the actual datagram transmission was then disabled and the test repeated. The resulting overhead was approximately 5 ms per message, leaving 18 ms for the actual transmission. This seemed to be slow considering a 16 MHz processor and a 16 Mbps Token Ring LAN. This same test was also performed using the 10 Mbps Ethernet protocol. The results from this test were better. It took 14 ms per message, with 5 ms of overhead. However, 9 ms (Ethernet) and 18 ms (Token Ring) still seemed excessive.

To discover whether the Windows operating environment had an effect on transmission rates, an MS-DOS test program was created that measured the time required to transmit 100 datagrams (100 bytes each). This program was able to transmit the 100 messages in less than 300 ms (Token Ring) and 100 ms (Ethernet), or about 3 and 1 ms per message respectively (1/6 the time of Windows). Due to the extreme difference between Windows and native MS-DOS, we wanted to pinpoint the problem in Windows by eliminating the Novell interface to the network and substituting the IBM NETBIOS interface. This third test gave results identical to the IPXSPX version. We believe we have verified the existence of a Microsoft Windows-induced delay in the transmission process. One source of the delay comes from having to transfer data packets and command blocks from extended memory to low DOS memory, where the adapter can access it. We also suspect that the Windows kernel only processes interrupts after a certain arbitrary number of clock ticks. Reducing the NIO's overhead by 75 per cent would allow transmission of over 50 datagrams second—far more than we feel is required for gesturing. However, discovering the reason for the delay inside of Windows' kernel and reducing it by just 20 per cent would yield the same results.

4.5.3 Multiple sender interaction

A third set of tests determined the network interaction effects that would occur if several stations were gesturing at the same time. In this test the number of sending stations was varied from 1 to 4 and the message transmission rate was varied from 5 to 25 messages per second. Each message was 100 bytes long and sent using broadcast datagrams. The results of the test revealed two important thresholds; one when the network protocol began favouring one station over another, and a second when datagrams began to be lost. These thresholds are shown in Figures 4.3 and 4.4.

Datagram packets are lost whenever there is a transmission error or if a station does not have an outstanding receive request. Since real-time gesture messages will probably be transmitted at a rate greater than 10 per second, we believe the loss of a few is not a major



Figure 4.3 Network favouritism threshold.

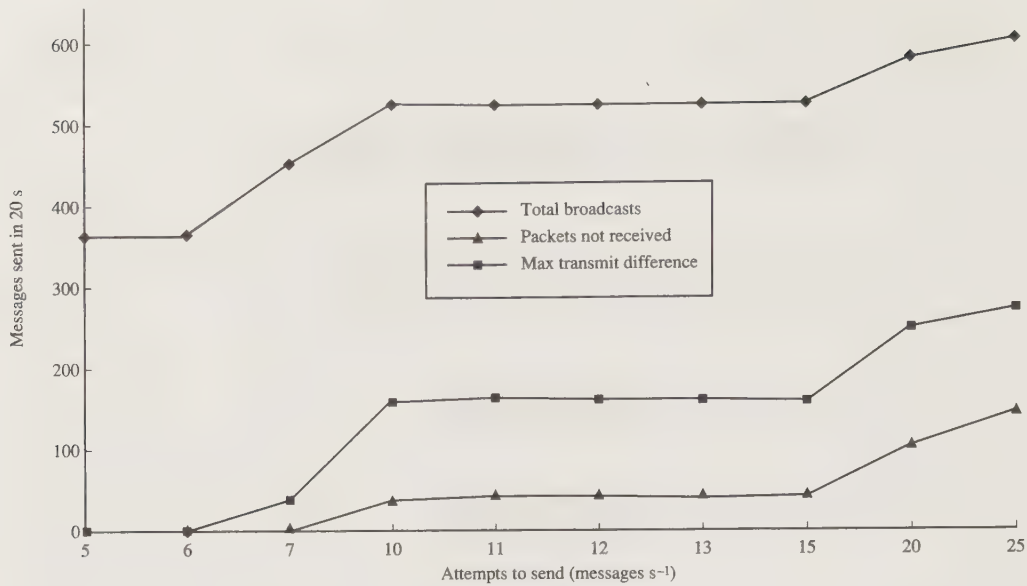


Figure 4.4 Datagrams last threshold.

concern. The favoured station threshold does present a problem. Table 4.2 presents the data from a 20 s run where four stations were transmitting (and receiving) at the same time. Station A was not able to transmit at the same rate as the other stations once the send rate was increased to seven messages per second. Station B began being penalized at 9 or 10 messages a second, while Stations C and D were able to transmit at a high rate for all speeds. At the speeds required for smooth gesturing (minimum of 10–15 per second), Station A was only able to transmit 1 message per second. This results in a very jerky gesture motion. Since a station's ability to transmit/receive is based on processor speed and location on the ring, certain stations will always be at a disadvantage. This indicates that the software controlling the gesturing should be intelligent enough to govern its

Table 4.2 Concurrent transmitters

Rate	Messages sent in 20 s			
	Stn A	Stn B	Stn C	Stn D
5	91	91	91	91
6	91	91	91	91
7	88	121	121	121
10	22	145	178	179
11	21	139	181	181
12	22	141	181	179
13	22	143	178	179
15	25	138	180	182
20	20	32	261	269
25	22	29	293	265

transmission rate whenever two or more stations are gesturing at the same time. This test was repeated on the Ethernet LAN, and yielded similar results.

4.5.4 Video display

A final performance test was conducted in order to measure the processor overhead required for displaying different types of gesture pointers. This test consisted of drawing and erasing a 24×36 pixel bitmap or a five-sided filled arrow polygon, of roughly the same area, 2000 times. We felt that the overhead of a *bitBlt* call might be larger than the overhead of a *drawPolygon* call. The test was run on four different CPUs (see Figure 4.5). For all CPUs, the drawing of the polygon was 4 to 5 times faster than the bitmap. The slowest CPU (Model 55) required 14 ms for the bitmap and 3.8 ms for the polygon, the fastest CPU (Gateway 486/33) required 4.1 ms for the bitmap and 0.89 ms for the polygon. As polygons are much faster to draw, gesturing programs should use this technique when running on slower processors. We acknowledge that these results are highly dependent on the video display hardware, but the test represents the current state of PC technology.

4.5.5 Summary

The above tests indicated that unreliable datagram communications are appropriate for gesturing and that a definite limit exists for transmitting and receiving gestures. This limit is determined by processor and network speed.

Table 4.3 shows equation results for maximum gesture rates of bitmap and polygon cursors. If a rate of 10 messages per second is required for smooth gesturing, then low-end 386SX/16 systems could handle three concurrent gesturing stations and high-end 486/25 systems could handle 10 stations (using bitmap gestures). These calculations assume that there are no background tasks active. If large groups are to be supported on PC-based systems, gesturing needs to be viewed as a separate operation from drawing and other schemes such as 'point and jitter' need to be explored. It is not *required* that all stations'

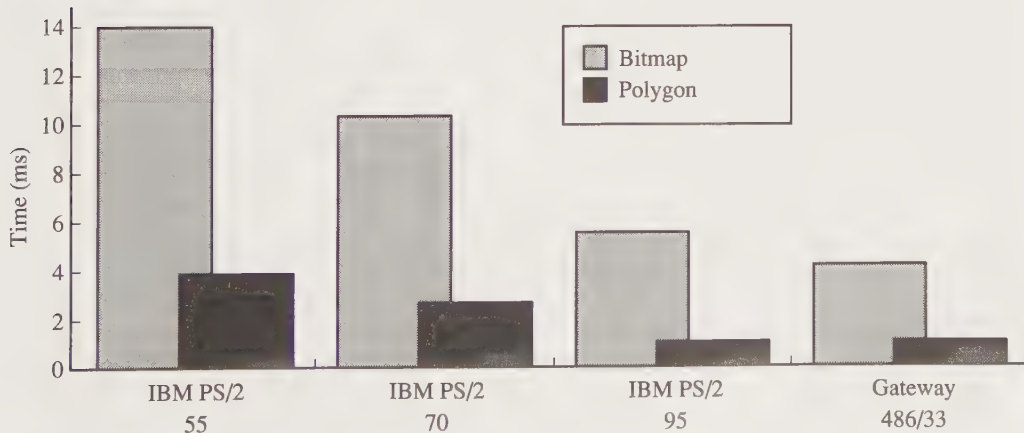


Figure 4.5 Video performance.

cursors be displayed at all times, but based on previous research and our own experience, the richness of the interaction is significantly reduced. Whether this becomes an impediment to successful group interaction is an empirical question.

Table 4.3 Maximum gesture rates

<i>386SX/16</i>			<i>486/25</i>		
<i>Polygon</i>	<i>n</i>	<i>Bitmap</i>	<i>Polygon</i>	<i>n</i>	<i>Bitmap</i>
1	38	38	1	100	100
2	18	21	2	54	71
3	11	15	3	37	55
4	8	11	4	28	45
5	6	9	5	22	38
6	5	8	6	19	33
7	4	7	7	16	29
8	4	6	8	14	26
9	3	5	9	12	23
10	3	4	10	11	21

4.6 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we have addressed some of the practical considerations for gesturing within group support systems applications. We have outlined a framework for implementing gesturing that spans low to high network bandwidth and processing power. The framework also addresses issues relevant to group size and interaction style.

We argue that the appearance of the gesture cursor should vary based on group size, user activity and screen size. For small groups the cursor should be large enough to find and have some sort of owner identification (name, initials etc.). For large groups (6+) and/or small screens (14" and smaller), cursors should be made smaller (16 × 16) and inactive cursors should decay or disappear. Ideally, the cursor shape will provide information about the current actions of the user (drawing, pointing or requesting the floor). Finally, cursor update modes can be varied to accommodate system limitations. Full motion gesturing should be used whenever possible, with limited motion or point and quiver techniques reserved for slower networks.

During performance tests on PC networks we discovered that there is a large penalty for operating in the Windows GUI environment when transmitting messages. We also determined that the Ethernet or CSMA/CD protocol to be somewhat superior in small groups that the Token-Ring protocol using full motion gesturing; either provides more than enough bandwidth. As one might expect, the faster processors were able to transmit and receive more gesture messages per second. More importantly, slower CPUs will spend most of their time processing incoming gesture messages and will not be able to transmit messages of their own. Therefore, if equal participation is required, one must impose an adaptable governing mechanism that limits the transmission rates based on the number of stations participating or else limit the communication via the use of point and jitter and some smoothing mechanisms.

One might argue that all these problems can be solved by using high-performance workstations. However, for groupware to become commonly used in commercial environments, problems of this sort need to be solved. Further, when groupware applications are distributed geographically, relying on low-speed networks, processor problems will be replaced with transmission latency problems. Perhaps the solutions we suggest to address slow processors can be applied to slow networks as well.

REFERENCES

- Bier, E. A. and Freedman, S. (1991) MMM: a user interface architecture for sharing editors on a single screen, in *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, South Carolina, USA, 11–13 November, ACM Press, pp. 79–86.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems*, April 1990, pp. 184–92.
- Dennis, A. R., Valacich, J. S. and Nunamaker, J. F. (1990) An experimental investigation of the effects of group size in an electronic meeting environment, *IEEE Transactions on Systems, Man and Cybernetics*, **20**(5) pp. 1049–57.
- DeSanctis, G. and Gallupe, B. R. (1987) A foundation for the study of group decision support systems, *Management Science*, **33**(5) pp. 589–609.
- Dourish, P. (1992) ShDr version 1.3a. Software available from parcftp.xerox.com via anonymous ftp.
- 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 Collaborative Work*, September 1988, pp. 13–24.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, *Communications of the ACM*, **34**(1) pp. 38–58.
- Farallon (1988) *Timbuktu user's guide*.
- Greenberg, S. (1990) Sharing views and interactions with single user applications, in *Proceedings of ACM/IEEE Conference on Office Information Systems*, April 1990, pp. 227–37.
- Greenberg, S. (1991) Computer supported cooperative work and groupware: an introduction to the special edition, *International Journal of Man Machine Studies*, **34**(2), pp. 133–43; republished in Greenberg, S. (ed.) (1991) *Computer Supported Cooperative Work and Groupware*, Academic Press, New York.
- Greenberg, S. and Bohnet, R. (1991) Groupsketch: a multi-user sketchpad for geographically distributed groups, in *Proceedings of Graphics Interface 1991*, June 1991, pp. 207–15.
- Greenberg, S., Bohnet, R., Roseman, M. and Webster, D. (1992b) Groupsketch: in *CSCW '92 Technical video Program, SIGGRAPH Video Review*, Issue 87, a supplement of ACM Computer Graphics.
- Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992a) Human and technical factors of distributed group drawing tools, *Interacting with Computers*, **4**(3) pp. 364–94.
- Group Technologies (1991) *Aspects: the first simultaneous conference software for the Macintosh, version 1*, User's manual.
- Hayne, S. (1992) Computer support for graphical brainstorming: a group support system, *Working paper #WPS-10-92*, University of Calgary.
- Hayne, S. and Pendergast, M. (1992) Techniques and experiences with object oriented windows-based group support software development, *Working Paper*, Department of MIS, University of Calgary.
- Ishii, H. and Kobayashi, M. (1992) Clearboard: a seamless medium for sharing drawing and conversation with eye contact, in *ACM SIGCHI Conference on Human Factors in Computing Systems*, 3–7 May, Monterey, California, ACM Press.
- Ishii, H. and Miyake, N. (1991) TeamWorkStation: towards an open shared workspace, *Communications of the ACM*, December, pp. 37–50.
- Minneman, S. L. and Bly, S. A. (1990) Experiences in the development of a multi-user drawing tool, in *The 3rd Geulph Symposium on Computer Mediated Communication*, 15–17 May, Guelph, Ontario, University of Geulph Continuing Education Division, pp. 154–67.
- Nunamaker, J. F., Dennis, A., Valacich, J., Vogel, D. and George, J. (1991) Electronic meeting systems to support group work, *Communications of the ACM*, **34**(7) pp. 40–61.

- Olsen, J. S., Olsen, M. H., Mack, L. A. and Wellner, P. (1990) Concurrent editing: the group's interface, in *Human-Computer Interaction: INTERACT '90*, Elsevier.
- Pendergast, M. and Hayne, S. C. (1992) Alleviating convergence problems in group support systems: the shared context approach, *Working Paper*, Department of MIS, University of Calgary.
- Rein, G. L. and Ellis, C. A. (1991) RIBIS: a real-time group hypertext system, *International Journal of Man Machine Studies*, **34**(3) pp.349–68, republished in Greenberg, S. (ed.) (1991) *Computer Supported Cooperative Work and Groupware*, Academic Press, New York.
- 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 *Proceedings of the 1st European Conference on Computer Supported Cooperative Work (EC-CSCW '89)*, 13–15 September, Gatwick, UK, Computer Sciences House, Slough, UK.
- Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tartar, D. (1987b) WYSIWIS revised: early experiences with multi user interfaces, *ACM Transactions on Office Information Systems*, **5**(2), pp. 147–86.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. (1987a) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, January, pp.32–48.
- Tang, J. C. (1989) Listing, drawing and gesturing in design: a study of the use of shared workspaces by design teams. *Ph.d. thesis*, Mechanical Engineering, Stanford University, California. 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, *International Journal on Man Machine Studies*, **34**(2) pp.143–60; republished in Greenberg, S. (ed.) (1991) *Computer Supported Cooperative Work and groupware*, Academic Press, New York.
- 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*, ACM Press, pp.313–20.
- Tang, J. C. and Minneman, S. L. (1991) VideoWhiteboard: video shadows to support remote collaboration, in *ACM SIGCHI Conference on Human Factors in Computing Systems*, 28 April–2 May, New Orleans, ACM Press, pp.315–22.
- Valacich, J. S., George, J., Nunamaker, J. F. and Vogel, D. (1990) Group size and proximity effects on computer mediated idea generation, *Working Paper*, University of Arizona.
- Wilson, B. (1992) Creating a highly portable shared whiteboard based on X-Windows, in *CSCW '92 Workshop on RealTime Group Drawing and Writing Tools*, Toronto, Canada.

USING A SHARED PEN-BASED TOOL FOR MEETING SUPPORT

Catherine G. Wolf, James R. Rhyne
and Laura K. Briggs

ABSTRACT

We-Met (Window Environment-Meeting Enhancement Tools) is a prototype pen-based tool designed to support both the communication and information retrieval needs of small group meetings. The first part of this chapter describes We-Met and the rationale for its design, the second discusses findings from an empirical study of the use of We-Met for group communication, and the third discusses findings from a study of the search and retrieval of information from non-computer-based meetings conducted to provide insight into how to facilitate these activities in We-Met. This chapter identifies potential communication process gains due to the pen-based interface style, opportunities for the facilitation of information retrieval in a pen-based tool, and the functionality interface challenges in the design of a tool to support small group meetings.

5.1 INTRODUCTION

In the late 1980s, research began to explore the use of conventional keyboard-based computers for the support of face-to-face (Mantei, 1988; Stefik *et al.*, 1987) and remote meetings (Sarin and Greif, 1988). These early systems served to elucidate important issues in the design of meeting support systems, and analysis of their short-comings have contributed to our understanding of key requirements (see Tator *et al.*, 1991, for example). These keyboard-based systems did not enable users to communicate using sketches, a very

The original version of this paper appeared in the *Proceedings of CSCW '92*. Copyright 1992, Association for Computing Machinery, Inc. Reprinted by permission.

common medium of communication in many meetings. Recognizing the importance of this mode of communication, several researchers have built shared drawing tools to support small group drawing activities by participants in different locations (Bly, 1988, 1990; Greenberg *et al.*, 1992; Minneman and Bly, 1991; Tang and Minneman, 1991). These systems were found to facilitate a style of interaction which was in many ways typical of face-to-face meetings.

The information generated in a meeting is typically used outside the meeting as well, and another line of research has sought to facilitate the use of collaboratively produced information. There is some evidence that the requirement to facilitate communication within a meeting may often conflict with the requirement to capture meeting information in a form which facilitates its review. Conklin (1989) described a structured format for capturing design rationale that facilitates the later use of information. However, the cognitive overhead in using the method made it inappropriate for use during meetings. Videotape has been used to capture meeting information with little disruption of the meeting process (see Muller, 1991, for example), but is very cumbersome to review.

We believe that an effective meeting support tool must facilitate both group communication and the capture and review of information. Over the past several years, we have been building We-Met (Window Environment-Meeting Enhancement Tools), a prototype pen-based system designed to meet both of these needs for small, informal work groups (Wolf *et al.*, 1991).

This chapter is organized into three main parts. The first describes We-Met and the rationale for its design, the second describes a study of the use of We-Met for group communication, and the third describes a study of the search and retrieval of information from non-computer-based meetings which was conducted to give us insight into how to facilitate these activities in We-Met.

5.2 WE-MET

The We-Met user interface is shown in Figure 5.1. The We-Met prototype is under development, and some of the features described here are not yet operational.

We-Met provides a shared drawing area in which several users can work at the same time. When one person writes, the other participants see each stroke as it is completed. Users have access to multiple shared drawing areas and can scroll within a drawing area. The displayed marks can be edited (see buttons in lower left of Figure 5.1). Additional features include writer identification, the importation of bitmap images and methods for joining and leaving a We-Met meeting.

Numamaker and his colleagues (1991) have distinguished features of a meeting system that support the communication process from features that structure it. Process support features enhance a meeting by facilitating communication among people (for example, a slide projector may serve this purpose). Process structure features effect the communication process by directing the pattern, timing, or content of communication (for example, following Roberts' Rules of Order may have this effect). Because our goal was to facilitate the meetings of small, highly interactive, unpredictable groups, we deliberately chose to provide process support rather than process structure. One consequence of this decision is that We-Met does not impose floor control on access to the drawing area or interface controls.

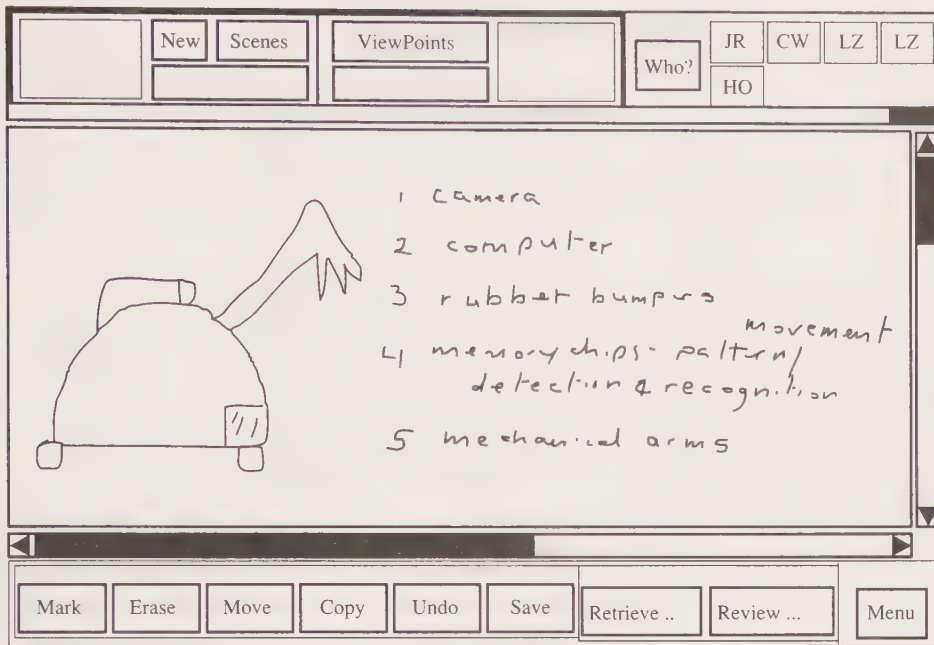


Figure 5.1 The We-Met interface: scene from the clutter collector robot task.

All meeting events are recorded and can be reviewed during or after the meeting. Our own observations as well as recent reports (Bly, 1988; MacLean *et al.*, 1989) have suggested that the process by which information is created and used can be important for the understanding of the end product of the workgroup. The We-Met record preserves temporal information by creating a time-stamped history of meeting events (see Rhyne and Wolf (1992) for a detailed description of this history mechanism). Users can travel backwards and forwards in the history and replay selected segments. Because the drawings and writings created in the shared work area have a temporal as well as spatial dimension, we refer to the shared work area as a 'scene'. In order to enable users to identify particular points (called 'viewpoints') in a scene for later review, users may give these points a text name or mark them with a hand-drawn squiggle (such as a star). People make such marks in notebooks and whiteboards, and we expect that the ease of use of this technique will transfer to our computer-based tool.

We-Met runs on workstations with an attached LCD/digitizing tablet over a local area network. It can be used for co-located meetings, in which the participants are in the same room, or distributed meetings, in which the participants are geographically separate from each other. We envision the future use of We-Met on notepad computers connected by a wireless local area network.

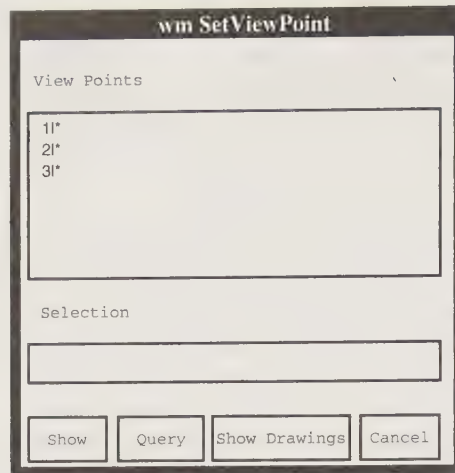


Figure 5.2 Scene Selection Dialog Box.

5.3 USE OF WE-MET FOR GROUP COMMUNICATION

As the We-Met prototype has evolved, we have conducted informal experiments with our colleagues, visitors and ourselves. We plan to put it in the hands of existing groups to use in their work over extended periods of time. In order to get more detailed user feedback at an early stage of development, a study was conducted using a prepared task. This study is described here.

The following features could be used by participants in the study:

- Shared scene for pen input
- New scene: to get a new empty scene, a participant touched the *New* button (see Figure 5.1). All participants then saw the new scene on their displays.
- View another scene: to view another, previously created scene, a participant touched the *Scenes* button. A selection box was presented and the user selected the desired scene from a numbered list (see Figure 5.2). The selection box was presented only to the person who had initiated the dialogue. After selection, all participants saw the selected scene.
- Scroll: to scroll spatially within the current scene, a participant used the scroll bars at the right and bottom of the application window. When an individual scrolled a scene, it did not scroll the scene presented to other participants.

All participants had equal access to the drawing area and controls, and could draw or operate a control at any time.

5.3.1 Study procedure

The operation of the system was described and demonstrated to the group before they started the primary task.

The participants in the study consisted of nine groups of three people each. The participants were employees of the IBM T.J. Watson Research Center and included programmers, computer scientists, psychologists, engineers, mathematicians and summer students. The members of each group worked together and had frequent meetings.

For four of the groups, the participants were *co-located* in the same room. The participants were seated in a U-shaped arrangement. In addition to their individual LCD/digitizing tablets, the output from one of the workstations was projected onto a wall for common viewing.

In the other five groups, the participants were distributed in three separate rooms. They were connected by a telephone conference call in addition to the We-Met application.

The group was told that they were a committee assigned to work on the design of a Clutter Collector Robot which would pick up household clutter and put it in its proper place (a task created by one of the authors). They were told that they were not expected to reach a conclusion on the design, but to generate a number of ideas. The task was designed to encourage the production of sketches, diagrams and writing. The topic was chosen on the assumption that everyone was familiar with the problems of household clutter and, thus, had a basis for contributing to the discussion.

The groups worked on the task for about 30 minutes. They then filled out a brief questionnaire and discussed their reactions to the system.

One of the authors observed the meeting and took notes. In addition, the time-stamped recording of meeting events was operational and used to create a log of the meeting for analysis. The participants and the We-Met screen were both videotaped.

5.3.2 Findings

The findings reported here are based on the participants' questionnaire responses, the post-task discussion, the observer's log, and preliminary analyses of the videotapes and computer-based log.

The videotapes were analysed to gain insights into the patterns of interaction among participants and the nature of the drawing/writing activity. The unit of activity was the turn, defined as a sequence of strokes and/or scene changes by one participant. The turns of participants sometimes alternated with one another (i.e. first one person wrote, then another) and at other times over-lapped each other (i.e. two or more people wrote at the same time).² The writing, drawings and other marks made on a turn were transcribed. The speech of participants was not transcribed.

The computer-based log was used in conjunction with the videotape to extract timing and user identification information for each turn. The computer-based log recorded all stroke events, new scene creations and switches among scenes. For each event, the type of event, time in seconds, user and scene identification number were recorded. For stroke

² There are often brief pauses between strokes when people write. Thus, when several people write at more or less the same time, some guidelines were needed to determine when the successive strokes of one user which were temporarily interleaved with the strokes of another user should be coded as a single turn. This decision was made based on the perceived intention of the writer (e.g. strokes comprising a word or phrase were grouped as one turn) and the temporal characteristics of the strokes (during episodes of concurrent writing by two or more participants, if the interval of time between successive strokes by a user exceeded 3s, the strokes were considered to be part of separate turns).

events, the number of points in the stroke and the coordinates of the bounding box were also recorded.

The findings are discussed in terms of the apparent process gains or losses the We-Met meetings exhibited compared with face-to-face non-computer-based meetings or meetings supported by a keyboard-based system. We find it useful to characterize the findings in terms of process gains/losses even though this study did not include the two comparison conditions, because it leads to the formulation of hypotheses that could be tested in future studies.

It should be noted that there was considerable variability among groups both in quantitative measures of behaviour and qualitative meeting style, a common finding in studies of group behaviour. There were no measures or characteristics which reliably distinguished the co-located from the distributed groups. Consequently, in the following discussion, findings from these two conditions are combined.

Modes of communication

All participants found the pen-based interface easy to use. They found it easy to speak or listen and write or draw at the same time. For many people, the task of typing takes up too much cognitive capacity to allow them to simultaneously type and participate fully in the spoken discussion as listeners or speakers. Thus, the pen-based interface may produce a process gain compared to a keyboard interface because it permits parallel activity within individuals.

The pen-based interface facilitated the use of multiple channels of communication. As in a non-computer-based meeting, participants spoke, drew, wrote, made hand gestures in the air, and made frequent eye contact (see Tang (1991) for a description of activities around a drawing pad in non-computer-based meetings).

The strokes that a participant made on each turn were classified as writing, drawing or gesture marks. This last category consisted of marks which were not intended as part of the writing or drawing but served to draw attention to a part of it, or indicate a relationship between parts. Many turns consisted of more than one category of marks. The percentage of marks in each category is shown in Table 5.1.

Table 5.1 Percentage of turns containing marks of each type

<i>Mark</i>	<i>Percentage</i>
Drawing	32
Writing	41
Gesture	13
Drawing, Writing	7
Drawing, Gesture	1
Writing, Gesture	6
Drawing, Writing, Gesture	1

While the forms of communication used by people in a meeting are highly dependent on the task as well as the meeting support capabilities available, the high percentage of turns

which contained drawing suggest that participants readily made use of the drawing capabilities afforded by the pen.

Twenty-one per cent of the turns included gesture marks. People spontaneously employed a rich vocabulary of gesture marks to meet their communication needs. Circular marks and arrows were used to point to objects, asterisks were used to mark important items, checks were used to identify completed topics, and double-headed arrows were used to show a relationship between two objects. In one group, one participant guided another to a part of a scene not currently visible by the second participant by drawing a line from a visible part of the scene to the desired part.

Such hand-drawn gestures can be made more easily with a pen than a mouse. In mouse-based shared drawing systems, multiple telepointers (one representing each participant) have been used for some of these purposes (Greenberg *et al.*, 1992; Wolf and Rhyne, 1993).

Because the pen could be used with equal facility for writing, drawing and gesturing, there was no overhead in shifting among these activities. Transitions among these modes of communication were seamless, and 15 per cent of the turns contained more than one pen-based mode of communication. For some tasks, the multiple capabilities of the pen may be better suited to the communication needs of the group than a keyboard/mouse interface, and thus provide a process gain.

Interaction styles

Like Ellis *et al.* (1991), we found that group interaction was characterized by a number of styles.

Much of the writing and drawing was characterized by temporally overlapping activity among participants. An average of 26 per cent of all turns fell into this category. Studies which have examined overlapping speech (see Levinson (1983) cited in Whitaker *et al.* (1991) and Seller (1992)).

The overlapping drawing/writing activity consisted of two different styles of interaction.

Several groups worked in a highly *collaborative* fashion throughout most of the session. For example, one person might add to a list of requirements for the Clutter Collector Robot and another might expand that item. In three groups, one person often took on the role of writer and another took on the role of sketcher. As the writer listed requirements for the robot, the sketcher drew it. All groups commented that it was much easier to make a contribution than in a conventional whiteboard meeting. In a conventional meeting, taking the floor means taking it away from someone else. In the We-Met meeting, the floor could easily be shared. Thus, the threshold for making a contribution seemed to be lower than in a conventional whiteboard meeting. The resulting synergistic ideas generation can be viewed as a process gain compared with conventional meetings.

Another style of interaction consisted of deliberately *separate parallel* activity. For example, some groups decided to work independently for several minutes and then share their individual creations sequentially. Another type of parallel activity occurred when one participant drew or wrote in an area of the scene not being viewed by others, while monitoring a conversation between the other two. In these instances, the individual was working on a topic different from the one under discussion by the other two. Typically, when the two finished their discussion, the third participant pointed out the new contribution.

Whitaker *et al.* (1991) also reported a large amount of separate parallel activity when three participants used a shared drawing tool without access to a speech channel. These authors suggested that the persistence of the written record encouraged users to work in parallel, since they knew they could attend to the contributions of others at their convenience. Such separate parallel activity enables the generation of a greater number of ideas in a fixed period of time and represents a potential process gain over conventional meetings or computer-based systems which permit only one person at a time to take the floor. When the shared drawing area and speech channel are used together as in this study, participants have the dual benefits of parallel work and the ability to direct other participants' attention to a particular contribution as desired.

In addition to the overlapping modes of writing drawing activity, We-Met also was used in a more conventional *moderator-led* style. One group, for example, adopted a moderator-led style for the initial 15 minutes in which all three participated in the spoken conversation but one person did all the writing. This participant also controlled the discussion. This is the typical model for a conventional whiteboard meeting. As it happened, this participant's display was the one projected onto the wall for public viewing, and after a few minutes the other participants directed their attention to the wall as the moderator scrolled the scene. It is possible that control of the public display plus the knowledge that the others were looking at the display empowered this participant to control the floor (an interesting topic for future research).

Common work product

In response to the question, 'What did you like about We-Met?' participants in each group commented that they liked the ability to produce a common work product. Compared with a conventional meeting, they thought that the 'shared understanding' would help avoid misunderstanding and miscommunication among participants about the group's work.

Space management

The version of We-Met tested here did not enable participants to have a private scene for their own use. A number of participants commented on the need for a private scene for two purposes: private note-taking and to work on ideas they were not ready to share with the group. For the latter purpose, people also wanted to be able to share the contents of the private scene with the group at will. In order to cope with the lack of a private drawing area, some users took advantage of the independent scrolling. They scrolled to an area of the screen not currently being used by others for their independent work. The group that decided to work in a separate parallel mode for a while actually drew lines across the scene to mark off a private area for each person. The lack of a private scene may produce a loss if it inhibits the development of ideas which individuals are not yet ready to share with others, or causes users to forget ideas that they would have written down as personal notes.

Both display resolution and the size of the drawing area that could be viewed at one time were limited by current display technology, and were not sufficient for many of the participants. The resolution (VGA) limitations meant that writing had to be larger in size

than on a sheet of paper in order to be legible. The size (7.5 inches by 3.38 inches) of the drawing area that could be viewed at one time is equivalent to about 1/3 of a sheet of paper. Users had two options for dealing with the lack of space: to scroll within the current scene, or to start a new scene. There seemed to be some individual differences concerning when to start a new scene vs. scroll for more space. There were episodes in which one member of a group suggested starting a new scene and another countered with the suggestion that they scroll within the existing scene. Compared with conventional whiteboard meetings, the lack of space is a process loss since it caused users to spend time trying to compensate for it, and may have curtailed their generation of ideas.

There is evidence that scenes were considered to be a means of organizing the logical structure of the meeting for some users. In one group, when the group ran out of space in a scene, a participant commented, 'Are we done now with this topic?'. They did not continue with this topic after starting the new scene. In another group, a participant commented that changing scenes seemed to force closure on a topic. Thus, it seems that an interface characteristic, the amount of space available in a scene, may effect a characteristic of the meeting, the degree to which a topic is explored. Observations with groups meeting on real work topics where participants have more at stake are necessary to determine how the interface affects both the meeting process and eventual work product.

Coordination among participants

Participants had no problem coordinating written/drawn contributions with those of other participants. There were only two instances of two people simultaneously trying to write in the same place (both in the distributed condition). In both cases, the collisions were immediately recognized and resolved. Because they saw each other's marks on a stroke-by-stroke basis, individuals had almost instantaneous information about the writing of others. Even when all three participants contributed to a common list, the ability to immediately see others' writing plus social protocol seemed to coordinate their contributions.

There were two aspects of We-Met which caused some coordination problems: changing scenes and scrolling.

To start a new scene, a user simply touches the *New* button; the new scene is then displayed to all participants. To change scenes, a user touches the *Scenes* button and then selects the desired scene from a list presented in a selection box (see Figure 5.2). The selection box is displayed only to the user who initiated the scene selection dialogue.

Six of the nine sessions experienced difficulty with scene changing.

There are three aspects of the scene change process which could potentially cause problems to users: agreeing on whether or not to change scenes, making the change and realizing that the scene has just been changed. The first aspect was generally handled without problem by speech in this study. When a participant wanted to change scenes or start a new scene, he or she usually announced the intention with a comment such as 'Should we go to a new scene now?'.

The second, making the change, was occasionally a source of confusion. On several occasions, two people started a new scene after the group agreed to do so, each unaware of the other's action. This created one more scene than the group intended. When participants later tried to select a scene from the list, they were confused because there were more scenes than expected. In one case, a participant made five consecutive scene

changes as he searched for the desired scene. This was followed by a scene change initiated by another person as he independently tried to find the correct scene. The additional unexpected change caused the first participant some annoyance.

Some possible means of control – locking out subsequent scene changes for an interval of time following a change or requiring confirmation of an impending change by all participants – have the unpleasant side effect of slowing the pace of interaction and burdening users with interface machinery. A combination of social protocol and improved feedback for in-progress scene changes are more appealing solutions.

The third aspect of a scene change, realizing that the scene had been changed, caused occasional difficulties. Although We-Met is designed so that the name of a scene (a number by default, or a user-created text or drawing) is displayed above the drawing area, this feature was not implemented in the version tested. Thus, users did not always have adequate information for easy identification of scenes. Identification of scenes by names may make it easier to tell that a scene has been changed. Another method of providing feedback for scene changes which is consistent with the scene metaphor would be to paint the screen with the new scene with a gradual ‘wipe’.

Scrolling was a source of difficulty for some participants. Each person could scroll the scene independently and that caused some problems coordinating the views of individual participants. Some users initially assumed that scrolling was not independent (in spite of instructions to the contrary). They may have assumed that WYSIWIS (what you see is what I see) applied to scrolling because all participants initially had the same view and because scene changes were not independent. They quickly learned that this was not the case.

Several people commented that they had difficulty keeping track of where others were writing, or when new information existed due to the independent scrolling. Unlike paper documents, in which the sights and sounds of page-turning inform a meeting participant that others are attending to a different part of the document, the physical cues for scrolling were few. This difficulty of keeping track of where others were writing seemed to detract from the feeling of shared understanding. Others have also observed scroll coordination problems in shared drawing tools (Greenberg *et al.*, 1992).

The most serious scrolling problem was finding the desired portion of the scene. Although other participants tried to help a member who was lost in space with advice such as ‘scroll down’ or by drawing the path to the desired location as described above, they had no quick way of communicating the location information in an adequately precise way. On occasions the person whose display was projected on the wall assumed the role of controlling the projected display for the benefit of all.

Scrolling is an interface technique developed for single-user applications to deal with the need for more screen space. Although users get lost when scrolling in single-user applications, the problem is perhaps more annoying in a shared application because the user must drop out of the ongoing discussion while searching for the desired position. Some obvious solutions to scroll coordination problems – such as scrolling all scenes in lock-step, or getting rid of the need for scrolling by limiting the size of a scene to the dimensions of the scene window – seem to have undesirable characteristics for a tool designed to support small, highly interactive groups, in that they take control away from the user. Approaches that contribute to mutual awareness of location and scrolling actions by providing additional physical cues (see for example, Gaver (1986)), or facilitate navigation to the desired location seem more promising.

The fact that people had little problem coordinating their written contributions but did have problems coordinating their shared control of the We-Met interface is very likely due to their differential experience with these tasks. Experience in meetings has taught people that they must coordinate their speaking and writing activities with those of others. However, most people have no experience in coordinating control of a computer interface with others. Some insight into how experienced users might manage control of a shared application comes from one group in which one participant assumed the role of 'scene coordinator' and executed all scene changes for the group. He informed the group of an impending scene, by announcing 'switching'. This individual had a PhD in human factors and training in human communication. He was consequently much more aware of the need for coordination in the control of the interface than most users.

Time lost in coordination confusion and the recovery from confusion constitute a process loss. However, time spent in proactive coordination, such as discussing whether the group should move to a new scene, may be more effective than any control mechanism or rule that could be built into a meeting support tool. As others have noted (Johnson-Lenz and Johnson-Lenz, 1991), developing effective CSCW tools means developing protocols and strategies for their use as well as developing technology.

5.3.3 Conclusion

The study provided encouraging evidence for the ease of use of the tool, and suggested that We-Met supports a variety of interaction styles and modes of communication in both distributed and co-located meetings. The study alerted us to some potential process losses due to difficulties with available space and coordination of the use of interface controls across users. It also hinted at some interesting interactions between technology and meeting process which should be explored in future studies.

The research described here has focused on the meeting process. We have not attempted to evaluate the quality of problem solution achieved using We-Met (see Olson *et al.* (1992) for a good example of this). We hope to address this issue in future work.

5.4 SEARCH AND RETRIEVAL OF MEETING INFORMATION

In order to gain insight into how to facilitate the search and retrieval of information from We-Met meetings, we conducted a study of conventional meetings. The goals were two-fold: to identify methods of searching for meeting information which could be automated by a pen-based meeting tool, and to identify difficulties in searching for this information which were amenable to computer support. This study was done before the system described in the previous section was operational.

5.4.1 Study procedure

Ten meetings occurring at the IBM T. J. Watson Research Center were observed and videotaped. The groups consisted of both technical and non-technical staff. The meeting purposes included research design and planning, information gathering or presentation, project update and project presentation. The modal number of people in a meeting was 3.

with a range of 2 to 10 participants. A total of 23 participants from nine of these meetings were interviewed individually several days to two weeks later.

During the interview, a participant was first asked to recall the main points of the meeting, as well as the specific ideas, proposals or plans discussed. In the second stage of the interview, participants were asked to search from one of these specific items in the videotape of the meeting. The participants searched the videotape by manipulating the play, reverse, fast forward and slow speed search controls. They were told how long the meeting had been and could also use the time counter in their searches. As they searched the videotape, the experimenter encouraged the participants to describe how they were searching for the required information.

5.4.2 Findings

In considering the findings, one should keep in mind that the characteristics of videotape as a search medium constrain the possible search strategies. One cannot access semantic information directly, but must rely on more salient physical characteristics of the meeting. Furthermore, since videotape is a sequential medium, one must search the tape in time sequential order (backwards, forwards). However, since the drawings and writings in a We-Met scene are stored as stokes, rather than coded text or graphics, some of the difficulties of searching an analogue medium apply to We-Met as well.

The study revealed that people search for meeting information using a number of characteristics.

- *Participant behaviour*: people looked for information based on the behaviour of others ('I'm looking for times when the people (two other participants) were speaking'), or themselves ('I'm looking at myself taking notes').
- *Communication medium*: people recalled the medium by which the information was presented (e.g. foil, whiteboard) and also searched for a particular communication medium to locate a spoken discussion ('I know it is around the chart presentation').
- *Time*: several aspects of time were used to search for information: relative time ('It's probably midway into the meeting'), duration (25 minutes into the discussion), and clock time ('We only got through item 1.2 by 5:00').
- *Relation to other events*: as they played the videotape, people frequently remarked that the event they were looking for occurred before or after an event currently being viewed (e.g. when a person left the meeting). These events served as markers which guided the participant to move backwards or forwards in the videotape record.

Many of these search strategies can be automated with a computer, since these characteristics can be recorded as part of the meeting record. The We-Met record contains a time-stamp for each event, so time could easily be used as a search strategy. Writer identification is recorded and this aspect of participant behaviour could also be used. Much of the meeting information is contained in the spoken discussion, however, and the ability to search by speaker and play back speech in conjunction with the stroke-based record would be quite valuable. Communication medium could also be recorded and used for search. Characteristics such as these could be used in combination to limit the portion of the meeting searched.

In addition, participants used a strategy we call temporal sampling — viewing segments of the videotape, fast forwarding, and then viewing another segment of the videotape. This was a cumbersome process since the video could not be viewed in this mode of operation, and it was hard to control the distance moved with much precision. An interesting side effect of temporal sampling was that it occasionally caused people to revise their recollection of the meeting when they saw events they had forgotten or misremembered.

We have implemented a simple version of temporal sampling in our current prototype. When the horizontal slider at the top of the drawing area is moved to the left, the meeting record is replayed in reverse temporal order and, as it is moved to the right, it is played in the forward direction. Our initial experience with this feature has been positive and we are eager to learn how users will employ this feature.

The search methods described above are a means to get to the desired semantic information. It would be quite desirable to enable the user to access this semantic information directly. Text-based databases typically use content or keyword searches for this purpose. Since the writing and drawings in the We-Met record are not recognized, a method analogous to content searching is not available to us. However, the ability to identify a viewpoint with a text name or hand-drawn squiggle may serve a purpose similar to searches based on explicitly defined keywords.

5.4.3 Conclusion

This study gave us insight into the methods people use to search for meeting information in a non-coded medium. We plan to explore some of these methods in future versions of We-Met. The research also suggested how the power of the computer could be used in new ways to facilitate searching of the meeting record. Finally, our observations have underscored the importance of dealing with the audio record, in addition to the stroke-based record, in future versions of We-Met.

5.5 SUMMARY

We have described We-Met, a pen-based prototype designed to explore the communication and information retrieval needs of small group meetings. The study of the use of We-Met suggests the pen-based interface style produces communication process gains, and supports a variety of participant interaction styles. This study also exposed some problems in the use of the limited display space and in the coordination of the use of interface controls across users. The study of strategies users employ when searching for meeting information on videotape identified a number of methods (such as searching by time or writer) which could be automated in a computer-based tool. Some of the problems encountered in searching videotape (such as the ability to access information directly) may be amenable to computer support in We-Met.

The empirical studies reported here have given us some early insights into the features required to support small group meetings. We look forward to reporting on our progress as we continue our development and evaluation efforts.

ACKNOWLEDGEMENTS

We gratefully acknowledge the contributions of Angela McBride who coded the videotapes and assisted in the development of the coding scheme. We thank the Integrated Systems Solutions Corporation for their support of this work.

REFERENCES

- Bly, S. A. (1988) A use of drawing surfaces in different collaborative settings, in *CSCW '88: Proceedings of the Conference on Computer-Supported Cooperative Work*, Portland, Oregon, pp. 250–6.
- Bly, S. A. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems*, 26–28 September, ACM Press, Portland, Oregon, pp. 184–92.
- Conklin, J. (1989) Design rationale and maintainability, in *Proceedings of the 22nd HICCS*, IEEE Computer Society Press.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, *Communications of the ACM*, **34** (January), pp. 39–58.
- Gaver, W. W. (1986) Auditory icons, using sound in computer interfaces, *Human-Computer Interaction*, **2**, pp. 167–77.
- Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992) Human and technical factors of distributed group drawing tools, *Interacting with Computers*, **4**, pp. 364–92.
- Johnson-Lenz, P. and Johnson-Lenz, T. (1991) Post-mechanistic groupware primitives: rhythms, boundaries and containers, *International Journal of Man-Machine Studies*, **34**, pp. 395–417.
- Levinson, S. C. (1983) *Pragmatics*, Cambridge University Press, Cambridge.
- MacLean, A., Young, R. M. and Moran, T. P. (1989) Design rationale: the argument behind the artefact, *Proceedings of CHI '89*, 30 April–4 May, ACM Press, Austin, Texas, pp. 247–52.
- Mantei, M. (1988) Capturing the Capture Lab concepts: a case study in the design of computer supported meeting environments, in *CSCW '88: Proceedings of the Conference on Computer-Supported Cooperative Work*, Portland, Oregon, pp. 257–70.
- Minneman, S. L. and Bly, S. A. (1991) Managing a trois: a study of a multi-user drawing tool in distributed design work, in *Proceedings of CHI '91*, 15–17 May, Guelph, Ontario, Canada, pp. 217–24.
- Muller, M. J. (1991) PICTIVE – An exploration in participatory design, in *Proceedings of CHI '91*, 27 April–2 May, New Orleans, Louisiana, pp. 225–31.
- Nunamaker Jr, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R. and George, J. F. (1991) Electronic meetings systems to support group work, *Communications of the ACM*, **34**, pp. 41–61.
- Olson, J. S., Olson, G. M., Storrosten, M. and Carter, M. (1992) How a group-editor changes the character of a design meeting as well as its outcome, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '92)* 31 October–4 November, ACM Press, Toronto, Canada, pp. 91–106.
- Rhyne, J. R. and Wolf, C. G. (1992) Tools for supporting the collaborative process, in *Proceedings of the Fifth Annual Symposium on User Interface Software and Technology (UIST '92)* 15–18 November, ACM Press, Monterey, California, pp. 161–70.
- Sarin, S. and Greif, I. (1988) Computer-based real-time conferencing systems, In Greif, I. (ed.) *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann, San Mateo, pp. 397–420.
- Sellen, A. J. (1992) Speech patterns in video-mediated conversations, in *Proceedings of CHI '92*, 3–7 May, ACM Press, Monterey, California, pp. 49–59.
- Stefik, M., Foster, G., Bobrow, D. G., Kahne, K., Lanning, S. and Suchman, L. (1987) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, **30**, pp. 32–47.
- Tang, J. C. (1991) Findings from observational studies of collaborative work, *International Journal of Man-Machine Studies*, **34**, 143–60.
- Tang, J. C. and Minneman, S. (1991) Video WhiteBoard: video shadows to support remote collaboration, in *Proceedings of CHI '91*, April, ACM Press, New Orleans, pp. 315–22.
- Tatar, D. G., Foster, G. and Bobrow, D. G. (1991) Design for conversation: lessons from Cognoter, *International Journal of Man-Machine Studies*, **34**, pp. 185–209.

- Whitaker, S., Brennan, S. E. and Clark, H. H. (1991) Co-ordinating activity: an analysis of interaction in computer-supported co-operative work, in *Proceedings of CHI '91*, 27 April–2 May, ACM Press, New Orleans, Louisiana, pp. 361–7.
- Wolf, C. G. and Rhyne, J. R. (1993) Gesturing with shared drawing tools, in *InterChi '93 Adjunct Proceedings*, 24–29 April, ACM Press, Amsterdam, The Netherlands, pp. 137–8.
- Wolf, C. G., Rhyne, J. R. and Briggs, L. K. (1992) Communication and information retrieval with a pen-based meeting support tool, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '92)*, 31 October–4 November, ACM Press, Toronto, Canada, pp. 322–9.
- Wolf, C. G., Rhyne, J. R., Zorman, L. A. and Ossher, H. O. (1991) WE-MET (Window Environment-Meeting Enhancement Tools), *CHI '91 Proceedings*, pp. 441–2.

INTEGRATION OF INTERPERSONAL SPACE AND SHARED WORKSPACE: CLEARBOARD DESIGN AND EXPERIMENTS¹

Hiroshi Ishii, Minoru Kobayashi and
Jonathan Grudin²

ABSTRACT

We describe the evolution of the novel shared drawing medium ClearBoard, which was designed to seamlessly integrate an interpersonal space and a shared workspace. ClearBoard permits co-workers in two locations to draw with coloured markers or with electronic pens and software tools while maintaining direct eye contact and the ability to employ natural gestures. The ClearBoard design is based on the key metaphor of 'talking through and drawing on a transparent glass window'. We described the evolution from ClearBoard-1 (which enables shared video drawing) to ClearBoard-2 (which incorporates TeamPaint, a multi-user paint editor). Initial observations and findings gained through the experimental use of the prototype, including the feature of 'gaze awareness', are discussed. Further experiments are conducted with ClearBoard-0 (a simple mockup), ClearBoard-1 and an actual desktop as a control. In the settings we examined, the ClearBoard environment led to more eye contact and potential awareness of collaborator's gaze direction over the traditional desktop environment.

Reprinted, with permission, from *ACM Transactions on Information Systems*, October 1993. Association for Computing Machinery, Inc.

¹ A previous version of this paper was presented at the 1992 ACM Conference on Computer-Supported Cooperative Work (CSCW '92) (Ishii *et al.*, 1992a). Descriptions of ClearBoard-1 given in an ACM CHI '92 paper (Ishii and Kobayashi, 1992) are incorporated into this paper.

² Jonathon Grudin participated in the experimental and observational phase of the ClearBoard research during his stay at NTT Human Interface Laboratories in November 1991 as a visiting professor and contributed to the preparation of this article.

6.1 ITERATIVE DESIGN OF SEAMLESS COLLABORATION MEDIA

Ideally, tools enable us to work smoothly and without interruption. Complex computer-based tools, however, often require too much of our attention, distracting us temporarily from the work we are doing. 'Seamlessness', in the sense of eliminating unnecessary obstructing perceptual seams, has been a key concept of our evolving media design. Collaborative work is particularly marked by spatial, temporal and functional constraints that force us to shift among a variety of spaces or modes of operation. Seamless design undertakes to decrease the cognitive loads of users as they move dynamically across different aspects of their work.

Our research progressed through iterative design steps from TeamWorkStation-1 and TeamWorkStation-2 to ClearBoard-1 and ClearBoard-2 (Ishii *et al.*, 1992b). Ishii designed TeamWorkStation-1 (TWS-1) to enable smooth transitions over:

1. the seam between individual and shared workspaces
2. the seam between computer and desktop tools (Ishii, 1990; Ishii and Miyake, 1991).

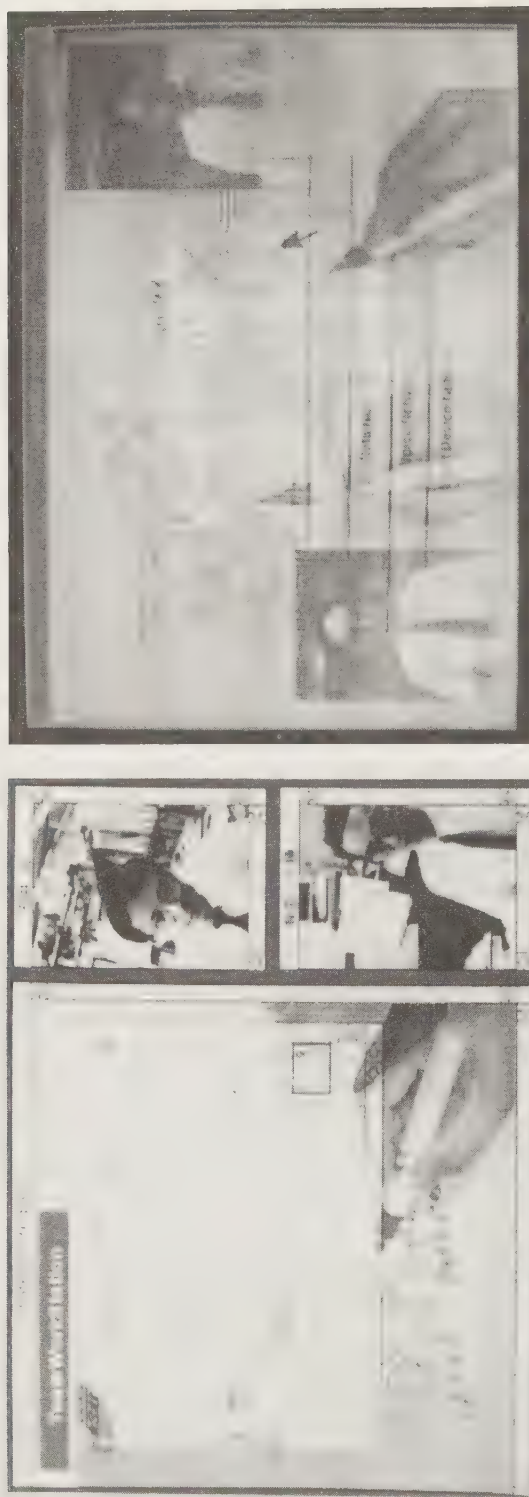
TWS-1 provides a shared workspace in which participants can freely use a variety of everyday media, such as computer tools, handwriting, printed materials and hand gestures (see Figure 6.1(a)). The two windows on the right provide live video images of the participants' faces. The left window provides a *shared workspace* that is made by *translucent overlay* of the individual workspace images (live video images of desktop surfaces and/or computer screen images).

TeamWorkStation-2 (TWS-2) is developed to provide the shared workspace over narrowband ISDN (Integrated Services Digital Network) using the CCITT H.261 standard (Ishii *et al.*, 1993). ClearFace, a new multi-user interface for TWS-2, superimposes translucent live video facial images over a workspace image to enable the more effective use of the normally limited screen space (see Figure 6.1(b)). Users have little difficulty in selectively viewing either the facial image or the workspace image (Ishii and Arita, 1991).

These multi-user interfaces for TWS-1 and TWS-2, however, suffer from an undesirable seam between the face images and the shared workspace. We realized that the problem was not just the superficial physical discontinuity of spatially separated windows: absent are the cues that would enable a smooth shift of focus between the shared workspace and the partner's face image. Camera positioning prevents one person from knowing the direction of the other's gaze; it could be directed toward the face image, toward objects in the shared workspace window, or elsewhere. A shift in focus is not apparent until accompanied by a visible gesture, a mouse movement or an audible remark. Awareness of gaze direction and mutual eye contact are impossible.

To overcome this limitation of TeamWorkStation, and to realize a seamless environment for real-time collaboration, we started the ClearBoard project. Based on the new metaphor of a transparent glass board, we have developed two working prototypes, ClearBoard-1 and 2, and conducted some experiments with them (Ishii and Kobayashi, 1992; Ishii *et al.*, 1992a).

This article first discusses the notions of *shared workspace* and *interpersonal space* and reviews existing support technologies for activities in these spaces (and their limitations) to fully explain the goal of the ClearBoard project. In the next section, we introduce the metaphor upon which ClearBoard is based: *looking through* and *drawing on* a glass board.



(b)

(a)

Figure 6.1 Shared screen examples of TeamWorkStation (a) original multi-window interface for TeamWorkStation-1; (b) new ClearFace interface for TeamWorkStation-2.

We then introduce the architecture of ClearBoard-1, a prototype that supports remote collaboration through *shared video drawing*. Next we describe ClearBoard-2, which utilizes computer-based shared drawing to overcome the limitations of ClearBoard-1. We briefly outline an alternative architecture that represents an interesting technological possibility. Finally, we briefly describe the experimental use of these prototypes. Both informal use and formal experiments are covered; the latter designed to explore the feature of 'gaze-awareness'.

6.2 SHARED WORKSPACE AND INTERPERSONAL SPACE

One major focus of groupware development has been the creation of virtual '*shared workspaces*' in distributed computer environments. Some groupware definitions take this workspace-oriented view, such as:

Groupware . . . the computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment. (Ellis *et al.*, 1991)

Whiteboards and overhead projections of transparencies are examples of shared workspaces in face-to-face meetings. Participants can see, point to or draw on a whiteboard simultaneously. An overhead projector makes handwritten or computer-generated documents visible to all participants in a room while permitting the speaker to point or draw. Shared workspace activities include sharing information, pointing to specific items, marking, annotating and editing.

In a distributed, real-time collaboration, these activities can be supported by computer-based groupware, including:

1. Shared screen systems such as Timbuktu (Farallon, 1991).
2. Shared window systems such as VConf and Dialogo (Lauwers *et al.*, 1990).
3. Multi-user editors such as Cognoter (Foster and Stefik, 1986), GROVE (Ellis *et al.*, 1991), ShrEdit (Olson *et al.*, 1993), Commune (Bly and Minneman, 1990), CaveDraw (Lu and Mantei, 1991), Aspects (Group Technologies, 1990), GroupSketch (Greenberg *et al.*, 1992), GroupDraw (Greenberg *et al.*, 1992), We-Met (Wolf and Rhyne, 1992), WSCRAWL, integrated in SEPIA (Streitz *et al.*, 1992), Tivoli (Pedersen *et al.*, 1993) and TeamPaint (described later). Use of hand gestures in a shared workspace can be supported by shared video drawing media such as VideoDraw (Tang and Minneman, 1991a) and TeamWorkStation (Ishii, 1990).

In face-to-face meetings, we speak, make eye contact and observe each other's facial expressions and gestures. These verbal and non-verbal channels are important in building confidence and establishing trust (Argyle, 1975; Buxton, 1992; Mantei *et al.*, 1991). The focus of telecommunication technologies such as the videophone and video conferencing has been the creation of 'interpersonal spaces' that maintain a sense of 'tele-presence' or 'being there' (Hollan and Stornetta, 1992) through the visibility of gestures and facial expressions of distributed group members. Media Space (Bly *et al.*, 1993), CRUISER (Fish *et al.*, 1993), and VideoWindow (Fish *et al.*, 1990) are examples of such technologies.

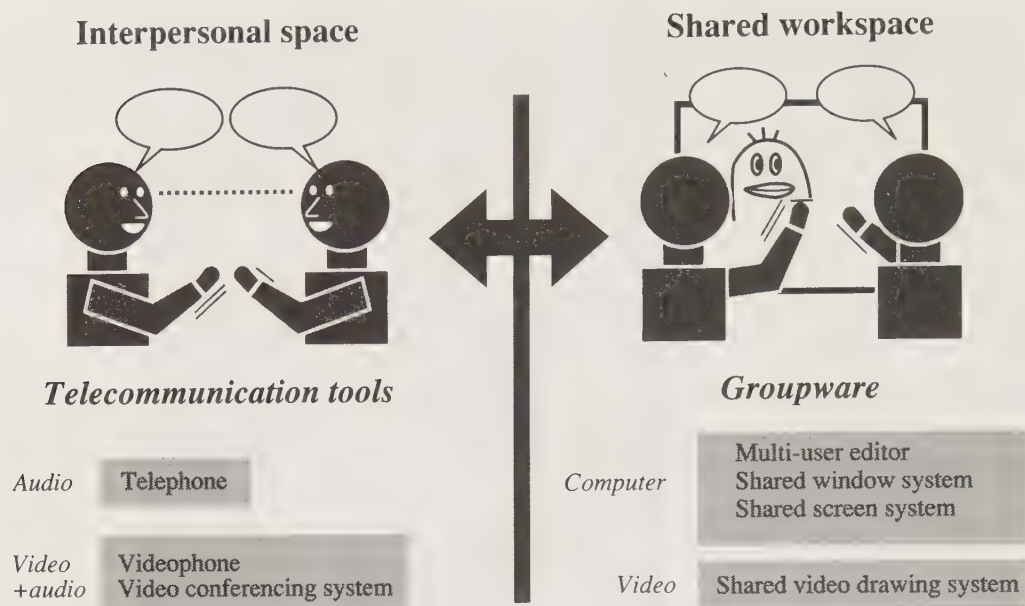


Figure 6.2 Interpersonal space, shared workspace and support technologies in real-time remote collaboration.

Figure 6.2 illustrates these concepts and identifies the technical support available for real-time remote collaboration.³

6.3 LIMITATIONS OF EXISTING SUPPORT TECHNOLOGIES

Both shared workspace and interpersonal space are present in ordinary face-to-face meetings and may be essential for remote real-time collaboration. Several media space⁴ technologies support both spaces.

Figure 6.3 illustrates three typical display arrangements of media spaces. In Figure 6.3(a), a display providing a live video image of the partner's face adjoins a display showing the shared work. The ARKola simulation (Gaver *et al.*, 1991) in the IIIF environment (Buxton and Moran, 1990) and some nodes of CAVECAT (Mantei *et al.*, 1991) adopt this arrangement. SEPIA-IPSI media space (Streitz *et al.*, 1992) locates small custom-built desktop video devices (small monitors and camera) on top of the computer screen. In Figure 6.3(b), the displays are repositioned to resemble the situation of interacting across a table. VideoDraw (Tang and Minneman, 1991a) and Commune (Bly and Minneman, 1990; Minneman and Bly, 1991) experiments adopt this arrangement. In

³ This framework was developed through a discussion with William Buxton who pointed out the importance of a smooth transition between what he calls 'shared task space' and 'person space' (Buxton, 1992).

⁴ 'Media space', originally the name of a specific system (Bly *et al.*, 1993), is used here in the sense of Mantei *et al.* (1991) as a general term to represent computer-controlled video environments.

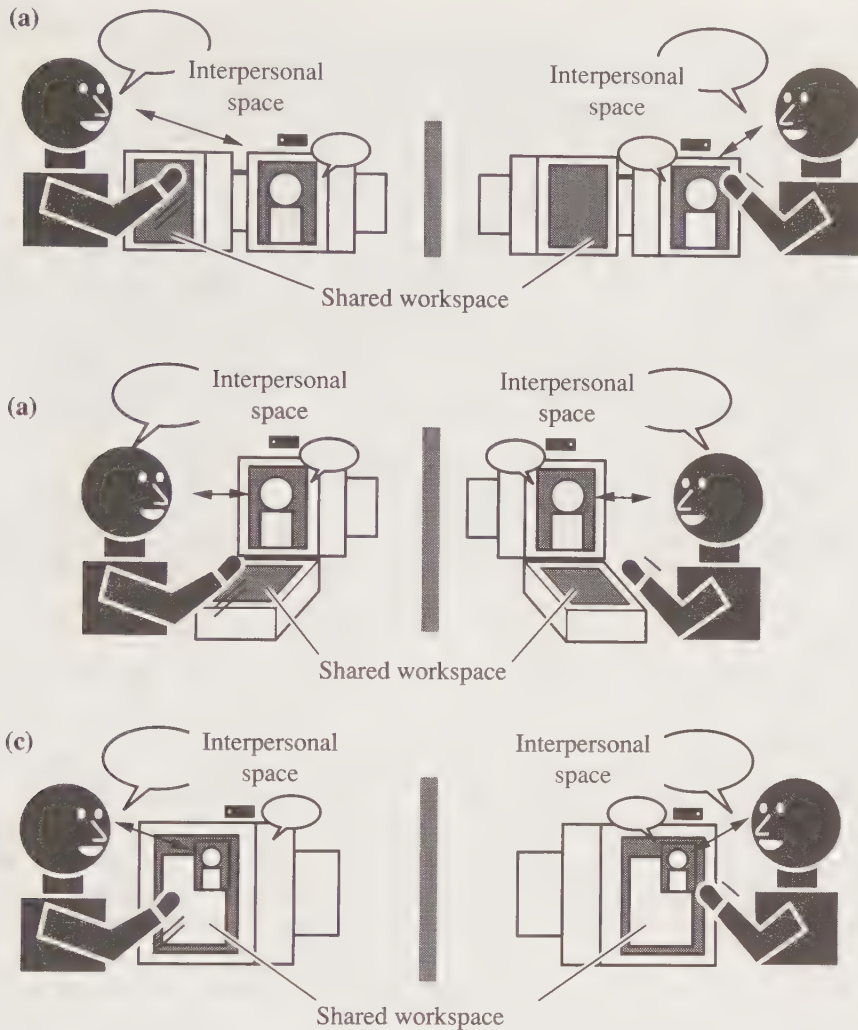


Figure 6.3 Typical screen arrangements in media space.

Figure 6.3(c), the live video images and the shared workspaces are incorporated into different windows of a single screen. TeamWorkStation (Figure 6.1), PMTC (Tanigawa *et al.*, 1991), MERMAID (Watabe *et al.*, 1990) and some CAVECAT nodes employ this desktop video technology.

These designs, although they represent an advance, share a major limitation: an arbitrary seam between shared workspace and interpersonal space. Experiments on TeamWorkStation proved that the problem is not just the superficial physical discontinuity of spatially separated windows. Users experience an undesirable gap between the two functional spaces: shared workspace and interpersonal space.

In a face-to-face design meeting, while using a whiteboard or drawing surface we frequently switch our focus between the two spaces. Even when drawing, we briefly glance

at our partner's face to attract attention or gauge comprehension. Similarly, our partner's head turning, eye movement and gestures also attract our attention and trigger our focus shift. This dynamic and interactive focus switching between shared workspace and interpersonal space is made possible in ordinary meetings by the presence of a variety of *non-verbal cues*.

Current media space technologies do not provide strong clues. Spaces created by these technologies are discontinuous and arbitrary (Gaver, 1992). Users cannot switch their focus between the two spaces naturally and smoothly. ClearBoard is designed for pairs of users and overcomes these limitations by seamlessly connecting interpersonal space and shared workspace. ClearBoard allows users to shift easily between interpersonal space and shared workspace using familiar everyday cues such as the partner's gestures, head movements, eye contact and gaze direction.

6.4 GLASS BOARD METAPHOR OF CLEARBOARD

Our first step was to consider the metaphors needed to create a medium that allows people to use everyday skills without special training. We first investigated two familiar metaphors: (a) talking over a table and (b) talking in front of a whiteboard.

Work carried out across a table, or with a system based on this metaphor of sitting on opposite sides of a table and talking over it, is suitable for face-to-face communication because the participants can easily see each other's faces. Unfortunately, the drawing orientation is upside-down for one party. Work carried out with a whiteboard or a system based on a whiteboard metaphor (or by participants on the same side of a desk or table) has the advantage that the participants share a common board orientation.

With real tables and whiteboards, participants share the same physical space and may touch their partners. However, it is hard to implement a mechanism than can usefully recreate this shared space. The only way we found of realizing this metaphor is to employ 'virtual reality' technology with force feedback mechanisms. However, we do not think it is a good idea to force users to wear awkward head-mounted displays, special gloves, and a suit just to share some drawings. This solution places users in a computer-generated virtual world that definitely increases cognitive loads.

Kobayashi came up with the new metaphor of 'looking *through* and drawing *on* a big glass board' and gave it the name 'ClearBoard' (Ishii and Kobayashi, 1992). Figure 6.4 shows ClearBoard-0, which is the simple mockup of this ClearBoard concept for a co-located pair of users. ClearBoard-0 consists of a glass board positioned between the partners on which they draw or post objects. This prototype represents the best possible case for visual clarity. In addition to reinforcing the actual physical separation of the partners, ClearBoard requires less eye and head movement to switch focus between the drawing surface and the partner's face than is needed in either the whiteboard or desktop environment. A real glass board has the problem that written text appears reversed to one's partner; we were able to solve this problem by *mirror-reversing* video images in ClearBoard-1 and 2, as described below.

The existing systems most similar to ClearBoard are VideoWhiteBoard (Tang and Minneman, 1991b) and LookingGlass (Clark and Scrivener, 1992). VideoWhiteBoard utilizes the users' shadows to convey their gestures during shared drawing activity. VideoWhiteBoard looks like a *frosted* glass board, in contrast to the *transparent* glass



Figure 6.4 A simple mockup of the ClearBoard metaphor: ClearBoard-0.

board that ClearBoard reproduces. VideoWhiteBoard merges elements of interpersonal space and shared workspace; however, the shadow images do not convey facial expressions, eye movement or eye contact, all of which are present with ClearBoard.

LookingGlass displays the full-screen window of a ROCOCO sketchpad (shared drawing software) over a full-screen video image of a remote partner. Using a half mirror, LookingGlass supports eye contact. However, the use of indirect drawing devices (mouse, digitizer) separates hand movements on a desktop from cursor movement on the computer screens, and users cannot see the actual hand gesture behind the cursor movement. The users cannot place their hands close to marks on the computer screen surface for pointing because the half-silvered mirror is angled backwards at 45° in front of the vertical screen surface. In contrast, ClearBoard allows users to draw on and gesture directly over the screen surface.

6.5 THE DESIGN OF CLEARBOARD-1

In order to implement the remote version of ClearBoard, we identified three design requirements:

1. Direct drawing on the display screen surface must be supported.
2. The video image of user must be taken *through* the screen surface to achieve eye contact.
3. A common drawing orientation must be provided.

In order to satisfy these three requirements with simple technologies, Ishii devised the system architecture called the 'drafter-mirror' architecture. The first prototype, ClearBoard-1, was implemented in November 1990 by Kobayashi.

6.5.1 ClearBoard-1 architecture

Figure 6.5 illustrates the drafter-mirror architecture of ClearBoard-1. It looks like a 'drafter' (a desk for architectural drawing) and it uses a half-silvered mirror to satisfy requirement 2. Each terminal is equipped with a tilted screen, a video projector and a video camera. The screen is angled backwards at about 45° , and is composed of a projection screen, a polarized film and a half-silvered mirror (which reflects half of the light reaching it and allows half to pass through). Video feedback between the two cameras and screen pairs is prevented by the polarizing filter placed over each camera lens and the nearly orthogonal polarizing filter that covers the surface of each screen. Users can write and draw on the surface of the screen using coloured paint markers⁵ and cloth erasers.

Figure 6.6 illustrates how this drafter-mirror architecture works. The video camera located above the screen captures the drawing marks on the screen surface and the image of the user reflected by the half-silvered mirror as a continuous video image. This image is sent to the other terminal through a video network, and projected on the partner's screen from the rear. The partner can draw directly over this transmitted video image⁶.

The image of the partner and the partner's drawing is mirror-reversed so that ClearBoard-1 provides both users with a common drawing orientation on their screens. Since the user's image is also mirror-reversed, a right-handed partner will appear to be left-handed. The drafter-mirror architecture results in the video camera capturing double hand images as it draws – one being the direct image, and the other being the image reflected by the half-silvered mirror, as shown in Figure 6.6.

The drawing image captured by the camera is trapezoidally distorted due to perspective because the screen is at an angle to the camera. In order to support shared drawing on the screen, the drawing image must be recreated with the original shape and size on the partner's screen. In ClearBoard-1, the distortion is offset by the opposite distortion caused by projecting the image onto the tilted screen. In order to give a suitable distortion rate, the camera and the projector are symmetrically arranged with respect to the screen.

6.5.2 Results of ClearBoard-1 experiments

Figure 6.7 shows snapshots of the ClearBoard-1 prototype in an experimental session that will be described later. In summary, we observed effortless focus switching between the task and the partner's face. Users could read their partner's facial expression, achieve eye contact and utilize their awareness of the direction of their partner's gaze (we call this characteristic 'gaze awareness' (Ishii and Kobayashi, 1992)). These last two features are the novel aspects of the ClearBoard prototypes and their importance is discussed later.

Most users did not notice double hand images. Some users, however, reported an initial period of distraction. No subjects reported difficulty with the mirror-reversal of the partner. This may be because our faces are quite symmetrical, or our own images are reversed in mirrors.

⁵ Water-based fluorescent paint markers were used in our experiment because these colours are easy to distinguish from the images of the user and the user's background.

⁶ This shared video drawing technique, which allows remote partners to draw directly over the video image of their co-workers' drawing surface, was originally demonstrated in VideoDraw (Tang and Minneman, 1990).

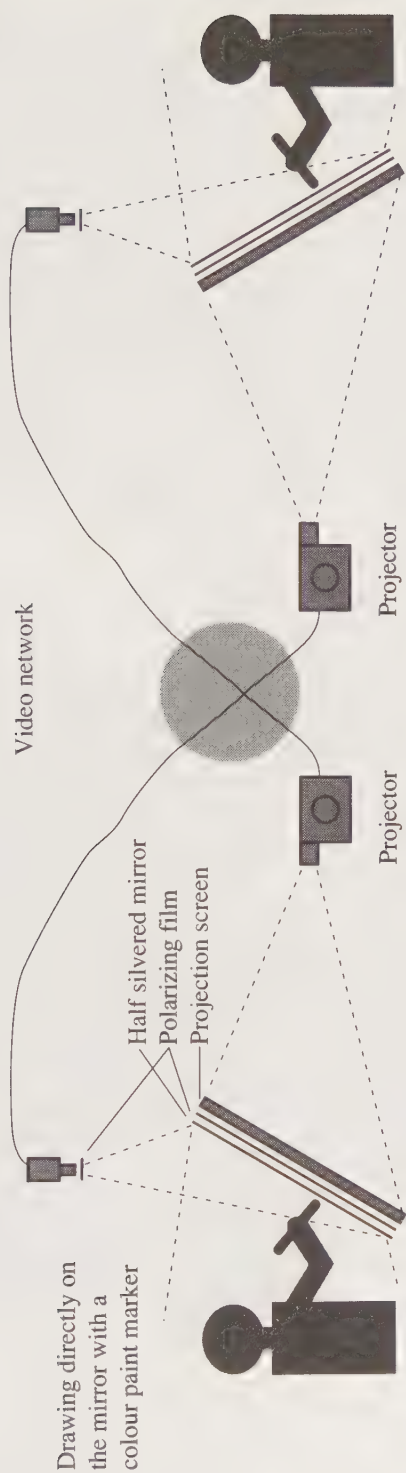


Figure 6.5 System architecture of ClearBoard-1.



Figure 6.7 ClearBoard-1 in use.

An interesting and less critical confusion manifested itself when users directly drew over their partner's image, playfully adding eye glasses or moustache, for example. Clearly they had a WYSIWIS (what you see is what I see) expectation, not realizing that although the drawing is shared, the facial images are not, with each person seeing only the other's image. Thus, the metaphor of the ClearBoard is not always entirely assimilated.

We also encountered problems in using ClearBoard-1: the most frequently reported problem was that video images on the screen are darker and less clear than is desirable. This is because:

1. The brightness of the projected video image is reduced by the use of half-silvered mirrors and polarizing films (more than half of the light is lost).
2. The tilt of the screen places the bottom edge about 40 cm further from the camera than the top, making it hard to keep the entire drawing surface (and the drawer's face) in sharp focus.
3. The video resolution was limited to the liquid crystal video projector's 90 000 pixels (in contrast to the approximately 400 000 pixels of the CCD camera).

The lack of video resolution forced the use of thick colour paint markers; drawing with them is not precise and quickly uses up the available display space (50 cm x 55 cm). This problem is exacerbated by the low erasure efficiency with the cloth erasers and the difficulty of recording the resulting drawings. (We mainly used Polaroid cameras or video printers.) Another limitation of shared video drawing is that a user cannot erase the partner's drawing (Tang and Minneman, 1991a). Marks drawn by each user exist only on their respective screen surfaces. When users run out of free drawing space, they often hesitate to ask each other to erase marks or are embarrassed by requests to erase their own marks because of the lack of appropriate recording and reusing capabilities.

It would be desirable to be able to bring both computer files and printed materials directly into the ClearBoard shared drawing space, much as is possible in TeamWorkStation. Computer file access can be incorporated by using a computer-based groupware technology that permits shared drawing over the documents stored in computer files; the use of printed materials remains a problem if we stick to direct drawing. If user A puts a sheet of paper on his or her ClearBoard surface and marks on it, user B can see it, but subsequent marks by B will not be seen by A because the paper blocks the rear-projected image.

6.6 THE DESIGN OF CLEARBOARD-2 WITH TEAMPAIN

To overcome many of the problems of ClearBoard-1, we designed a new computer-based prototype, ClearBoard-2. Instead of video drawing with coloured paint markers, ClearBoard-2 provides users with digitizer pens and TeamPaint, a multi-user computer-based paint editor running on networked Macintosh computers.

6.6.1 ClearBoard-2 architecture

Figure 6.8 illustrates the system architecture of ClearBoard-2. To improve the clarity of the screen image, we used a CRT-based rear projection display with a transparent digitizer sheet. The digitizer is mounted to the surface of a flat panel display.

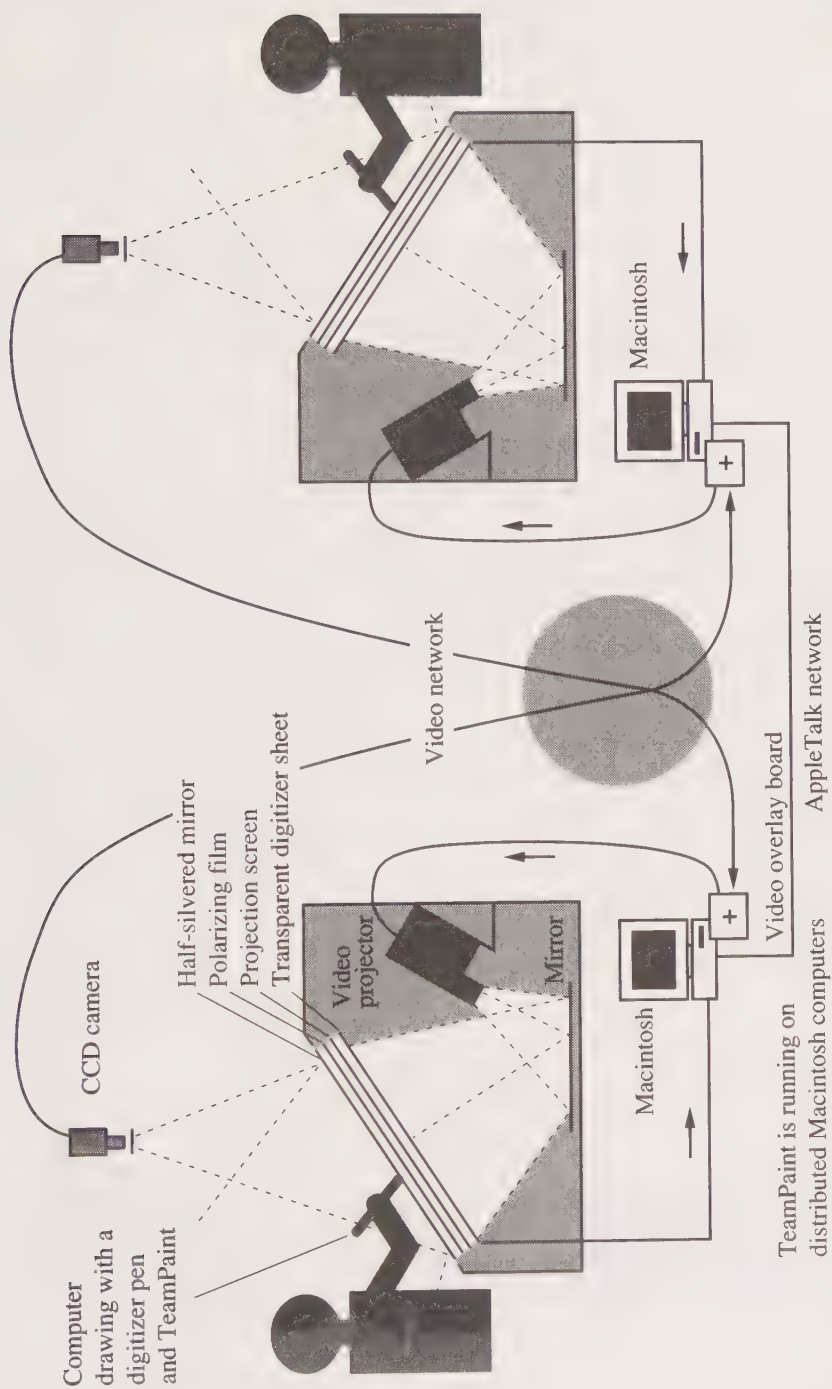


Figure 6.8 System architecture of ClearBoard-2.

The screen size is 80 cm x 60 cm, 1.7 times bigger than the ClearBoard-1 screen. Although ClearBoard-2 is based on the same 'drafter-mirror' architecture as ClearBoard-1, the digitizer pen and TeamPaint add the following capabilities:

1. Collaboratively created drawings can be saved as computer files and re-accessed later.
2. Documents created with other editors can be imported.
3. It is easy to get a new blank sheet (drawing space).
4. It is easy to edit and erase marks.

The shared drawing image (RGB video) is overlaid on the video image of the co-worker (NTSC) using a special video overlay board and the mixed RGB video image is projected on the screen by a video projector. Chroma-keying in the overlay sharpens the drawing image against the image of the co-worker.

6.6.2 TeamPaint: a multi-user paint editor

TeamPaint is not a special component of ClearBoard-2; it is a groupware application that runs on AppleTalk-networked Macintosh computers without any special hardware. It can be used by any number of users simultaneously, but performance drops off somewhat as the number of users increases. A mouse or tablet can be used for indirect drawing; the digitizer screen supports direct drawing. Figure 6.9 shows an example of a TeamPaint screen. TeamPaint design is based on the following principles:

1. *A simple human interface*

TeamPaint provides an intuitive interface based on the metaphor of drawing on a sketchpad with a colour pencil. Scissors provide the functions of cutting, copying and moving marks. A new blank sheet can be obtained by a single mouse click on the bottom left corner of the pad. To maximize transparency and speed of drawing, it is a simple bitmap paint editor, not an object-oriented draw editor. TeamPaint enables simultaneous gesturing and drawing by multiple users.

2. *Replicated architecture*

To support time-constrained design sessions, TeamPaint maximizes run-time performance by employing a replicated architecture (Lauwers *et al.*, 1990). It does not require a central server.

3. *Multi-layer structure*

Each user is provided with individual transparent layers or drawing surfaces. The default is that users draw on only their own layers but see a composite of all layers. Members normally use different colours to distinguish the ownership of marks. Because each layer is isolated from the others, no access control is necessary. It is also possible to share a layer, allowing other users to draw and erase the image on that layer. The usefulness of this multi-layer architecture was originally demonstrated in TeamWorkStation using a translucent video overlay technique (Ishii and Miyake, 1991). We found people respect the ownership of the drawing even in a close collaborative session, and the direct data sharing and editing function is not always required. It is, however, still a research issue (Lu and Mantei, 1991).

4. *Gesture and process awareness*

Gestures in the form of cursor movements, and through them the drawing process, are

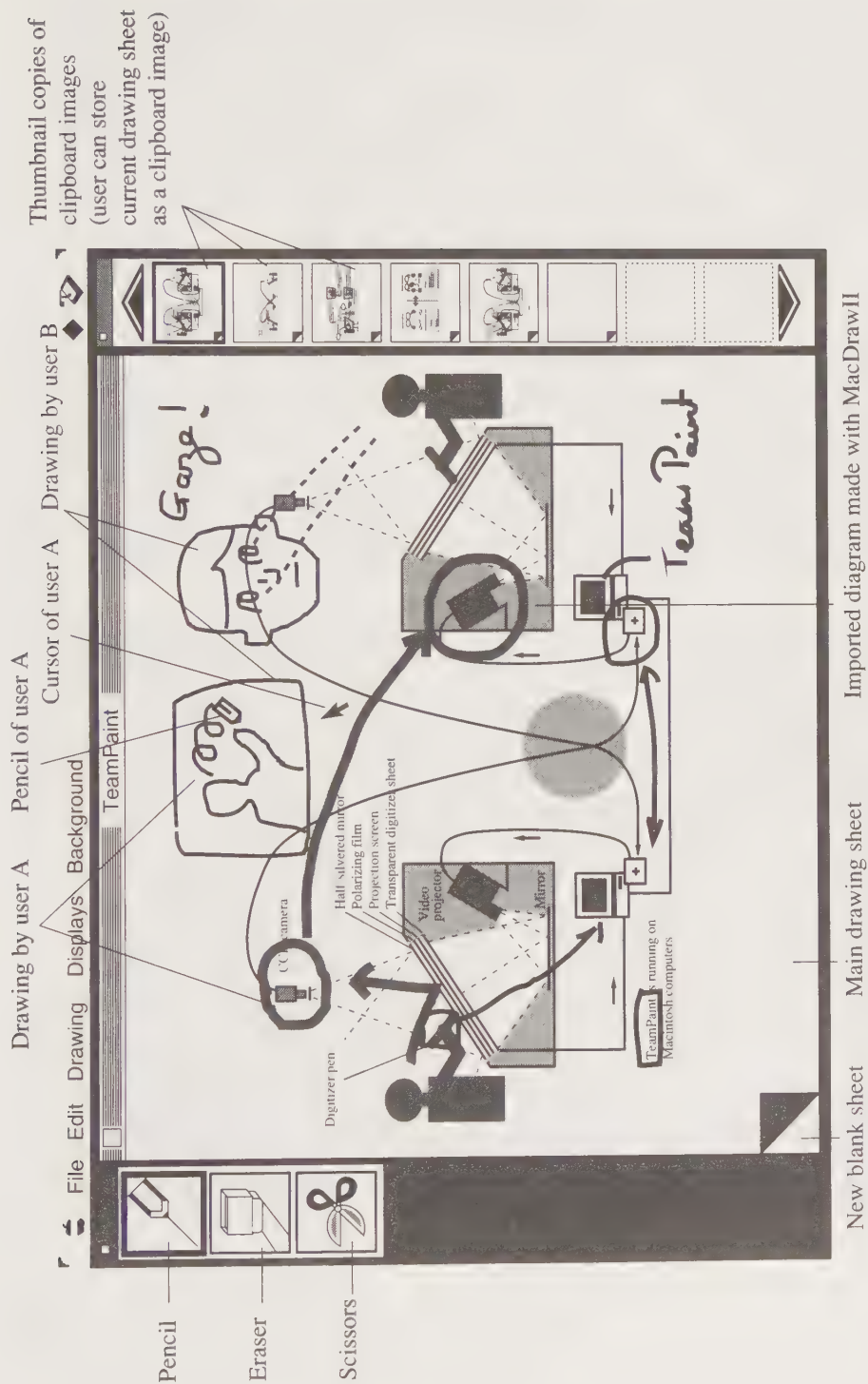


Figure 6.9 Example of a TeamPaint screen.

visually shared by all members. This feature is important in enhancing the sense of a distributed group process (Hayne *et al.*, 1993)⁷.

5. *Data exchange via standard format file*

TeamPaint can store the shared drawing results in the PICT file format that can be read by standard Macintosh programs. TeamPaint can also read PICT files created by other editors, such as MacDraw II.

6.6.3 Initial ClearBoard-2 experience

We implemented an initial ClearBoard-2 prototype in February 1992. Figure 6.10 shows snapshots of the authors using ClearBoard-2 in a design session. Its ability to record results and reuse data from previous sessions or from other application programs promises to add tremendous value to an already practical tool. The use of RGB video and the chroma-keying overlay technique do increase image clarity. Since the partner's video image is seen in the background of the TeamPaint groupware, which runs in a Macintosh full-screen window, one has the feeling of interacting with the partner through a window instead of through a big glass board. The change of screen angle (45° in ClearBoard-1 and 35° in ClearBoard-2) decreases arm fatigue and prevents users from covering the screen with their body and hindering the camera view. It creates an impression that the partner is more *under* the screen than *behind* it as in ClearBoard-1.

We found that because ClearBoard-2 provides a precise shared drawing surface through the use of TeamPaint software, its video calibration need not be as strict as that required by ClearBoard-1. It was often observed that the user's gaze follows the partner's pen movements. We confirmed that ClearBoard-2 supports *gaze awareness* as well as ClearBoard-1 did. A user can know what object in the TeamPaint screen the partner is looking at. The most serious problem of the current prototype is that drawing sometimes halted because of the poor sensitivity of the switch at the digitizer pen tip. Users complained about the situation using the expression 'out of ink'.

6.7 AN ALTERNATIVE ARCHITECTURE BASED ON A LIQUID CRYSTAL SCREEN

The half-silvered mirror and polarizing film still block the projected light and darken the ClearBoard-2 display. Moreover, overhead light sources are reflected by the half-silvered mirror, which degrades the visibility of the projected image. Fully overcoming this problem and increasing the robustness against the lighting conditions will require us to abandon the drafter-mirror architecture. To address this, we are designing a new architecture based on a liquid crystal screen (LCS) that can be rapidly switched between transparent and light scattering states by the application of a suitable control voltage. Figure 6.11 illustrates the new system architecture.

⁷ Awareness based on such a tele-pointer may have limitations. Actual hand gestures have much more power of expression, and with ClearBoard-2, the real hand and pen gesture images that lie behind the tele-pointer augment the awareness provided by TeamPaint. This video-augmented computer-drawing technique was originally demonstrated in VideoCom, presented by Minneman and Bly at CHI '91. In VideoCom, however, the hand image and face image are separated, and presented in different displays.



Figure 6.10 ClearBoard-2 in use.

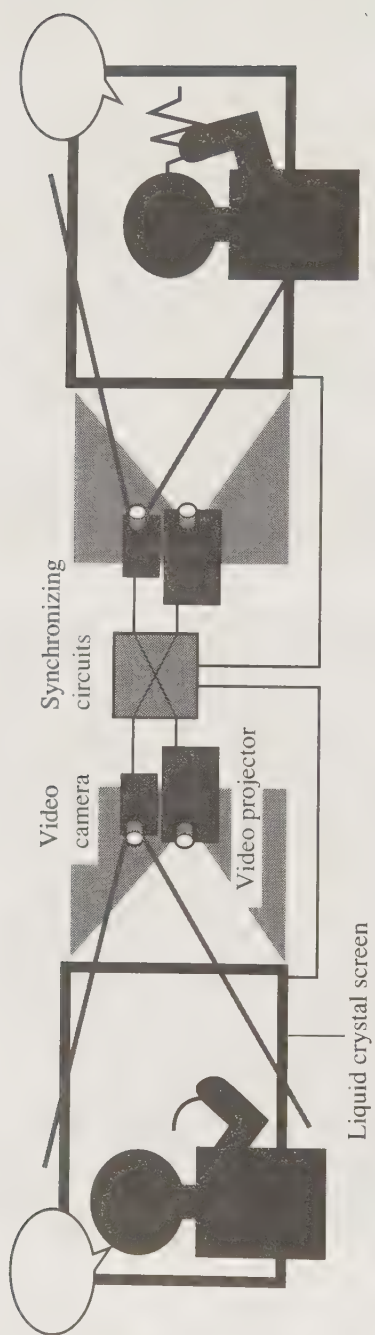


Figure 6.11 Liquid crystal screen architecture.

The architecture works by switching between the (1) light-scattering (opaque) and (2) transparent states. In state (1), the screen works as a rear projection screen on which the image of a co-worker and his or her drawing is displayed. In state (2), the user's image is captured by a video camera behind the now-transparent screen. The timing of image capture and image display is synchronized to the cycling of the LCS states. This technique was originally a means of enabling eye contact (Shiwa and Ishibashi, 1991). By allowing users to draw directly on the screen, this architecture can be extended to implement the ClearBoard concept.

Today, this LCS architecture is limited by an undesirable *flickering* that increases with the screen size; the transition frequency of the liquid crystal screen depends on its size. The expected evolution of LCS technologies should overcome this problem and realize the ClearBoard concept at a reasonable performance to price ratio in the near future.

6.8 EMPIRICAL RESEARCH USING CLEARBOARD PROTOTYPES

We have used the ClearBoard prototypes in work or work-like situations and in tasks constructed to explore certain aspects of the technology. In some cases we ourselves have been the users, while in others the users were people not involved in the development of the technology. This section outlines some of the purposes of these studies and gives examples of the results.

One purpose of our experiments was to obtain a quick first impression of the usability of the prototypes and of any obvious technical or behavioural problems. Many of the ClearBoard-1 problems and resulting ClearBoard-2 requirements described earlier were discovered through such use. A second purpose was to explore in more detail the way in which users react to the new aspects and capabilities of the technology. More careful study is needed to see how people react to overlaid images and to remote collaboration with eye contact and awareness of gaze direction, for example. A third purpose was to gain increased understanding of how people work together and how technology might ideally support such work. There are basic research questions that must be addressed before we can understand how our technologies affect or could affect collaboration.

6.8.1 Results from informal studies of use

The authors and six colleagues not involved in this research used the ClearBoard-1 prototype in conceptual design exercises (the limited marker resolution prevented the detail required by complicated electronic circuit diagrams, for example). We found that users easily and frequently glanced at each other's face and achieved eye-contact while conversing and while drawing. Switching focus from the drawing to the partner's face required almost no head movement. The effect seemed to be an increased feeling of intimacy and co-presence. These impressions were addressed more quantitatively in experiments described later.

Ishii and Arita found that users of ClearFace hesitated to draw over the image of the partner's face, where the partner's image was in a translucent small window that appeared to be superimposed on a larger drawing image. With ClearBoard, users seemed to see the partner as *behind* the drawing and thus were not reticent in drawing on the board *in front* of the partner. This may be attributable to the transparent glass metaphor and to the

relatively large size of the partner's image and head movements. Even with overlapping images, users did not report having trouble distinguishing drawing marks from the video background.

6.8.2 River crossing problem solving and gaze awareness

The importance of eye-contact is often discussed in the context of video communication tools (Acker and Levitt, 1987). However, we found that even more important may be the more general capability we call 'gaze awareness': the ability to monitor the direction of a partner's gaze and thus his or her focus of attention. A ClearBoard user can tell what screen objects the partner is gazing at during a conversation more easily than is possible in an ordinary meeting environment with a whiteboard.

We conducted collaborative problem-solving experiments on ClearBoard-1 using the river crossing problem (Ishii and Kobayashi, 1992).⁸

It has been shown that the success of this game depends heavily on the *points-of-view* of the players (Hutchins and Levin, 1981). It is thus advantageous for the collaborative players to know what the partner is gazing at. Figure 6.12 shows a snapshot of one such experiment. Participant A is gazing at one side of the river, and Participant B is looking at A's face to read his gaze. These experiments confirmed that it is easy for the players to tell which side of the river their partner is gazing at and that this information seems to help them understand and guide the partner.

We believe the concept of gaze awareness is more general and thus a more important concept than eye contact. Gaze awareness lets viewers know what their partners are looking at, whether it be someone's face or anything else in the shared workspace. You can tell when your partner is looking at you, and when your partner looks at an object in the shared workspace you can see what the object is. Eye contact is a special case of gaze awareness.

We think that providing gaze awareness will be an important goal of the next generation of shared drawing tools. It cannot be easily obtained in conventional meeting environments; only CSCW technology can provide it. ClearBoard enables distributed users to establish gaze awareness.

6.8.3 Backgammon instruction and gaze pattern

More extensive experiments were carried out by videotaping backgammon instruction sessions on a modified game board using ClearBoard-0, ClearBoard-1 and an ordinary table top as a control. The backgammon positions were laid out in a square as shown in the photographs from a ClearBoard-0 session in Figure 6.13. By confining the instructional activity to the periphery, we could differentiate patterns of visual attention to the workspace and to the partner, as seen in Figure 6.13. The backgammon game succeeded in engaging the subjects and motivated them to focus on the task. We observed

⁸ The river crossing problem is a puzzle in which the challenge is to devise a plan for getting the members of two groups, A and B, across a river in a boat. (Traditionally, the groups were missionaries and cannibals.) The boat can hold only two members at a time and must have one person in it to cross the river. Members of group A can never outnumber members of B on either bank. We played the puzzle by drawing the river on ClearBoard and using Post-It notes to represent people.



Figure 6.12 ClearBoard-1 used for solving the river crossing problem.



Figure 6.13 Gaze awareness in backgammon instruction with ClearBoard-0.

complex uses of gaze awareness. For example, in the left picture of Figure 6.13, the instructor continues to point to a position on the display while monitoring the gaze of the student. This enables him to verify that the student is looking at the correct position.

In one experiment a teacher instructed a student in different backgammon tactics in three settings: ClearBoard-0, ClearBoard-1 and with a table. Neither the teacher, a backgammon expert, nor the student were involved in the research effort. Each session took about 20 minutes, with the first half mainly being used for teaching rules and tactics, and the latter half spent playing a game using the knowledge acquired by the student. In the game-playing phase, both teacher and student were often absorbed in the game, and rarely looked at each other's face in any of the settings. In the teaching phase, however, we found big differences in the patterns of focus shifting.

We observed the patterns of gaze, gesture and speaking in each of the three teaching phases, and found that there was considerably more shifting of focus between shared workspace and interpersonal space in ClearBoard-0 and 1 settings than when using the table. Figure 6.14 shows a coded transcript of the patterns of conversation, gesture and gaze in sample 140 s segments in each settings. The experiment was designed to minimize random eye contact, and in finding that the pattern of eye contact varied remarkably with the task we are confident that it did not result from random glances.

In this experimental setting, the participants showed a greater incidence of eye contact and focus shift between shared workspace and interpersonal space with the ClearBoard technologies. There is a decrease with ClearBoard-1, perhaps because the partner's face is less clear than with ClearBoard-0, but the incidence is still considerably greater than the tabletop, where the separation of workspace and interpersonal space is greater. Results from a single pair of subjects are more suggestive than definitive. Although the value of monitoring a partner's gaze direction seems clear, the uses of eye contact in such settings are matters for further research. Nevertheless, this and other trials with ClearBoard have been very encouraging.

6.9 FUTURE WORK

We have described the designs and experiments of the shared drawing media ClearBoard-1 and ClearBoard-2, which seamlessly integrate shared workspace and interpersonal space. Many interesting technical and behavioural issues remain to be investigated.

6.9.1 Multi-user and multi-point ClearBoard

One interesting design question is how to extend the ClearBoard concept to support multiple users and multiple points. Although we originally designed ClearBoard-2 for a pair of users, often two or three users gathered in front of a terminal and worked together with remote users. The larger screen size of ClearBoard-2 allows multiple users to join a ClearBoard-2 session comfortably. We think the use of wider wall-sized screens will make ClearBoard-2 a practical multi-user medium with which to connect *two* distributed meeting rooms. ClearBoard-2 extensions will support the simultaneous use of multiple electronic pens at each terminal for this purpose. TeamPaint software will also be extended to support multiple inputs that correspond to multiple pens at each computer.

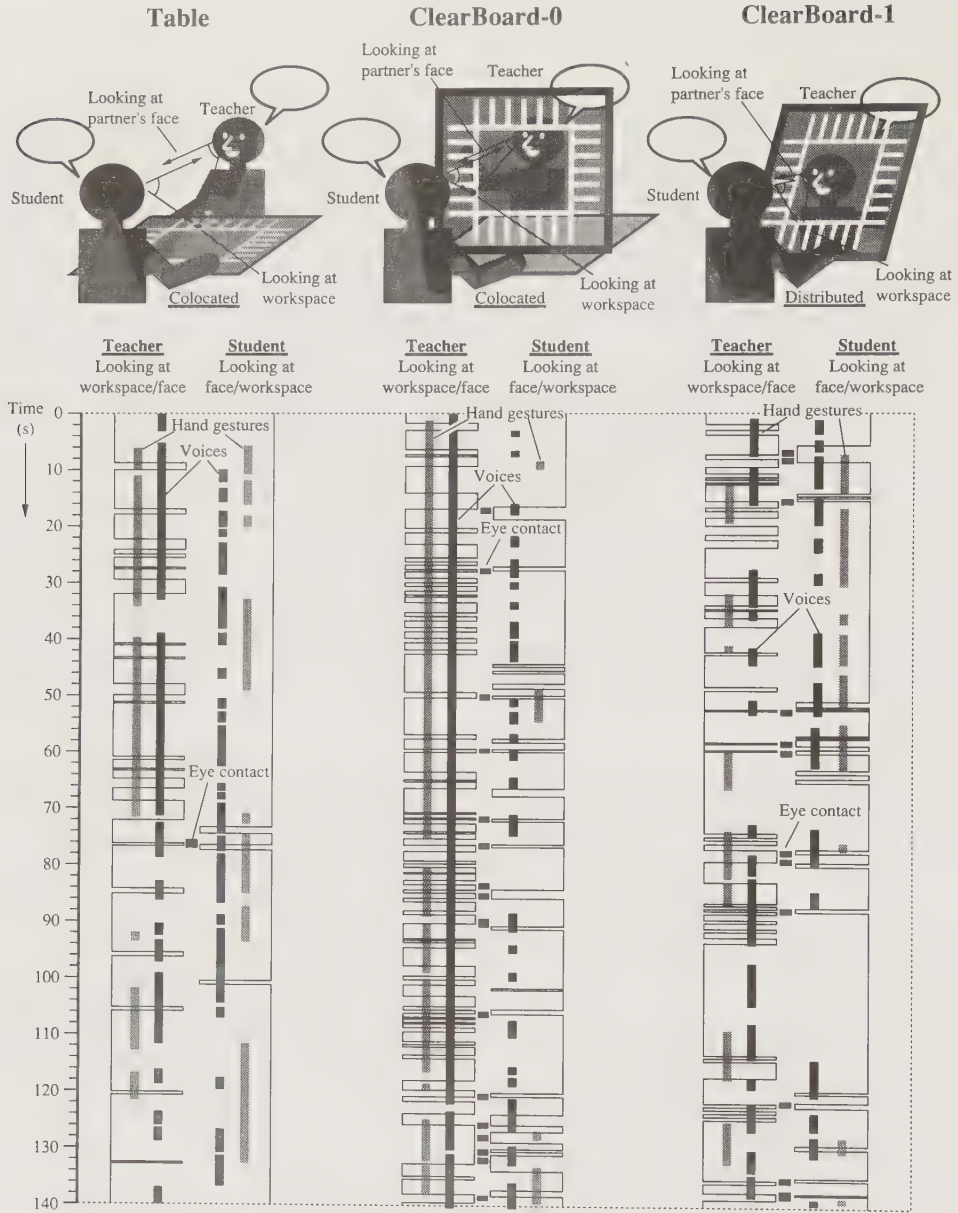


Figure 6.14 Time chart of gaze, gesture and speaking patterns in sample segments in backgammon teaching phase.

How to extend ClearBoard to connect three or more points is a more challenging question. Severe technical and usability limitations prohibit us from overlaying many more video images. The metaphor of ClearBoard is to connect two distributed spaces through a virtual glass board; to support three or more points seems to require a major change in the metaphor itself. In multi-user and multi-point situations, it is more difficult to know who is watching whom, and eye contact confusion is a potential problem. Hydra, designed by Buxton, provides an ingenious solution to this multi-user gaze problem in multi-point video conferencing, although it does not provide a shared workspace (Buxton, 1992). We are exploring ways to extend the ClearBoard metaphor seamlessly to support multiple users at more than two points without losing its simplicity and virtue of gaze awareness.

6.9.2 New display technology

Implementation of the ClearBoard prototypes convinced us that the invention of new display technologies is critical to make the ClearBoard concept widely accepted. For example, the multi-user ClearBoard-2 indicates the need for wall-sized flat display technology with the following functions:

1. to display the overlaid image of computer screen and the video at very high resolution and with enough brightness for ordinary office environments
2. to input concurrent gestures of drawing and pointing at display surfaces with multiple electronic pens
3. to capture a bright video image of users in front of the display surface

To implement the prototypes described in this paper, we used a variety of commercially available components, such as half-silvered mirrors, polarized films, transparent digitizer sheets, video overlay boards, video projectors and video cameras. We manually integrated them into a system based on the new architecture we had invented. However, the limitation of this approach is apparent. We feel it is necessary to start from the electronic engineering design of a new device that satisfies the display requirements just described.

Bitmap displays played a crucial role in the development of GUIs (graphical user interfaces); head-mounted displays did the same for VR (virtual reality). We expect ClearBoard to provide a new goal for display technologies that may change our concept of a wall from being a passive partition to being a dynamic collaboration medium.

6.9.3 Interpersonal distance

Interpersonal distance is an interesting issue in the ClearBoard design. Edward T. Hall studied interpersonal distance and developed the categories of *intimate distance* (0–18 in or 0–46 cm), *personal distance* (1.5–4 ft, 46–122 cm), *social distance* (4–12 ft, 1.22–3.66 m) and *public distance* (more than 12 ft, 3.66 m) (Hall, 1966).

ClearBoard creates the impression of participants standing about *one metre* apart, because both sit (or stand) close enough to the screen to draw directly on its surface. This virtual distance belongs to the *personal distance* category in Hall's classification. When people use ClearBoard with close friends or colleagues, this distance seems appropriate. However, for a formal meeting with a person of much higher rank, this virtual interpersonal distance might seem too small and the participants might be uncomfortable.

Therefore, we would like the media to provide users with some control over the virtual interpersonal distance. We are planning to provide an option of indirect drawing using a wireless tablet or pen-based personal computer for that purpose.

6.9.4 Studies of visual behaviour

Interpersonal distance is one example of behavioural issues that can be explored using ClearBoard technology and that will be important in governing how it is received and used. We have noted that awareness of gaze direction is potentially very important. This has clear uses in identifying the context for spoken remarks and gestures. More ambiguous is the role of direct eye contact. ClearBoard makes eye contact easy to establish and may even make it more difficult to avoid. It has been shown that the use of eye contact varies with culture (e.g. Argyle, 1975); these are issues for further exploration in ClearBoard settings.

Another important behavioural issue is that of visual attention in the context of overlaid images. For example, it has been shown that while people can selectively attend to one of two overlaid visual images, they may have trouble monitoring two unrelated images even when they try to do so (Neisser and Becklen, 1975). ClearBoard presents two images in the context of an overarching task; how that will affect behaviour remains to be explored. Depth is another variable to be considered. In ClearBoard-0, the shared workspace and the partner were at different focal planes. The subsequent ClearBoard prototypes place both images on the same focal plane.⁹

We are planning to study visual behaviour further in the context of overlaid images. Existing work on the roles of visual attention and eye contact indicate the need for further research into the effects of facilitated eye contact and gaze awareness provided by these technologies.

6.10 CONCLUSION

We have described the design of the novel collaboration media ClearBoard-1 and ClearBoard-2. These media seamlessly integrate a shared workspace and interpersonal space, and transitions of focus between the spaces are smooth. ClearBoard is based on the metaphor of talking through and drawing on a transparent glass window, while providing a common right-left orientation.

Our studies suggest that ClearBoard realizes *gaze awareness*, components of which include eye contact and monitoring the partner's direction of gaze. Gaze awareness may be crucial to the next generation of shared drawing media and is a potentially useful capability that CSCW technology can greatly enhance. ClearBoard offers a means to further explore the role of eye contact and gaze awareness in collaborative work, by making them possible in new situations that are amenable to study. The use of gaze

⁹ Kobayashi and Ishii are developing a new display technology called 'Displayers' that can place multiple video images on different focal planes using multiple crystal screens (Kobayashi and Ishii, 1992). A user can see the composite of all video images at different distances. We plan to use the Displayers technology to place the drawing image and the partner's image on different focal planes, and to study the effect of selective attention in a ClearBoard environment.

awareness is reported by athletes, teachers and others, who may position themselves to take advantage of it. How extensive its use is, and when eye contact might be disruptive, are topics for exploration. Our observation that eye contact occurred more often in instruction than in competition suggests that there will be differences and also suggests that ClearBoard users can avoid eye contact when it is not needed. Our users did not report disruptive eye contact, but in some situations it could be an issue.

We see the evolution from ClearBoard-1 to ClearBoard-2 as being very important. Computer and video communication technologies have, until now, evolved independently. Although they have been loosely coupled using arbitrary multi-window interfaces in many desktop multimedia conferencing systems, they have never been integrated seamlessly from the users' cognitive point of view. We feel ClearBoard-2 is the first system that fully succeeds in naturally integrating the technology of computer-based groupware with that of video conferencing. We expect that the seamless integration of computer and video communication technologies will realize the next generation of collaboration media.

Moreover, ClearBoard-2 can be seen as one instance of the paradigm shift from traditional HCI (human-computer interaction) to HHI (human-human interaction mediated by computers). We are interacting not *with* computers, but *through* computers. ClearBoard design is not only 'beyond being there' (Hollan and Stornetta, 1992) but also a step beyond the traditional desktop metaphor based on a multi-window interface. We expect ClearBoard to be useful both as a collaboration medium and as a vehicle to investigate the nature of dynamic human interaction.

ACKNOWLEDGEMENTS

William Buxton at the University of Toronto must be mentioned for his stimulating discussions with Ishii on the shared workspace/interpersonal space framework. We thank Naomi Miyake at Chukyo University for her insightful comments on the role of gaze and her participation in the river crossing experiments on ClearBoard-1. We thank Saul Greenberg at University of Calgary for his detailed comments to a draft of this paper. Thanks are also due to A. Sasaki and Y. Hanamoto for their collaboration in developing TeamPaint software, and to Takaya Endo and Takahiko Kamae for supporting this research.

REFERENCES

- Acker, S. and Levitt, S. (1987) Designing videoconference facilities for improved eye contact, *Journal of Broadcasting and Electronic Media*, **31** (2) pp. 181-91.
- Argyle, M. (1975) *Bodily Communication*, Methuen & Co. Ltd., London.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, *Proceedings of COIS '90*, ACM Press, New York, pp. 184-92.
- Bly, S. A., Harrison, S. R. and Irwin, S. (1993) Media spaces: video, audio and computing, *Communications of the ACM*, **36** (1) pp. 28-47.
- Buxton, W. (1992) Telepresence: integrating shared task and person spaces, in *Proceedings of Graphics Interface '92*, Morgan Kaufmann Publishing, Los Altos, pp. 123-9.
- Buxton, W. and Moran, T. (1990) EuroPARC's integrated interactive intermedia facility (IIIF): early experiences, in *Proceedings of the IFIP WG8.4 Conference on Multi-User Interfaces and Applications*, North-Holland, Amsterdam, pp. 11-34.

- Clark, S. and Scrivener, S. (1992) The ROCOCO sketchpad distributed shared drawing surface, *LUTCHI Report No. 92/C/LUTCHI/0150*, LUTCHI Research Centre, Loughborough University of Technology, UK.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, *Communications of the ACM*, **34** (1) pp. 39–58.
- Farallon Computing Inc. (1991) *Timbuktu 4.0 User's Guide*, Farallon Computing Inc., Emeryville, CA.
- Fish, R. S., Kraut, R. E. and Chalfonte, B. L. (1990) The VideoWindow system in informal communications, in *Proceedings of CSCW '90*, ACM Press, New York, pp. 1–11.
- Fish, R. S., Kraut, R. E., Root, R. W. and Rice, R. E. (1993) Video as a technology for informal communication, *Communications of the ACM*, **36** (1) pp. 48–61.
- Foster, G. and Stefik, M. (1986) Cognoter, theory and practice of a collaborative tool, *Proceedings of CSCW '86*, ACM Press, New York, pp. 7–15.
- Gaver, W. (1992) The affordance of media spaces for collaboration, in *Proceedings of CSCW '92*, ACM Press, New York, pp. 17–24.
- Gaver, W., Smith, R. and O'Shea, T. (1991) Effective sounds in complex systems: the ARKola simulation, in *Proceedings of CHI '91*, ACM Press, New York, pp. 85–90.
- Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992) Issues and experiences designing and implementing two group drawing tools, *Proceedings of HICSS '92*, IEEE Computer Society, Los Alamitos, CA, pp. 139–50.
- Group Technologies, Inc. (1990) Aspects: the first simultaneous conference software for the Macintosh, *Aspects User's Manual*, Group Technologies, Inc., Arlington, VA.
- Hall, E. T. (1966) *The Hidden Dimension*, Doubleday & Company, New York.
- Hayne, S., Pendergast, M. and Greenberg, S. (1993) Gesturing through cursors: implementing multiple pointers in group support systems, in *Proceedings of HICSS '93*, IV, IEEE Computer Society, Los Alamitos, CA, pp. 4–12.
- Hollan, J. and Stornetta, S. (1992) Beyond Being There, in *Proceedings of CHI '92*, ACM Press, New York, pp. 119–25.
- Hutchins, E. L. and Levin, J. A. (1981) Point of view in problem solving, *CHIP Technical Report No. 105*, University of California at San Diego.
- Ishii, H. (1990) TeamWorkStation: towards a seamless shared workspace, in *Proceedings of CSCW '90*, ACM Press, New York, pp. 13–26.
- Ishii, H. and Arita, K. (1991) ClearFace: translucent multiuser interface for TeamWorkStation, in *Proceedings of ECSCW '91*, Kluwer Academic Press, Dordrecht, pp. 163–74.
- Ishii, H. and Kobayashi, M. (1992) ClearBoard: a seamless medium for shared drawing and conversation with eye contact, in *Proceedings of CHI '92*, ACM Press, New York, pp. 525–32.
- Ishii, H. and Miyake, N. (1991) Toward and open shared workspace: computer and video fusion approach of TeamWorkStation, *Communications of the ACM*, **34** (12) pp. 37–50.
- Ishii, H., Arita, K. and Kobayashi, M. (1992b) Towards seamless collaboration media: from TeamWorkStation to ClearBoard, in *SIGGRAPH Video Review*, CSCW '92 Technical Video Program, Issue 87, Item 6, ACM Press, New York.
- Ishii, H., Arita, K. and Yagi, T. (1993) Beyond videophones: TeamWorkStation-2 for narrowband ISDN, in *Proceedings of ECSCW '93*, Kluwer Academic Press, Dordrecht.
- Ishii, H., Kobayashi, M. and Grudin, J. (1992a) Integration of interpersonal space and shared workspace: ClearBoard design and experiments, in *Proceedings of CSCW '92*, ACM Press, New York, pp. 33–42.
- Kobayashi, M. and Ishii, H. (1992) DisPLayers: multi-layer display technique to enhance selective looking of overlaid images, in *Posters and Short Talks of CHI '92*, ACM Press, New York, p. 66.
- 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 COIS '90*, ACM Press, New York, pp. 249–60.
- Lu, I. and Mantei, M. (1991) Idea management in a shared drawing tool, in *Proceedings of ECSCW '91*, Kluwer Academic Press, Dordrecht, pp. 97–112.
- Mantei, M., Baecher, R., Sellen, A., Buxton, W. and Milligan, T. (1991) Experiences in the use of a media space, in *Proceedings of CHI '91*, ACM Press, New York, pp. 203–8.
- Minneman, S. L. and Bly, S. A. (1991) Managing a trois: a study of a multi-user drawing tool in distributed design work, in *Proceedings of CHI '91*, ACM Press, New York, pp. 217–24.
- Neisser, U. and Becklen, R. (1975) Selective looking: attending to visually specified events, *Cognitive Psychology*, **7**, pp. 480–94.

- Olson, J. S., Olson, G. M., Storrosten, M. and Carter, M. (1993) Group work close up: a comparison of the group design process with and without a simple group editor, *ACM Transactions on Information Systems*, **11** (4) pp. 321–48.
- Pedersen, E., McCall, K., Moran, T. and Halasz, F. (1993) Tivoli: an electronic whiteboard for informal workgroup meetings, in *Proceedings of INTERCHI '93*, IFIP and ACM, pp. 391–8.
- Shiwa, S. and Ishibashi, M. (1991) A large-screen visual telecommunication device enabling eye contact, in *Digest of technical papers of Society for Information Display International Symposium 1991* (SID '91), pp. 327–8.
- Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuler, W., Schuett, H. and Thuring, M. (1992) SEPIA: a cooperative hypermedia environment, in *Proceedings of the Fourth international ACM Conference on Hypertext (EHT '92)*, Milan, 30 November–4 December, ACM Press, New York, pp. 11–22.
- Tang, J. C. and Minneman, S. L. (1991a) VideoDraw: a video interface for collaborative drawing, *ACM Transactions on Information Systems*, **9** (2) pp. 170–84.
- Tang, J. C. and Minneman, S. L. (1991b) VideoWhiteBoard: video shadows to support remote collaboration, in *Proceedings of CHI '91*, ACM Press, New York, pp. 315–22.
- Tanigawa, H., Arikawa, T., Masaki, S. and Shimamura, K. (1991) Personal multimedia-multipoint teleconference systems, in *Proceedings of INFOCOM '91*, IEEE Communications Society, pp. 1127–34.
- Watabe, K., Sakata, S., Maeno, K., Fukuoka, H. and Ohmori, T. (1990) Distributed multiparty desktop conferencing system: MERMAID, in *Proceedings of CSCW '90*, ACM Press, New York, pp. 27–38.
- Wolf, C. and Rhyne, J. (1992) Communications and information retrieval with a pen-based meeting support tool, in *Proceedings of CSCW '92*, ACM Press, New York, pp. 322–9.

SECTION TWO

Systems, Architectures and Toolkits for Groupware

Section 1 provided many insights, observations and guiding principles on how people draw together. We also saw visions of many innovative groupware systems and concepts that extend the drawing process onto new media. However, we still need to understand the technical nature of these media.

The chapters in this section concentrate on the technical aspects of groupware. They illustrate how particular systems can be constructed, explain specific architectures for groupware, and generalize concepts into toolkits that allow rapid construction of multi-user applications.

The section begins with Brian Wilson's explanation of the architecture of WScrawl 2.0 (Chapter 7), a shared whiteboard whose implementation takes advantage of the networking capabilities of X-Windows. Although shared whiteboards can be implemented in many ways, WScrawl is notable because it was developed without custom communication protocols and uses only standard, widely available libraries (C, Motif and X). The result is a program that is only slightly more complex than a single-user program with similar functionality. Wilson also argues that much of the simplicity of WScrawl comes from adopting a centralized architecture over a replicated one.

The idea of a groupware toolkit is tackled head-on by Roseman and Greenberg in Chapter 8. They describe some basic necessities of a toolkit from both a programmer's and an end-user's viewpoint. A programmer, for example, worries about management of multiple processes, interprocess communication, conference set-up and tear-down, how graphical objects behave as data structures, and so on. A user, on the other hand, is more interested in how the groupware supports individual and group needs. Concerns here include how one can gesture and annotate over the work surface, how floor control is handled, how one enters and leaves a conference, and how other work can be imported into a group meeting. Roseman and Greenberg argue that a usable groupware toolkit must make it easy for programmer to handle not only the technical requirements, but to include features desired by users as well. GroupKit, a groupware toolkit using a replicated architecture, is described to show how this can be done in practice. A run-time architecture supports processes and communication; transparent overlays offer a convenient method for adding gesturing and annotation to groupware; and open protocols allow a designer to create a wide range of interface and interaction policies on top of a single system.

Karsenty and Beaudouin-Lafon continue the discussion of a toolkit at a slightly different level (Chapter 9). They introduce the notion of shared manipulation, an extension of Shneiderman's direct manipulation paradigm that includes sharing objects of interest and providing awareness of the actions of the group. They incorporate the principles of shared manipulation in the SLICE model (shared layers in cooperative editing). The architecture of SLICE divides groupware into layers, each responsible for a different function, e.g. document abstraction, its visual representation, its manipulation, its view and multiple cursors. Parts of the SLICE model are incorporated into GroupDesign – a shared editor for structured diagrams – and the authors discuss how the model is implemented in practice. As it is a fully replicated system, particular attention is paid to currency control and the issues that arise from it.

WSCRAWL 2.0: A SHARED WHITEBOARD BASED ON X-WINDOWS

Brian Wilson

ABSTRACT

WScrawl 2.0 is a shared whiteboard based on the networking capabilities of X-Windows. This paper describes the simple, fast, clear architecture of WScrawl 2.0 using Motif 1.1 and X11R4. The simplicity of this design allows for a powerful shared whiteboard to be constructed. Directions for future work are proposed.

7.1 INTRODUCTION

WScrawl 2.0 is a shared whiteboard based on Motif 1.1 and X-Windows that allows any number of users to cooperate in creating a single drawing. It can be thought of as a colour paint program that runs on several users' displays at once. If one participant draws something, all participants in the session see the change to the drawing instantly. WScrawl 2.0 is the successor to the original WScrawl (version 1.0) which was first made publicly available in 1989 and has been distributed and used widely in the last three years.

There are a growing number of applications like WScrawl called real-time or 'synchronous' cooperative tools. Researchers in the field of computer supported cooperative work (CSCW) are examining how to build these tools efficiently, and more importantly how these tools can be of use to end users. WScrawl 2.0 is unique among these tools in that it is built entirely on the network capabilities of X-Windows and Motif 1.1. By exclusively using X-Windows (sometimes referred to simply as 'X'), it is not necessary to create custom communication protocols to achieve cooperative applications such as

WScrawl. As this paper will show, cooperative applications are straightforward to design and implement in this manner using the common tools of X, Motif and the 'C' programming language. Furthermore, because there are implementations of X on virtually every type of personal computer and workstation sold today, WScrawl is extremely portable and in fact has been compiled and run on many platforms including Apple Macintosh, Sun SparcStation, HP 700, IBM RS 6000, Silicon Graphics Indigo and others.

This implementation of WScrawl 2.0 proves that real-time cooperative tools can be built with more than adequate performance on today's hardware, communicating across today's networks, and implemented with today's widely available standard graphical toolkits that give excellent support for distributed graphics.

In the rest of this paper, I describe the philosophy, the design, and the implementation of WScrawl 2.0 in depth. Source code to WScrawl 2.0 is free and available via anonymous ftp from the machine 'sax.stanford.edu'.

7.2 RELATED WORK

Computer systems that recognize and deal with collaborating users are labelled 'group aware'. In this section, I will only address previous work relating to 'real-time' or 'synchronous' distributed group systems in which the collaborating users are physically separated yet using the system at the same point in time. Also, I will limit the discussion to work that relates to shared drawing surfaces.

7.2.1 Cooperative custom hardware systems

Several interesting experimental cooperative systems have been built with custom hardware. These systems tend to be 'proof of concept' systems using communication bandwidths not widely available in the commercial world, or perhaps using new input devices that are not in wide use. Hopefully, as networks get faster or new commercial computers are designed and built, the concepts studied in these systems will find their way to end users. For example, as far back as the 1960s in the NLS project, users could link two terminal screens together for real-time conferencing, but it was years before networks made linking terminals together realistic outside of research labs (Engelbart and Lehtman, 1988).

The CoLab meeting room project at Xerox PARC (Stefik *et al.*, 1987) defined the concepts of 'What You See Is What I See' (WYSIWIS) and the 'shared TelePointer'. A shared TelePointer is a cursor controlled by one user that appears on all of the collaborating users' screens. The controlling user points at objects on the screen, usually providing a verbal explanation at the same time. Also developed at Xerox PARC was Commune, a shared drawing surface in conjunction with full-duplex audio and video connections (Minneman and Bly, 1991).

In the VideoWhiteBoard (Tang and Minneman, 1991), cameras show the user's hand motions in addition to the actual drawing. An improvement on this idea is Ishii's ClearBoard where cameras also capture the user's facial expressions by essentially providing a piece of virtual glass that the two users are standing on opposite sides of (Ishii and Kobayashi, 1992).

Many software systems have been augmented with custom hardware systems to provide some additional functionality not possibly available through software. At GMD-IPSI, WScrawl was linked with custom video conferencing equipment to examine the use of audio and video in conjunction with shared drawing (Streitz *et al.*, 1991). Users could select other users out of a menu in WScrawl and get a desk-to-desk teleconference and shared whiteboard.

7.2.2 Cooperative software systems

In the past few years there has been a significant rise in synchronous groupware projects based partially or even entirely on custom software written for existing hardware. This is encouraged by a recent emphasis in computer fields for standards and facilitated by pervasive networking and powerful yet relatively inexpensive workstations.

Some of these systems are fairly general, such as Timbuktu (Farallon, 1988) and Hewlett-Packard's SharedX product (Lauwers, 1990). SharedX allows users to run any existing X-Windows application on several workstations at once. SharedX duplicates every drawing request to each workstation in the session so that a carbon copy of the application appears on each display. SharedX then multiplexes the input from all the participating displays down to the one application. For example, users could run the commercial product Island Paint (a fairly standard commercial paint program that runs in X-Windows) under SharedX to get a limited shared whiteboard. SharedX shows a copy of the drawing on several workstation displays at once, and any user may draw since SharedX multiplexes their input down to the one copy of Island Paint. Notice that under this system only one user may draw at one time.

The advantages of the above system are that through SharedX, all of your existing programs can be synchronous groupware. The disadvantage is that the applications themselves are not group-aware and therefore at best will support primitive cooperative work and at worst will crash or hang because they receive input in a way they were not designed to handle.

A plethora of fairly specific group-aware systems have been written, such as Bellcore's Conversation Board (Brinck and Gomez, 1992), CaveDraw (Lu and Mantei, 1991), Grove (Ellis *et al.*, 1989) and XSketch (Lee, 1990). These are all shared drawing or editing programs which have their merits of one sort or another, but seem to suffer from a severe lack of functionality, bad performance, a hard to use interface, or some combination of these problems. Some of them, like XSketch and GROVE, have gone to the considerable effort of creating their own custom communication protocols, and others, such as the Conversation Board, are built on custom libraries providing distributed system support.

Grudin (1988) points out that if a CSCW system requires more effort to use than a single-user system, users will likely reject it. I would expand on this statement saying that if the shared system is sluggish to use or limited in functionality, users will also reject it. In WScrawl, I have created a high-performance, easy to use, shared whiteboard rich with functionality developed without custom communication protocols and using only standard, widely available libraries.

7.3 AN EXAMPLE WSCRAWL SESSION

Figure 7.1 shows what one user of WScrawl might see at some point during a WScrawl session. As you can see, WScrawl appears to the user like a standard drawing program. There are several pull-down menus along a menu bar labelled 'File', 'Edit', 'Options', 'Colors', 'Fonts' and 'Help'. A tool palette on the left of the main window allows the user to pick various tools such as a paintbrush or an eraser. A large shared drawing area is viewed through a scrolled window. In this WScrawl session, two users named 'Brian' and 'Dave' are playing Tic-Tac-Toe (or noughts and crosses) (the current members of the wscrawl session are listed in the 'STATUS' window directly beneath the menu bar).

WScrawl is rich with the functionality one might expect in a regular (non-shared) paint program. In Figure 7.1, we see a Fatbits Editor for detailed work, a Color Picker, and a small control panel for configuring some pointer settings. WScrawl can save images in several formats, has extensive online help, can import arbitrary regions of the screen outside the WScrawl window through a friendly drag and drop interface, can type in many fonts, paint using patterns, and more.

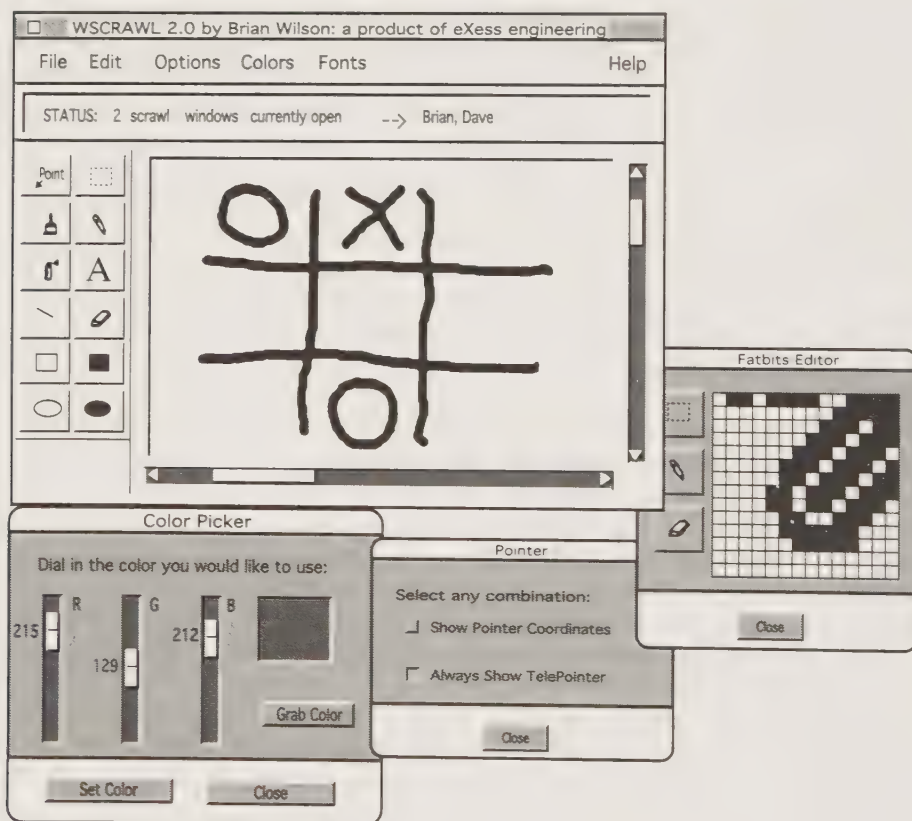


Figure 7.1 One user's view of a WScrawl session.

7.4 USER INTERFACE DESIGN PHILOSOPHY

In WScrawl, I have strived to create a responsive, interactive, fully functional paint program that people will want to use precisely because it is a good paint program. This paint program has one extra menu option: 'Add Display', which allows several people to edit the picture simultaneously. Although the shared capabilities of WScrawl took effort to design and implement, the illusion I wish to offer the user is one of effortless and natural transition from isolated editing to shared editing (Haake and Wilson, 1992). The commands should not change, the user should not be prevented from doing anything he or she could do in the isolated editing case, and most important of all, the response time of the paint program should not change significantly.

There are two components of WScrawl presented to any one user: the 'user-interface' component (made up of the pull-down menus, the tool palette, the user's current colour and tool, etc.), and the 'shared drawing board' component.

The user-interface component is entirely controlled by, and private to, the one user. For example, each user can choose which language he or she wishes the menus and dialogs to be in. (English and German are currently supported.) In our example in Figure 7.1, the WScrawl we see has the user-interface in English, but there is no way for this user to control what the other user's interface language is in, or even know if the other user has their interface in English or German. Also, if user 'Brian' is currently browsing the menus or choosing the paintbrush tool, user 'Dave' can be drawing with a pencil the entire time, oblivious to what user 'Brian' is doing.

In contrast with the user interface component, the shared drawing board component always appears identical to all members of the WScrawl session. If user 'Brian' sketches a red circle in the drawing area, both user 'Brian' and user 'Dave' instantly see each pixel appear as it is drawn. The drawing area is conceptually one surface that is viewed by several users at once (even though it is implemented with separate copies of the drawing surface at each display).

In certain other shared drawing systems, such as running Island Paint under SharedX, users must take turns drawing. For example, while user 'Brian' is selecting a paintbrush, user 'Dave' can do nothing. Furthermore, when this action is completed, both user 'Brian' and user 'Dave' have a paintbrush selected as their current tool. Because WScrawl is group-aware, user 'Brian' could be drawing a red circle at the exact same time as user 'Dave' is erasing an area of the screen, with all changes to the drawing appearing to both users in real-time.

For a discussion of how people might use WScrawl, Whittaker *et al.* (1990) studied users collaborating with WScrawl 1.0.

7.5 IMPLEMENTATION

This section describes WScrawl's basic architecture, the use of Motif as a distributed application toolkit, the functionality WScrawl currently has, and finally some short-term future directions for WScrawl. In this section, the term 'instance' of a WScrawl interface refers to the WScrawl windows that are presented to a single participant of a WScrawl session and all of the internal code to support that participant's interface. However, as mentioned before, WScrawl is a single Unix process.

7.5.1 X-Windows as a distributed architecture

WScrawl uses the network capabilities built into the X-Windows system. To understand the basic architecture of WScrawl, you must first understand the basic architecture of a typical X-Windows client. Figure 7.2 shows a typical X-client running on CPU 1 and displaying on Display 3.

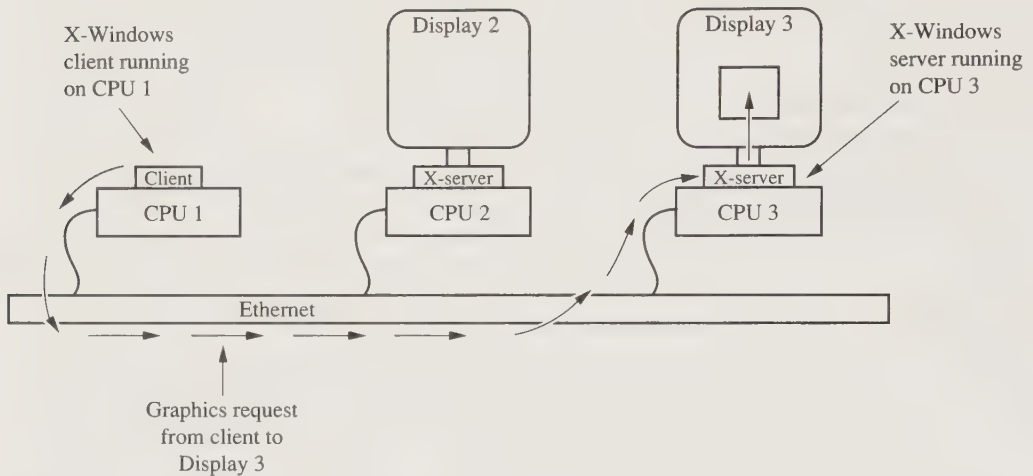


Figure 7.2 Typical X-Windows program.

The concept of any X-Windows client is that it executes on a CPU somewhere on the network (in this case CPU 1) and takes input from and displays graphics on a display running an X-server somewhere on the network (in this case Display 3). At start-up time, the client executes the Xlib (X-Library) call `XOpenDisplay` to specify which display on the network the client wishes to display graphics on. In our example in Figure 7.2, the client executes the pseudo-X code:

```
main()
{
    Display *disp;                                /* declare vars */

    disp = XOpenDisplay ("Display3");              /* open display 3 */
    XDrawPoint (disp, 20, 100);                   /* draw a point */
    XCloseDisplay (disp);                          /* close display */
}
```

The most important items to notice are that in this typical X-client, the `Display3` was opened by name, and that in every graphics call the `disp` variable was the first argument.

Now that we know how an X-client works, WScrawl is just an X-client that executes more than one `XOpenDisplay` call. It is a single Unix process, just like any other

X-client. It draws to displays across the network just like any other X-client, except that it draws to more than one. Figure 7.3 shows a WScrawl session where WScrawl is running on CPU 1 and has windows open on Display 2 and Display 3.

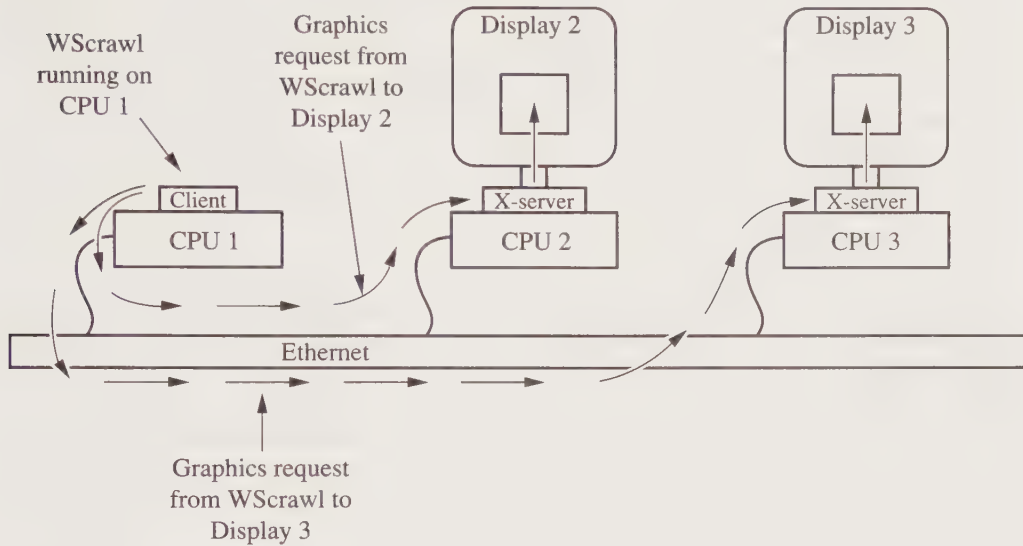


Figure 7.3 WScrawl session.

WScrawl takes input from and displays graphics on several displays at once. I refer to this as having several ‘instances’ of the WScrawl interface. Any X-Windows client *could* do this, but most choose not to because in most cases it is not useful. The WScrawl in Figure 7.3 executes the pseudo-X code:

```
main()
{
    int i, NUM_DISPS = 2;
    Display *disp[2];                /* declare vars */

    disp[0] = XOpenDisplay("Display2"); /* open display 2 */
    disp[1] = XOpenDisplay("Display3"); /* open display 3 */

    for (i=0; i < NUM_DISPS; i++)
        XDrawPoint(disp[i], 20, 100); /* draw a point */

    XCloseDisplay(disp[0]);           /* close display 2 */
    XCloseDisplay(disp[1]);           /* close display 3 */
}
```

The identical Xlib calls are used in the WScrawl case as in the normal client case. In other words, no extra communications protocol is specified at all. Also notice that except for the set-up and tear-down of the WScrawl session, the code is the same for any number of users. The loop is generic in that it works for one or more users, and in fact the source code to WScrawl is filled with loops that look precisely like the one presented in the pseudo-code above. Therefore, X-Windows contains all the network capability necessary to write distributed applications, and best of all, X is the standard window system for Unix! (The X-Windows libraries are implemented on top of TCP/IP with TCP_NODELAY set, but X insulates the programmer entirely from the implementation, so this is just an interesting technical fact that is not relevant to WScrawl.)

In the above example, WScrawl outputs graphics to multiple displays, but did not accept input. When an input event such as a mouse button press is received, it contains the information of which display it occurred on. The decoding of this information is covered in more depth in Section 7.5.2.

Finally, the above example demonstrates that WScrawl is not truly multi-tasking. Instead, WScrawl gives the illusion of being multi-tasking by accepting input from one user, taking the appropriate actions, and then accepting input from the next user and taking the appropriate actions, and so on in a round-robin fashion. On all but the very slowest CPUs and displays, WScrawl performs quite adequately, presenting the illusion of all users having their own drawing programs running on their own CPUs.

7.5.2 Motif as a distributed application toolkit

WScrawl was developed with the Motif 1.1 toolkit, which itself sits on top of the Xt Intrinsics from MIT. Development time was considerably shortened by using a mature toolkit like Motif, and although there is nothing specific about other toolkits that prevented their use, Motif is the currently touted 'standard' X-Windows toolkit, which makes WScrawl more attractive for certain groups of users. There are also several sophisticated layout tools for creating Motif applications which makes Motif attractive.

When WScrawl is first run, one interface instance is created on the invoking user's display. At that point, the first user can use WScrawl as a single-person paint program or add another user through the Add Display menu item in the File menu. In Xt Intrinsics terms, a WScrawl session has a single application context for all interface instances, but it creates one `applicationShellWidgetClass` widget for each instance as that instance's top level widget. Using this scheme, Motif and Xt worked exceptionally well with little custom code to support multiple displays.

After the interface instances are created, WScrawl waits for user input. The Motif libraries provide the application programmer with the concept of 'callback' functions. At initialization time, WScrawl specifies what function should be called when a certain event occurs. To preserve the illusion of multi-tasking between display instances, each callback executes a small amount of code and then returns control to the central event loop. In this way, the CPU is being time-sliced among all the members of a WScrawl session in a fine enough granularity to preserve smooth animation and good response time to any one user's actions.

The only major problem encountered using Motif was with the Motif menu convenience functions which required some workarounds to preserve the illusion of multi-tasking. 'Normal' Motif menus use a small menu event loop where as soon as the user extends a

menu, Motif loops tightly to track the pointer motion. For WScrawl, using Motif menus would mean that as soon as one user pulled down a menu, no other participant in the WScrawl session would be able to do anything until that user released the menu. Therefore, I implemented the menus from Form Widgets and Pushbuttons to look and behave like standard menus but still give the desired illusion of multi-tasking. In every other case, such as dialog boxes, push buttons and text edit widgets, Motif uses the main event loop, giving the desired illusion of multi-tasking to the members of the session.

It is important to identify that most of the WScrawl Motif code would be written in exactly the same way for a normal (non-shared) paint program. By carefully structuring the data and preparing the interface carefully, only a few lines were added to make WScrawl a 'group-aware' shared drawing program.

7.5.3 WScrawl's capabilities

WScrawl has several quite advanced capabilities that are not available in any other shared drawing system to date. Keep in mind while reading this section that WScrawl's performance is not only 'acceptable' for a shared drawing tool, it is excellent even for a single-user drawing program. Also, the source code is available so that anyone can make additions or changes to WScrawl. Since WScrawl is in C and Motif, the libraries necessary for development are available on virtually every hardware platform sold today.

One of the more impressive capabilities WScrawl possesses is the ability to carry on a session between multiple depth displays. WScrawl is coded generically for any depth screen, and for example can have a window open on a monochrome display (1 bit) and on a 256 colour display (8 bit) and on a 24 bit (true colour or direct colour) display all at the same time.

In some cases, it is impossible to preserve consistency between two different depth screens. For example, say one WScrawl session contains a monochrome display and also contains a 256 colour display. There are two ways to handle the case where the user at the colour display changes pencil colour to blue: (1) disallow this action, since it is not possible on all screens, or (2) find the closest approximation to blue on the monochrome screen and continue on. WScrawl chooses the latter of these approaches, which I call 'best-attempt screen consistency'. This works surprisingly well in most cases, allowing the members of the session who have colour screens to have colour-capability, while allowing the members of the session with monochrome screens to still see approximately what is occurring and contribute.

In a normal use of WScrawl, one of the users already in the session 'adds' someone outside the session. However, WScrawl also has the ability to 'join' an existing WScrawl session from the outside. The WScrawl client is always ready to receive input on a well-known TCP/IP socket. Another WScrawl executes a 'knock-knock' on that socket and then exists, and the WScrawl that was 'knocked-upon' adds the user who 'knocked'.

WScrawl's handling of resources is quite sophisticated. Since a user cannot assume that the application resources are installed correctly on some system that he or she is trying to WScrawl with, WScrawl is able to detect that the resources are not set up correctly and install a sane set of resources in the server. In addition, any changes to settings such as the user name or the interface language appear immediately to the user, but are also set in the server resources. Then, even if WScrawl quits and starts again, the settings are preserved.

This works particularly well with environments that save resources when they are shut down, such as the Hewlett-Packard Visual User Environment (HP-VUE).

WScrawl has a TelePointer that allows one user to point to specific regions of the screen for the benefit of the other members of the session. This TelePointer can be turned on 'continuously', so, for example, users can see exactly who is drawing some new line or erasing some section of the whiteboard.

Another interesting piece of functionality is WScrawl's ability to grab arbitrary regions of the display *outside* WScrawl's window and bring them into the shared drawing area. This allows the one user to show other users exactly what is happening on his or her display. In conjunction with the other sophisticated drawing features of WScrawl, this 'super-selection' also allows users to get a closer look at items on the screen.

In WScrawl, there are two different policies for using the scrollbars in the shared drawing area. The default policy allows each user to scroll to a different area of the whiteboard independently. The other policy available allows all users to lock their scrollbars together for what Rein and Ellis call a 'Closely Coupled' session (Rein and Ellis, 1991). If one user scrolls, all the drawings scroll the same amount in the same direction. This preserves the identical view of the whiteboard for all participants.

Finally, WScrawl contains a wealth of features that users might want in any raster-type paint program. There is a Fatbits Editor that allows the user to edit very fine detail on the screen. There is a Color Picker that allows the user to 'grab' any colour he or she sees on the display, or alternatively dial in any R, G and B values needed. WScrawl's interface can be in any arbitrary ISO-Latin representable language (translations have been done to English and German so far). WScrawl can paste standard X-Windows selections, and can cut, copy and paste its own custom selections. Users can type in multiple fonts, they can save images in X-Windows bitmap form or colour X-Window-Dump (XWD) standard form, or in greyscale PostScript for printer output. They can continuously show their pointer coordinates for accurate alignment and selections. Users can paint with a variety of paintbrush shapes, can draw any width straight lines, any width and any shape ovals and squares, and can use a spray can. There is also extensive online help available by clicking on the 'Help' button, which is always visible.

7.6 CENTRALIZED VS. DISTRIBUTED ARCHITECTURE

In Section 7.5, it was shown that WScrawl has a centralized architecture based on the network capabilities of the X-Windows system (Scheifler and Gettys, 1986). Some researchers in this area are put off by the centralized architecture (Halasz, 1992; Ellis and Gibbs, 1989). A more decentralized alternative might be to have a separate UNIX process running locally on each user's CPU, communicating through Berkeley sockets. These processes must specify and then communicate through a custom protocol, thus discarding all the effort MIT went to in designing and implementing the perfectly usable X-Protocol.

To facilitate discussion of the issues surrounding these two approaches, I present here an example 'occurrence' during a shared whiteboard session. In this example, a 'User A' in the session draws a red rectangle which appears on all the participants' displays. After User A selects the rectangle, the following steps occur in the WScrawl case:

1. The one WScrawl Unix process sends an Xlib `XFillRectangle()` protocol request to every display (X-server) in the WScrawl session.
2. Each X-server receives the request and renders the red rectangle.

This is much like the activity shown in Figure 7.3. The decentralized approach mentioned above would go through similar, yet more complex steps. After User A selects the rectangle, the following steps might occur in the decentralized case:

1. The Unix process running on User A's CPU sends a custom 'filled-rectangle' packet across Berkeley sockets to every other CPU, with approximately the same information that is contained in the Xlib call `XFillRectangle()`.
2. Each Unix process running on a separate CPU receives one request to draw a red rectangle.
3. Each Unix process calls `XFillRectangle()` to its local X-server.
4. Each X-server receives the request and renders the red rectangle.
5. Each Unix process checks to see if some race condition has occurred during this operation, and if so backtracks (could be several steps).

In the above example, WScrawl's centralized approach has several distinct advantages over the decentralized approach. First of all, the decentralized approach invented a new protocol to send the 'fill-rectangle' request (step 1), when WScrawl's use of the X-Protocol is not only adequate but faster since WScrawl sends the requests directly to the destination X-servers, which must be told to draw the rectangle in either case. WScrawl's approach is simpler since the centralized architecture trivially eliminates race conditions, which must be checked for in the distributed case (step 5). Finally, the WScrawl approach took a tiny percentage of the time to implement and debug, since the communications protocol worked perfectly the first time, as it always will in the extremely standard and debugged X-Windows. In the decentralized case, the programmer would have spent hours testing and debugging all the possible permutations of messages that might occur through the new custom protocol, especially when considering the possible race conditions.

Although the above example represents a common action in a shared drawing session, there are other common actions where WScrawl's centralized architecture is less advantageous. For example, the case where one user wishes to view the colour choices by browsing the Colors menu in the menu bar. In this new example, the one WScrawl process (possibly across the network) must handle extension of the Colors menu and highlight each menu item as the pointer passes over it. In the decentralized case, the one UNIX process local to that user could take care of the menu extension and highlighting while not sending any packets across the network to the other Unix processes. This results in a more responsive interface than in the centralized WScrawl case. Exactly how much more responsive depends on the network distances the shared program is running across. On a local network with modern workstations, the two cases are indistinguishable, since WScrawl's menus extend as fast and browse as responsibly as any single-user application's menus do. However, across wide area networks, such as when WScrawl was tested between the USA and Germany, network traffic and other inconsistencies cause 'hangs' for as long as 30 s at a time, where none of the participants in the WScrawl session can do anything. If implemented carefully, a decentralized architecture could buffer the users from such hangs, still allowing them to get local feedback to their actions, such as seeing

menus extend and seeing paint appear on the drawing surface. Then, as the network clears, the buffered requests could be pushed through. As mentioned above, at that point the extremely complex problem of resolving race conditions becomes the distributed application designer's foremost concern.

A decentralized implementation might also scale better to larger numbers of users in one session as the session nears the limits of the workstations' communication and processor speeds. However, research has found that rarely do more than two or three people wish to share a whiteboard (Brinck and Gomez, 1992).

In some ways, the wide area network problem is simply unsolvable. If packets do not get through for 30 s, then the users will not see what the other users are drawing and may continue drawing in a useless or even destructive way. Until networks get faster and more reliable, this problem cannot be solved no matter what type of distributed architecture the program has. When wide area networks do become faster and more reliable, the centralized WScrawl will work virtually as well as the distributed architectures will, but still be easier to design and implement.

In conclusion, I cannot stress emphatically enough how easy it is to add new functionality to WScrawl, since the extensive X graphics protocol already exists, and also because the centralized architecture trivially eliminates all race conditions. It is my opinion that the step to a decentralized architecture is so complicated that the final product will lack the long list of functionality that a centralized approach would contain given the same implementation time and effort. In other words, the distributed architecture is currently a trade-off giving slightly better performance in some cases and much less functionality overall.

7.7 SUMMARY

WScrawl 2.0 is an interactive, fully functional shared whiteboard implemented in Motif 1.1 and X-Windows without custom communication protocols. It is public domain, and available today as highly portable C source code that has been compiled and run successfully on (among others) Sun SparcStation, IBM RS 6000, DEC 3100, and HP series 300, 700 and 800.

This implementation of WScrawl 2.0 proves that real-time cooperative tools can be built with more than adequate performance on today's hardware, communicate across today's networks, and be implemented with today's widely available standard graphical toolkits, which give excellent support for distributed graphics.

ACKNOWLEDGEMENTS

The implementation of WScrawl and the writing of this chapter would not have been possible without the help, comments, contributed code and encouragement from more people than I can list here. The following deserve special mention: Harry Phinney for answering ten bazillion X-Windows low level technical questions and as far as I can tell never getting one wrong; Pam Raby my Motif guru; Mark Cook for his xwd file I/O functions; Craig Durland, Keith Marchington and Charlie Amacher for technical help; Sven Dack and Klaus Reichenberger for German translations and German beer; Randy

Wilson, trans-Atlantic test base; and finally, Dr Norbert Streitz and the German Institute GMD-IPSI, who supported the development of much of version 2.0. Thank you for providing an excellent environment to do research in.

REFERENCES

- Brinck, T. and Gomez, L. M. (1992) A collaborative medium for the support of conversational props, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '92)*, Toronto, Canada, 1-4 November.
- Ellis, C. A. and Gibbs, S. J. (1989) Concurrency control in groupware systems, in *Proceedings of the ACM SIGMOD '89 Conference on the Management of Data*, 2-4 May, Seattle WA, pp.399-407.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, in *Communications of the ACM*, **34** (1) pp.38-58.
- Engelbart, D. and Lehtman, H. (1988) Working together, in *BYTE*, December, pp.245-52.
- Farallon Computing Inc. (1988) *Timbuktu User's Guide*, Farallon Computing Inc., Berkeley CA.
- Grudin, J. (1988) Why CSCW applications fail: problems in the design and evaluation of organizational interfaces, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88)*, 26-29 September, Portland OR, pp.85-93.
- Haake, J. M. and Wilson, B. (1992) Supporting collaborative writing of hyperdocuments in SEPIA, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '92)*, 1-4 November, Toronto.
- Halasz, F. (1992) Personal communication, Xerox PARC, February.
- Ishii, H. and Kobayashi, M. (1992) ClearBoard: a seamless medium for shared drawing and conversation with eye contact, in *Proceedings of the Conference of Computer Human Interactions (CHI '92)*, 3-7 May, pp.525-32.
- Lauwers, J. C. (1990) Collaboration transparency in desktop teleconferencing environments, *Ph.D. Thesis*, Stanford CA.
- Lee, J. J. (1990) XSketch: a multi-user sketching tool for X11, in *Proceedings of COIS '90 Conference on Office Information Systems*, 25-27 April, Cambridge MA, pp.169-73.
- Lu, I. M. and Mantei, M. M. (1991) Idea management in a shared drawing tool, in Bannon, L., Robinson, M., Schmidt, K. (eds), *Proceedings of the 2nd European Conference on Computer-Supported Cooperative Work (E-CSCW '91)*, 25-27 September, Amsterdam, pp.97-112.
- Minneman, S. L. and Bly, S. (1991) Managing a trois: a study of a multi-user drawing tool in distributed design work, in *Proceedings of the Conference of Computer Human Interactions (CHI '91)*, 27 April-2 May, pp.217-24.
- Rein, G. L. and Ellis, C. A. (1991) rIBIS: a real-time group hypertext system, *International Journal of Man-Machine Studies*, **34**, pp.339-67.
- Scheifler, R. W. and Gettys, J. (1986) The X-Windows system, *ACM Transactions on Computer Graphics*, **5**, pp.79-109.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. (1987) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, **30** (1) pp.32-47.
- Streitz, N., Lemke, A. and Wilson, B. (1991) Integrating video conferencing with WScrawl, unpublished paper.
- Tang, J. C. and Minneman, S. L. (1991) VideoWhiteBoard: video shadows to support remote collaboration, in *Proceedings of the Conference of Computer Human Interactions (CHI '91)*, 27 April-2 May, pp.315-22.
- Whittaker, S., Brennan, S. E. and Clark, H. H. (1990) Co-ordinating activity: an analysis of interaction in computer-supported co-operative work, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90)*, 1-10 October, Los Angeles, pp.361-7.

GROUPKIT: A GROUPWARE TOOLKIT FOR BUILDING REAL-TIME CONFERENCING APPLICATIONS

Mark Roseman and Saul Greenberg

ABSTRACT

This chapter presents our approach to the design of groupware toolkits for real-time work, and how the design is instantiated in our toolkit, GroupKit. The design is based on both the technical underpinnings necessary for real-time groupware and on user-centred features identified by existing CSCW human factors work. We also present three strategies for building GroupKit's components. First, an extendible, object-oriented *run-time architecture* supports managing distributed processes and the communication between them. Second, *transparent overlays* offer a convenient method for adding general components to various groupware applications, for example supporting gestures via multiple cursors and annotation via sketching. Third, *open protocols* allow the groupware designer to create a wide range of interface and interaction policies, accommodating group differences in areas such as conference registration and floor control.

8.1 INTRODUCTION

Real-time computer conferencing applications have been a major focus of groupware development efforts. Often motivated by CSCW research, we have recently seen the creation of numerous systems, such as shared text editors (Neuwirth *et al.*, 1990), freehand sketching systems (Greenberg and Bohnet, 1991; Minneman and Bly, 1991), structured drawing programs (Greenberg *et al.*, in press) and group support systems (Nunamaker *et al.*, 1991). Yet construction of these applications is fraught with difficulties. Besides all the normal problems of building single-user applications, the groupware developer must be concerned with technical issues such as synchronization, concurrency, communications,

The original version of this paper appeared in the Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '92). Copyright 1992, Association for Computing Machinery, Inc. Reprinted by permission.

registration and more. The developer must also understand and incorporate the fundamental CSCW human factors issues identified for effective group work. *Groupware toolkits* are now emerging that address some of these issues. By providing the key components for common groupware needs, the toolkits can reduce development effort, enable rapid prototyping and increase product quality of multi-user applications.

Of course, there are many potential groupware applications, and no one toolkit could cover all possibilities. In our case, we are concerned with toolkits that assist in constructing real-time *work surfaces* - shared visual environments where one user's actions are made immediately visible to other users (Gibbs, 1989). These work surfaces are useful in both face to face and geographically distributed meetings. At its simplest, a work surface is a 'what you see is what I see' (WYSIWIS) space (Stefik *et al.*, 1987) that conference participants use for creating, importing, displaying and interacting with their artefacts - usually text and graphics of some sort. Thus work surfaces include generic applications such as shared windows, whiteboards, structured drawing systems, and shared editors. But work surfaces can be far more powerful and specialized than that. For example, WYSIWIS could be relaxed so that people could view the work surface from different perspectives; an example would be a competitive card game (Patterson, 1991). Specialized applications could expand the notion of work surface to include tools for voting, brainstorming, organizing ideas and so on.

This chapter discusses our toolkit design philosophy, as embodied in GroupKit. We begin by listing a preliminary set of requirements for groupware toolkits, derived from both human factors work and technical innovations in CSCW. The following section presents an overview of GroupKit, with a detailed description of its technical architecture. Examples of three strategies for designing toolkit components are then given.

8.2 TOOLKIT DESIGN REQUIREMENTS

In this section we propose and defend some of the underlying design requirements that we feel are important for a real-time conferencing toolkit. We do not believe this list is complete, nor will every item be necessary for every groupware application. However, we do feel the requirements listed will apply to a wide range of real-time conferencing applications, and that they provide a reasonable starting point for discussion. Requirements listed in the first section are human-centred, and those in the second are programmer-centred.

8.2.1 Human-centred design requirements

Supporting multi-user actions over a visual work surface

We believe there are several general but critical activities that people do over a shared work surface, regardless of its contents. Two of these, gesturing and annotation, are described below.

- (a) *Provide support for gesturing.* Researchers at Xerox PARC studied the use of conventional drawing surfaces by small groups (Tang, 1991). A critical finding was that participants frequently gesture over the drawing surface: to enact ideas, to signal

turn-taking, to focus the attention of the group, and to reference objects on the surface. Several recent computer systems emulating group drawing surfaces support gesturing with multiple cursors appearing on all displays (Greenberg and Bohnet, 1991; Bly and Minneman, 1990; Greenberg *et al.*, in press), and their usability studies confirm the ubiquity of gesturing. We believe that gesturing can enhance communication in many diverse types of conference, and should be supported at the toolkit level.

- (b) *Provide support for graphical annotation.* The Xerox studies also noticed many instances of annotations made to existing drawings, serving both as gestures (e.g. underlining text while saying 'this one here') and as meta-level notes. Several systems now incorporate graphical annotations for their objects. FreeStyle users, for example, can verbally and graphically annotate bitmap snapshots; the results can be mailed to others who can then play back the transcript (Francik, 1991). Both Proof-Marks in *vmacs* (Lakin, 1990), and the commercial MarkUp application allow comments and markup symbols to be added to written documents. As with gesturing, we believe that real-time group graphical annotation over a work surface is useful in many situations and should be supported by the toolkit.

Structuring group processes during a meeting

Some researchers believe that groupware should impose a social model of interaction on the group. This is an explicit attempt, based on management theory, to provide methods for keeping the group on task, enforcing roles and commitments, and making the group more efficient and productive. There is certainly controversy between those who believe that social protocol should be determined only by the group members (e.g. Dykstra and Carasik (1991)), by the software (e.g. Nunamaker *et al.* (1991)), and somewhere in between (e.g. Johnson-Lenz and Johnson-Lenz (1991) and Greenberg *et al.* (1991)). We believe that some group process primitives should be provided by the toolkit, accommodating groupware that wishes to control meeting structure. The list below discusses only a few group process requirements.

- (c) *Provide various floor control policies.* Floor control or turn-taking mechanisms provide a way to mediate access to shared work items. Lauwers (1990) and Greenberg (1991) recommend that systems should 'support a broad range of [floor control] policies' to suit the users' needs. Systems such as SharedX (Garfinkel *et al.*, 1989) and Aspects (from Group Technologies) support a few different policies, while Share (Greenberg, 1991) strives to provide complete flexibility. Floor control can be important in many situations, such as shared screens allowing only serial interaction, or systems following strict interaction models, such as a teacher controlling which students can access the work surface.
- (d) *Support different registration methods.* Another part of group process controls who is allowed to join the meeting. For some meetings, anyone may be allowed to join. For others only a select group can participate, or perhaps new users must be 'sponsored' by an existing user. Sometimes more spontaneous creation of conferences is desired (Kraut *et al.*, 1990) while other situations require a central facilitator to handle registration (Nunamaker *et al.*, 1991). Toolkits should provide the flexibility to support any reasonable registration process.

- (e) *Support latecomers to the conference.* A consequence of spontaneous conferences is that all users will not join the conference at the start. Provisions should exist allowing newcomers to join at any time, as well as allowing existing members to leave. Strategies must also be supported to assist the newcomers in 'getting up to speed'. This may involve simply sending the current conference state to the new user (Greenberg *et al.*, 1992) or providing summary information on how the conference has progressed over its lifetime.

Integration with conventional ways of doing work

Groupware should not impose a barrier between 'individual' and 'group' ways of working. For example, the system should provide group members with ready access to their individual work, and allow them to import it to a conference. Additionally, all normal communication channels (e.g. telephone, email) should be readily available.

- (f) *Integrate other forms of communication.* Voice communication is an important factor in most conferences (Chapanis, 1975), and given the ubiquity of telephones we should assume that a voice channel is available. While many real-time conferencing systems assume a voice channel is present, they do not explicitly support creating voice links. Ideally, there should be a mechanism in the conferencing system to establish voice conferencing – perhaps automatically when a data conference is started. Similar arguments hold for other channels that may be available such as video links.
- (g) *Allow use of single-user applications.* Soon, most computer applications may be designed to support multiple users. Unfortunately, most of today's applications support single users only. There are several reasons why single-user programs should be available in multi-user conference settings. Groupware counterparts to single-user programs may not exist; a person's work may be accessible only through a particular application; people are skilled on particular applications. Conference users should be able to view and interact with single-user systems through shared screens or shared windows e.g. Share (Greenberg *et al.*, 1990) and SharedX (Garfinkel *et al.*, 1989). The toolkit should provide shared windows or the means to incorporate other shared window systems.

8.2.2 Programmer-centred design requirements

Technical support of multiple and distributed processes

Most groupware, especially for geographically distributed conferences, will require an architecture where multiple (perhaps distributed) processes can communicate with each other. While most operating systems provide process control and interprocess communication, the programmer's job of initiating, maintaining and tearing down processes and their communication channels is a tedious one. Also, state information about sessions may need to survive beyond the lifetime of a single process or meeting.

- (h) *Provide processes for basic conference management.* Groupware applications must oversee all conference management, which include activities such as participant registration, initiation and teardown of meeting processes, communications, and so

on. Groupware toolkits have placed much emphasis on providing capabilities for conference management, e.g. Liza (Gibbs, 1989), Conference Toolkit (Bonfiglio *et al.*, 1989) and MMConf (Crowley *et al.*, 1990). We too believe that the basic run-time infrastructure for conference management must be supplied by the toolkit.

- (i) *Provide a robust communications infrastructure.* Any groupware toolkit must provide the communications facilities on which to build conferencing components. At the very least, it must be possible for any process to send messages to specific processes owned by conference users, and it is preferable if a multi-cast facility is available to broadcast a single message to all. Of course, the communications demands will depend heavily on the way the process architecture is determined. The trade-offs between centralized and replicated architectures are well-documented (Lauwers *et al.*, 1990), with centralized architectures simplifying concurrency control and replicated architectures being more efficient and robust to machine failure.
- (j) *Provide support for persistent sessions.* Often computer conferences will span more than a single session, for example decision support meetings (Nunamaker *et al.*, 1991). It is desirable to maintain session state information over the full duration of the conference. There should exist a general mechanism whereby conference objects can be made persistent.

Technical support of a graphics model

A visual work surface will require graphical and textual primitives. Yet shared graphics require several capabilities that are not present in single user systems.

- (k) *Provide primitives to a shared graphics library.* Many groupware applications require graphical library primitives for creating multi-user objects such as shared lines, rectangles, circles and text. Greenberg *et al.*'s (1992) discussion of GroupDraw describes technical issues of a shared object-oriented drawing package, and provides their design of an abstract drawing object that can be sub-classed into concrete objects such as shared lines. Similarly, the fine-grained editing of simple graphics and text objects in Bier and Freeman's (1991) MMM system gives insight into how shared objects should behave. Similar extendable graphics libraries should be provided by toolkits, so that programmers can easily create shared interactive graphical objects on the display.
- (l) *Provide object concurrency control.* Many groupware conferencing systems support access to some type of shared object, be it structured graphics or a text buffer. Concurrency control is often needed to mediate access to the object, for example two people trying to manipulate the same point on a line. In fact several concurrency schemes have already been implemented in groupware toolkits (Patterson *et al.*, 1990; Patterson, 1991; Gibbs, 1989). Concurrency can be achieved through simple locking, transaction mechanisms or numerous other schemes (Ellis *et al.*, 1991). In addition, the degree of concurrency and access to shared objects can be specified through the notion of flexible coupling (Dewan, 1991).
- (m) *Separate the view of an object from its underlying representation.* Many single-user graphical systems separate the properties of an object from its view on the screen. Patterson argues that this separation is critical in groupware (Patterson *et al.*, 1990; Patterson, 1991), and that abstractions should be used to create an interface-

independent representation of data. As a consequence, users can have multiple perspectives on the same data.

8.3 GROUPKIT

GroupKit is a toolkit a programmer can use to develop real-time computer conferencing applications for geographically distributed or face-to-face meetings. It provides not only support for basic connectivity, but also support for higher-level human factors concerns. As it is still being developed, we have not yet incorporated all our design principles. However, components are provided for flexible policy conference registration, communications support, flexible floor control, and gestural and annotative communication.

GroupKit is written in C++, on top of the InterViews toolkit (Linton *et al.*, 1991). Applications built using GroupKit will run on any machine supporting InterViews, that is, UNIX workstations running X-Windows. GroupKit relies on the InterViews glyph mechanism for its user interface constructs. Glyphs are lightweight objects (similar to widgets in other toolkits) that are composed to make interfaces. The InterViews Dispatch library – a front end to normal UNIX sockets – is used for communications.

8.3.1 Process and communications run-time support

GroupKit's technical infrastructure is based upon a replicated architecture and communications support, as illustrated in Figure 8.1. The infrastructure supports conference registration and the subsequent communication between processes owned by conference participants. This is comparable to the communications support in Rendezvous (Patterson *et al.*, 1990) and the Conference Toolkit (Bonfiglio *et al.*, 1989). The remainder of this section describes the infrastructure components by walking through an example scenario.

Registration

Suppose a user wants to create a shared drawing surface, perhaps to discuss a design problem with a remotely located colleague. On initiating the GroupKit-based program, the user must first create a conference. This is done through the *Registrar Client*, which provides an interface to the central *Registrar*.

The Registrar allows the user to create, join or leave one or more conferences. The Registrar Client is responsible for implementing particular registration policies, e.g. deciding who enters the conference, how they do so and what the interface looks like. While GroupKit allows new Clients to be programmed, it also provides a library of predefined Registrar Clients implementing particular registration policies and user interfaces. One novel aspect of this scheme is that it allows group members to use different Registrar Clients to enter a single conference.

The Registrar is an independent process (invisible to the user) that maintains a list of all conferences and their users. One central Registrar would exist at each installation. The Registrar itself is *policy-free*, and leaves it to the Registrar Clients to implement a

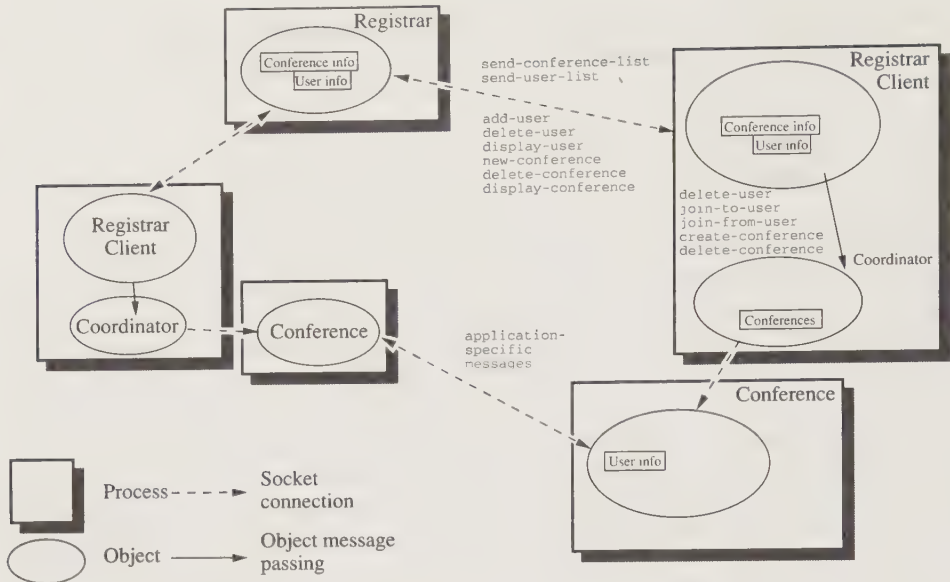


Figure 8.1 Communications infrastructure of GroupKit, showing message passing between objects. Here, objects owned by one user (rightmost Registrar, Coordinator and Conference) interact with objects owned by another user, as well as the central Registrar. Small text indicates the message passing protocol.

particular registration policy to present a reasonable interface to the user. This allows different policies and interfaces to be implemented in order to accommodate group differences.

Conference initiation

In our scenario, the user has just requested a new conference through the Registrar Client, which in turn passes the request on to the Registrar. Next, the Registrar Client asks the *Coordinator* to create a new *Conference* object. The Coordinator acts as an intermediary between the Registrar Client and application Conferences, permitting multiple conferences (e.g. sketching and editing applications) to share a common registration mechanism. Its main duties are to create Conferences at the request of the Registrar Client, and to direct requests from the Client to the appropriate Conference.

The Coordinator then spawns a new process which instantiates a Conference object. It is the Conference object that actually runs the specific groupware application. GroupKit provides a generic Conference object, which the developer may subclass to provide extra functionality required by the application. In this case, the generic Conference object is used, along with a programmer-defined glyph (graphical interface) supporting a shared drawing surface.

Conference maintenance

Other users can also create Conference objects. As each Conference locates the meeting participants via the Registrar, communications channels will be opened between all Conference objects. Facilities in the generic Conference are provided for exchanging messages with other user processes.

Communications between distributed processes are maintained by *messaging objects*. The two types of messaging objects ('Writer' and 'Reader') provide a convenient method of communicating with processes owned by remote conference participants. These objects, derived from the InterViews Dispatch library, provide a primitive Remote Procedure Call (RPC) facility. Writer objects specify messages they can send, while Reader objects provide functions to be called when messages are received.

Conference leaving and termination

As with initiation, conference leaving and termination is handled through the Registrar Client. If a user wishes to leave the conference without terminating it, their Registrar Client sends the `delete-user` message (Figure 8.1) to the Registrar. Some Clients may permit explicit conference termination, allowing any user to terminate the conference, while more typically the Registrar Client for the last user to leave will terminate the conference. This is done by sending a `delete-conference` message to the Registrar, which will be rebroadcast to any remaining users' Registrar Clients.

8.3.2 Overlays

GroupKit provides components that may be included in any Conference object through an *overlay* strategy. Overlays are transparent 'windows' placed on top of the main application graphics shown by the Conference object. Currently, overlay components have been implemented for gestural communication (via multiple cursors over a surface) and annotative communication (via freehand drawing over a surface). The motivation is that such components could be useful for a variety of groupware applications, as mentioned earlier.

Figure 8.2 provides a conceptual picture of adding an overlay for displaying multiple cursors on an existing application graphic. The transparent cursor overlay is written as an InterViews glyph that overlays any other glyph. Neither the cursor glyph nor the main application glyph need any knowledge of the other. Input events are received by the cursor glyph, which updates cursors as necessary. The event (e.g. cursor motion) is then passed to the application glyph, to use as needed. The application glyph need not even know that the event went through the cursor glyph. As with single-cursor window systems, event-driven drawing operations are performed normally by the application glyph, based on these events, with the cursor glyph sketching the cursors on top of the normal graphics.

To incorporate the multiple cursor overlay into an application, the programmer instantiates a Cursor Overlay which surrounds the application's graphic. Code in the overlay's constructor allows the Cursor Overlay to communicate with remote conference objects and inquire about new conference users.

This technique, also used to implement the freehand sketching overlay, seems very promising. Through the composition mechanism, adding overlay components to

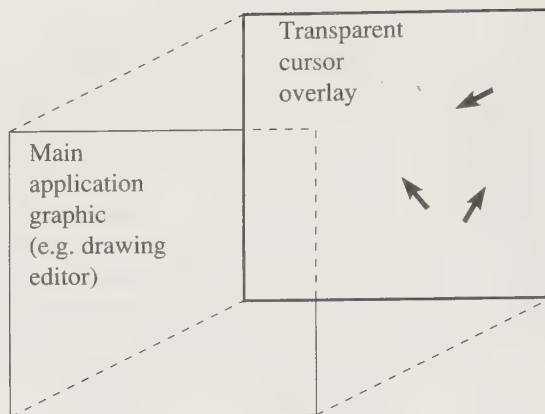


Figure 8.2 Adding a multiple cursor overlay.

applications is extremely straightforward. Also, the overlays are kept separate – conceptually and in the code – from the underlying applications. Its strength is that the overlay does not interfere with the underlying graphics of the application, even if those graphics are changing. Because of this, it is a trivial matter to add, for example, annotation capabilities on top of a ‘live’ shared terminal application. Unlike other systems that only allow annotation of screen snapshots (e.g. MMConf (Crowley *et al.*, 1990)), the underlying application can be fully active. It is expected that further research will suggest other components that could be transparently added to a variety of conferences.

8.3.3 Open protocols

One design requirement for GroupKit is to provide flexible policies where appropriate, allowing group processes to be structured during a meeting, and to accommodate group differences. *Open protocols* provide a means of implementing this flexibility. They have three components: a controlled object, a controller object and a protocol describing how the two communicate. The controller object’s behaviour does not incorporate any policy determining how its state can be manipulated. Instead, a protocol is defined, and the object will obey any external requests made to it to change its state. The controller is just an external object that implements a particular policy by the requests it sends to the controlled object.

Currently, two components have been implemented using open protocols, a floor control module and a registration module. We concentrate here on the registration module. Figure 8.3 shows the message protocol (beside the arrows) between the Registrar (the controlled object) and its Clients (the controllers). Here, the Registrar responds to *any* request from its clients, allowing any client to ask the Registrar to create a new conference, or conceivably even to delete any user from any conference. While this does make it possible to create a ‘super-user’ version of the Client, it also provides the flexibility to create any number of other Clients interfacing to the Registrar, without making any changes to the Registrar itself. As examples of this, the implementations of both a *free* and a *restricted* registration policy are now described. Under a free registration policy, new

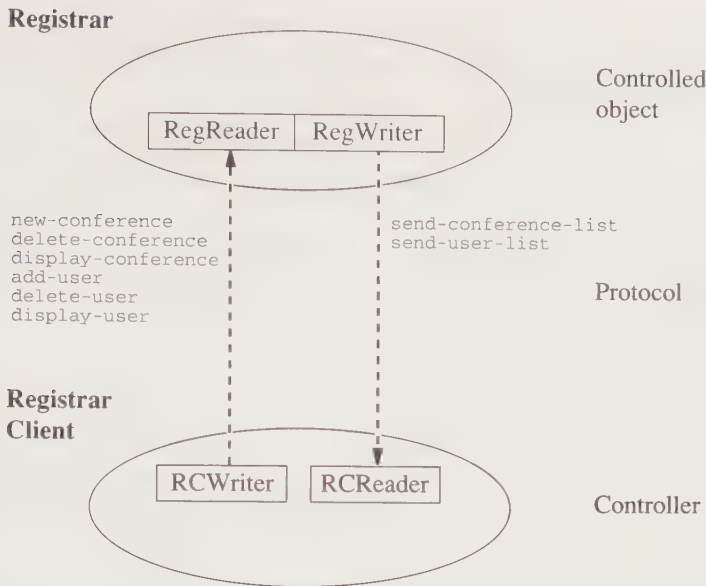


Figure 8.3 Registration Protocol.

users may join any existing conference. The implementation here is straightforward. The Registrar Client sends an `add-user` message to the Registrar, which is broadcast to the other Registrar Clients in the selected conference. The Registrar Client also requests the Coordinator to create a new application Conference. The Conference makes connections with the other users, and the interaction proceeds normally.

In contrast, a restricted registration policy does not permit new users to join an existing conference unless 'sponsored' by an existing conference participant. Here, the Registrar Client again sends an `add-user` message, which is rebroadcast to the other users. At this point, the local Registrar Client does *not* ask the Coordinator to create a new conference. The remote users are asked by their Registrar Clients if the new user should be accepted. A remote user can accept the new user, and sends them a message, prompting the new user to create the conference as before. If rejected (either explicitly or by timeout), the `delete-user` message is sent to the Registrar.

Floor control policies are handled similarly (Greenberg, 1991). Each user has a flag, specifying if their actions should affect the conference or not. Any user can examine or change any other user's flag. A preemptive floor control scheme for serial interaction, where a user can immediately seize control from the current floor holder, can be implemented by setting the local flag to `write` and setting all others to `no-write`. The end result is that the local user gains control of the floor until preempted by another user. A ring-passing policy, where the floor may be seized only if it is free (i.e. the previous floor holder has released it), is implemented by setting the local flag to `write` only if all others are currently set to `no-write`. MMConf (Crowley *et al.*, 1990) uses a similar strategy for floor control, but manipulates only a single token, thus allowing less flexibility in protocols.

Again, this strategy seems useful in general. By providing a simple protocol to change states, building new policies becomes a simple matter of expressing the policy's semantics in the language of the protocol.

8.4 BUILDING AN EXAMPLE APPLICATION

This section describes the steps necessary to build applications using GroupKit. The steps include specifying the application-specific graphical presentation and interaction, initializing objects to send and receive application messages, selecting an application Conference object and initializing a Coordinator object.

As an illustration, we describe the construction of a simple multi-user freehand sketching program using the multiple cursor overlay. The interface is similar to GroupSketch (Greenberg and Bohnet, 1991): multiple cursors are always visible, any user can draw at any time, and fine-grained actions are immediately visible on all displays. Note that although GroupKit provides a sketching overlay, for illustration we will not use it.

8.4.1 Graphical presentation

The application needs one or more InterViews glyphs to manage the graphical presentation and user interaction aspects of the interface, as well as any internal data structures. For a freehand sketching application, this involves creating a glyph holding a bitmap, and providing tools (pencil and eraser) that respond to mouse events for changing the bitmap. To incorporate the cursor overlay, the bitmap glyph is 'composed within' the cursor glyph.

Designing this part of the sketching application is comparable to designing a single user version of the group application. There are some general conventions that are helpful to follow, such as separating event handling (cause) and the result of events (effect) into different routines. This facilitates use of common routines for local and remote invocations (Greenberg *et al.*, 1992).

8.4.2 Messaging objects

Writer and Reader objects are used to send and receive application-specific messages. The routines in Writer objects are invoked as a result of local actions, for example, transmitting coordinates of a drawn line segment to the other replicated applications. Callbacks in the Reader objects interpret these messages, usually calling routines in the graphical presentation object to handle requests. In the example, the Reader instructs the sketchpad glyph to draw the line specified in the message. The standard objects must be initialized to include the required callbacks.

8.4.3 Coordinator

The Coordinator connects the registration mechanism (via the Registrar Client) to the application Conference objects running as separate processes. Available conference types must be specified to the Coordinator, using the standard X resource mechanisms (i.e. XDefaults).

8.4.4 Application conference

The application Conference maintains communications channels with other distributed applications. In our example, the generic GroupKit conference has sufficient functionality. The main program instantiates this object, and 'attaches' to it the bitmap glyph described earlier, so that the glyph can send and receive messages.

The Conference is notified when users join or leave. Routines in the base Conference class manage the low-level socket connections between users. However, other classes may be notified when new users join or leave, to manipulate application data structures maintained for each conference user. For example, the cursor overlay uses this information to add or remove cursors as users join or leave.

FUTURE WORK

The work presented here should be seen as an initial attempt to formalize the design and implementation of general groupware conferencing toolkits. The design requirements emphasize the important abstractions needed in real-time CSCW applications, and provide a basis for generalizing existing application or toolkit features. The three strategies presented – a run-time process and communication architecture, overlays and flexible policies – should be seen as general strategies that can be used to implement certain design features. It is expected that further design principles and strategies will evolve.

GroupKit has proven to be a flexible platform for testing our ideas. We have already built prototype drawing programs and shared terminals, and we will be constructing more elaborate and robust applications shortly. Currently, several of the design principles have not yet been embodied in the toolkit. Our immediate plans are to address the concurrency control issues, building a layer of support for generic shared graphical objects, that follow ideas presented in Greenberg *et al.* (1992). Ideally, a framework for building domain-specific group graphical editors could be created, drawn from the ideas in Unidraw (Vlissides *et al.*, 1989).

Some other work has focused on blurring the distinction between synchronous and asynchronous groupware, by providing a system that determines the appropriate means of conferencing (synchronous or asynchronous) and makes available various communication channels (text, voice, video). The choices offered the user depend on environmental information (i.e. who is currently around, what communication channels are available). The registration mechanisms in GroupKit provide the flexibility to implement such a scheme.

Groupware toolkits still have a long way to go to catch up to their single-user counterparts. We look forward to the day when all toolkits will incorporate multi-user features, and their interface components (such as control panels and editing widgets) have multi-user capabilities built into them. When that day comes, the artificial distinction between single and multi-user systems will disappear.

Note: GroupKit is available by anonymous ftp from the Department of Computer Science, University of Calgary ([ftp.cpsc.ucalgary.ca:pub/projects/grouplab/software](ftp://cpsc.ucalgary.ca/pub/projects/grouplab/software)).

REFERENCES

- Bier, E. A. and Freeman, S. (1991) MMM: a user interface architecture for shared editors on a single screen, in *User Interface Software and Technology (UIST '91)*, 11–13 November, pp. 79–86.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems (COIS '90)*, 25–27 April, pp. 184–92.
- Bonfiglio A., Malatesa, G. and Tisato, F. (1989) Conference Toolkit: a framework for real-time conferencing, in *Proceedings of the 1st European Conference on Computer Supported Cooperative Work (EC-CSCW '89)*, September.
- Chapanis, A. (1975) Interactive human communication, *Scientific American*, **232** (3) pp. 36–42.
- Crowley, T., Baker, E., Forsdick, H., Milazzo, P. and Tomlinson, R. (1990) MMConf: an infrastructure for building shared applications, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90)*, 7–10 October.
- Dewan, P. (1991) Flexible user interface coupling in collaborative systems, in *Proceedings of the ACM CHI'91 Conference on Human Factors in Computing Systems*, 27 April–2 May, pp. 41–8.
- Dykstra, E. A. and Carasik, R. P. (1991) Structure and support in cooperative environments: the Amsterdam Conversation Environment, *International Journal of Man-Machine Studies*, **34** (3), pp. 419–34.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences. *Communications of the ACM*, **34** (1) pp. 38–58.
- Francik, E., Rudman, S. E., Cooper, D. and Levine, S. (1991) Putting innovation to work: adoption strategies for multi-media communication systems, *Communications of the ACM*, **34** (12) pp. 37–63.
- Garfinkel, D., Gust, P., Lemon, M. and Lowder, S. (1989) The SharedX multi-user interface user's guide, version 2.0, *HP Research report No. STL-TM-89-07*, Palo Alto, California.
- Gibbs, S. J. (1989) Liza: an extensible groupware toolkit, in *Proceedings of the ACM CHI'89 Conference on Human Factors in Computing Systems*, 30 April–4 May, pp. 29–35.
- Greenberg, S. (1990) Sharing views and interactions with single-user applications, in *Proceedings of the Conference on Office Information Systems (COIS '90)* 25–27 April, pp. 227–37.
- Greenberg, S. and Bohnet, R. (1991) GroupSketch: a multi-user sketchpad for geographically-distributed small groups, in *Proceedings of Graphics Interface '91*, 5–7 June, Calgary, Alberta, pp. 207–15.
- Greenberg, S. (1991) Personalizable groupware: accommodating individual roles and group differences, in *Proceedings of the 2nd European Conference on Computer Supported Cooperative Work (EC-CSCW '91)*, September.
- Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992) Human and technical factors of distributed drawing tools, in *Interacting with Computers (Special Edition on CSCW)*, **4** (3), pp. 364–94.
- Johnson-Lenz, P. and Johnson-Lenz, T. (1991) Post-mechanistic groupware primitives: rhythms, boundaries and containers, *International Journal of Man-Machine Studies*, **34** (3) pp. 385–418.
- Kraut, R. E., Egido, C. and Galegher, J. (1990) Patterns of contact and communication in scientific research collaborations, in *Intellectual Teamwork: Social Foundations of Cooperative Work*, Lawrence Erlbaum Associates, New York, pp. 149–72.
- Lakin, F. (1990) Visual languages for cooperation: a performing medium approach to systems for cooperative work, in *Intellectual Teamwork: Social Foundations of Cooperative Work*, Lawrence Erlbaum Associates, New York, pp. 453–88.
- Lauwers, J. C. (1990) Collaboration transparency in desktop teleconferencing environments, *Ph.D. dissertation*, Technical Report CSL-TR-90-435, Stanford University, Computer Systems Laboratory.
- 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 (COIS '90)*, 25–27 April, pp. 249–60.
- Linton, M. A., Calder, P. R., Interrante, J. A., Tang, S. and Vlissides, J. M. (1991) *InterViews Reference Manual*, Stanford University.
- Minneman, S. L. and Bly, S. A. (1991) Managing a trois: a study of a multi-user drawing tool in distributed design work, in *Proceedings of the ACM CHI'91 Conference on Human Factors in Computing Systems*, 27 April–2 May, pp. 217–24.
- Neuwirth, C. M., Kaufer, D. S., Chandhok, R. and Morris, J. H. (1990) Issues in the design of computer support for co-authoring and commenting, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90)*, 7–10 October.

- Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R. and George, J. F. (1991) Electronic meeting systems to support group work, *Communications of the ACM*, **34** (7) pp. 40–61.
- Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S. (1990) Rendezvous: an architecture for synchronous multi-user applications, in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90)* 7–10 October.
- Patterson, J. F. (1991) Comparing the programming demands of single-user and multi-user applications, in *Proceedings of User Interface Software and Technology (UIST '91)*, 11–13 November, pp. 87–91.
- Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tatar, D. (1987) WYSIWIS revised: early experiences with multiuser interfaces, *ACM Transactions on Office Information Systems*, **5** (2) pp. 147–67.
- Tang, J. C. (1991) Findings from observational studies of collaborative work, *International Journal of Man-Machine Studies*, **34** (2) pp. 143–60.
- Vlissides, J. M. and Linton, M. A. (1989) Unidraw: a framework for building domain-specific graphical editors, in *Proceedings of User Interface Software and Technology*, October.

SLICE: A LOGICAL MODEL FOR SHARED EDITORS

Alain Karsenty and Michel Beaudouin-Lafon

ABSTRACT

We present a model for shared editing of documents. Our approach is to extend the principles of direct manipulation to groupware systems and to apply them to shared editing. We define a model of shared manipulation based on the decomposition of an application in different layers. We show how different groupware functionalities can be obtained by either sharing or slicing each layer and we illustrate these functionalities with GroupDesign, a shared editor for structured diagrams that we have developed. We also address the use of auditory icons for improving awareness of the group in shared tasks. Finally, we describe the implementation of GroupDesign with a purely replicated architecture. We describe, in particular, the concurrency control algorithm used to achieve data consistency, which is based on the semantics of the application.

9.1 SHARED MANIPULATION

Direct manipulation was introduced by Shneiderman (1983) as a syntactic and semantic model of interaction based on the following principles:

- Continuous representation of the objects of interest.
- Physical actions instead of complex syntax.
- Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
- Layered or spiral approach to learning that permits usage with minimal knowledge.

Many interactive systems have been implemented based on the principles of direct manipulation. Editors are probably the most popular of these systems: general purpose editors (e.g. text editors, bitmap editors and structured drawing editors) as well as domain specific editors (e.g. music editors, CAD systems) all use, at least partly, direct

manipulation interfaces. The concept of WYSIWYG system (what you see is what you get) is often associated with direct manipulation editors. A WYSIWYG editor displays on the screen a representation of the document that is identical (or close to) the printed version of that document. In fact, the full power of direct manipulation as an interaction paradigm is obtained with WYSIWYG systems since their combination enhances the user's illusion of editing a real (paper) document.

Since direct manipulation interfaces have proved very effective for single-user applications, it is tempting to apply their principles to multi-user applications. Indeed, a number of existing real-time groupware systems have a direct manipulation interface. BoardNoter (Stefik *et al.*, 1987a) and Commune (Minneman and Bly, 1991) are bitmap editors used as sketching tools in design sessions. GROVE (Ellis *et al.*, 1991) and SASSE (Baecker *et al.*, 1993) are cooperative text editors that use conventional select-and-edit direct manipulation models. MMM (Bier and Freeman, 1991), CaveDraw (Lu and Mantei, 1991) and GroupSketch (Greenberg and Bohnet, 1991) are drawing tools that use a direct manipulation model close to that of MacDraw. The commercial product Aspects (Group Technologies, 1991) includes two direct manipulation editors: a text editor and a diagram editor. GroupDesign (Beaudouin-Lafon and Karsenty, 1992) also has a direct manipulation interface for editing multi-page diagrams. Cognoter and Argnoter (Stefik *et al.*, 1987a) are more specialized editors also based on the principles of direct manipulation.

These systems, and others, implement different variations of WYSIWIS (what you see is what I see) interfaces (Stefik *et al.*, 1987b). This concept captures the idea of sharing the representation of the document being edited among the users of the group. It is now well established that strict WYSIWIS interfaces present a number of constraints that are not compatible with most shared editing tasks and in fact most systems implement *relaxed* WYSIWIS interfaces. Stefik *et al.* (1987a) have identified four dimensions for relaxing a WYSIWIS interface: time, population, congruence of views and display space. This provides a useful grid for both the analysis and design of shared editors.

The WYSIWIS properties of a shared editor only address the *presentation* of a shared artefact to a group of users. They do not address the editing paradigm used by the group for modifying the document. This can be paralleled with WYSIWYG properties of a conventional editor, which only address the congruence between the display presented to the user and the final printed form of the document. This prompts us to look more closely at the editing techniques used in shared editors and their interpretation of the principles of direct manipulation. The above-mentioned systems implement several variations of the same editing paradigm. Some systems require turn-taking while others run an open floor. In most systems the users edit the document independently by means of an independent selection, but MMM (Bier and Freeman, 1991) for example allows several users to grab and move or resize the same rectangle simultaneously. Some systems implement ownership by allowing a user to only edit his or her objects (e.g. CaveDraw (Lu and Mantei, 1991)) while other systems provide access to any object by any user. These differences cannot be captured adequately by the sole concept of relaxed WYSIWIS.

We propose to extend the principles of direct manipulation to real-time groupware systems. Furthermore, since these principles are very general, we introduce a model for shared editors.

We call *shared manipulation* the extension of direct manipulation to groupware systems. Groupware systems have been defined by Ellis *et al.* (1991) as 'computer-based systems

that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment'. This definition and the numerous systems that have been implemented lead us to define shared manipulation by adding two principles to those of direct manipulation:

- sharing the objects of interest
- providing awareness of the actions of the group

The first additional principle accounts for the fact that, in a shared editor, the notion of shared environment referred to by Ellis *et al.* (1991) is reflected by the objects of interest referred to in the principles of direct manipulation. The objects of interest are the objects that the users may need to manipulate in order to carry out their task. Hence this principle can be interpreted in two ways: (i) the objects of interest must be shared in order for the users to work on the same data; and (ii) the shared objects that are needed by the users for their tasks should be objects of interests, i.e. according to the principles of direct manipulation, they must be made visible and directly editable.

The second additional principle accounts for the notion of common task, which requires coordination among users. Such coordination can occur only if the users are aware of each other's actions. This awareness cannot be achieved by simply sharing the objects of interest: when an object changes it is important to be able to know who is issuing the change and when, in order to understand why this change is being done and to communicate about the change. Although communication about the document and the editing task may use external media, e.g. an audio/video link, it must be based on a shared perception of the *actions* of the group, not only of their effects.

9.2 SLICE: A MODEL OF SHARED EDITORS

The principles of shared manipulation that we have introduced in the previous section are very general and can be interpreted in a variety of ways. They prove useful only when put into practice in a particular system. In fact the various systems that have been developed for cooperative editing already implement these principles more or less thoroughly. Our goal in this article is to introduce a unifying logical model for shared editors that (i) allows us to analyse existing systems and (ii) provides guidelines for designing new systems. By modelling the editor we can capture both the shared document and the interaction on the document. This reflects the two additional principles of shared manipulation.

Our model is called SLICE (sharing layers in cooperative editing). It consists of a set of layers (see Figure 9.1), each of which contains objects. The layers are stacked like a set of transparencies. The display is thus (conceptually) the superposition of the displays of each layer. For some layers this superposition is obtained by a simple visual overlay. For other layers this is not meaningful and the superposition is more akin to filtering or transformation: the layer displays the layers below it by filtering/transforming them.

Each layer can be either *shared* or *sliced*. If the layer is shared it is the same for each user and each user has equal access to its objects. If the layer is sliced there is an instance (or slice) of the layer for each user and each user only has access to the objects of his or her slice. In addition, depending on the user's and/or designer's choice, a user may see his or her slice or the whole layer, i.e. the superposition of the slices that constitute the layer. In a

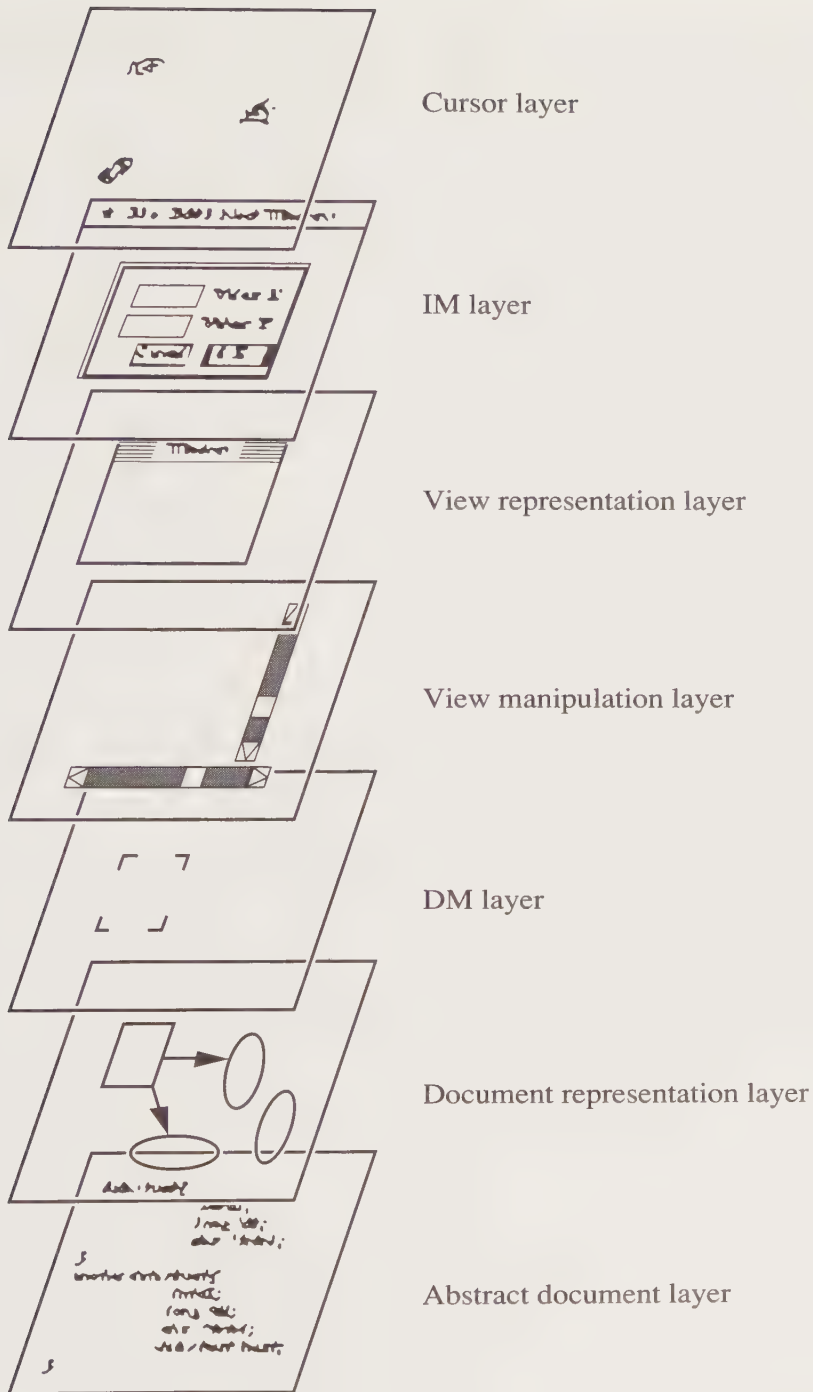


Figure 9.1 The SLICE model.

similar way to layer superposition, slice superposition can be a simple visual overlay or a more complex transformation.

We use seven layers, defined from bottom to top as follows:

- *Abstract document layer*: this layer is the internal representation of the document. It is a special layer since it has no graphical representation *per se*. In addition to the information representing the current state of the document, this layer contains specific data for supporting shared editing tasks, e.g. a history of commands with their authors. We call this data *group information*.
- *Document representation layer*: this layer holds the displayed representation of the document. The same logical document can be represented in a variety of ways. For example a formatted text can be represented in a WYSIWYG way such as in MacWrite or as a text with embedded formatting commands such as a LaTeX source file. Even a WYSIWYG representation can have variations, such as the size of the page, the number of columns, or the fonts used.
- *Direct manipulation (DM) layer*: the objects of the representation layer are manipulated using a direct manipulation interface. The DM layer holds the transient objects that are often used in direct manipulation systems to provide appropriate feedback of the state of the interaction to the user. This includes selection handles, rubber band rectangles, construction lines etc.
- *View representation layer*: this layer defines windows that display parts of the document (also called *views* of the document).
- *View manipulation layer*: the views are manipulated through objects displayed in this layer. Typical objects are scroll bars and various icons that allow one to move, resize and zoom in/out the views.
- *Indirect manipulation (IM) layer*: many commands in so-called direct manipulation interfaces are actually issued using an indirection through menus and/or dialog boxes. This is because direct manipulation is inadequate to support precise or abstract operations. Menus, dialog boxes, tool palettes and other artefacts of indirect manipulation are held in this layer.
- *Cursor layer*: this layer contains the cursor that follows the movement of the mouse. If several location input devices are available (e.g. a mouse and a trackball), each of them can be represented by a corresponding cursor in this layer.

Layers have already been used to describe the logical and/or physical structure of an interactive system. We have implemented a graphical toolkit called Xtv (Beaudouin-Lafon *et al.*, 1990) where several views can be superposed in a single window. Based on this approach, Fekete (1992) presents a five-layer model for describing drawing tools. A recent version of Interviews (Linton *et al.*, 1989) has introduced facilities to manipulate multiple layers. These facilities have been used in Groupkit (Roseman and Greenberg, 1992) to implement a cursor layer containing the cursors of the different users of a groupware application as well as annotations.

We have used our Xtv toolkit to implement multi-user applications with three layers, corresponding to the document representation layer, direct manipulation layer and cursor layer of the above model. The document representation layer was shared between the users, whereas each user had his or her own direct manipulation layer. The cursor layer was shared, and contained the cursor of each user.

The SLICE model presented in this article is the result of these experiments and of the design and implementation of GroupDesign (Beaudouin-Lafon and Karsenty, 1992), a shared editor for structured diagrams. GroupDesign was developed in order to better understand the user interface requirements for shared editors and the issues raised by their implementation. In contrast with most shared drawing tools that have been developed to date, GroupDesign is not meant to be used as a sketching tool to support meetings. Rather, it is designed to be used by groups of users working in a loosely coupled way on a document, similarly to the way people work with a shared text editor. A document is a set of pages connected hierarchically. Each page contains structured graphics similar to MacDraw. Objects can be connected together by lines or arrows, and objects can be attached to pages of the document. When an object is attached to a page, double-clicking it opens the associated page. GroupDesign uses colour to identify users: each user is assigned a colour displayed in the Users menu. Several groupware features change the colour of objects in the document to convey information on the group's activity.

In order to illustrate the SLICE model, we use GroupDesign as the main example. We show how the groupware features of GroupDesign (e.g. echo, localization, private editing) can be interpreted in SLICE. We also show how SLICE can describe features that are present in other shared editors. We proceed by discussing the possibility and implications of sharing and slicing each.

9.2.1 Abstract document layer

Since this layer has no graphical representation, it plays a special role. Sharing at least part of this layer is a basic requirement, otherwise, the users would simply work on different documents. Parts of the abstract document may not be shared between the users. This allows each user to edit a private part of the document, usually in a separate window. In this case, however, private parts are disconnected from shared parts.

In GroupDesign the notion of *private editing* allows a user to edit a part of the shared document privately. This corresponds to a transaction style of editing: changes to the abstract document are visible to the other users only when they are committed. Committing the changes may create conflicts that have to be handled. A locking mechanism (not used in GroupDesign) can be used to avoid conflicts at the expense of limiting access to the document by other users. Private Editing addresses the issue of privacy: a user might not want the group to see the intermediate hesitating steps that lead to the final result.

9.2.2 Document representation layer

Sharing: sharing the document representation is not mandatory to implement a shared editor. For example, in a heterogeneous environment (Karsenty *et al.*, 1993) two users could work on a shared document using different interfaces. Even in a homogeneous environment, it can be valuable to allow the users to edit a shared document with their favourite editors. The drawback is that many features of a shared editor rely on using the same interface and unless users want to work in a loosely coupled way, it is preferable to work with the same document representation.

Slicing: one way to slice the document representation layer is to have different objects in each slice and use superposition to make all objects visible to each user. Such slicing restricts the editable objects to those in the user's slice. This corresponds to the model of CaveDraw (Lu and Mantei, 1991), where each user draws on a separate layer and the slices are overlaid so that all users can see the document but cannot modify data they did not create.

Another way to slice the document representation is to put all the editable objects in each slice and not to use superposition of the slices. This allows each user to control the display of the document, in particular with respect to the group information. Group information covers a wide range of possible information that can complement the document's abstract description for the purpose of group work. Of particular importance is the history of the document. This history contains the description of the evolution of the document since its creation, providing information about:

- Authorship: user who created each object and who last modified an object.
- Partial history: history of any sub-part of the document, including individual objects.
- Age: time when objects have been created and/or last modified.

Group information can be represented in different ways: using colour; using annotations, i.e. attaching text or graphic to each object; or using sounds, e.g. playing the name of the user who created an object when double-clicking on it. In GroupDesign we chose to use colour because it provides the user with the clearest view. There are two drawbacks to using colour: it is incompatible with other uses of colour in the application, and it does not scale well, i.e. colour coding users is not effective when there are more than a few users.

We designed three modes to represent the group information in the document representation layer. The *identification* mode represents *who* created an object or who last modified it: each object has the object creator's colour attribute. The *age* mode represents *when* objects have been created: the objects' colour attributes vary from red (recently modified) to blue. The *trace* mode represents *what* has been recently done: the objects leave a trace on the document. For example deleted objects leave a greyed mark and objects that have been moved leave a grey trace at their original position. Furthermore, those modes can be combined. An example is given in Figure 9.2. The trace mode shows that objects A and B have been previously moved, the trace of A is lighter than that of object B, indicating that A was moved before B. Finally, although this cannot be shown in black and white, A and B have the colour attributes of the user who moved the objects.

The above modes provide the user with permanent information about the history of the document. It is also possible to provide the user with the group information using a navigation system. In GroupDesign, the history function can be used to replay the history of the document. Navigation in time is controlled with a control panel similar to a tape player. The history can be filtered: one can observe the evolution of a particular area of the document, or only see the last actions of a given user.

The identification and age modes and the history function have been implemented in GroupDesign. The trace mode is a new concept that we plan to implement. Other systems have used similar concepts. SASSE (Baecker *et al.*, 1993) uses the identification mode, in the same way as we do but for a text editor. GROVE (Ellis *et al.*, 1991) suggested the notion of aged text, which is similar to our age mode.

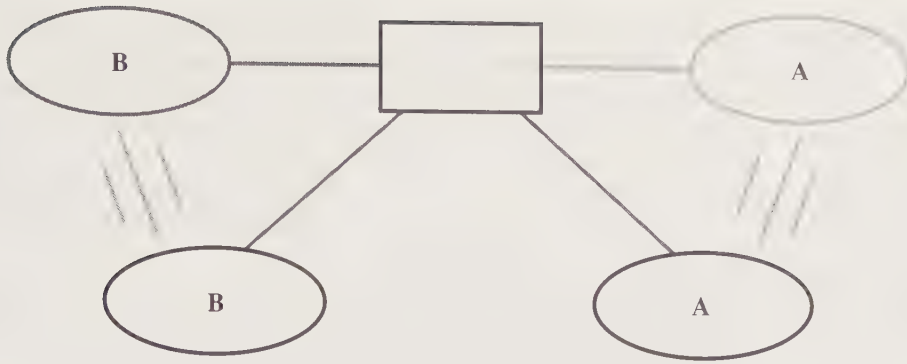


Figure 9.2 Trace mode.

9.2.3 Direct manipulation layer

DM interfaces use a number of techniques to provide the user with an appropriate feedback. For example, structured graphical editors use handles for the selection and manipulation of objects and ghosts while resizing or moving an object. Similarly, the selection is represented in a text editor as a blinking cursor and uses inverse video when spanning a piece of text.

Sharing: if the direct manipulation layer is shared, only one object can be selected and modified at a time. This gives a tightly coupled editing style, and also avoids conflicts. MMM (Bier and Freeman, 1991) uses this model: multiple users can simultaneously select and resize an object by dragging different handles of the same object.

Slicing: if the direct manipulation is sliced, the users have their own independent selection. As a consequence, conflicts may occur, since several users may try to modify the same object simultaneously in incompatible ways. If the slices are not superposed, the users do not see other users' feedback. This can make it difficult to follow other users' actions. For example, it is convenient to know which objects are selected by other users in order to coordinate actions and reduce conflicts.

Superposing the slices allows a user to get an awareness of the group work by visualizing the other users' actions. We can choose to visualize the other participants' slices directly, using a simple filter that transforms the other participants' feedback in their assigned colour so as to make it easier to understand. But, as shown with GROVE (Ellis *et al.*, 1991), users are moderately interested in other people's changes; thus those changes should be incorporated smoothly in the document. Thus, we advocate a more sophisticated filter that transforms the feedback into what we call an *echo*.

The echo is how a user sees someone else's feedback. In GroupDesign, echo is a two-step process. When a user starts a command, the other users see only an icon displayed on top of the object depicting the command being carried out (Figure 9.3). This tells them that something is going to happen. It also locks the object about to be modified for incompatible operations, thus avoiding conflicts. For example, an object being moved cannot be moved by another user, but its colour can be changed. When the command is completed, the icon disappears and an animation shows the result of the command (Figure 9.4). Such an animation is necessary to make the other users aware of what has happened

to the document. It is not disruptive because it integrates the changes smoothly in the document. Without animation, the change of state is hardly noticed, or users have difficulties in telling precisely *what* happened. Echo is independent of the way the operation is issued: whether it is direct or indirect manipulation, the echo is still the same. For example, if a user undoes a move operation using a keyboard shortcut, no feedback is provided and the object jumps to its previous position. This is a sufficient feedback for the user who initiated the action but not for the other users. Therefore, echo also uses animation so as to make the change understandable.

When sharing views (see next sections) in order to work in a tightly coupled way, providing the user with an echo similar to the direct manipulation feedback can be useful. In order to allow a smooth transition from working in a tightly coupled way to a more loosely coupled way, the echo should be adjustable. We have not experimented yet with such an adjustable echo in GroupDesign.

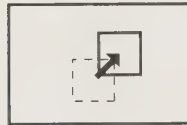


Figure 9.3 Echo: busy 'move' icon.

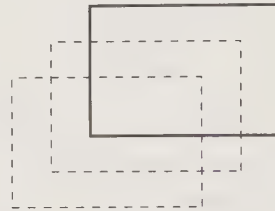


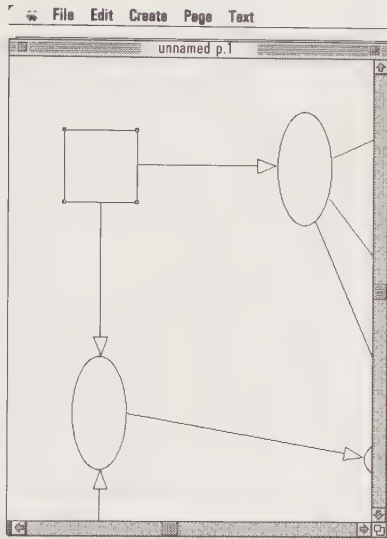
Figure 9.4 Echo: animation.

9.2.4 View representation layer

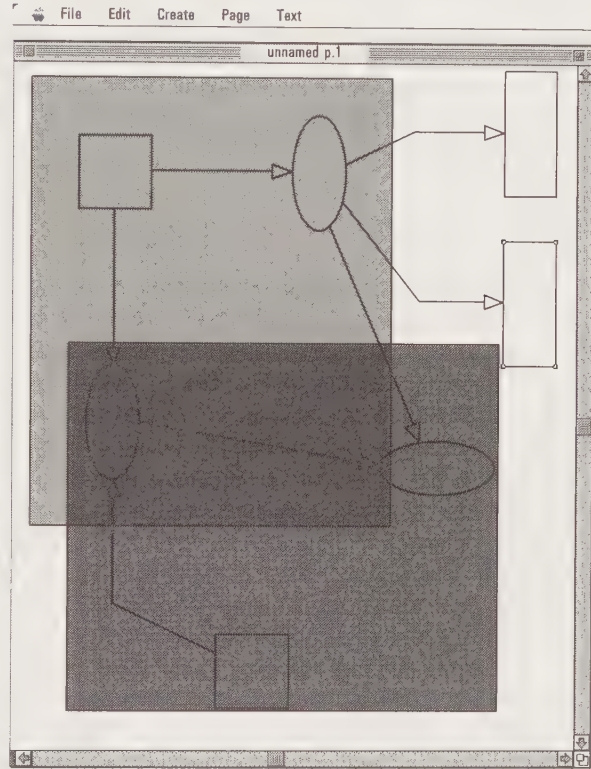
Sharing: view representations consist of the window frames that contain the view manipulation objects (see next section) and the document. Sharing this layer means that the view representation is the same for each user. This may not be practical (or even possible) in a heterogeneous environment, since the view representation usually depends on the look and feel available on (or imposed by) the platform, e.g. Motif, Macintosh, Open Look, NextStep.

Slicing: when slicing the view representation layer, users have their own view representation. Superposing the slices can lead to useful features. This superposition can be done in different ways, all of which use transformations. One method, implemented in SASSE, (Baecker *et al.*, 1993) is to use a *gestalt* view. This method consists in displaying a reduced view of the document where one can visualize each user's location. In GroupDesign we implemented a different method called *localization* (Figure 9.5). In this mode, the participants' front windows are displayed in the document itself as a rectangle with their assigned colours. Therefore, a user can see who is working in the same area as he or she is. This gives the users a sense of territory when they modify objects; they are aware of whether other users are currently viewing the changes.

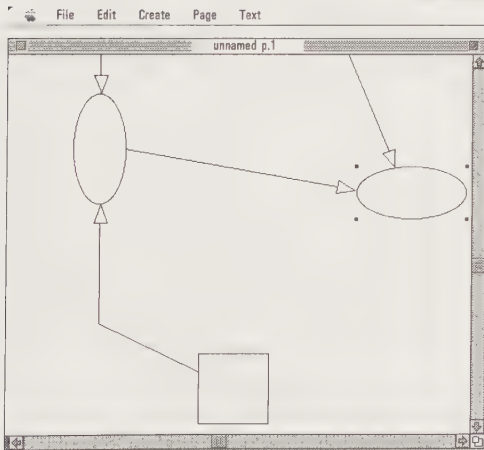
Whether to use localization mode or *gestalt* view depends on what kind of information the user wants: the localization mode gives a sense of who is seeing what the user sees, whereas a *gestalt* view allows the location of each participant.



Alain's view



Heather's view



Michel's view

Figure 9.5 Localization.

In GroupDesign we have complemented the localization mode with a function called *teleportation*: selecting a user's name in a menu teleports one's view to the area currently viewed by that user. Thus, each user can have both an independent view of the diagram and be able to synchronize views with another user.

9.2.5 View manipulation layer

Sharing: sharing the view manipulation layer means that all users visualize the same part of the document. This is useful when participants want to work in a tightly coupled manner. We call this mode *teleconference*, since during a teleconference users want to be able to point to parts of the document knowing that the other users can see what they see. This is also called the strict WYSIWIS paradigm (Stefik *et al.*, 1987b). Regarding the user interface, the issues are similar to those of the direct manipulation layer: feedback of the operations on the window needs to be translated into an echo for the other users. In GroupDesign we implemented a two-phase echo as with the direct manipulation layer. When a user starts dragging the scroll bar to scroll the document, the echo consists of displaying an icon on top of the bar, locking the scroll bar. When the user releases the mouse, the echo is an animation moving the document from its initial to its final position. The same principle applies to resizing or zooming the window.

An issue that should be addressed is whether or not to share window position. This can be difficult to implement if participants use computers with different screen sizes. However, this might be necessary. For example, in a teleconference a user might want to display different views of the document, and the way windows are positioned with respect to each other is important. One solution is to implement a window placement algorithm that depends on the screen size. For example, if a user with a large screen changes a part of the document that a smaller screen does not display, the view on the latter could be automatically adjusted in order to make the changes visible.

Slicing: when slicing the view manipulation layer, users gain freedom to move independently in the document. However, users lose awareness of the positions of each other in the document. One way to provide this awareness is to combine the slicing of the view manipulation layer with the localization mode offered by slicing and superposing the view representation layer. Another way, implemented in SASSE (Beacker *et al.*, 1993) is to superpose slices using a filter that transforms the objects in a useful way: each user's scroll bar is displayed so that the users know the location of the other users.

9.2.6 Indirect manipulation layer

Sharing: sharing the indirect manipulation layer means that there is only one active menu or dialog box at a time. One user can pop up a menu or pull it down from a menu bar while another user selects a command in it. Similarly all users can modify the same dialog box simultaneously. Moreover, an echo must be designed, similar to the other layers. This provides a strong coupling that can prove useful, for example, in teaching how to use a system. However, this sharing is difficult to implement because most of today's toolboxes do not take multiple input devices into account in menus and dialog boxes.

Slicing: slicing the indirect manipulation layer means that each user accesses the menus and dialog boxes independently. When the slices are not superposed we obtain the result of most shared editors, including GroupDesign: a user cannot see another user pulling down

a menu or editing a dialog box. Superposing the slices would gain an awareness of other user's actions: each user could see the other users pulling down menus and editing dialog boxes without being able to interfere with their actions. Appropriate transformations such as colouring would then probably be needed to avoid confusion.

The properties obtained by sharing, slicing and superposing the slices are similar to those obtained for the direct manipulation layer: sharing gives a tighter coupling, useful in teaching for example, while slicing gives a loose coupling. Superposing the slices adds awareness and can also be useful in specific situations such as making a demo of a software. So far, we know of no system that implements the sharing/slicing of the indirect manipulation layer. For teaching or demonstrating a system, people would rather use application independent tools such as Timbuktu (Farallon, 1988).

9.2.7 Cursor layer

Sharing: sharing the cursor layer means that all users have the same cursor, which is controlled by each user's mouse. This is neither practical nor easy to implement, although it could have some applications in tutoring systems. Sharing the cursor is usually implemented by a floor-control mechanism: users have to gain access to the cursor by asking for the floor. As a consequence only one user at a time can modify the document.

Slicing: slicing the cursor layer provides each user with his or her own cursor. Superposing the slices allows a user to see other participants' cursor activity. The other users' slices can be transformed by colouring their cursors or displaying the user names next to the cursors. Such multiple cursors (also called *telepointers*) have been implemented in several systems (e.g. Greenberg *et al.*, 1992; Group Technologies, 1991). We did not implement them in GroupDesign for the same reason we did not implement the shared indirect manipulation layer – the lack of support in the graphical toolkit.

Slicing the cursor layer and superposing the slices complement the sharing/slicing of other layers and thus can be activated automatically. For example, when sharing the direct and/or indirect manipulation layer, viewing every participant's cursor helps understand the other users' actions. As another example, in teleconference mode, it is necessary to show cursors in order to be able to point to an area of the screen while communicating via an audio/video link with other participants. However, it is not possible to superpose the cursor slices directly if the document representation layer is not shared, unless the cursor position is translated between representations.

The cursor layer can prove useful for other purposes. It can display the marks in a pen-based interface such as We-Met (Wolf *et al.*, 1992). It could also be used to write annotations or to circle things on top of the document as done in GroupKit (Roseman and Greenberg, 1992). Alternatively, a specific new layer could be devoted to annotations. This suggests that the layers we propose in the SLICE model are not fixed. In fact, the model is flexible enough to accommodate new layers or a different set of layers.

9.3 USE OF AUDITORY ICONS

The layers of the SLICE model focus on graphical representation and interaction. Graphics have some well-known limitations: the display might become overloaded with

graphical information, making it difficult to read by the user, abstract operations are difficult to represent graphically, and operations that occur out of view are not visible.

Auditory icons have proved useful for supporting cooperative work (Gaver *et al.*, 1991; Gaver, 1992). They help develop awareness between participants by associating appropriate sounds (Gaver, 1993) with actions done by the user. Editing a shared document in a loosely coupled way means that many invisible actions occur while a user is working on a document. Associating appropriate sounds to those actions help the participants know what is going on without using graphical representations that clutter the display and increase the cognitive load. Sound requires fine tuning to convey the appropriate semantic information and not disturb the users. For example, in the Arkola experiment, the system would at first emit continuous sounds to reflect the state of a simulated machine, but the users eventually found irregular sounds more appropriate.

In GroupDesign an audio echo complements the graphical echo described earlier. When views are not shared, two users can work on the same page of a document and yet have non-overlapping views. Since objects that are on the same page of a document are related, a user might want to be aware that another user is making changes next to his or her view. Audio echo associates a sound with every operation (e.g. create, delete, move, resize) and makes the amplitude of the sound proportional to the distance from the object to the view. Thus, when a user is starting to work closer to one's view, the sounds increase until the view overlaps the co-located user's view, in which case the audio echo stops and the graphical echo is used instead.

We plan to experiment with auditory icons in other situations. In the localization mode, users are made aware using graphics that their view overlaps someone else's view. An auditory version of the localization concept could be used to replace or complement the graphical version: when a user enters another user's view, the sound of footsteps coming in could be played – the sound of footsteps going away would be played when the user is scrolling to a non-overlapping part of the document. In a similar way, the sound of a door opening/closing can be played when a participant enters/leaves a session.

9.4 ARCHITECTURE

The SLICE model is a logical model that does not depend on a specific architecture for the implementation of a shared editor. The model, however, does impose some constraints on the architecture. In particular, the architecture must ensure a good response time of the system from the users' actions upon the interface. Many features that we have introduced (e.g. echo, telepointers) are effective only if the response time is short.

The two main approaches are centralized and replicated architectures. The centralized approach is easy to implement but does not give a good response time, since one process handles the requests from all the users. The replicated architecture gives a response time as good as for a single-user interface, but it is more complex to implement: a concurrency control algorithm must ensure that the abstract document is the same at each site. The replicated architecture also has the advantage of being more robust, since there is no single point of failure as in the centralized approach.

In order to obtain a good response time and a robust system, which we think are crucial for the acceptance of groupware, we have developed a *purely replicated architecture* for the implementation of GroupDesign. The complexity of the implementation lies in the

replication algorithm which is outlined below and described more thoroughly in Karsenty and Beaudouin-Lafon (1993). A purely replicated architecture consists of running a replica of the application on each computer. There is no central process for the coordination of the replicas and all the replicas are strictly identical. The replicas communicate by sending each other events through LocalTalk, using the facilities of the Apple Event Manager of Apple's new System 7.0 (Apple 1991).

Most authors recognize that a replicated architecture is worth the difficulties it raises. MMConf (Crowley *et al.*, 1990) has an architecture close to ours, but it encourages turn-taking floor control since the events are not guaranteed to arrive in the same order at all sites when running an open floor. On the other hand MMConf is a toolkit, whereas the groupware features of our system are currently integrated in the application. DistEdit (Knister and Prakash, 1990) is a toolkit for programming multi-user text editors. It relies on the ISIS toolkit (Birman *et al.*, 1990) for the distributed aspect of the architecture. This may cause performance problems, as stated by the authors, because of the concurrency control algorithms implemented by ISIS. GROVE also uses a replicated architecture (Ellis and Gibbs, 1989), with concurrency control being achieved by a technique called operation transformation: operations which arrive out of date are transformed so that they can be executed without disrupting the session. If there are n operations, this technique requires n transformation procedures, some of which are not trivial to write. Our system uses a similar architecture with a simpler concurrency control scheme. GroupKit (Roseman and Greenberg, 1992) also uses a replicated architecture but leaves the concurrency control to the programmer. Finally, Liza (Gibbs, 1989) is a toolkit that uses a central process with replicated clients, and RendezVous (Patterson *et al.*, 1990) is an example of a centralized application.

9.4.1 From design to GroupDesign

We neither built GroupDesign from scratch nor modified the code of an existing program; rather, we used an existing application, MetaDesign, which we modified through its programming interface, called Design/OA. MetaDesign is a graphical editor from MetaSoftware Corp. used to create logical diagrams made of nodes and connectors. The tool we used to build GroupDesign is Design/OA (Meta Software, 1989), a set of functions to customize, extend and tailor MetaDesign. It contains three categories of functionality to install filter functions to intercept and control user actions, such as creating a node; to extend and customize MetaDesign's menus; and to create and modify diagrams, the same way one would do manually with MetaDesign.

We have built GroupDesign with Design/OA as follows. Filter functions have been installed to intercept the user actions and broadcast them to the other replicas. We have added two menus, one for the control of the groupware features, the other to display the user names and their associated colours. Finally, we have used the functions that modify the diagram to handle the events coming from the other replicas.

Using Design/OA made our task much easier than starting from scratch. Design/OA makes it possible to completely change the features, look and feel of MetaDesign so that we have not been limited by the interface when building GroupDesign.

9.4.2 Managing the replication

The main advantage of a replicated architecture is the possibility of having an interface with a response time as short as for a single-user interface. This is achieved by using a high-level asynchronous protocol. The protocol is implemented in a software layer which is mainly independent of the application. This means that the technology we have used to turn a single-user application into a multi-user applies to many other applications, although we have not come to the point where we can provide a groupware toolkit for that purpose, such as DistEdit (Knister and Prakash, 1990), MMConf (Crowley *et al.*, 1990) or GroupKit (Roseman and Greenberg, 1992).

The protocol consists of events being sent by the replicas to describe the commands carried out by each user. An event contains an operation, defined as a triple (object, opcode, arguments). The protocol is asynchronous because a replica never waits for a reply from the other replicas. This requires a distributed algorithm that ensures that the events are processed at each site in a compatible way.

A precise description of the algorithm can be found in Karsenty and Beaudouin-Lafon (1993). The basic idea is to use a logical clock (Lamport, 1978) at each site to time stamp events sent by this site. Each replica keeps a list of received events. When a replica receives an event with time-stamp t , it should undo any operations triggered by events with a more recent time-stamp, then handle the received event, and finally redo the operations. However, if the operation contained in the received event commutes with the more recent ones, then it is not necessary to undo and redo those operations. Similarly, if the received operation is masked by a more recent one, i.e. if applying $op1$ and then $op2$ is identical to applying $op2$, then the received operation can be discarded. For example, if one user moves a rectangle and another changes its colour, the operations commute and thus can occur in different orders at different sites. If a user sets the colour of an object to red (event $e1$ at time $t1$) and another sets it to green (event $e2$ at time $t2 > t1$), a third replica receiving $e2$ then $e1$ will handle $e2$ and discard $e1$.

In GroupDesign, except for the creation of objects, for any couple of operations ($op1$, $op2$), if $op1$ and $op2$ are equal then $op2$ masks $op1$, otherwise $op1$ and $op2$ commute. Two operations are equal if their object and opcode are equal. In particular, this means that objects are independent, i.e. for any two different objects A and B , any operation on A commutes with any operation on B . This also means that operations are independent, i.e. the result of applying two different operations to the same object does not depend on the order of the operations. Under the above condition, a received event is always handled immediately, without having to undo and redo operations. The creation of objects is a special case, since it is impossible to apply operations on objects which have not been created yet. Events carrying such operations are stored in a separate list which is examined each time an object is created.

The management of logical clocks makes it possible for two events to be sent by different replicas at the same logical time, causing a conflict. Fortunately, the probability of such a conflict is quite low because the time window within which it can occur is very small. For example, user A could change the colour of an object to green, and user B could change it to red before the event is received from A. In case this were to happen, we have used the total ordering of events defined by Lamport (1978). As a consequence, the operation of one of the users is executed while the other one is discarded and the user notified.

In the private editing mode (when a user makes changes privately and commits them at a later time), conflicts may occur more often because they are detected only when the user commits. Hence, there is a conflict for each object modified by a user in this mode which has also been modified by another user. We have decided that the user in the private editing mode has his or her conflicting operations cancelled, in order to disrupt the least number of users. This is not a satisfactory solution, and we need more experience with the usage of this feature to devise a better one. The conflicts could be minimized by signalling such users to the group.

9.5 CONCLUSION

Research on real-time shared editors has reached a state where more and more systems exist and a few toolkits are becoming available. It is an appropriate time to start working on logical models that will eventually lead to better tools. This is what we have attempted in this chapter. We have introduced the notion of shared manipulation as an extension to the concept of direct manipulation and we have presented the SLICE model to describe shared editors. The model has helped us to define more precisely the shared manipulation paradigm, thus allowing us to identify a number of features of shared editors. We have implemented some of these features in GroupDesign, a shared diagram editor. We have also used GroupDesign to validate the concept of purely replicated architecture. With this architecture, we have obtained an efficient implementation at minimal cost by starting from an existing application. We have also successfully extended the approach to heterogeneous environments (Karsenty *et al.*, 1993).

We can see three main directions to continue this work. First, the SLICE model needs to be refined and evaluated on a wider range of systems. As shown by CaveDraw (Lu and Mantei, 1991) and GroupKit (Roseman and Greenberg, 1992), the concept of layers is already present in real-time multi-user editors. This chapter has shown that the concept of layers can bring much more to shared editors. Second, user testing is necessary to evaluate the features we have described. This will allow us to refine our model and more precisely define the needs of multi-user interfaces. Finally, a part of the SLICE model could be implemented as a toolkit so as to facilitate the development of shared editors.

REFERENCES

- Apple Inc. (1991) *Inside Macintosh*, Vol.VI, Addison-Wesley, Reading MA.
- Baecker, R. M. Nastos, D., Posner, L. R. and Mawby, K. L. (1993) The user-centred iterative design of collaborative writing software, in *Proceedings of the ACM Conference on Human Factors in Computer Systems (INTERCHI'93)*, 24–29 April Amsterdam, The Netherlands, Addison-Wesley, Reading MA, pp. 399–405.
- Beaudouin-Lafon, M., Berteaud, Y. and Chatty, S. (1990) Creating direct manipulation applications with Xtv, in *Proceedings of the European X-Windows System Conference (EX'90)*, November.
- Beaudouin-Lafon, M. and Karsenty, A. (1992) Transparency and awareness in a real-time groupware system, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)* 15–18 November, Monterey CA, ACM Press, New York, pp. 171–80.
- Bier, E. A. and Freeman, S. (1991) MMM: A user interface architecture for shared editors on a single screen, in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'91)*, 11–13 November, South Carolina, ACM Press, New York, pp. 79–86.

- Birman, K., Cooper, R., Joseph, T., Marzullo, K., Makpangou, M., Kane, K., Schmuck, F. and Wood, M. (1990) *The ISIS System Manual*, The ISIS Project, Cornell University.
- Crowley, T., Milazzo, P., Baker, E., Forsdick, H. and Molinson, R. (1990) MMConf: an infrastructure for building shared multimedia applications, in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'90)*, 7–10 October, Los Angeles, CA, ACM Press, New York, pp. 329–342.
- Ellis, C. A. and Gibbs, S. J. (1989) Concurrency control in groupware systems, in *Proceedings of the ACM Conference on the Management of Data (SIGMOD'89)*, 2–4 May, Seattle, WA.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, *Communications of the ACM*, **34**(1) 38–58.
- Farallon Computing Inc. (1988) *Timbuktu User's Guide*, Farallon Computing Inc., Berkeley CA.
- Fekete, J. D. (1990) A multi-layer graphic model for building interactive graphical applications, in *Proceedings of Graphics Interface (GI'92)*, May, Vancouver, pp. 294–300.
- Gaver, W. W., Smith, R. B. and O'Shea, T. (1991) Effective sounds in complex systems: the Arkola simulation, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, New Orleans, 27 April–2 May, Addison-Wesley, Reading MA, pp. 85–90.
- Gaver, W. W. (1992) The affordances of media spaces for collaboration, in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, 31 October–4 November, ACM Press, New York pp. 17–24.
- Gaver, W. W. (1993) Synthesizing auditory icons, in *Proceedings of the ACM Conference on Human Factors in Computer Systems (INTERCHI'93)*, April, Amsterdam, Addison-Wesley, Reading MA, pp. 228–35.
- Gibbs, S. J. (1989) Liza: an extensible groupware toolkit, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'89)*, 30 April–4 May, Austin TX, Addison-Wesley, Reading MA, pp. 29–35.
- Greenberg, S. and Bohnet, R. (1991) GroupSketch: a multi-user sketchpad for geographically distributed groups, in *Proceedings of Graphics Interface (GI'91)*, June, pp. 207–215.
- Greenberg, S., Roseman, M. and Webster, D. (1992) Human and technical factors of distributed group drawing tools, *Interacting with Computers*, **4**(3), 364–92.
- Group Technologies Inc. (1991) *Aspects: the first simultaneous conference software for the Macintosh, Version 1*, Group Technologies Inc., Arlington.
- Karsenty, A., Tronche, C. and Beaudouin-Lafon, M. (1993) GroupDesign: shared editing in a heterogeneous environment, *Usenix Journal of Computing Systems*, **6**(2), 167–195.
- Karsenty, A. and Beaudouin-Lafon, M. (1993) An algorithm for distributed groupware applications, in *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS'93)*, 25–28 May, Pittsburgh PA.
- Knister, M. J. and Prakash, A. (1990) DistEdit: a distributed toolkit for supporting multiple group editors, in *Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW'90)*, 7–10 October, Los Angeles CA, ACM Press, New York, pp. 343–55.
- Lamport, L. (1978) Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, **21**(7), 558–65.
- Linton, M. A., Vlissides, J. M. and Calder, P. R. (1989) Composing user interfaces with InterViews, *IEEE Computer* (February) 8–22.
- Lu, I. M. and Mantei, M. M. (1991) Idea management in a shared drawing tool, in *Proceedings of the Second European Conference on Computer-Supported Cooperative Work (ECSCW'91)*, 25–27 September, Amsterdam, Kluwer Academic, Dordrecht, pp. 97–112.
- MetaSoftware Corp. (1989) *Design/OA Manual*, MetaSoftware Corp., 150 Cambridge Park Drive, Cambridge MA.
- Minneman, S. L. and Bly, S. A. (1991) Managing a trois: a study of a multi-user drawing tool in distributed design work, in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, 27 April–2 May, New Orleans, Addison-Wesley, Reading MA, pp. 217–224.
- Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S. (1990) RendezVous: an architecture for synchronous multiuser applications, in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'90)*, 7–10 October, Los Angeles, pp. 317–28.
- Roseman, M. and Greenberg, S. (1992) GroupKit: a groupware toolkit for building real-time conferencing applications, in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, 31 October–4 November, Toronto, ACM Press, New York, pp. 43–50.

- Shneiderman, B. (1983) Direct manipulation: a step beyond programming languages, *IEEE Computer* (August) 57-69.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. (1987a) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, **30**(1), 32-47.
- Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tatar, D. (1987b) WYSIWIS revisited: early experiences with multiuser interfaces, *ACM Transactions on Office Information Systems*, **5**(2), 147-167.
- Wolf, C. G., Rhyne, J. R., and Briggs, L. K. (1992) Communication and information retrieval with a pen-based meeting support tool, in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, 31 October-4 November, Toronto, ACM Press, New York, pp. 322-9.

SECTION THREE

Applications of Collaborative Tools

The technological breakthroughs necessary to build computer-based shared drawing spaces are almost complete. However, what works well under controlled circumstances in the laboratory may not perform as well in the field. The transition from research principles and prototypes to robust commercial systems is a difficult one. The papers in this section not only describe experiences of walking the commercial path, they provide proof of the relevance of the work conducted by the research community. While the market is still somewhat limited, the ability to work together may still be the biggest payoff of the ubiquitous use of computer technology.

David Martin has successfully piloted the SMART 2000 electronic whiteboard (bitmap based) for multi-site meeting support from conception to commercialization. Chapter 10 describes the system, its applications and lessons learnt from the development process. He has overcome technical problems such as separation of display optics from touch screen, synchronization of Windows applications through a generic message handler, and self-correcting multi-point data connections through phone lines. Usability difficulties were overcome by moving to rear screen projection and interface developments. Experiences using SMART 2000 for distance learning seems to show that teachers can adapt classroom techniques and students and remote sites can receive effective instruction: a virtual classroom. Many other observations are reported when conducting same time, different place, meetings. Developing for the commercial market-place obviously brings challenges for quality assurance processes and testing that are not found in pure research. This paper is a refreshing documentary of a successful project.

In contrast to end-user systems described in other chapters, Dorab Patel and Scott Kolter have built and commercialized a robust tool kit for development of text-based groupware (Chapter 11). Their implementations of fairly simple tools lead to the discovery of what may be a minimal set of functions for constructing loosely coupled systems. They have addressed: determining what group resources are available, what different users are doing, multi-casting of messages and guaranteed replicated state consistency. All of these functions allow groupware developers to build complex applications with different work policies embedded. Developers benefit from canned solutions to difficult parts of groupware implementation, thus allowing them to concentrate their efforts on application-specific problems.

Structured graphic editors differ from bitmap editors because discrete objects, e.g. boxes and circles, can be created, modified and deleted. Mark Pendergast has demonstrated that the concepts that work well in simple applications can be successful in a robust commercial system (Chapter 12). By implementing a full-featured group graphic editor for the PC environment (Windows and a variety of network protocols) called GroupGraphics, he has taken large strides towards breaking down barriers to acceptance of CSCW systems – no longer do users need to complain that certain fonts, colours or objects are not supported. This paper also outlines a series of mechanisms and intended effects for group drawing, some of which are born out by his observations. Since Pendergast is expecting large groups (more than 15 people) to be able to use GroupGraphics simultaneously, he has built in several work protocols that are not often incorporated into systems intended for small groups. This work is clearly an excellent effort to include all current and practical CSCW functions in one system.

Designing interfaces that will be accepted by the user community has long been a difficult problem. The graphical user interface has compounded the exercise. David Miller, John Smith and Michael Muller have built the TelePICTIVE system (Chapter 13) to allow

user and designer to collaborate in the development of an interface. The software is based on a sound 'paper and pencil' methodology, typically practised with novice users. This approach to participatory design is novel and initial reports indicate that users had the most difficulty with not knowing what the other users were doing when they have an object 'locked'. This issue has led to the exploration of privacy questions and multiple levels of locking. This is a natural progression of the use of text-based group support systems for software requirements definition. Having users more involved in the design process can only lead to greater system acceptance.

SMART 2000 CONFERENCING SYSTEM: WHAT WE LEARNED FROM DEVELOPING AND USING THE SYSTEM

David A. Martin

ABSTRACT

The development of a new suite of technology aimed at solving a long existing problem: 'How do we provide non-intrusive interactive technology tools that allow groups of people in remote locations to work effectively together?' is discussed, including:

- The history of the development process
- Exploring the uses of the product – what worked and what did not.

10.1 INTRODUCTION

The author was involved in 1987 in the development and commercialization of a new type of liquid crystal display (LCD) technology. This was one of the first large-screen displays – 720 pixels by 480 pixels and 8.5 inches by 11 inches. The technology was originally developed at ITT in Tempe, Arizona, and could be used in either a projected mode on top of a standard overhead projector or used as a backlit display, much like a monitor.

In searching for applications for this LCD technology, a team of people at Imagineering Communications Inc. came up with the idea of creating an interactive touch display that would be tied via standard communications to rooms of people that would be able to see the same information at the originating site and interact with the information.

By 1990, the author had found the technology pieces needed to pull together a prototype for the product. With the introduction of the Microsoft Windows operating environment

in May of that year and the availability of high-resolution colour LCD projector panel displays, SMART Technologies, the parent company to Imagineering Communications, started on the process to bring a *practical* conference room solution to the problems of interacting at a distance with other workgroups.

From the first beginnings of this project, all team members were focused on developing an easy to use product that would work with existing software in use in the classroom and the office. We focused on customers that we knew and ignored the body of knowledge that had been built around computer-supported collaborative work (CSCW). As we will discuss in this paper, this gave us a blank sheet of paper in our design efforts and prevented us from being weighed down by a design paradigm as to what would work (or not work). This approach of ignoring published material on CSCW has also had a detriment in that it took us longer to understand some of the principles that would have been evident had we been insiders in the world of CSCW.

10.2 PRODUCT DESCRIPTION

SMART Technologies Inc. has created a family of real-time meeting support tools that can be used in one location or up to 50 remote locations.

The focus of this technology has been a new generation of electronic whiteboards which allow us to place a PC-based set of meeting support tools into a conference room without participants needing to focus on keyboard skills or new technologies. Whatever is written on the electronic whiteboard is saved on an electronic page. The electronic ink can be printed out on standard laser printers or on colour printers. The images can be saved on floppy disks and given to meeting participants as they leave as a record of the meeting results, or, the results can be sent over local area networks to anyone that needs to know the results of the meeting.

Dr George Goodman, a researcher at the Advanced Development Labs at Intel Corporation and a user of the SMART 2000 Conferencing and Presentation Systems, says: 'The whiteboard is natural to use; no long learning curve is required to get to know how it functions – it is intuitive, we all know it works. It is everything that the electronic copy boards should have been, it captures all the ideas of the meeting participants and stores them on the computer for later recall. When you print out the information, it is printed on good quality paper and it can be printed in colour. We actually prefer to use this device whenever we are holding a meeting'.

As well as being a large display, the SMART whiteboard is also an interactive display. This meeting support tool also functions as an input device to run Windows-based software. In short, it functions as a mouse. Anything pointed at on the screen is executed when touched; in effect it is a large touchscreen.

This feature is used most often to give presentations. The SMART software is run as a task alongside presentation graphics software like PowerPoint or Persuasion and allows the computer information to be highlighted with electronic ink. Because the images are displayed as 3 ft by 4 ft images, they can be easily seen by all meeting participants.

The third component of the SMART system is the ability to connect up to 16 remote sites over telecommunication lines. The product was first implemented over standard Hayes-compatible modems. Technology was developed to allow one conference chair site to be in control of all remote PCs that also had whiteboards. Using the Windows message

handling queue, messages are simultaneously sent to the chair's PC and to the remote PCs and executed in the message handling queue of each system.

This approach allows the use of low bandwidth telecommunication lines and gets the performance equivalent of running from a local hard disk, because the programs and the data are in fact stored on the hard disk of each PC. Information written on the SMART Notepad application is sent to remote sites with a requirement that a 1200 baud communication speed be supported. We have developed the concept of a network keyboard where the chair can input data through the keyboard and have the data go to all PCs and be recognized there.

Keyboard data requires a communications bandwidth of 600 to 1200 baud. Network and application control events require less than 100 baud.

10.3 APPLICATIONS OF SMART 2000

Our design goals for SMART 2000 were for the technology to be used in a classroom or conference room for:

- business presentations
- interactive business meetings
- distance education
- remote training
- brainstorming

Our assumption was that people in the above application areas probably already knew each other and were members of workgroups where they were focused on documents or information rather than people. Our target audience was very motivated in keeping away from travel and using technology to solve problems, learn something, or run a meeting.

In effect, SMART Technologies saw our business opportunity as being the development of new technology solutions for what had been called audiographics and what is now known as data conferencing or multimedia conferencing.

To date, we have successfully installed systems and support them in Australia, Japan, the USA and Canada. We have just announced the availability of our products in Europe and have seen lots of attention being generated by our innovative approach to the providing of conferencing solutions.

One of those industry analysts who have worked with and reviewed SMART 2000 is Rich Bader. In the 23 June 1992 issue of the *PC Industry Newsletter*, he wrote, 'I have found it – the compelling multimedia application'. In the article about SMART 2000, Rich Bader, former general manager of the PC Enhancements division of Intel, described the SMART 2000 architecture as being very robust and well architected. Rich also indicated that until he had seen SMART 2000, he had been very sceptical about how multimedia applications would actually be used by anyone to do useful work.

As a footnote to our design goals stated above, it is interesting to note that over 40 per cent of products sold by SMART Technologies have been for standalone applications where the interactive display is used by people in one location not connected to remote sites. As a developer, we saw that our technology could be used in this fashion, but did not expect to see much interest in this application area.

10.4 TECHNOLOGY HURDLES OVERCOME

The list below indicates the technical hurdles we had to overcome when developing our product.

- Large interactive display – first of class.
- Building a large surface touch display.
- Separation of display optics from touchscreen.
- Orientation of projected image to touchscreen.
- Reflectivity of projected image so it can be seen and written on the ‘hot spot’ problem.
- Setting-up of audio in a noisy environment.
- Developing rear screen optics.
- Applying large surface touchscreen to glass rear screen optics.
- Synchronization of Windows applications through message handling system.
- User interface that works in the conference room and on the desktop.
- Connecting multiple users to a meeting in a mixed communications environment.

10.5 WHAT WE LEARNED IN THE DEVELOPMENT PROCESS

We had some clear ideas and vision regarding the development of hardware and software systems that would allow people to interchange information over modem-based communications from one conference room to another.

As we developed the initial prototypes, first in MS-DOS and then in Windows, we knew that we wanted to develop products that could and would be used in real-life situations. This meant that we were driven on the one hand to develop commercial-level products that would be error-free and robust and supportable, while, on the other hand, we were always struggling with the reality that there was more work to be done.

Doing the next generation work meant that software needed to be rewritten and reorganized. This process was painful, because although we had been careful in our architecture selection and design, we were often telling our customers that they would have to wait 6 to 12 months for our next major release before they would see the features that they wanted. In other words, we chose to not continue development in the early versions of the early efforts. We did some low effort maintenance on the early releases and asked our customers to wait while we took apart our software and rebuilt our solutions on the new architecture.

Because we had chosen a business model that meant we needed to *sell* conferencing product continually during the development, it was appropriate that we focused on conference room solutions rather than desktop solutions. This allowed us to collect a larger number of dollars per sale and concentrate on helping the customer get the systems installed and people trained. While this was happening, we also managed to understand the customer needs and what we needed to do to improve the hardware and software in the next generation. This approach resulted in a very focused product development cycle tied to customer needs.

An example of this type of customer was the Department of Education in Australia. SMART had been preoccupied with the Windows synchronization technology. What we learned from customers like the Department of Education was that there was an opportunity for us to offer a simpler solution.

While we were focused on elegant software that allowed users to control remote Windows applications, our customers were looking for tools that they could use in the conference room as simple devices. They wanted a limited number of 'buttons' and interfaces. They wanted management people or teachers to have the ability to see the product once and be able to work on the system without prompting within five minutes. Our customers wanted to use our SMART whiteboard as a utility; they wanted anyone attending a meeting to feel comfortable that the device in the meeting was as usable as a meeting appliance can be.

As developers, our focus was initially different. We were caught up in the design of elegant solutions that needed to meet a knowledgeable user – a user like us.

This is a lesson that all of us that develop products and do research into new product areas often forget; our customers do not have the time to spend on learning our products that we imagine.

10.6 FRONT SCREEN VERSUS REAR SCREEN

In our initial conceptualizations of the SMART 2000 product, we knew that we wanted to project computer information on a touch-sensitive screen. We always saw this as a front-projected image because we felt that this was the lowest cost solution in the long run and, as users, we had found it to be very practical.

At trade shows and in meetings with potential customers, we found a number of people who were very excited about being able to work with a product that projected images from the rear. At first we were defensive about this and could not see their point of view. Eventually, though, we got around to seeing that there were some advantages to this design.

We conducted a thorough search of the literature to locate the new materials that we needed. Our front screen product used a black material that was electrically conductive. It was difficult for us to imagine that we would be able to find an equivalent material electrically that we would also be able to see through!

Nevertheless, after much searching we found the material we needed just by chance. As we built our first prototype, we found ourselves intrigued by the feeling of looking through a computer screen and writing on the screen with our finger.

As we continued our development on this product, we decided that we would first offer the product in a roll-about cabinet form. We chose to show this first device at an industry trade show for conferencing solutions in San Jose, California. As the date for the trade show got nearer, we realized that this was a more important product than we first realized and that we would need to pull out the stops to demonstrate our first rear-screen electronic whiteboard.

Our senior research engineer flew down to California from our Ottawa development facility and met the rest of the trade show staff. As we opened the boxes on the trade show floor, the tension mounted as we realized that this product was so new that it had never been assembled before in a working model!

Our efforts paid off and on Sunday, a full 24 hours before the trade show, we were able to experience for the first time the process of writing magically on the SMART rear screen writing board.

We found that this cabinet design solved the problems of routing of cables. It also took a noisy overhead projector and placed it away from the conference users where the fan

noise could not be heard. Of course, the shadow effect that we saw with the front screen went away and computer information could be seen at all times without it being blocked by people standing in front of the writing board. We also found that the writing surface was smoother than the front screen and easier to write on – just like writing on paper!

This product ended up being able to address four problem areas with the front screen projection system and has added significant functionality. Yes, it is more expensive, but it delivers more benefits than the increased cost. But our initial design focus and prejudices caused us *not to pursue* this product for at least a year, even though we had very strong indicators from the market place that this was wanted.

This product has become our most popular product for us and our customers. What is interesting is that our initial rejection of the concept was so very strong!

10.7 CONFERENCE ROOM TO DESKTOP

Our initial research and development was focused on the conference room. How could we provide conference room devices (i.e. large-screen interactive displays), that could be used naturally in a group meeting environment?

As we showed our first products to potential customers, we inevitably got asked, 'But I don't need one of these large devices when I am sitting at my desk. Do you have a solution for me to use at my desktop PC?'

We realized that there was an opportunity for SMART to release a product that would provide a lot of the same functionality on the desktop that we were demonstrating for the conference room. But there were some problems. Our architecture and interface was built for the conference room. There would have to be changes for us to run our software on a large number of PCs that exist on people's desks. The most important of these changes was related to device drivers. We had chosen to build a specific mouse and keyboard driver to replace the standard Microsoft drivers. The new SMART drivers allowed us to support the large interactive SMART display. But, at the desktop, these were proving to be very difficult to install and support. There were too many potential conflicts with other specialized mouse drivers.

Beyond the software architecture issues were the business issues. SMART was set up to provide solutions for the conference room. We did not see ourselves as a provider of shrink-wrap software with the ability to handle hundreds of support calls each day for a desktop product.

We decided to tackle this question in two ways. We wrote a specification for a new generation of SMART 2000 software that would allow us to do away with specialized drivers and we redesigned the interface so that it would be more Windows – as for desktop users. In effect, we unhooked the user interface from the rest of the software and created two interfaces; one for the desktop and one for the conference room. Both interfaces sit on top of the conferencing engine and can be used interchangeably.

During this time, we have moved from standard C to C++. We still develop for the Windows environment as our main market, but we now have projects to move our software to Apple Macintosh and Sun workstation platforms. We now have a code base that is split into three components as shown in Figure 10.1.

Nevertheless, our focus on conference room solutions resulted in a good conference room design but delayed our entry into the broader market for conferencing products.

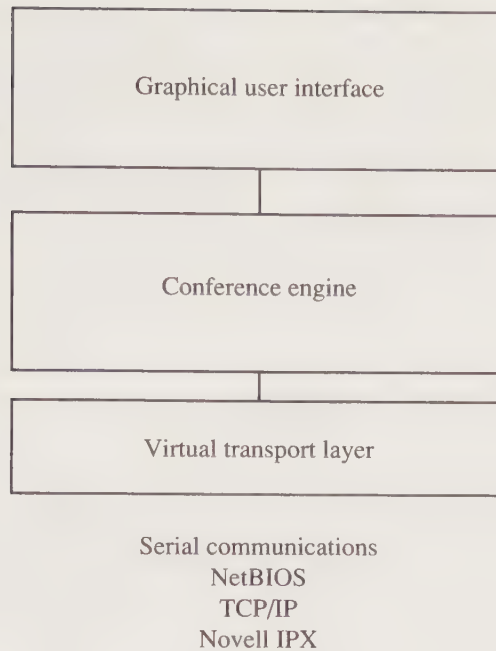


Figure 10.1 SMART 2000 architecture.

Should we have built in solutions earlier for the desktop? Probably. Could we have done this while we were focused on the conference room? Probably not.

We are really happy with the eventual result. From our experience with complex software design projects, you can never be certain about the final product design of a product for a new product category. Our approach was to build sufficiently robust solutions for our customers to use and monitor their usage and their comments. No surprises here: you must build solutions that work and then *listen really hard* to your customers to what is important to them and *implement improvements in a timely fashion*.

10.8 INDUSTRY PARTNERS

The second action item was for us to approach potential industry partners and determine whether there was a larger company that we could work with to launch a desktop product that would interoperate with SMART's conference room technology.

We discussed this concept with four or five potential partners and found a Fortune 100 technology company that was focused on releasing products into this market. They wanted to focus on the desktop; we did not. We wanted to build solutions for the conference room, they did not.

After nine months of negotiations, SMART was able to conclude a deal with this organization and start joint work on a new release of software that builds on the strengths of SMART's architecture and code base. So, three years after starting work on the

conferencing software, version 1.0, we have gone through three rewrites of the software and we are about to introduce version 4.0 of the architecture.

Our agreement with our industry partner includes long-term software co-development, joint marketing, sharing of relevant technology that is developed, introduction to strategic accounts etc. With a central position in the personal computer industry, our industry partner has the needed clout to launch a new product category and sustain the effort needed to educate the market about solutions that have been built and why they should be considered. This partner also has the financial staying power and senior management commitment to see the conferencing technology through its various phases.

10.9 MODEMS VERSUS LAN PROTOCOLS

Our initial focus was on remote meetings and training. To make our product as ubiquitous as possible, we chose to focus our development on an architecture that assumed that a telephone link was available between the remote locations.

By focusing on modem communication, we discovered that this met the needs of our customer base; for instance, the US government spends millions of dollars on telecommunications each year, but 78 per cent of the installed base of systems that are available is analogue telephone or POTS lines.

By taking the approach of focusing on software solutions that are low bandwidth, we found that we ended up with a different architecture of software compared with the CSCW software that has been built where it was assumed that the collaborative work would occur on local area networks (LANs) at speeds up to 10 Mbps. In our case by comparison, we focused on solutions where the available bandwidth is 10 kbps – a factor of 10 difference.

Once we had this framework for modem communication in place, we were slow to start the process of supporting additional network protocols. We could not understand why customers would want to use LANs as a transport medium. By local, we felt that users would be better to just go and meet the local person down the hall or on the next floor.

What we were slow to understand was that larger companies had invested lots of infrastructure dollars on interconnecting these local area networks and now someone connected to a network could interchange information with another person using the network protocols.

So we started the effort to develop support for Novell's IPX, NetBios and TCP/IP. Before jumping into this, we redesigned our communications layer so that we could easily add additional network protocols at a later date. We also built in error detection and correction, a SMART multipoint layer to support multiple users in a conference, flow control, and network status information that would be available to the application layers of the conferencing software.

Having built in support for these protocols, we are discovering that most of our corporate customers are able to use our conferencing products right out of the box on their networks. The networks can bridge or route the conference packet information straight to the destination of the other conference attendees. Because of our multipoint layer, it is possible over the network connection for up to 32 conference sites to join a meeting.

By having focused initially on low bandwidth telephone connections, the performance that we are able to bring to higher speed networks is excellent.

In retrospect, we should have built support for networks into our design earlier. However, the discipline brought by only having 9600 bps of bandwidth available has allowed us to create a unique and better product.

10.10 WHAT USERS LEARNT IN USING TOOLS DEVELOPED BY SMART

This section lists some of the human considerations in our product.

- Audio quality was of first importance.
- In a group environment, there are many distractions, extra pressures. This requires that any solutions need to be as simple as possible to use; the tools must be intuitive.
- If tools break down, then there is frustration and no use. People are busy and do not have time to learn new things, so tools must look familiar if they are to be adapted widely.
- Software must work with familiar software tools. People spend time learning tools like Excel, so they do not want to learn a new tool that may be similar but not have the same functions. Technology components to be added to existing office products for widest acceptance should work with existing PCs, audio systems, software and networks.
- For international markets:
 - English works well in most markets for this type of product.
 - Specialized versions were needed for Japan and France.
 - Japanese require higher levels of integration.
 - Australian customers want to put the pieces together themselves.
 - ISDN is not ISDN is not ISDN is not ISDN (Australia is different from North America is different from Japan is different from Europe).
 - Overseas communications requires extra effort to get modems communicating.
- Tools that appear to work locally need to work seamlessly over the network. One early customer saw a SMART presentation a full year after they had been using the product. The SMART engineer was conducting an information session to three sites using the SMART whiteboard products. Instead of pointing to the displayed images at the local sites using his finger, he used the remote pointer feature to make certain that the remote sites also saw what he was trying to emphasize.

The people at the remote sites had never seen this used before, and it was a revelation to them. They found it very useful and requested that all future presenters use this feature.

For SMART, this type of thing underlined two things:

1. How could we make our interface more natural so that pointing at the screen caused the remote sites to 'see' the point being made?
 2. No matter how intuitive an interface, it is necessary to train new users on the features of the product.
- 80/20 rules:
 1. Most multipoint conferences are 3–4 sites, not 100 sites connected simultaneously.
 2. Most of the features of a new product will not be used by most clients – keep it simple.

- Identification of who is writing is not that important. This feature of CSCW systems is nice but not necessary. Humans are good at interpreting what is being said and displayed. Our research indicates that who is writing can often be distracting.

In most business conferences, the protocol of when to speak and not to speak is well understood and adhered to. For distance education, the lecturer speaks most of the time (> 80 per cent), and controls who gets to ask questions when. This works best when the lecturer is asking conference attendees questions on what is being displayed on a regular basis.

10.11 DISTANCE LEARNING APPLICATIONS

This approach has allowed us to install systems at a variety of distance learning locations and allow teachers to give classes at up to three to four sites. Recent experience at the University of Nevada has shown that:

- Teachers can get operational in a very short time.
- People can start using a new system if it follows familiar objects.
- People can start using a new system if it has an interface that they are already familiar with from previous use at the desktop.
- Distance learning can be accomplished where students at remote sites can do at least as well as students at the lecturer's location (in our current experience, better).
- Two-way interaction is important, not just observer status if you want to optimize the process.
- Meeting facilitation becomes more complex with each additional support technology.
- Preparation is important for best results, but *ad hoc* tools are useful.
- Having technical people available is useful; they should help the meeting by addressing any technical glitches, but should do so in the background if possible.
- Restarting a conference that has been interrupted should be possible without extra effort.
- Having a course on protocols to use when using a new technology is helpful. For example a University of Ottawa video tells students how to identify themselves when asking a question.
- Moving the lecturer around to different remote sites is a useful thing. Broadcast from each remote site.
- Lecturers seem to do best when they teach to a local student group, not an empty classroom.
- Technology works best when the users do not have to translate too many of their regular techniques to the new environment.
- Communications-based technology allows you to train facilitators at remote sites.
- Prepared models that could be used to help the group problem solving process are extremely helpful.

The experience at the university shows that some teachers can easily adapt their classroom techniques to the SMART system with no apparent difficulties. Students at remote sites can get an effective education without travel to a city. Resource people with knowledge can be brought into a virtual class setting simply by dialling them.

Two aspects of the SMART system's features have been used to date in the various classes that have been held. In the first class that was held in medical pathology, various 35 mm slides were digitized and displayed at each of the class sites. The high-colour, high-resolution images were able to be talked about by each of the students at the various sites. The information on the slide could be pointed to or marked with a colour marker and commented on. Students participating in these sessions have indicated that this was very effective and a great way to learn.

Other sessions have been held where there has not been as much preparation, but the lecturers have simply used the electronic whiteboard as they would conduct a regular lecture. In this situation, they use a number of classroom techniques, such as asking students in the remote sites for input into the process. This can be done by students inputting information on their whiteboards, but is most often done by the lecturer listening to the verbal response and writing the information on the whiteboard at the main lecture site.

Having obtained reasonable performance over low-speed telecommunication lines, we are now turning our attention to the support of higher speed telecommunication links. This will allow us to provide even more *ad hoc* capabilities to the SMART system and provide tools like colour and black and white scanners with near real-time network access.

10.12 SAME TIME, DIFFERENT PLACE, MEETING USAGE

The SMART 2000 system has been used in a second major application area; that of connecting groups of people who wish to have an interactive meeting. Examples of people who are users include Intel Corporation's Advanced Development Labs and The Ontario Lotteries Corporation.

In this scenario, co-workers, who probably already know each other, are able to connect whiteboards in their meeting room to whiteboards in other remote meeting rooms. By creating a conference call for voice communication and a second call for the data/graphics information, meeting attendees are able to both see and hear what is being discussed.

Items displayed on the local meeting whiteboard are also on the remote whiteboard display. Items pointed at can be seen at the remote location. Information written on the local whiteboard using a white board pen can be *simultaneously* seen at the remote display.

(Note: *simultaneously* means for our purposes within a second from the time it is displayed on the local screen. Assumption: data call originates and terminates within North America.)

We have discovered that the following items need to be kept in mind when using our technology in this mode.

Audio is very important; it cannot be allowed to interfere with the process

Some tests that we conducted at the Institute for the Future in Menlo Park, California, indicated that audio must work seamlessly. When it does not work, the visual tools become close to unusable. In the Institute for the Future tests, two teams were assembled in remote locations, one in a standard size conference room, the other team in a large cavernous room with terrible 'bouncy' acoustics.

To make matters worse, the two audio conference units chosen were not compatible: one was an older half-duplex audio system based on a convenor concept, the other was a

newer generation, pseudo-full-duplex room audio system. These two devices tried to monitor the telephone line levels and 'fought' each other so badly that audio was impossible.

The distraction that this caused was so great that the benefit of the visual part of the conference was greatly reduced.

Lesson: make certain that audio support for the group conference is acceptable before adding visual conferencing. The audio quality must not be distracting. For an interactive meeting such as occurs in business, it is likely that full-duplex is required. For lecture or presentation situations, it is possible that half-duplex audio speaker/microphone situations are sufficient because one speaker speaks more than 80 per cent of the time and interruptions are not important. This last point is perhaps moot as DSP-based group full-duplex audio solutions are now available at reasonable prices.

Meeting facilitation becomes more complex with each additional support technology

In some technology-assisted meetings, the facilitator and the technologist are the same person. In most meetings, the conference chair is the facilitator, with technical help available if needed. In our experience and in our expectations, it should be the situation that the chairperson simply calls a meeting and runs it without help.

However, familiarity with technology and a new way of running a meeting can vary substantially. During the early process of learning how to run a meeting, it is wise to make certain that a technical person is available to troubleshoot any problems on an exception basis.

If the number of new technology items is large, such as new audio or new modem or network links, then a gradual run-in process should be recognized as necessary. The problem here is that trying to get work done and trying to make the technology work can run counter to each other: one distracts from the other. While we expect our non-intrusive conference room technology to be self-evident in its operation, the level of impatience of a business person can be high, and therefore any technology needs to be introduced gradually.

In the education environment, we have found a wide range of teacher acceptance of technology. For some, the Windows interface of SMART 2000 was a very natural environment because they had extensive experience with this on their desktops. For others techno-fright took over, and they needed hand-holding to get through the first few lectures that they gave. From experiences at the University of Nevada and the University of Ottawa, it is felt that a technical person should be available for all distance education classes to solve any issues that may arise. In the case of the University of Ottawa, this technical person monitors all audio and data links and on an exception basis is able to solve any disconnects before they are noticed. In the scale that they operate, this one person is responsible for handling 3 to 5 classes during any time period. This represents over 42 courses held last semester.

Have a break point scheduled in the proceedings to evaluate how things are going and make course changes if needed

With data and audio conferencing, there are fewer feedback indicators than exist in face-to-face or video conference meetings. Therefore, it is important that the chair or facilitator

solicit opinions on the direction of the meeting at regular intervals to see whether what is being discussed is relevant.

Develop team-building and trust-building before introducing meeting support technologies

If the electronic meeting tools are being used to manage a new workgroup, then all team members should meet in person early on in the process and get to know everyone before relying on the electronic technology for the ongoing meetings.

This is common sense, but crucial for the success of the meeting process. Most humans are good at visualization of the other people around a conference table. This is much easier when a mental picture of the other people is reinforced by actual face-to-face meetings at the beginning of a project and can be helped by meeting during a project.

The idea is not to eradicate travel completely, but to supplement the face-to-face meetings with conferencing systems so as to cut down on travel and still get productive work done.

In SMART's case, we manage an engineering team in Ottawa and another two thousand miles away in Calgary. Most of the telecommunications and interactive display hardware is done in Ottawa and the software application development is done in Calgary.

From our experience, we found that it was important to meet at the beginning to familiarize every member of the project team with each other. During the project, we met over the phone a number of times using our own technology. We were able to share designs, make notes and record the progress of the meeting as milestones were achieved, etc. However, we still found that it was necessary to have senior team members travel to the project sites to meet face-to-face. Depending upon the state of the project, this was done every six weeks in a hectic period or every 3–4 months in a calmer period, we have been doing this for four year now and have found that running design teams remote from each other is not easy: it is something you have to work at, but it is increasingly how work will be done.

At the universities we work with on distance education courses, we find that the distance education programmes that are most successful are the ones where the instructor visits the remote classrooms at least once during the 13 week semester. In the case of our technology, the professor is able to originate a lecture from any site, so the remote site simply becomes the originating lecture point. This has proved effective in getting the remote students involved with the course and allowing the lecturer to know 'who is out there'.

Carefully consider room size, acoustics and suitability to task

For group meeting situations, it is important to review the meeting room configuration before you try to stuff new equipment into the room. Does it have adequate cooling? Does it allow for integration (separation?) of audio with data display? Does it have enough electrical and telephone (network) support?

For temporary meetings, this last problem, has occurred on most occasions where we have set up. The analogue phone connections that we normally request for modem connection are usually routed through a digital PBX and are therefore not usable. We have often found in temporary situations that small things have not been planned for. This

results in frustrations ranging from such events as someone picking up an extension phone during a data conference to trying to squeeze 20 people and equipment into a room designed for five people.

Beyond the temporary room situation, if you are involved with planning a room for regular data conferencing, please contact your equipment suppliers: they will likely have some good ideas to share with you.

Group size is important; for intense communications, have a maximum size of eight people at each site

Do not expect to overcome the basic laws of running a meeting just because you have meeting support tools. You will be very productive with group data conferencing technology if you restrict the number of group participants in those situations where you need lots of participation.

One idea that might work is to separate the meeting into two or three sequential events. Try recording the first event using standard VCR equipment and a VGA to NTSC (or PAL) recorder to capture screen information and audio to tape. Then play this back for the participants of the second meeting so that they have an information base to work from.

Focus on meeting topics: is the task at hand important to the participants?

When we have run tests of our data conference products, we found that people need to be motivated to work together in a meeting. We had one meeting at the Institute for the Future where the meeting participants at remote sites were asked to solve a marketing problem for a large consumer products company. The meeting attendees at both sites were not interested in solving this particular problem and sat back and did not participate. The data that we were trying to track on their use of the technology proved useless.

If you have busy people participating in tests of new technology, do not waste their time!

Can you have fun while problem solving?

While trying to create solid solutions for meeting room tools that aid in productivity, we have found that other meeting tools and processes are nowhere near as fun as using our interactive display technology!

We have sometimes described our technology as colour Etch-a-Sketch. Can having fun improve the meeting process? Or does it get in the way.

We are still doing research on this. It is conceivable that once the novelty wears off, the better tools available in the electronic meeting room will make the meeting more effective and people will be able to visualize issues better – and still have fun.

People in problem-solving situations do best when the technology is in the background and non-obtrusive

Our experience with the technology is that if you are trying to sort out the interface issues with a group very familiar with the product, you will be blocked from achieving the best

solutions. That is why we focused on the whiteboard metaphor early on so as to give a familiar look and feel to the product.

Since then we feel that our efforts to improve functionality have sometimes resulted in too high a level of complexity, and we have had to scale back our capabilities or offer more than one level of interface – one level for the expert, one for the simple user.

People in problem-solving situations do best when they follow a group leader or conference chair

Having the ability to force the same view is important for a lot of situations: anarchy creates confusion. This is self-evident for those who have studied meeting facilitation techniques. With our type of conferencing technology, we have experimented with no control by the chair to absolute control. Our current feeling is that we must provide levels of both ends of the spectrum. Some meetings work best with control, others are best run with peer-to-peer control.

Passing control of the chair is important

Passing control of the meeting from the chair works in most corporate settings, where other hierarchical corporate organization protocols underlie the meeting protocols. But in some situations, the passing of the gavel would be best addressed by having the ability to take back control rather than simply requesting it back.

A meeting facilitator can be useful

We have experienced some meetings where we used a professional facilitator. In one case, this facilitator was a trained graphics facilitator. The quality of the meeting was very high. The facilitator was able to capture the tone of the meeting very well and use graphics to help the participants visualize the meeting objectives and the intersection of the possible solutions.

Although I do not recommend that you use a professional facilitator for all meetings, you should try a facilitator in some meetings as a learning experience.

A facilitator with a graphical orientation is best.

Two-way interaction is important

As in most learning experiences, it is best to do, not just watch. Therefore, it is best to structure technology-supported meetings so that both sides of a conference are contributing to the discussion with both audio and written material.

In one situation, those of us in the development team at SMART sat in on a course on Biological Media given at the University of Nevada. We were surprised by the amount of interaction going on between the students and the professor. Of course, this was nothing more than what a good college professor would do in following a Socratic style of teaching: draw something on the whiteboard or show an image to all course attendees and then ask specific students to comment on what they saw. Get your audience involved if you want the most success!

Preparation is important for best results, but *ad hoc* tools are useful

Our initial product design assumed that people would prepare their presentation material ahead of time. For some meetings and training sessions, this model fits exactly the way in which conferences are run. But most meetings and lectures are run with a combination of *ad hoc* and prepared material.

Having changed the model to include both prepared and *ad hoc* material, we have had a much better reception from our user community.

Of interest though is the finding that some users of the technology get into a rut and only use the *ad hoc* feature and do not do advance preparation of lecture material. This requires some thinking on the part of school administrators and course planners.

Having technical people available is useful

Technical staff should help the meeting by addressing any technical glitches, but should do so in the background if possible.

If too many new items are introduced, users regress

Although most of us are of the video and television era, where fast-moving images are standard fare, it is possible to see information overload for new users of data conferencing technology. In this case, users will regress to using audio only.

This phenomenon is similar to the experiences of anyone who has gone skiing. On the easy slopes, it is easy to look like a professional skier. On steeper, bumpier pistes, most of us regress to just trying to get down the hill – without consideration of style!

If you are responsible for introducing new conferencing technology to your organization, realize that this will occur. It is a problem that can be solved with time and some training.

Some meetings require anonymous input, others require knowledge of who is originating the idea

As system designers, we have discovered that there are many meeting styles. Some work best where the identity of the remote contributor is hidden. In other cases, you *want* to know who the author is of a particular idea!

There is considerable discussion of this question in the literature. In our case, we first implemented the anonymous input model and found that we did not have a lot of requests for us to provide an identification label. This may have been because our early experiences have been with members of groups working together rather than groups working together composed of individuals. Our research continues.

Cross-platform solutions are important

We chose to implement our solutions on one of the most important graphical-based systems – PCs running Windows. Having done this, it is clear that everyone who has seen the technology and does not have a PC has immediately asked us if SMART 2000 runs on their particular computer platform, say a Macintosh or a Sun workstation.

With the proliferation of computer platforms in universities and corporations, it is clear that any software provider of conferencing solutions must provide answers on multiple platforms. In our case, we are working on Apple Macintosh and Sun workstation software versions.

The world of conferencing is the world of integration of computers and telecommunications. It is world similar to the problems of telephony – it will not grow unless my telephone talks to yours. This problem will be solved in a number of ways, standards of course being one important aspect. It is clear though that software providers must provide solutions for all important platforms to be successful.

Provide links to other data sources, e.g. Macintosh, Mainframe, LAN

If your system provider cannot provide a conferencing solution, then there must at least be a way of getting information from the data source in a format compatible with the conferencing system.

Provide printing as part of the tools, leave the meeting with electronically captured information

Our early adopters like Intel Corporation found our meeting tools very useful when they were able to leave the meeting with a complete record of the meeting in hard copy. This gave senior management tangible evidence of the benefits of the technology.

Today, we are investigating the benefits of using pen recognition technology to allow recognition to occur in the meeting to computer-readable formats (ASCII text). This should speed up the transcribing process as the recognition algorithms get more robust.

If keyboard activity is needed have a specific person assigned to this

This is common sense again. Do not try to have a fully-fledged meeting participant try to participate and keep notes. By having a meeting note-taker work on a separate PC, you can get additional information in a different format which can be valuable when read back into the conferencing system.

Group memory is important (e.g. tear-off sheets of written information)

We have all had the experience of participating in a brainstorming meeting where all sorts of ideas are considered over a day or two. In most of these meetings, someone writes down ideas of the group on a large paper pad, tears off the sheets and tapes them to the wall. Since this information is usually displayed in a chronological fashion from left to right, meeting attendees can often get visual cues on what is displayed in order.

SMART has experimented with an electronic version of this idea. We attached a 'poster printer' to a SMART whiteboard and have been able to take ideas written on the SMART board and print 2 ft by 3 ft paper output which we then tape in order on the wall.

Because the original information is still recorded in the SMART whiteboard PC, this information can be reordered or changed and reprinted.

Prepared computer models that could be used to help the group problem solving process are extremely helpful

Our experience at the Institute for the Future was very positive with respect to prepared information. By developing models of group solving, all participants were able to see the different solutions from a simple computer model. Both sides could manipulate the computer model and see the results of their efforts. This resulted in a clearer understanding of the solution process from meeting attendees at both meeting sites.

Pen computing may be important to group meetings

Our experiences of developing conferencing software for the Japanese market indicates that problems with keyboarding of information in Kanji might make the pen recognition technology built-in to the SMART whiteboard a good solution for capturing of meeting notes.

We have found that Japanese business people tend to write Kanji very precisely. This fact, coupled with the good quality of the Japanese language recognizers results in the electronic whiteboard being a great device for the meeting room in Japan and other Asian countries.

10.13 SUMMARY

SMART has created the first of what will be a family of products that allow individuals or groups to interact with other individuals or groups over telecommunication networks. This sharing of information is delivered in an innovative way by tying together touch-sensitive whiteboards with displayed computer graphics. By providing an easy to comprehend user interface, we have been able to have teachers give courses remotely, or business people have interactive meetings, even in geographically separated locations.

We believe that our users are attracted to our software/hardware integration because:

- It is inherently not difficult to understand and use.
- What is on my screen is on your screen.
- Write here, see there, is a given.
- The whiteboard metaphor is something even your boss understands.
- It does not force people to learn a new software interface; they can use their existing software.

Developing technologies for a new application area is an iterative journey. I do not believe that anyone is smart enough to figure out all the issues ahead of time and build a robust architecture that allows workgroups to operate on problem solving in a real-time conference. The development of software for this task must be done – and redone. You must have the patience to build solutions, verify the usage of the software and rebuild the software again and again until it works.

In other words, a clear objective must be set, but you must be prepared to change your ideas as you listen to the needs of the market.

You must also be prepared to sift through often conflicting ideas about what are the needs of the market and make decisions about which areas to emphasize and which areas to focus on first.

Developing for the commercial market obviously brings challenges for quality assurance processes and testing that are not found in the pure research area. You must separate out the testing from the development and conduct extensive tests of the software in multiple computer and communications settings.

While the above is easily said, this process of coming to a software level of robustness that matches the documentation level is probably the hardest thing to achieve while product development is still active. This is probably the most difficult thing for those in CSCW research to come to grips with as the industry matures and products get released into the hands of customers.

This journey of 'pushing the envelope' has just begun. What is clear is that we have lots of people excited about a new way of interacting that saves money and allows people to be more productive. This excitement is universal; whether it is Australia, Japan, Taiwan or Sweden, people are generally excited about the SMART 'Etch-a-Sketch'. . . and having fun!

COMMERCIALIZING A REAL-TIME COLLABORATIVE TOOLKIT

Dorab Patel and Scott D. Kalter

ABSTRACT

Groupware is a growing industry and many developers are striving to succeed in this market. One reason is that the benefits of groupware applications have not been clearly demonstrated to end users. Demonstrating the benefits of a collaborative toolkit is more challenging, since its immediate users are application developers rather than end users.

This chapter chronicles the conversion of one of our research architectures into a commercial toolkit for groupware developers. We review the research environment that led to this concept, the product that was developed, and the issues that we faced. We contrast the goals and environment of the research prototypes with those of the commercial product development in which we later engaged. A list of issues in the commercialization of a real-time collaborative toolkit is discussed, and the lessons learned from this exercise are summarized.

11.1 INTRODUCTION

The groupware industry has recently taken off. However, many vendors are still striving for wider acceptance of their products. Some of the reasons for this situation are examined by Grudin (1988). One of the reasons is that benefits are not adequately emphasized and communicated to users. Lotus has been successful by addressing some of these issues in the context of Notes (Marshak, 1990), an asynchronous groupware environment. Pendergast (see Chapter 12) discusses some of these issues in the context of the commercialization of a real-time, synchronous collaborative application.

Most of the activity in the market has been in the realm of asynchronous groupware. Other than teleconferencing and meeting support, there are few examples of synchronous groupware products. When we worked together, we found ourselves in need of tools to

share information in real time. After building a few prototypes of these tools, we realized that we could abstract out the common aspects into a collaborative toolkit. This toolkit would ease the task of developers building real-time synchronous groupware applications.

Successfully commercializing a collaborative application requires significant effort. Commercializing a collaborative toolkit required even more. The main reason is that there are two sets of users for a toolkit. The direct users are application developers. However, the end users of a collaborative application built with the toolkit are also indirect users of the toolkit. Toolkit developers have to take both kinds of users into account. This paper discusses these and other issues in the commercialization of a real-time, synchronous collaborative toolkit.

To provide a context for the discussion, we use our collaborative toolkit, COeX, as an example. We first describe our research environment and the prototype applications we built and with which we experimented. Next, we briefly describe our toolkit product, the commercial environment, and the changes we had to make to our research prototypes as a result of that environment. We then discuss the issues raised by the commercialization. Some of these issues are faced by all vendors; others are specific to developers of collaborative toolkits. We concentrate on the latter. We conclude with a summary of the lessons learned, and provide advice to help other vendors of collaborative toolkits succeed in the marketplace.

11.2 RESEARCH ENVIRONMENT

The experimental research we conducted before trying to build a commercial product consisted of creating prototype tools that worked with software we used daily. Our goal was to try out different architectures, user interface designs and channels of communication quickly and easily. Much of the motivation came from having inadequate tools for working together to write papers and code. We often wanted to work closely together, on the same file and at the same desk. When we were restricted to single-user tools, only one person could type. The other would have to distract the typist to fix simple errors or other trivial details. Now that we have the technology, we are able to use a desk with two machines on it to work together cooperatively on the same file during times of intensive collaboration. The tight coupling of our tools reflects our cooperative work style.

We built these tools for experimentation and our own use. This freed us from trying to ascertain the requirements of a potentially diverse community of users. We had only to implement these tools on our own platforms, about which we knew the most, and did not have to be concerned about portability. These simplified requirements were instrumental in trying many ideas, and discarding the prototypes that did not work out as we would have liked.

Many of the tools are built in GNU Emacs (Stallman, 1987) within a framework called ESC (Emacs Server Client). ESC is a framework that allows an Emacs process to send Elisp S-expressions to other Emacs processes and have them evaluated in the remote Emacs. The ESC layer and most of our tools are written in Eoops (Houser and Kalter, 1992) an object-oriented variant of Elisp created at Twin Sun. A major benefit of using Eoops over Elisp was that it allowed us to develop code rapidly in an incremental, dynamic, object-oriented way without having to concern ourselves about name space pollution.

The tools we built are robust enough that we use them regularly. Over the course of the last two years we have subjected ourselves to our own tools and find them increasingly useful in spite of their relative simplicity. This experience showed us that synchronous collaborative applications are useful to support our cooperative work style. Here is a brief description of the more interesting tools. These tools are more fully described in Patel and Kalter (1993b).

Toss/fetch: Allows a user to ‘toss’ a point-of-view (POV). A POV consists of a buffer and a location within it. Tossing a POV makes it available for others to ‘fetch’. Fetching a POV recreates a copy of the tossed buffer in the fetcher’s environment. This scheme allows collaborating users to share contextual information with low overhead. A major benefit of this approach is that minimal modifications to the application are necessary to support it.¹

Coed: Allows multiple users to share the same document simultaneously. Users have to lock a region before modification. Within a region, the full set of Emacs commands are available. Multiple regions can be locked by the same person simultaneously.

There are two versions of Coed. The first one is built around a server that stores a centralized master copy of the document state. The second one more closely models our commercial toolkit by using a distributed state architecture. We used the second prototype to explore various designs before committing to a commercial quality version.

Jot: Fine-grained sharing of a document by multiple users. No locking is necessary, but only a limited subset of the Emacs commands are available.

Spike: Similar to Toss/fetch, except that the toss is directed at a particular recipient, who is automatically notified of the toss. This notification displays the tossed information in a separate window.

Slam dunk: Similar to Spike, except that a region is spiked rather than a buffer. The spiked region is inserted at the current location of the receiver’s cursor. We found this simple technique to be most useful when two people are working physically next to each other.

Clipboards: We experimented with a variety of shared clipboard architectures in an attempt to generalize on the Slam dunk model.

11.3 PRODUCTION ENVIRONMENT

Based on the experience using these prototypes, we developed an application-independent collaborative toolkit, called COeX, that implements low level communication and sharing mechanisms common to all groupware applications. COeX implements mechanisms without excessively restricting the policy options of the application developer.

¹ This tool was built independently of the ESC framework, and implemented in three different editors: GNU Emacs, vi, and sted.

The COeX toolkit represents our attempt to distil the important ideas from prototypes built in a research environment, produce a functioning product, and transfer the new technology to the marketplace in a relatively short period of time. The product environment caused us to change some fairly fundamental assumptions in the design of the research prototypes. We start by examining the COeX architecture and how it differs from its closest research predecessor, Coed.

11.3.1 COeX toolkit architecture

The COeX toolkit (Figure 11.1) consists of a library and a server. The library consists of a kernel and an application-specific module (ASM). The kernel is application-independent and provides basic communication and sharing facilities. The server is also application-independent. It provides multicast, synchronization, and serialization facilities. The ASM provides sharing management functions that are tailored to a class of applications like spreadsheets or text editors. COeX provides collaborative functionality for resource discovery, rendezvous, session management and concurrency control. The client application interfaces to the ASM. COeX supports all the mechanisms stated by Pendergast (1993) that are useful for providing collaborative benefits.

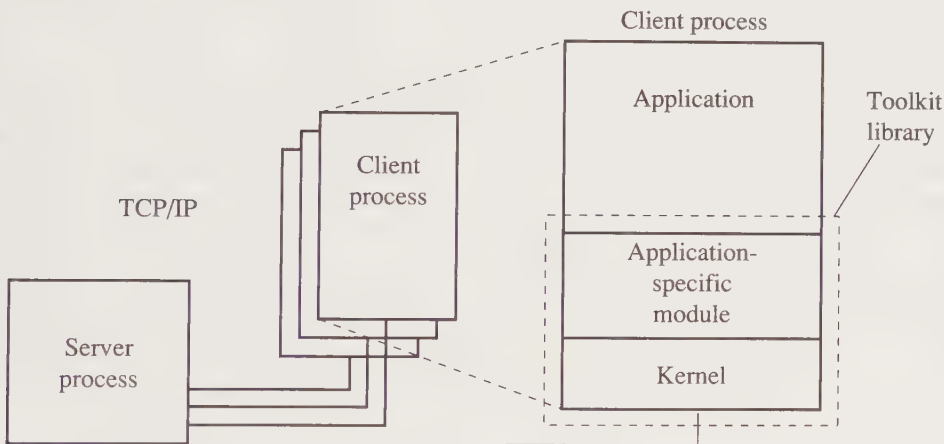


Figure 11.1 COeX toolkit architecture.

There are several significant differences between this architecture and that of the original Coed application. The primary difference is that the original Coed used a centralized state model of sharing; whereas COeX uses a distributed state model. This change was necessary since the applications, with which we anticipated integrating, would have required excessive modifications to support a remotely stored state. Furthermore, the COeX server would have become application-dependent. The current architecture makes the server entirely independent of the application.

Application independence also motivated us to define an application programming interface (API) that would be general enough for a variety of applications. As the ASM in the above architecture indicates, we do not yet have the experience to design a single general API, but the current API is a step in the right direction.

The internal protocol 'over the wire' between the COeX server and library also needed to be redefined, documented, and thoroughly tested. The refinement process made the protocol more robust, but also less flexible than that used for Coed.

As the above examples show, the commercialization resulted in changes that enhanced the generality and robustness of the system. On the other hand, moving from an implementation in Eoops to one in C required us to deal with the name space, portability, and error handling limitations of C. This had the benefit of making us much more disciplined in our coding, since we could not rely as much on the underlying environment.

11.3.2 Demonstration applications

Since applications are necessary to demonstrate a toolkit, we integrated COeX with the Sted text editor and the Idraw object-oriented graphic editor that are distributed with the InterViews (Linton and Calder, 1987) C++ X toolkit from Stanford University. The resulting collaborative applications are the Costed collaborative text editor, and the Codraw collaborative graphics editor. COeX was also integrated into IslandCalc, a commercial X-based spreadsheet.

All these applications work in real time and allow multiple users to work on the same document simultaneously. We used these examples to experiment with a range of sharing and user interface policies that can be built using the COeX toolkit. For example, we implemented methods for graphically displaying shared regions on applications with a variety of capabilities (e.g. text versus graphics, colour versus monochrome) and came up with informative but unobtrusive display mechanisms appropriate for each particular situation. We also implemented a method for visualizing and browsing the change history of a shared document. Our intention in constructing these applications was to demonstrate the kind of functionality that could be built with the toolkit. We also used these applications to test the proper operation of the toolkit and to verify its application independence. Although the testing and verification were successful, the presentation of the toolkit could be improved.

- People who see the demonstration may not be able to distinguish between the functionality provided by COeX and that of the program. This leaves them confused about what COeX contributes, and what the application developer is expected to provide. Having several different applications, differing only in added functionality, helps to make the distinction clearer.
- Demo applications must be polished, flashy and feature-laden in order for the demonstration to be effective. Demos are crucial for conveying benefits to users and appropriate resources should be devoted to their development.
- People were often inspired by our demo applications to visualize various additional features that they found useful. This demonstrated that they found the toolkit features beneficial. However, some were disappointed to realize that they themselves would have to build those application-dependent features using the base toolkit functionality.

Providing canned example code (analogous to application notes for integrated circuit chips) for common application-dependent features helps encourage such people.

11.4 ISSUES IN THE COMMERCIALIZATION OF A TOOLKIT

Many issues must be addressed when converting research prototypes into products. The process requires attention to detail and always seems to take much longer than anticipated. Research prototypes demonstrate technical feasibility. Hence, the emphasis is on showing that the concept is viable, rather than on the usability of the interface or on features that can be easily built on the base functionality. Filling out the details, such as error reporting, customization, internationalization, and stress testing for robustness should all be planned for.

Writing software today is a series of compromises. A software product must be written for popular hardware and operating system combinations so that the potential market will be large. Often, many platforms must be targeted for a product to be successful. These platforms may not be the most convenient or appropriate on which to write the software. The users of the software must be identified and their goals must be met – even if this requires designs that appear inappropriate to the software designers. The tools that are used to build the software should be popular, as tools that are used more often are more likely to be reliable. More appropriate tools are often not well enough established, or are not sufficiently supported to make them appropriate for commercial software development.

Although most software developers are constrained to use popular tools, they still have significantly more latitude than a toolkit developer. When considering the development tools for COeX, we perceived a need to use C or C++, since our immediate customers would probably be using them. Since our toolkit might be used with applications written in C, we could not write the toolkit in C++ and use all its features. Hence even C++, a fairly standard environment, was ruled out. Additionally, at the time, there was concern about its availability on a large enough number of platforms. In retrospect, using C++ might have helped data abstraction in the internal structure of the toolkit.

The issues mentioned above must be dealt with by all vendors. However, as toolkit developers we also faced additional challenges.

11.4.1 Customers are developers

There are two classes of toolkit users. The direct user is usually a software developer. However, the derived functionality may be intended for the end user, so the toolkit author needs to identify and understand the needs of both kinds of users. One of the implications is that though the toolkit provides the communication and sharing mechanisms, the developers must use these mechanisms to build services and features required by end users. Providing example code for common features helps to give a more concrete feeling for what is attainable.

The marketing effort for a toolkit must be conscious of the two classes of toolkit users. How much should the marketing emphasize the different aspects of the toolkit for the developer versus the end user? Sample demonstration applications also have to deal with this issue. Are the examples meant to demonstrate the ease of integration or the flashy

features that appeal to the end user? The obvious answer to both questions is 'Both!', so the marketing material needs to be flexible enough to tailor the presentation to the desired audience.

Since a toolkit's customers are developers, and there are far fewer developers than end users, user studies pose a greater challenge. Beta customers help in uncovering the most obvious issues, but studies across a wider variety of customers and situations are more beneficial for a toolkit. Similar issues are faced in assessing the documentation's effectiveness. One solution would be to develop relationships with other vendors and provide them with groupware consulting services. This would provide an avenue to better appreciate the issues involved in the use of a toolkit and the perceived benefits from the customer's point of view.

11.4.2 Application independence

For a toolkit to be useful in a horizontal market like groupware, it must be as application-independent as possible. This leads to two issues for commercialization. First, the integration of an inherently asynchronous collaborative toolkit into a previously synchronous application can be problematic. The success of the integration is dependent on the structure of the application software. Some application characteristics that are desirable for supporting collaboration are listed in Patel and Kalter (1992). Of course, a newly written application could take this characteristic into account. Second, a toolkit cannot handle all situations. Toolkit vendors cannot provide tailored solutions to an application without access to its source code. Conversely, providing access to the toolkit source code may be necessary to accommodate certain application requirements. Both these issues can be addressed by having a close relationship between the toolkit vendor and application developer.

In addition to being application-independent, our toolkit attempts to implement a communication and sharing mechanism while imposing a minimal policy on the application. This is a goal similar to that of the X-Window system toolkits. Policy independence is useful since it does not constrain developers to a particular policy. However, since policies are not provided, developers must come up with their own. This conflicts with a primary goal of a toolkit to provide as much canned functionality as possible so that a developer can reuse it easily in order to save time. A solution would be to provide sample code for a set of commonly useful policies. Users could modify that sample code to suit their particular needs.

In our toolkit architecture, we chose to share the application's data across various application instances, but opted not to share individual views by default. Since we do not provide support for sharing views, our toolkit is independent of the window system or window system toolkit used by the application. This has a beneficial impact, since it provides a wider applicability for our toolkit and puts us out of the fray in the window system toolkit wars. On the other hand, since the toolkit is independent of the window system, it does not provide direct support for user interface artefacts like telepointers that are useful in real-time collaboration. Of course, support for telepointers in a particular window system can be built by developers or provided by the toolkit vendor on top of the facilities provided for sharing data. In contrast, collaborative toolkits that are restricted to a particular windowing toolkit e.g. GroupKit (Roseman and Greenberg, 1992), which is

based on InterViews (Linton and Calder, 1987), can provide a convenient mechanism for developers to build collaborative user interface features like telepointers.

11.4.3 Porting and portability

Part of the product cycle consists of porting an application to different environments. The porting mainly consists of adjusting for different locations and contents include files and libraries. Most applications have only a minimal interaction with the operating system – usually dealing with the file system. Our collaborative toolkit, however, has to deal with two low-level aspects of the operating system. First, we have to deal with the low-level TCP communication abstractions provided by the operating system. Fortunately, most UNIX systems provide a fairly standard *socket* abstraction for dealing with TCP communications. Second, our toolkit interacts with a remote mount daemon and mounted file systems to generate a network-unique identifier for a file. We found that there was significant variation in the facilities provided for such functions. For example, most UNIXes provide sequential access to the entries in the mount table, but some return all entries at once. Details of these issues can be found in Patel and Kalter (1993a).

11.4.4 User education

The major non-technical issue in commercialization of collaborative software is that of user education. There appears to be more demand for asynchronous rather than synchronous collaborative tools: witness the popularity of email, main-enabled applications, workflow, and platforms like Lotus Notes (Marshak, 1990). The exceptions to this are synchronous tools for meeting support (Nunamaker *et al.*, 1991) and tools that support remote conferencing (Sarin and Greif, 1985). More needs to be done by vendors to demonstrate the benefits of synchronous collaborative software outside the confines of a meeting or teleconference.

Users are strongly attached to their single-user applications and are not willing to give them up in spite of the benefits of multi-user tools. There does seem to be a demand for end-users to be able to use their familiar applications without any modifications *and* to be able to share data among instances in a tightly coupled real-time manner. This is a rather unrealistic expectation by end-users. It is possible to provide one or the other. For example, unmodified applications can be shared using a shared window system like XVT (Abdel-Wahab and Feit, 1991) or SharedX (Garfinkel *et al.*, 1989) with its attendant limitations. Alternatively, a single-user application can be modified using a toolkit such as COeX to provide a tightly coupled sharing capability. Groupware toolkit vendors can work with established application vendors to augment their single-user applications with collaborative capabilities and hence provide value-added functionality.

For a collaborative toolkit to be successful it needs to be integrated into successful (i.e. widely used, and hence popular) applications. These products are usually sold by larger companies. These companies are reluctant to use a third-party toolkit in their product, since they do not want to expose themselves to the risk of using someone else's libraries. It is up to the toolkit vendors to convince them otherwise. Again, developing a relationship of trust between application and toolkit vendors is important.

Users are comfortable with their current single-user applications and are loath to change applications or usage patterns. Our architecture supports the construction of collabo-

rative applications that behave exactly like their single-user counterparts when the collaborative applications are in single-user mode. This feature eases the transition from single-user applications to multi-user ones.

11.4.5 Specialized toolkits

All the above issues have been discussed in the context of a general-purpose collaborative toolkit. However, there are two other kinds of toolkit whose benefits are more obvious: a cross-platform toolkit, and a window-system or application-specific toolkit. Their disadvantage is that they have a more restricted domain of applicability.

The benefit of a cross-platform collaborative toolkit would be that developers could write to one standard higher level API and be insulated from the underlying system. One approach to the cross-platform strategy is to use the lowest common denominator of the features provided by the various systems. This approach may not provide enough useful capabilities. An alternative approach would be to build up those capabilities that are lacking in the base systems. However, developers may have to duplicate many operating system features for this scheme to work.

The benefit of a window system-specific collaborative toolkit would be that the toolkit could provide user interface features that were appropriately tuned to the window system or a window system toolkit. This would make it easy for a developer using that window system to build in collaborative user interface features. GroupKit (Roseman and Greenberg, 1992) is a good example. Alternatively, a toolkit could be specific to an application class and provide specific collaborative features for that application class. DistEdit (Knister and Prakash, 1990) is a good example.

11.5 CONCLUSIONS

Though the groupware market is growing, the successes are mainly in the domain of asynchronous groupware. Other than meeting support and teleconferencing applications, there have not been many products for real-time synchronous groupware. There is a lack of synchronous collaborative applications to support people who work closely together, often at the same desk. Our toolkit, COeX, provides pre-packaged collaborative functionality for use by developers of synchronous groupware applications.

This chapter discusses many issues relating to the commercialization of a real-time collaborative toolkit. They can be categorized as toolkit issues, groupware toolkit issues, and issues specific to COeX. Most of the issues (e.g. the toolkit's direct users are developers) are common to any kind of toolkit, but are discussed in the context of groupware. Others (e.g. user education) are more specific to groupware toolkits. A few (e.g. sharing of data, not views) are specific to COeX. Another way of categorizing the issues is to look at them as technical issues (e.g. easing the transition from single-user to multi-user applications), or commercial issues (e.g. the need to have good demonstration applications).

The success of an application depends on being able to convincingly demonstrate its benefits to users. Moreover, a toolkit needs to satisfy two sets of users, developers and end users. The market for general-purpose synchronous groupware applications is not

developed yet. Hence vendors must educate users to realize the significant benefits of such applications.

Groupware toolkits need to be flexible, and hence do not lend themselves to being shrink-wrapped. It is essential to develop a trusting relationship between the toolkit vendor and the application developer. The relation is closer to a consulting situation rather than a traditional vendor client relationship.

There is a tension between making a toolkit generic and providing domain-specific functionality. The application independence of a toolkit helps to make it more widely applicable. A domain-specific toolkit provides convenient support tailored to the application's needs. A hybrid approach is to provide generic functionality as a base, and libraries or sample source code for add-on domain-specific functionality.

Vendors contemplating the development of a collaborative toolkit should be conversant with all the above issues. The advice provided here should help their product succeed in the market.

ACKNOWLEDGEMENTS

The presentation of this paper was greatly improved by suggestion from Paul Eggert and Hidekaze Yasuda.

REFERENCES

- Abdel-Wahab, H. M. and Feit, M. A. (1991) Xtc: a framework for sharing X-Window clients in remote synchronous collaboration, in *Proceedings of the IEEE Tricomm'91: Communications for Distributed Applications and Systems*, April, Chapel Hill NC, pp. 159–67.
- Garfinkel, D., Gust, P., Lemon, M. and Lowder, S. (1989) *The SharedX multi-user interface user's guide, version 2.0*, Technical Report STL-TM-89-07, Hewlett-Packard Laboratories, Palo Alto CA.
- Grudin, J. (1988) Why CSCW applications fail: problems in the design and evaluation of organizational interfaces, in Suchman, L. (ed.) *CSCW'88: Proceedings of the Conference on Computer-Supported Cooperative Work*, 26–28 September, Portland OR, pp. 85–93.
- Houser, C. and Kalter, S. D. (1992) Eoops: an object-oriented programming system for Emacs-Lisp, *LISP Pointers*, 5(3), 25–33.
- Knister, M. J. and Prakash, A. (1990) DistEdit: a distributed toolkit for supporting multiple group editors, in Halasz, F. (ed.) *CSCW90: Proceedings of the Conference on Computer-Supported Cooperative Work*, 7–10 October, Los Angeles CA, pp. 343–55.
- Linton, M. A. and Calder, P. R. (1987) The design and implementation of InterViews, in *Proceedings of the C++ Workshop*, 9–10 September, Santa Fe NM, Usenix Association, pp. 256–67.
- Marshak, D. S. (1990) Lotus Notes: a platform for developing groupware applications, *Patricia Seybold's Office Computing Report*, July.
- Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R. and George, J. F. (1991) Electronic meeting systems to support group work, *Communications of the ACM*, 34(7), 40–61.
- Patel, D. and Kalter, S. D. (1992) A toolkit for synchronous distributed groupware applications, in Coleman, D. D. (ed.) *Proceedings of Groupware '92*, 2–5 August, San Jose CA, Morgan Kaufman, pp. 225–7.
- Patel, D. and Kalter, S. D. (1993a) A UNIX toolkit for distributed synchronous collaborative applications, *Computing Systems*, 6(2), 105–133.
- Patel, D. and Kalter, S. D. (1993b) Low overhead, loosely coupled communication channels in collaboration, in de Michelis, G., Simone, C. and Schmidt, K. C. (ed.) *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, 13–17 September, Milan, Klawer Academic, pp. 203–18.

- Roseman, M. and Greenberg, S. (1992) GroupKit: a groupware toolkit for building real-time conferencing applications, in Turner, J. and Kraut, L. (eds.) *CSCW'92: Proceedings of the Conference on Computer-Supported Cooperative Work*, 31 October–4 November, Toronto, pp. 43–50.
- Sarin, S. and Greif, I. (1985) Computer-based real-time conferencing systems, *IEEE Computer*, **18**(10), 33–45.
- Stallman, R. (1987) *GNU Emacs Manual*, 6th edn., Free Software Foundation, Cambridge MA.

GROUPGRAPHICS: PROTOTYPE TO PRODUCT

Mark O. Pendergast

ABSTRACT

The ultimate success of CSCW applications will be determined when and if they are accepted in industry. In order for CSCW to have an impact in industry, CSCW researchers need to clarify the benefits of their prototypes and demonstrate that concepts which work well in simple applications can be successful in robust ones.

This chapter presents a groupware application, GroupGraphics, which is currently undergoing the productization process. GroupGraphics began in 1990 as the academic prototype MUGE (multi-user graphics editor) at the University of Arizona. It was developed to run in the PC LAN environment and support activities such as enterprise analysis, graphical brainstorming, systems re-engineering and graphical meeting languages. GroupGraphics creates object-based drawings, as opposed to bitmap sketches. In 1991 Ventana Corporation, Tucson AZ, undertook the job of turning the prototype into a product which would complement its EMS product, GroupSystems. This has entailed deciding which features of single-user drawing tools need to be added to GroupGraphics in order to make it acceptable to the market-place, and which mechanisms of groupware tools to adopt in order to make it (a) stand out from other graphics tools, and (b) enhance groupwork. This paper reviews past groupware applications with regard to the mechanisms they employ, the effects on work, and the potential benefits of those effects. GroupGraphics is then presented with respect to those mechanisms, effects and benefits, and special issues related to large group management are discussed. Finally, a summary of lessons learned and future research plans is given.

12.1 INTRODUCTION

Nearly a quarter of a century has elapsed since Douglas Englebart first published his work on computer-supported cooperative work (Englebart and English, 1968), yet few groupware applications with any level of WYSIWIS interface are in use in industry. Currently available 'groupware' applications for the most part are limited to electronic mail, bulletin boards, file servers, databases and file transfer. While it is true that all these

applications facilitate groupware via data sharing, none really aids in coordinating groupwork or providing group interfaces. These systems score low on the groupware shared environment spectrum defined by Ellis *et al.* (1991). There are several electronic meeting systems (EMS) available (GroupSystems (Ventana Corp.), TeamFocus (IBM) and VisionQuest (CTC)), but all are struggling in the market-place. Products touted as groupware by large PC software vendors are better classified as platforms for potential groupware applications, e.g. Lotus Notes and Windows for Workgroups. The value of WYSIWIS interfaces and coordinated activities seems obvious to researchers, but has yet to make an impression on industry. Why?

One potential reason for this is the PC revolution of the 1980s. Group tasks such as writing and design migrated from mini computers and mainframes to personal computers with little processor power and limited connectivity. As personal computers and PC LANs evolved, software vendors concentrated their resources on developing more powerful single-user applications. At the same time, CSCW researchers have shied away from research on PC systems because of their limited power and lack of a multitasking operating systems. Personal computing is beginning to meet the requirements of CSCW, but most CSCW prototypes are designed around UNIX workstations.

Another reason that industry is taking a wait-and-see attitude towards CSCW is the fickleness of CSCW research. One year it's object-oriented design and SmallTalk, the next it's Hypertext, then C++ and X-Windows, and now multimedia. This fickleness, coupled with a lack of solid empirical evidence supporting groupware, has left industry with little to grab hold of. A prohibiting factor is the nature of CSCW prototypes. Shared text editors are impressive in their interaction, but users have become accustomed to powerful word processors. Researchers have rightfully spent their time perfecting multi-user interaction, leaving fonts, formatting and other features to industry. The same can be said of group painting and sketching tools. This puts industry developers and users in the position of comparing formatting and drawing features against coordination features. Colours and fonts will win every time.

In order for CSCW to have an impact in industry, CSCW researchers need to clarify the benefits of their prototypes and demonstrate that concepts which work well in simple applications can be successful in robust ones. This paper will first correlate common mechanisms employed by CSCW prototypes with their effects, then map the effects to tangible benefits. Section 12.3 presents an application, GroupGraphics, which was developed for the market place by Ventana Corporation (March 1993). Work on GroupGraphics began as an academic prototype (MUGE: Pendergast and Beranek, (1991)) at the University of Arizona and University of Florida. Section 12.4 details group management and facilitation procedures, and Section 12.5 reviews lessons learned during the development and productization of GroupGraphics.

12.2 REVIEW

In the last few years there has been an extensive amount of effort in developing real-time groupware applications. Most of this effort has been directed at simulating face-to-face interaction via shared graphical or textual workspaces. This effort has been intensified by the introduction of new technologies such as full motion video, improved pen interfaces and low-cost high-resolution displays. While there has been a great deal of work done on

the development of groupware tools, little effort has been expended on enumerating and quantifying its benefits. This section will review past work with respect to perceived benefits to groupwork.

12.2.1 Benefits

The ultimate success of groupware research will be determined when and if the functionality of the academic prototypes is adopted by the software industry and embraced by corporate computer users. For this to occur, there must be a cost/benefit justification for implementing groupware systems. Therefore, benefits of groupware systems must either directly, via improved quality or reduced time, or indirectly, via greater user satisfaction, outweigh the costs of implementation. With the exception of word processing and electronic mail, office automation software has had trouble justifying its cost. Recently, GroupSystems, from the University of Arizona, has shown considerable cost savings for GDSS at both Boeing and IBM (Nunamaker *et al.*, 1989). If the benefits realized by GDSS systems can be carried over to groupware as a whole, then current research efforts will be justified. Mapping potential corporate benefits back to technology-driven prototypes requires a double translation:

Mechanism → Effect → Benefits

Benefits can be realized by reducing the amount of time required to perform a task, creating a higher quality product or output, and by increasing worker efficiency and satisfaction. Effects are the changes in work habits and techniques induced by mechanisms or features of a collaborative application.

12.2.2 Effects

The effects achieved are:

- *Increase in the level of parallel activity.* This effect is most noticeable for tools which automate aspects of face-to-face meetings (Dennis *et al.*, 1988). The increase in parallel activity for applications which are distributed in nature, such as writing, would be less pronounced. This is due to the fact that most tasks can be decomposed to a state where each member can work on a separate part, merging the results at the end.
- *Greater participation by all team members.* Again this is more pronounced in automated face-to-face meetings. Tools which allow concurrent use and which provide mechanisms which promote contributions, e.g. anonymity, should encourage silent members to contribute.
- *Fewer work integration steps.* This applies to applications which use a shared workspace or shared context to store information. Groupware tools which require users to work individually, then periodically merge or integrate their work can actually add integration steps (Pendergast and Hayne, 1992).
- *Distributed work.* When groupware is implemented in a wide area network fashion this would save travel time. In a local setting, distributed features enable work to be carried out at the desk of each participant, removing the need for specialized electronic meeting rooms.

- *Synergy among workers.* Synergy is the increase in effectiveness which comes about when people work in groups rather than individually. Groupware applications can either enhance this effect by providing mechanisms to allow the free flow of ideas, work and communication, or inhibit synergy by creating artificial boundaries, rules or protocols.
- *Greater consistency of work.* Consistency in groupwork involves such things as using the same diagramming conventions, writing from the same perspective (first person, third person etc.) and nomenclature. Some aspects, such as diagram conventions, can be enforced by the software, other consistency gains are a result of better communications and access to work in progress.
- *Continuous validation of contributions by group members.* Groupware tools which promote concurrent access to data and provide real-time updates enable the entire team to access and review the work as it progresses. This eliminates the need for periodic review sessions and reduces the number of 'surprises' which can result from people working autonomously on separate portions of a project (Posner *et al.*, 1992).
- *Increased access to information.* This is accomplished by both allowing concurrent access to data and providing better methods to access the data, e.g. hypertext (Rein and Ellis, 1991; Delisle, 1990).
- *Greater coordination of groupwork.* Coordination of work implies both the use of the shared workspace, i.e. spatial arrangement (Tang and Minneman, 1990), and the coordination of actions. Coordinating actions in groupware can be a group protocol built in to the application, i.e. only allowing certain members to do certain functions at a given time, and/or can be supported by social protocols established between users via tele-presence (Greenberg, 1991; Ellis *et al.*, 1991).

Mapping the observed effects of collaborative applications to their tangible benefits gives us Table 12.1. This table shows that the effects which could lead to the most benefits (assuming equal weights) are: allowing parallel activity, increasing participation, providing more access to information, and better coordination of users. By concentrating on the effects which produce the greatest benefits, groupware developers can increase the likelihood that their innovations will be accepted and reduce the trend towards gadgetry.

Table 12.1 Groupware benefits versus effects

	Elapsed time	Output quality	Efficiency/satisfaction
Parallel activity	✓		✓
Participation	✓	✓	✓
Reduced integration	✓		
Distributed work	✓		
Synergy		✓	
Greater consistency		✓	
Continuous validation		✓	
Information access	✓	✓	
Coordination	✓		✓

12.2.3 Mechanisms

A review of existing collaborative applications reveals a long list of mechanisms which are being tested by researchers. Some of the mechanisms are obvious necessities of any collaborative application (networked workstations, shared data stores); others endeavour to simulate aspects of face-to-face meetings into CSCW; while some are novel attempts to apply newly available technology to CSCW problems. There is very little empirical evidence which links these mechanisms to effects; however, a review of concepts and anecdotal evidence presented in literature leads to the creation of Table 12.2.

Table 12.2 Mechanisms versus effects. Number in parentheses (*n*) is the number of benefits

	Gesturing Floor control Audio and video Structured data Structured tasks Independent views View synchronization Real-time updates Concurrent access								
Increased parallel activity (2)	✓			✓					
Increased participation (3)	✓	✓		✓			✓		
Reduced integration (1)	✓	✓	✓			✓		✓	
Distributed work (1)	✓		✓	✓			✓	✓	✓
Synergy (1)	✓	✓			✓				✓
Greater consistency (1)	✓	✓			✓	✓			
Continuous validation (1)	✓	✓							
Information access (3)	✓	✓		✓					
Increased coordination (2)		✓	✓		✓	✓	✓	✓	✓

- Real-time updates.** Real-time updates are probably the most significant aspect of groupware which separates academic prototypes and industry products. In industry, groupware has meant that the group's data is accessible via a LAN and has some mechanism to control access. Academic prototypes have stressed the need for real-time updates (data and screen) and it is taken for granted that some variation of WYSIWIS (Stefik *et al.*, 1987a) is a necessity. Various studies have suggested that real-time updates aid in coordinating work, increase participation and enable continual validation of the work.

- *Concurrent data access.* Concurrent access of data is a necessary partner of real-time updates. By itself it enables an increase in parallel activity, provides greater access to information, and helps to promote consistency of work. Coupled with real-time updates, it also aids in coordination of work, increases participation, and enables continual validation.
- *Gesturing.* In its simplest form, gesturing consists of a single group telepointer which can be used by participants in order to help direct attention to an item in the shared workspace. This would primarily be used in conjunction with verbal communications and take the place of actually pointing at something on a chalkboard. As collaborative applications have matured, telepointers have become more sophisticated in their design and more robust in their use. Multiple telepointers with user identification; telepointers which change shape to indicate the user's current mode of operation (pointing, creating, drawing, erasing etc.); telepointers which change orientation to allow multiple users to point at an object without overlapping; telepointers which change size based on the number of active users; and telepointers which disappear when inactive. With these new innovations, telepointers can be used for more than just pointing. For example, a waving pointer might indicate that a user would like control of an audio link, analogous to raising one's hand; a telepointer which indicates operation (drawing) would indicate to other users that this area of the shared workspace is being changed and that they should work elsewhere; and the mere presence of a labelled pointer would indicate that a particular team member is now participating. Early work with shared drawing tools such as MBlink (Sarin and Greif, 1985) and Boardnoter (Stefik *et al.*, 1987b) demonstrated the feasibility and usefulness of telepointers, and current work with gesturing in GroupSketch (Greenberg, 1991) and VideoDraw (Tang and Minneman, 1990) has extended gesturing and pointer techniques allowing them to convey more information. The primary effect which gesturing and telepointers have on group work is coordination of the users.
- *Independent view.* Independent view is a relaxed form of WYSIWIS whereby each participant is allowed to view a different portion of the shared workspace. Since users are not locked into a single screen or view, this should increase the level of parallel activity possible, and allow increased access to information. A drawback to individual views is that the users are no longer aware of where others are working, so coordination and consistency of work can be reduced.
- *View synchronization.* View synchronization or screen slaving is a technique which allows groupware applications that have implemented a relaxed form of WYSIWIS implementation to change over to a strict WYSIWIS mode. This increases coordination and enables such tasks as group reviews of a document or design and distributed presentations.
- *Structured data.* The process of turning ideas, concepts, models and information into a formal design or document requires users to collect, integrate, sequence and store the data. To do this users often maintain a complex mental model of the data and the structure of the artefact. This is fine for individual work, but is a hindrance in groupwork. Simple lists accessible via shared editors cannot represent the complex relationships between ideas. Adding structure and shared access to the data via multi-user hypertext systems shows much promise in this area. Notecards (Halasz *et al.*, 1987) provides users with a semantic network of electronic notecards connected by typed links. Other systems such as SEPIA (Streitz *et al.*, 1989), Neptune (Delisle and

Schwartz, 1989) gIBIS (Conklin and Begeman, 1988) and rIBIS (Rein and Ellis, 1991) have further demonstrated the usefulness of hypertext systems for structuring group work. Storing relationships of data with different types may also mean linking applications as well as data.

- *Structured tasks.* For some work it is appropriate to implement a structured methodology to guide groupwork, e.g. group decision support (Dennis *et al.*, 1988). The overall effect of these tools is to increase coordination of group members by sequencing their tasks and organizing their interaction. GroupSystems and VisionQuest are examples of such systems.
- *Floor control.* Initially floor control was used as a mechanism for concurrency control. Techniques such as 'gravel passing' and 'picking up the chalk' were used to grant exclusive write access to a shared data area. When used for this purpose floor control limits the amount of parallel activity. With more complex algorithms for handling data consistency (Ellis and Gibbs, 1989), automatic locking with fine granularity (Greif and Sarin, 1987) and more powerful systems, floor control mechanisms are no longer needed to insure consistent data. Rather, specific floor control policies can now be used to aid coordinating user interactions and in implementing procedures for highly coordinated activity, e.g. rIBIS's tightly coupled mode (Rein and Ellis, 1991).
- *Video and audio.* A current trend in CSCW systems is the integration of computer and video images via video overlay technology. This is exemplified by LookingGlass (Clark and Scrivener, 1992), TeamWorkStation (Ishii, 1991), IIIF (Buxton and Moran, 1990) and VideoDraw (Tang and Minneman, 1990). The addition of multimedia to CSCW aids in the support of tele-presence or co-presence, which in turn increases coordination and makes a more congenial environment for distributed work.

In this section I have attempted to enumerate the benefits, effects and mechanisms which have been observed in CSCW applications. Not all mechanisms are appropriate or advisable for all applications, but by taking a realistic look at their potential benefits to users, CSCW researchers can gain an insight into which mechanisms deserve further development. The next section details the functions and features of a group drawing tool. As this tool is in the process of becoming a commercial product, the translation mechanism → effect → benefit played a major role in its design.

12.3 GROUPGRAPHICS

GroupGraphics began in 1990 as the academic prototype MUGE (multi-user graphics editor) at the University of Arizona. It was developed to run in the PC LAN environment and support activities such as enterprise analysis, graphical brainstorming, systems re-engineering and graphical meeting languages. GroupGraphics creates object-based drawings, as opposed to bitmap sketches. In 1991 Ventana Corporation, Tucson AZ, undertook the job of turning the prototype into a product which would complement its EMS product, GroupSystems. This initiated the task of deciding which features of single-user drawing tools needed to be added to GroupGraphics in order to make it acceptable to the market-place, and which mechanisms of groupware tools to adopt in order to make it (a) stand out from other graphics tools, and (b) enhance groupwork.

GroupGraphics allows users to create diagrams in a virtual workspace which is 32K by 32K pixels in size (approximately 50 by 50 screens). Each user is allowed to view any portion of the diagram, at any scale factor, independently of other users (relaxed WYSIWIS). Diagram elements are stored as objects in shared files. These files may be opened on either a shared or private basis. Each user may have multiple copies of the same file opened at the same time, and may have multiple files open at the same time. This provides the users with the potential for having multiple views of multiple files. When a file is updated, the change is broadcast to all active users of the file to allow for real-time screen updates, and updates of local data stores. The operating environment is shown in Table 12.3.

Table 12.3 GroupGraphics operating environment

Microsoft Windows 3.0 or higher (3.1 preferred)
MS-DOS 3.3 or higher
Mouse or pen
16 MHz 80386SX processor (33 MHz preferred)
4 Mbyte memory (6 Mbyte preferred)
LAN file server
Reliable session connections via IPX/SPX or NetBios

The most noticeable change to the prototype was the addition of a toolbar down the left side of the display and a colour palette along the bottom. This arrangement is common among PC tools, making it acceptable to the market-place and intuitive to users (Figure 12.1). The next most noticeable change was the addition of extra graphical objects, raising the number of standard objects to 22. This allows GroupGraphics to support many design methods including flow charts, structure charts, data flow diagrams, fishbone diagrams, process flow charts and entity-relationship diagrams. In addition to the standard objects, GroupGraphics allows users to paste bitmaps onto diagrams. Sets of bitmaps can be defined and placed on auxiliary toolbars. This allows groups to define their own graphical languages or use a bitmap as a background for graphical annotation. Items under the Shapes menu can be enabled or disabled, assigned different names (e.g. Diamond could become Decision), and assigned a default colour. This rich set of objects allows groups to select from a large set of structured drawing techniques.

Objects are created by selecting them from the tools list on the left side of the screen or from the Shapes pull-down menu. Once created, objects can be manipulated by first selecting them with the left mouse button, then dragging them or resizing them by stretching their handles. Double clicking the left mouse button over a selected object will cause a dialog box to appear which allows the user to change all of the object's attributes (text attributes, shape, colours, shadow etc.). Lines and polylines can have arrowheads at one, both, or neither ends.

Up to 5000 characters of text may be associated with each object. Text is entered and modified using the Windows popup editor. When an object is selected, it will be displayed with solid black handles denoting the object's boundaries, and hollow black handles denoting the text area associated with the object. The text area can be moved or stretched using the right mouse, and text can be modified by double clicking the right button over

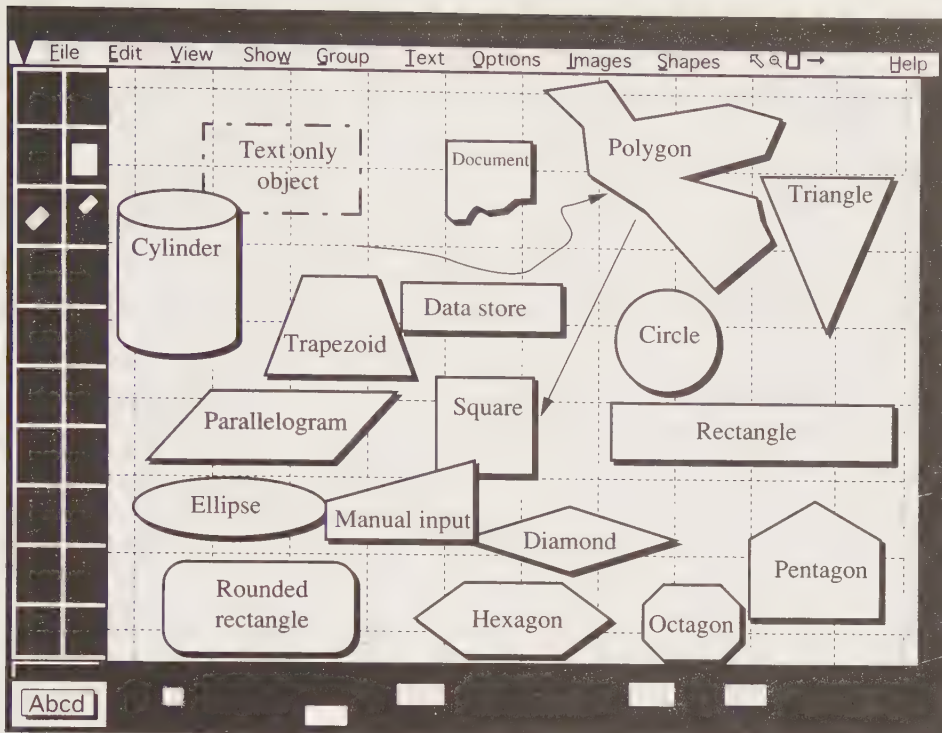


Figure 12.1 GroupGraphics interface and shapes.

the text area. Text associated with objects may be cut/copied/pasted from other applications. GroupGraphics supports standard as well as TrueType fonts. Text attributes include bold, italic, underlined or strikeout, and can be left, right or centre justified. Text which is associated with an object moves with that object. If automatic font sizing is enabled, the font size is automatically adjusted such that all the text is visible within the text region.

GroupGraphics provides standard editing features such as cut, paste, copy and undo. In addition, GroupGraphics allows cutting and pasting of bitmaps and text (Figure 12.2). Associated with the Edit menu are the Attach and Detach commands. In order to attach two objects, the user would select both, then invoke the attach operation. Attaching objects forms a logical and spatial relationship between them. If one object is moved, all attached objects are moved with it. Lines and polylines can automatically be attached to non-line objects when they are created if the 'auto attach' option is enabled. When an object is moved, the endpoints of any lines attached to it are moved. The Detach command allows the user to remove the relationship between the two objects. The attach option not only makes flow diagramming easier (since lines are stretched to follow objects), it also allows for extra structure to be added to a diagram. For example, if the workspace were filled with rectangles representing ideas, the rectangles could be clustered and then attached to one another, forming logical groups.

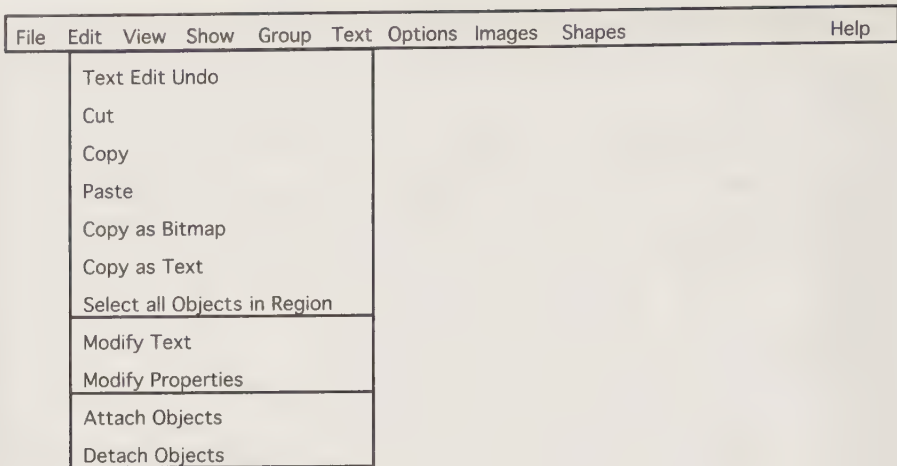


Figure 12.2 GroupGraphics object edit options.

Users are allowed to control the positioning and scaling of their views of the document independently. This is accomplished using scroll bars and arrow keys for changing position, and the Zoom function to change scaling. When using the Zoom function the mouse cursor shape will be changed to look like a small gunsight. The gunsight is used to indicate the precise spot where the screen should be zoomed in on/out from. User views can be coordinated using the group functions Lead, Follow and Match Screens. Other view options allow the user to turn on/off alignment grids, hide/show the toolbar and palette, zoom out far enough to view the entire diagram, reposition the screen to a diagram marker and automatically scale fonts. Diagram markers are a powerful tool in managing the workspace. They enable users to label parts of a diagram and to move directly to those labels. Since the workspace could potentially be an area 50 screens by 50 screens, diagram markers are vital in managing views and keeping users from getting lost.

Group features are those which help coordinate group activities and are especially helpful for distributed sessions. Screen displays are updated in real time with new or modified objects. Locking of diagram objects is done automatically when the user selects the object to be changed. Objects selected or locked by other users appear on the screen with red handles; objects selected or locked by a given user appear with black handles on his or her screen. Users can determine where others are working by turning on the Show Users option under the View menu. When this option is enabled, the location of each user is denoted with a small icon with his or her name next to it. The easiest way to determine where all users are working is to first select the Full View option from the View menu, then turn on the Show Users option. The Show Users option may be left on continually if desired. Doing this provides a constant reminder of which users are active in the same space.

Group pointers as well as full motion gesturing have been implemented. When the Group Pointer option is enabled a group pointer is broadcast each time the right mouse button is pressed. Those stations with Show Group Pointers enabled will see a hand-

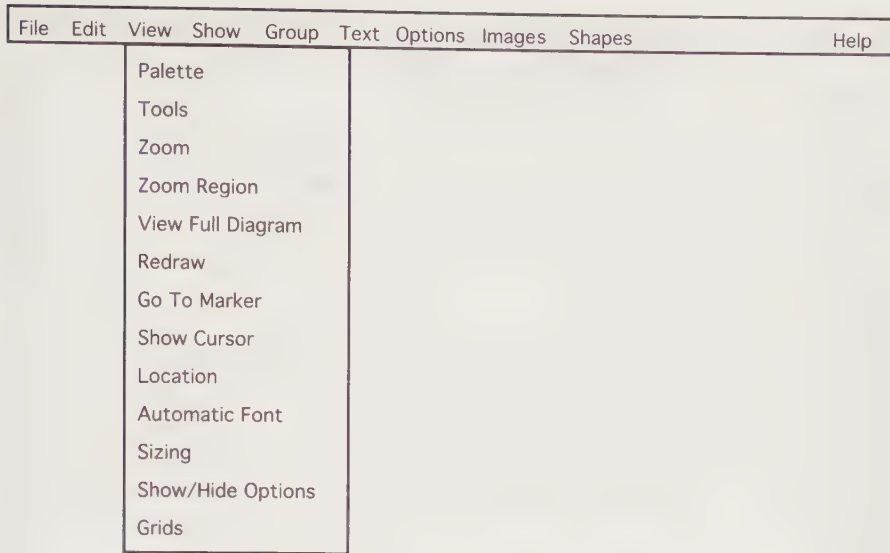


Figure 12.3 GroupGraphics user view options.

shaped icon (with the sender's initials on the cuff) appear on the screen moving up and down then settling on the appropriate spot. When gesturing is enabled, the hand icon is constantly moved, mimicking the movements of the mouse. Users may synchronize views by employing the Match Views or Lead/Follow options. When a user selects Match Views, his or her current view parameters are broadcast to other users, who in turn may accept or reject the match. If they accept the match, their views are zoomed and translated to match the senders. When the Lead Group option is enabled, each time the user changes view by zooming or scrolling, his or her parameters are broadcast to all other users. Those users which have the Follow Group option enabled will have their views translated accordingly. More than one user may be a leader at the same time, and a user may have both leading and following enabled at the same time. The group coordination functions are shown in Figure 12.4.

The colour of the pen (used to draw the border of objects) and the brush colour (used to fill objects) are independently controllable. Their colours can be selected either from the palette at the bottom of the screen or via pull-down menu options under the Options menu (Figure 12.5). GroupGraphics supports 8 standard brush patterns (solid, transparent and 6 hatch schemes), 3 pen widths, 5 pen patterns (solid, dashed, dotted etc.), 30 brush colours, 16 pen colours, front/back, shadows and arrowheads. A rich set of drawing options are expected by users, and the use of colour can be an important feature in controlling drawings. For example, each user could be assigned a different colour to work in, or parts of a diagram which have been accepted could be in black, new items in green, and disputed items in red. GroupGraphics provides the flexibility for groups to devise their own colour and object conventions to support the process at hand.

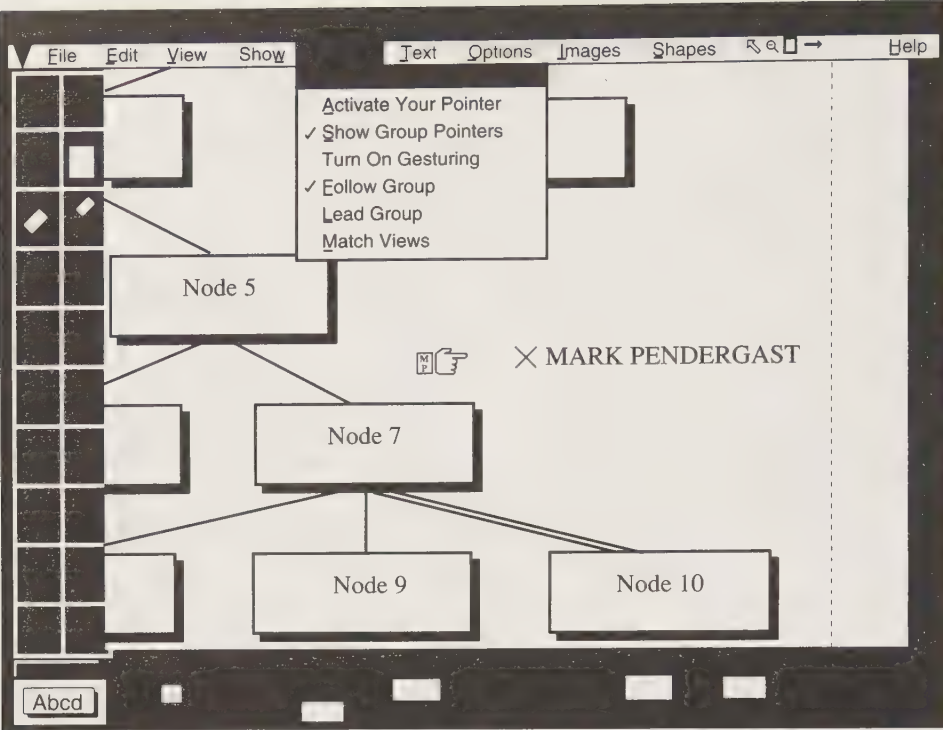


Figure 12.4 GroupGraphics group coordination functions.

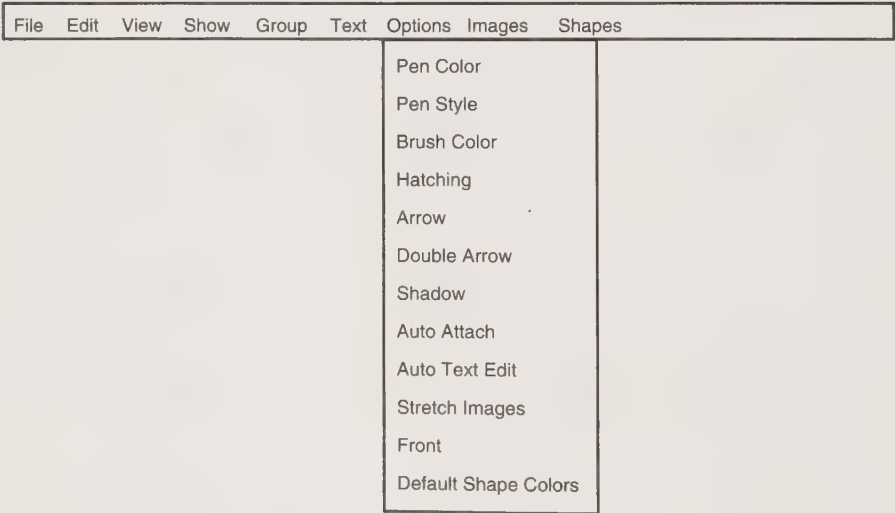


Figure 12.5 GroupGraphics object creation options.

12.3.1 Bitmap images

Bitmap images can be added to a diagram in one of three ways: pasting from the clipboard, selecting it from one of the image set buttons or selecting it from a list of bitmap files. In order to paste a bitmap from the clipboard the user must first copy it there from a bitmap-oriented program such as Paintbrush. The bitmap can then be pasted onto the diagram by stretching a region and selecting the Paste function from the Edit menu. Adding bitmaps from image set buttons is done in the same manner as other predefined objects, i.e. push the button, then stretch a region. To add a bitmap from a disk file the user must select the Pick Image From Library option on the Images menu, select the desired file, then stretch a region for it to be inserted into. The actual bitmaps are not stored in GroupGraphics diagram files; rather, references to files are stored. This helps reduce the size of diagram files and allows users to modify bitmaps which appear on the diagram using the painting tool of their choice. GroupGraphics can access images in either PCX or BMP formats. Image sets can be loaded onto the image buttons via the Modify Image Set 1 and Modify Image Set 2 options under the Images menu or loaded from sets defined in the .INI file.

12.3.2 Index/file folder objects

Index objects allow access to other programs from GroupGraphics. An index object is created in the same manner as other objects, except that its text is used as a command line to invoke another tool. The index object appears as a small yellow file folder. When the object is selected, its text line is displayed. When the left mouse button is double clicked on the object, the object's text is used to execute another program (GroupGraphics remains running). Example:

- VG.EXE DIAGRAM2.VMF starts another copy of GroupGraphics using
DIAGRAM2 data file
- NOTEPAD.EXE TEXT.TXT starts a copy of the Windows Notepad program,
editing TEXT.TXT
- HG.PIF would start Harvard Graphics in a DOS Window

12.3.3 Tool configuration

The GroupGraphics program default settings and menu options can be configured via an initialization (.INI) file. This file is accessed when the program is loaded and allows any menu or submenu to be hidden from the user, as well as setting default options such as pen colour, brush colour, work area dimensions etc. Additionally, the name given to each item in the Shapes menu can be changed to support a given process, e.g. for flowcharting 'Diamond' could be renamed 'Decision'. GroupGraphics supports many processes and the creation of many types of diagrams. In order to make the user interface less complex, different initialization files can be created to support different processes and different user skill levels. For example, an initialization file could be created to support creation of data flow diagrams. This initialization file would have only those objects enabled which are used on data flow diagrams (data stores, rectangles, lines, rounded rectangles etc.).

12.3.4 Show/presentation functions

These functions allow a large diagram file to be viewed full-screen (no menus, caption bars, scroll bars) as a series of screens. Shows will take one of two basic forms: (a) showing small segments of a large diagram in sequence; or (b) showing a series of small diagrams which are all in the same diagram file. A show is merely a sequence of view changes, progressing through a list of diagram markers (locations). The next/previous location is reached by pressing either the left/right mouse button, or left/right arrow key. A delay can be associated with each item in the show list in order to progress automatically to the next marker, and a continuous play option is available to allow the show to be automatically repeated. Shows can be a valuable aid for both distributed and face-to-face groups. When work is performed in a distributed mode (space and time), users can create shows which create a tour of the changes they have made to the diagram. In face-to-face mode, shows can be used to present a design, or to evaluate alternative designs.

Table 12.4 summarizes GroupGraphic's groupware mechanisms and their intended effects. In addition to mechanisms previously defined, GroupGraphics support a large workspace in order to allow participation of members in a large group, diagram markers

Table 12.4 GroupGraphics mechanisms and effects

	Concurrent access	Real-time updates	View synchronization	Independent views	Structured tasks	Structured data	Markers and shows	Large work space	Gesturing
Increased parallel activity (2)	✓				✓				✓
Increased participation (3)	✓	✓			✓			✓	✓
Reduced integration (1)	✓	✓	✓				✓		✓
Distributed work (1)	✓		✓	✓				✓	✓
Synergy (1)	✓	✓				✓			✓
Greater consistency (1)	✓	✓				✓	✓	✓	
Continuous validation (1)	✓	✓							
Information access (3)	✓	✓		✓					
Increased coordination (2)		✓	✓			✓	✓	✓	✓

and shows to help navigate and document changes to large diagrams, linkages between applications (file folder objects) as well as linkages between objects. GroupGraphics also supports features industry users have come to expect from single-user applications: a large colour set, variety of fonts, large object set, printing, context-sensitive help and cut/paste interfaces with bitmap and textual applications. GroupGraphics' configurability allows it to be used when constructing a large variety of structured diagrams.

12.4 GROUPWORK PROTOCOLS

Large groups (8+) and very large groups (20+) pose process management problems, namely, managing conflicts in the use of the workspace and conceptual conflicts in the creation of a diagram. Solving the first problem is a matter of providing a large enough work area so that everyone has room to work, showing where each person is working, and defining the bounds of each person's (or subgroup's) workspace. Solving the second problem requires providing the ability to simultaneously create alternative designs and then evaluate them using side by side comparison.

Distributed groups require extra features to take the place of ones commonly found in meeting rooms. This includes the functional equivalent of a large-screen display, the ability to identify objects on a common display, and an aid in controlling the voice channels which may exist (conference call). The functional equivalent of a large-screen display would be a window on every one's screen which a leader could control, identifying objects requires a telepointer for this window, and control of the voice channel requires a functional equivalent of someone raising their hands. GroupGraphics provides support for solving these problems:

- *Large configurable size workspace:* the maximum size of the workspace is 32K by 32K pixels or about 50 screens by 50 screens. This gives a group plenty of space to develop large diagrams or competing versions of the same diagram. The workspace can be made smaller to suit a group's needs by setting values in the .INI file. For example, a group of eight people, working in pairs, wants to develop four alternative diagrams (perhaps four different ways to restructure the company). This group may want to define a workspace which is 2 screens by 2 screens (1280 x 960 pixels). This smaller work area would prevent members from getting lost in the great expanse of a full-size work area, yet give each team its own space to work in.
- *Diagram markers:* provide a mechanism for labelling regions of the work area. For the example described above, the group leader could place a diagram marker in each of the four work areas, with names like 'Alternative 1', 'Alternative 2', . . . or more descriptive names, like 'Evolutionary Restructuring Plan', 'Eliminate Middle Management Plan'. The facilitator (or group members) could then use the GOTO Marker function to take a look at the competing plans. When the work is done, a presentation could be created showing each plan.
- *Grids:* provide a passive mechanism to define boundaries where individuals and subgroups should work. The small, medium and large grids are only useful for lining up objects on a diagram, but the presentation grid (one screen in size) and the page grid (one printed page, sizeable) could be used to define work areas. In our example, the group would most likely use the presentation grid to define its work area boundaries.

- *User markers*: user markers give a visual indication of where each user is working, and lets everyone working in a region of the diagram know which other users are there. The GOTO Marker function can be used by the facilitator (or other users) to check on the work of an individual. User markers used in conjunction with the Full View command provides an activity map, showing the location of every user.
- *Lead/follow*: allows users in subgroups to coordinate their actions, and provides a mechanism for users to follow a presentation. This is especially useful in the distributed mode or for facilities which do not have a large front screen. A facilitator could conduct a presentation of a diagram and users could watch in their own offices.
- *Group pointers, gesturing*: provide extra support for subgroup work, as well as supporting open discussion during a presentation. For distributed work, gesturing is especially important since, there is no physical way to point at something on someone else's screen or the group screen. Also, gesturing can be used as a mechanism to get the floor, i.e. moving your group pointer is analogous to raising your hand.
- *Multiple windows, independent views*: GroupGraphics allows a facilitator (or user) to run multiple copies of the program at the same time. Each copy can be editing a different diagram file or multiple copies of the same file. This gives the facilitator the ability to start up several copies of GroupGraphics for the same diagram, each showing different portions of the diagram. In the previous example, the facilitator may want to start four copies of GroupGraphics, each displaying the work area of a different subgroup (Figure 12.6). Windows provides a 'tile' function which would arrange the four windows in quadrants on the screen. This display could be left on the large screen so that users could look up and see what the other groups were doing. Using multiple windows also gives an easy way to do the side by side comparison of two or more alternative diagrams, without having to have two facilitator stations and two large screen displays.

12.5 LESSONS

Transforming an academic prototype to a commercial product is not an easy task. The program needs to be more robust with respect to features and reliability, it must be intuitive for the average user (i.e. someone not trained in computer science), it must be useful for day-to-day tasks, and it should incorporate mechanisms which are likely to create tangible benefits. Over the last year GroupGraphics has undergone this process.

The process of turning GroupGraphics into a commercial package included all the 'drudge' work one might expect: refining menu names, redesign of dialog boxes, context-sensitive help, user manuals and extra debugging. All this took time and resources, but this was expected. Conceptual issues which had to be resolved centred on where to draw the line between the individual's work environment and the group's environment. It is common for software packages to 'remember' menu and feature settings which were active the last time an application was executed for a particular data file. The problem which arises is how to determine what are group settings and what are individual preferences. If individual preferences are to be retained, should they be stored within the drawing file or in different locations? This dilemma is compounded by the fact that users will expect their settings to follow them from one workstation to the next. For reasons of simplicity and consistency, initialization settings are kept in a separate file which can be associated with a

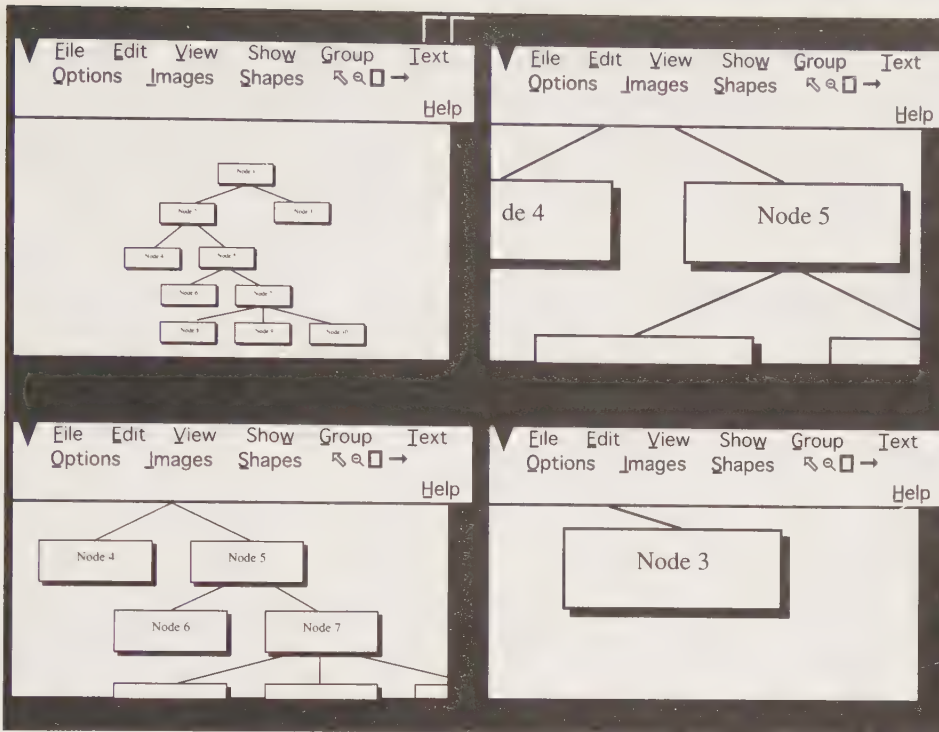


Figure 12.6 Multiple views of the same diagram.

particular drawing file at run time. A second problem involved whether or not to invoke restrictions on some operations when working in shared mode. Commands such as Clear All may be too destructive: allowing a user to 'select' all the objects and not allow access by others is also counter-productive. It is impossible for a program to prevent destructive behaviour completely, however, a program can provide a certain amount of protection to a group. For this reason, functions such as Clear All and Select All Objects in Region are disabled when running in the shared mode. An option which could be implemented would be to grant special access to some members of the group, either the team leader or the creator of the diagram. This may be added as an option in the future.

The use of a non-preemptive multitasking OS (Windows) is a major problem for drawing tools which attempt to support real-time WYSIWIS via peer-to-peer network connections. GroupGraphics is implemented as two programs, a drawing program and a network program. The network program runs in the background, receiving packets and assembling and queuing messages for applications. This separates the network functions from GroupGraphics, allowing better use of limited PC LAN adapter resources and multiple groupware programs to be active at the same time. When the graphics portion of GroupGraphics refreshes a screen with many objects and several large-scale bitmaps it can require upwards of 30 s of processor time (16MHz 386). If there are even a modest number of users (4 or 5), performing operations such as dragging or pasting sets of objects, then the available network receive buffers will fill quickly, causing transmit time-outs and

broken connections. To solve this problem it was necessary to quadruple the number of buffers available for receiving incoming packets, and place an explicit 'Yield' call in the repaint logic of GroupGraphics. This allows the network program to run and process the incoming packets. In a true multi-tasking OS, these actions might not have been necessary. Yield calls can be placed in new groupware applications, but if existing single-user applications hold the processor for too long, communications will be disrupted.

ACKNOWLEDGEMENTS

The research described in this article has been supported by Ventana Corporation, the University of Arizona and the University of Florida. Special thanks go to Stephen Hayne, Bill Saints, Jay Nunamaker, Doug Vogel and Craig Aytes for their participation in the development and testing of GroupGraphics.

REFERENCES

- Buxton, W. and Moran, T. (1990) EuroParc's integrated interactive intermedia facility (IIIF): early experiences, in Gibbs, S. and Verrijn-Stuart, A. A. (eds.), *Multi-User Interfaces and Applications*, North-Holland, Amsterdam.
- Conklin, J. and Begeman, M. L. (1988) gIBIS: a hypertext tool for exploratory policy discussion, *ACM Transactions on Office Information Systems*, October, pp. 140–52.
- Clark, S. and Scrivener, S. A. R. (1992) 'The ROCOCO sketchpad distributed shared drawing surface', LUTCHI Report Number 92/C/LUTCHI/0150, LUTCHI Research Centre, Loughborough University of Technology, Leicestershire, England.
- Delisle, N. M. and Schwartz, M. D. (1989) Collaborative writing with hypertext, *IEEE Transactions on Professional Communications*, **32**(3).
- Dennis, A. R., George, J. F., Jessup, L., Nunamaker, J. F. and Vogel, D. (1988) Information technology to support electronic meetings, *MIS Quarterly*, **12** (4) pp. 591–694.
- Ellis, C. A. and Gibbs, S. J. (1984) Concurrency control in groupware systems, in *Proceedings of the ACM International Conference of the Management of Data (SIGMOD)*, ACM Press, pp. 399–407.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991) Groupware: some issues and experiences, *Communications of the ACM*, **34**(1), pp. 38–58.
- Englebart, D. C. and English, W. K. (1968) A research centre for augmenting human intellect, *Proceedings of the Fall Joint Computing Conference*, Washington DC, December, pp. 395–410.
- Greenberg, S. (1991) Personalizable groupware: accommodating individual roles and group differences, in *Proceedings of the European Conference on CSCW*, Amsterdam, Netherlands.
- Greif, I. and Sarin, S. (1987) Data sharing in group work, *ACM Transactions on Office Information Systems*, **5**(2) pp. 187–211.
- Halasz, F. G., Moran, T. P. and Trigg, R. H. (1987) Notecards in a nutshell, in *ACM Conference on Human Factors in Computing Systems*, April, ACM Press.
- Ishii, H. and Miyake, N. (1991) Toward an open shared workspace: computer and video fusion approach of TeamWorkStation, *Communications of the ACM*, **34**(12), pp. 37–50.
- Nunamaker, J., Vogel, D., Heminger, A., Martz, B., Grohowski, R. and McGoff, C. (1989) Experiences at IBM with group support systems, a field study, *Decision Support Systems*, **5**, pp. 183–96.
- Pendergast, M. and Beranek, M. (1991) Coordination and control for collaborative workstation design, in *Proceedings of the HCI '91 Conference*, August.
- Pendergast, M. and Hayne, S. (1991) Assisting groups during the information on merging process, *Decision Sciences International Conference*, Miami, November, pp. 720–5.
- Posner, I., Baecker, R. and Mantei, M. (1992) How people write together, in *Proceedings of the Hawaii International Conference on System Sciences*, Hawaii, pp. 127–38.

- Rein, G. L. and Ellis, C. A. (1991) rIBIS: a real-time group hypertext system, *International Journal of Man Machine Studies*, **34**(3) pp. 349–68.
- Sarin, S. and Greif, I. (1985) Computer-based real-time conferencing systems, *IEEE Computer*, **18**(10) pp. 33–45.
- Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. and Tatar, D. (1987a) WYSIWIS revised: early experiences with multi-user interfaces, *ACM Transactions on Office Information Systems*, **5**(2) pp. 147–86.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. (1987b) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, **30**(1), pp. 32–48.
- Streitz, N., Hannemann, J. and Thuring, M. (1989) From ideas and arguments to hyperdocuments: travelling through activity spaces, *Hypertext 1989 Proceedings*, November, pp. 343–64.
- Tang, J. C. and Minneman, S. L. (1990) VideoDraw: a video interface for collaborative drawing, in *Proceedings of the Conference on Computer Human Interfaces*, April, pp. 313–20.

TELEPICTIVE: COMPUTER-SUPPORTED COLLABORATIVE GUI DESIGN FOR DESIGNERS WITH DIVERSE EXPERTISE

David S. Miller, John G. Smith and Michael J. Muller

ABSTRACT

It is generally accepted that it is important to involve the end users of a graphical user interface (GUI) in all stages of its design and development. However, traditional GUI development tools typically do not support collaborative design. TelePICTIVE is an experimental software prototype designed to allow computer-naïve users to collaborate with experts at possibly remote locations in designing GUIs. TelePICTIVE is based on the PICTIVE participatory design methodology, and has been prototyped using the Rendezvous system. In this paper we describe TelePICTIVE, and show how it is designed to support collaboration among a group of GUI designers with diverse levels of expertise. We also explore some of the issues that have come up during development and initial usability testing, such as how to coordinate simultaneous access to a shared design surface, and how to engage the participatory design of GUIs using a computer-supported cooperative work (CSCW) system.

13.1 INTRODUCTION

Over the last few years, many software tools have emerged which assist in the development of graphical user interfaces (GUIs). Most of these tools, however, concentrate on the implementation of GUIs rather than on their design. Yet the design phase is at least as important a part of the development process as the implementation (Goodwin, 1987; Gould and Lewis, 1985). In addition, most of the tools which do assist in the GUI design process do not encourage the practice of participatory design, which attempts to make interfaces more usable by involving users in the design process (Namioka and Schuler, 1992). Unfortunately, the users of applications are often not as familiar with GUI

The original version of this paper appeared in the *Proceedings of the ACM Symposium on UIST*. Copyright 1992, Association for Computing Machinery, Inc. Reprinted by permission.

technology as GUI developers are. Therefore, any GUI development tool which purports to encourage participatory design must be oriented towards naïve users in order to enable them to participate in the design process. In addition, it must support synchronous collaborative work in order to allow them to actively cooperate with experts in the design process.

TelePICTIVE has been designed to fill this gap. TelePICTIVE is an experimental object-oriented software prototype designed to allow naïve as well as expert users to work together in designing GUIs. It has been developed in the Rendezvous system which provides an architecture for multi-user multimedia synchronous (MUMMS) applications (Patterson *et al.*, 1990), and is based on the PICTIVE 'paper and pencil' participatory design methodology (Muller, 1991). The design objectives behind TelePICTIVE include:

- Make the TelePICTIVE interface powerful enough to assist in the design of complex interfaces, yet simple enough to allow naïve users to be comfortable with it.
- Provide as much communication bandwidth as possible in order to allow designers to work together from remote locations, yet prevent the designers from inadvertently interfering with each other's work.
- Provide intelligent assistance to the design process.
- Develop a front end to an environment which can provide information relevant to the design process, including other GUI designs, relevant style guides, usability data, repositories of design components and so on.

We believe that the use of TelePICTIVE will simplify as well as enhance the GUI development process. We expect that once a usable and complete GUI has been designed through the use of TelePICTIVE, it will be a relatively simple matter to use traditional interface development tools (IDTs) to actually implement the design. In this case, it will be important that the design be well specified before the implementation process begins. In the remainder of this paper, we will first discuss the PICTIVE methodology and Rendezvous system, which provide the infrastructure for TelePICTIVE. We will then outline how TelePICTIVE relates to other systems with similar goals. Next, we will provide an overview of the features of the current working experimental prototype. We will then discuss the results of initial usability testing, and some of the issues that have been raised by our initial experience with TelePICTIVE. Finally, we shall provide an overview of the architecture and future directions of TelePICTIVE.

13.2 BACKGROUND

13.2.1 PICTIVE participatory design

In general, participatory design attempts to empower and enfranchise users in the design process, either by making them direct participants in the design team, or by making design team members direct participants in the users' workplaces (e.g. Bjerknes *et al.*, 1987; Bodker *et al.*, 1988; Greenbaum and Kyng, 1991; Namioka and Schuler, 1992). The PICTIVE technique attempts to provide an 'equal opportunity' design environment in which a variety of stakeholders can contribute their ideas with equal ease and access. This is accomplished through the use of a deliberately low-tech set of design objects, consisting mostly of common office tools such as coloured markers, coloured paper, Post-It notes or

other removable labels, scissors and so on. In addition, some custom objects may be constructed for specific user needs or specific target environments, such as pre-printed icon sets or window borders (see Muller (1991, 1992) for details). The guiding notion is that, with the exception of certain disability conditions, all participants have approximately equal skill at 'operating' the PICTIVE design objects. This is also the guiding principle for TelePICTIVE, and is in contrast to current software rapid prototyping environments, in which users and some other stakeholder are often at a skill advantage with respect to software professionals. The motivation for building TelePICTIVE, is a computerized version of PICTIVE, included:

- Supporting 'design at a distance' so that designers at remote sites could collaborate in designing a GUI.
- Creating a fully accessible history of the design session.
- Providing intelligent assistance in the design process.

These points will be discussed further in the description of the features and architecture of TelePICTIVE. Finally, not only is TelePICTIVE based on the PICTIVE methodology, but it has been designed using the PICTIVE technique. The designers of TelePICTIVE include practitioners of the current PICTIVE methodology and future TelePICTIVE users, as well as human factors experts and experienced GUI developers.

13.2.2 Rendezvous groupware

TelePICTIVE was implemented using the Rendezvous groupware infrastructure (Patterson *et al.*, 1990), which is an object-oriented event-driven language for constructing MUMMS applications. It is built on top of the common lisp object system (CLOS) (Steele, 1990) and the graphics and network communications are based on the X-Window system (Scheffler and Gettys, 1986). The Rendezvous system provides a powerful and extensible object hierarchy through which we were able to implement representations of the PICTIVE objects as part of the prototype implementation of TelePICTIVE. In addition, the multi-user nature of the Rendezvous architecture greatly facilitated our ability to develop TelePICTIVE to support collaborative work.

13.3 RELATED WORK

TelePICTIVE is related to two classes of systems: GUI development systems such as IDTs and computer-supported cooperative work (CSCW) systems which support collaborative design. We are not aware of any other systems besides TelePICTIVE which specifically provide support for the collaborative design of GUIs. Research GUI development tools as well as commercial ISTs are not usually meant for computer-naïve users and do not provide support for collaborative design of interfaces. Systems such as Grow (Barth, 1986), Picasso (Rowe *et al.*, 1991), InterViews (Linton, 1989) and Garnet (Myers *et al.*, 1989) provide object-oriented environments for GUI development, but these environments are oriented towards assisting programmers with the implementation of GUIs. Quick (Douglas *et al.*, 1990) is a GUI development toolkit oriented towards non-programmers, but does not provide an environment for collaborative design. Tools such as these are

most useful for assisting with the implementation of a GUI once the design has been completed. The Colab project studied computer support for collaboration in meetings, and the results of that work suggest that user interface simplicity and simultaneous access are important to all multi-user tools (Stefik *et al.*, 1987). Many research CSCW systems support simultaneous access to a shared drawing space (Bly and Minneman, 1990; Mantei, 1988) or shared editors (Ellis *et al.*, 1990; Olson, 1990; Bier and Freeman, 1991), but these systems do not specifically support the collaborative design of GUIs, and few are oriented towards naïve users. These systems provide valuable insight into how to build systems meant for cooperative work, however, and we have attempted to incorporate some of this insight into the design of TelePICTIVE.

13.4 TELEPICTIVE

In the following section, we will give an overview of the features of TelePICTIVE. Unless specifically noted, all the features we discuss are implemented in the current experimental prototype. In order to provide a framework for examples in our discussion, we will begin with an example of how a typical design team would use TelePICTIVE.

13.4.1 Designing using TelePICTIVE

For the sake of this example, assume that the graphical interface of a telecommunications application needs to be developed. A preliminary design team has been assembled which includes a developer in New Jersey, a graphical designer in Texas and an eventual end user of the application in California. TelePICTIVE would be started up on the developer's workstation in New Jersey, and separate views would be sent to each of the participant's workstations (or X-terminals) using the X protocol. The participants first decide to work on the set of secondary windows for this application. The developer drags a window onto the shared design surface. The graphical designer then adds a label and a button panel to the window and sets its foreground colour using the options editor (see Figure 13.1 for a view of the TelePICTIVE session). In the meantime, the developer drags another window onto a separate section of the design surface. The end user, who is able to tell exactly where the developer is working on the design surface by looking in the UserInfo area (Figure 13.1), and by monitoring the developer's pointer, then expresses concern that the new window needs a text label to clarify its purpose. The end user then drags a text item onto the window and enters the appropriate text into it. When the end user makes a typographical error, the developer is tempted to fix it quickly, but cannot interfere with the object until the end user is finished with it. In the meantime, the graphical designer, who has forgotten exactly what secondary windows should look like, checks an on-line GUI style guide and usability data for further information on secondary windows. The participants confer over what the secondary windows for this application should look like, and then set the global properties of windows so that each new window that is dragged onto the design surface will conform to the established guidelines. The participants then finish designing the secondary windows for their application and save the design. This automatically saves a log of all changes made to the design since the last save. Future designers and developers could then build upon this initial design, and could revisit the design log to inspect or modify earlier versions of the design.

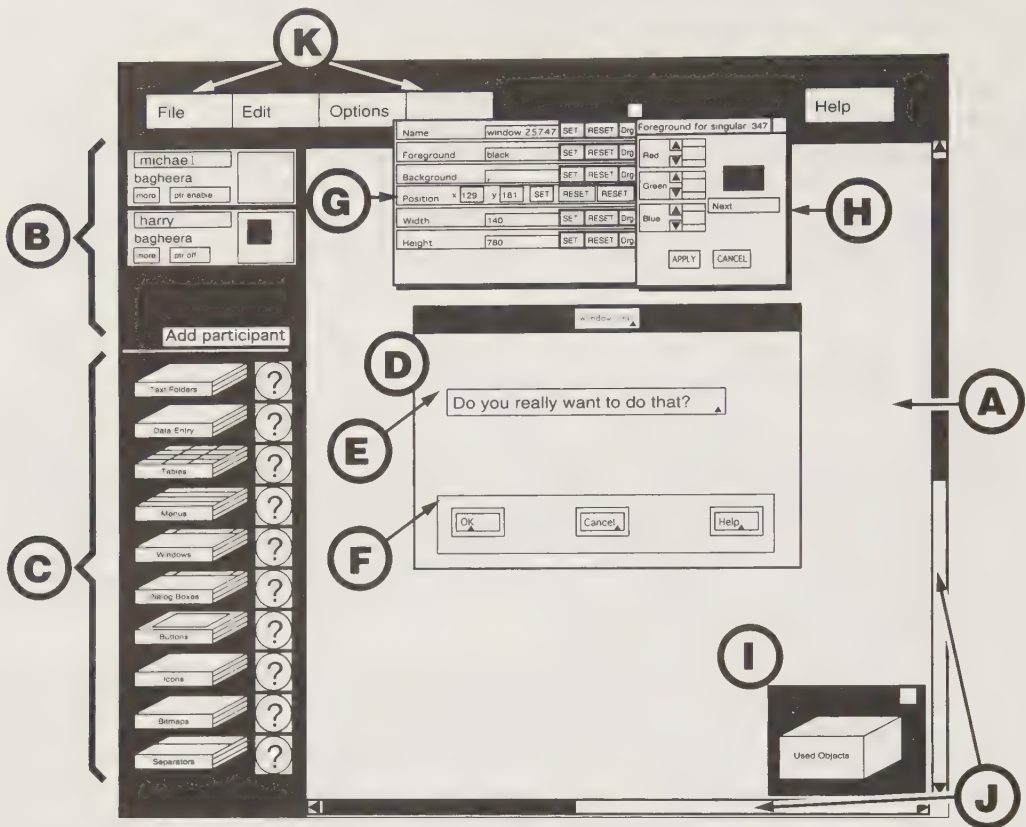


Figure 13.1 One participant's view of TelePICTIVE session.

13.4.2 Features of TelePICTIVE

The combination of features of TelePICTIVE that make it unique are that it is designed to support designers of diverse expertise, to support collaborative design from remote sites and to provide intelligent assistance to the design process.

Supporting designers with diverse expertise

As mentioned previously, the TelePICTIVE interface is being designed by its potential users and by users of the existing PICTIVE methodology, as well as by experienced GUI developers. This design process is intended to make TelePICTIVE usable by this subset of designers as well as by other designers with a wide range of expertise. To support users with little expertise in the use of graphical interfaces, all operations in TelePICTIVE can be

accomplished either through the use of menus or by drag and click operations with the left button of the mouse. For example, objects are 'added' to the design surface by dragging them off a pile of like objects in the Palette area (Figure 13.1(c)) onto the design surface, and are 'removed' from the design by dragging them into the Used Objects Box (Figure 13.1(i)). Operations such as these are designed to be intuitive and to require no expertise to use. In addition, objects, by default, can only be manipulated by one user at a time. Therefore, less experienced users do not have to be concerned about other users interfering with the objects they are working on. For users with more expertise, the TelePICTIVE interface is customizable. For example, text items in each view can be set to accept different sets of editing commands (e.g. Emacs, vi). This means that text items in the view of an experienced user can be set up to emulate his or her favourite editor, while the text items of an inexperienced user can be set up to accept a general set of editing commands that are easy to learn. Also, many TelePICTIVE operations can be accomplished through multiple methods (e.g. through a button click or a menu choice) for users with different preferences. For example, the option menu on a particular object can be invoked either through the top-level options menu (Figure 13.1(k)), or by pressing the right button of the mouse on the object.

Supporting collaborative design from remote sites

One of the major advantages that TelePICTIVE has over PICTIVE, is that it supports 'design at a distance'. Traditional participatory design is simply not possible if participants are geographically separated and are either unwilling or unable to travel. Hence, one of the major advantages of TelePICTIVE is being able to involve users in design sessions who otherwise would not be able to get involved at all. The current TelePICTIVE prototype supports collaborative design through the shared design surface, and the UserInfo area (Figure 13.1(b)). Whenever a participant manipulates an object on the shared design surface, all other participants simultaneously see the effect of the manipulations. The information about where each user is currently viewing is linked through the UserInfo area, allowing each participant to tell where any of the other participants is currently working on the design surface. This information is provided in the UserInfo area through 'box-within-a-box' notation, where the inner box represents the user's view and the outer box represents the total area of the shared design surface (Figure 13.1(b)). In addition, each user is assigned a unique colour, which is displayed as the colour of the inner box, and which highlights each object that the user selects. Any participant can explicitly match his or her view of the design surface to any other participant's current view by clicking on the representation of that user's design surface. Also, users can use the UserInfo area to obtain general information about other participants (such as where their machine is located), and to control whether or not to monitor their pointer through a telepointer. Telepointers display wherever in the design surface the corresponding participant's pointer is located. They are displayed in the colour of the participant whose pointer they are linked to and are tagged with the participant's user name. In the current implementation, participants can communicate with each other through text or through the manipulation of graphics, but still require a separate audio link for audio communication. This is not a major drawback to using the current system, though, since a multi-user audio link (a conference call) is readily available to most people, while a workstation equipped with audio capabilities is not.

We have experimented with using TelePICTIVE to design collaboratively over distances using a telephone as the audio link, and have not encountered any significant problems. We plan to integrate audio communication into TelePICTIVE in the near future, and video communication in the not so near future. Once audio and or video is integrated into TelePICTIVE, they will become part of the design log, and a full record (including audio conversations) of each design session could be saved and loaded. These design records (with or without audio records) support asynchronous collaborative design in the sense that designers could use the record of previous designs in their design sessions, as well as leave notes for future designers.

Intelligent assistance

Each TelePICTIVE GUI widget is represented by an object that was specifically designed to represent that widget and can behave appropriately. The TelePICTIVE object represents both the syntax of the GUI widget in terms of its appearance, available parameters, etc., as well as its semantics in terms of how the object behaves when placed on other objects, when other objects are placed on it etc. For example, in the current implementation of TelePICTIVE, placing a window object on a button object results in the window becoming a pop up window, and placing the button on the window results in the button becoming a child of the window. Each of the TelePICTIVE objects 'knows' its role in the interface. In addition, TelePICTIVE could be run in a demo mode, where objects assume their intended functionality in the interface being designed. For example, a text entry object, which is essentially a rectangle in regular design mode, would be fully editable in demo mode. Conversely, a text label object which is editable in design mode would not be editable in demo mode.

Finally, intelligent assistance from outside sources is being incorporated into TelePICTIVE. For example, we are in the process of integrating an on-line GUI style guide with TelePICTIVE which will provide rules for interface design that TelePICTIVE could read and use. Once this is accomplished, if a participant designs a widget that does not conform with style guide principles, for example, a warning could appear and the on-line style guide could be opened to the appropriate section.

13.5 INITIAL EXPERIENCE WITH TELEPICTIVE

13.5.1 Results of initial user tests

We have completed some initial user tests with the TelePICTIVE prototype and present the results here. Our initial experience with naïve users involved school teachers affiliated to the Bellcore Teacher Institute. After a brief introduction to the purpose and features of the system, two of the participants were given the task of designing the interface for a multimedia directory of historical persons. Even though these participants had little to no experience in the use of GUIs, they were able to make use of the system with minimal coaching and design a simple interface. Although they were hampered somewhat by performance problems, they reported afterwards that they found the system useful and fun. Our initial formal usability tests involved groups of two or three GUI designers, most of whom had experience with PICTIVE, but none of whom had experience using

TelePICTIVE. We placed the first two participants at the same location (with separate screens) and the third participant at a remote location with a monochrome monitor and with a voice connection through a pair of standard speaker phones. After some basic instructions, the participants were asked to complete several GUI design tasks through cooperative use of the system, and then were asked to evaluate the use of the system. The first series of questions we asked our users were how they would compare the use of TelePICTIVE with the use of PICTIVE.

TelePICTIVE was compared most favourably with PICTIVE in the following categories:

- ability to collaborate remotely
- changing the attributes of objects (e.g. size, position, colour)
- working collaboratively and avoiding conflicts

TelePICTIVE was compared least favourably with PICTIVE in:

- communicating ideas to others
- knowing what other participants are doing
- expressing ideas on design surface

We next asked our participants some questions about TelePICTIVE alone. The best aspects of the system were considered to be:

- made it easy to construct and share interface design objects
- design surface was equally accessible, and collaboration was enhanced as a result
- system was fun to use

The worst aspects of the system were:

- performance problems
- communicating with other participants and coordinating work

13.5.2 Issues in use of TelePICTIVE

We will now discuss some of the issues raised by our initial experience with TelePICTIVE.

Participatory design using CSCW

The goal of participatory design is to involve end users in the design process. However, these users may not initially be comfortable with computers, complicating the matter of designing a computerized tool to involve them in the design process. The rationale for the original PICTIVE paradigm was that users would be more effective in expressing their ideas and defending their interests if the design environment were composed of familiar, easily 'operated' objects. We attempted to duplicate this ease of operation as much as possible in the TelePICTIVE interface, and to provide support for naïve users as documented in Section 13.4.2. We believe that, although it is difficult, it is possible to design an interface that naïve users can quickly become comfortable with. Our experience suggests that this is important for any CSCW tool, since the work of a group will be limited if any members of the group have difficulty in understanding the interface. Our experience so far with naïve users (including some users who had never seen a GUI before)

suggests that TelePICTIVE is indeed easy to use. With minimal instruction and help, the naïve users were able to complete GUI design tasks using TelePICTIVE. The central complaint of these users concerned the performance and speed of the system, rather than any difficulties in learning the interface. In addition, a large majority of the users reported the system as ‘fun to use’, suggesting that they had little problem using the interface.

Collaborative design using CSCW

As detailed in Section 13.4.2, one of the major goals of TelePICTIVE is to support the design of GUIs in a CSCW environment. We hope for TelePICTIVE not only to support the collaborative process, but to actually enhance it. One way this can be accomplished is by using the system as a ‘mediator’ for the group. For example, the current version of TelePICTIVE prevents more than one user from controlling the same object at the same time. In addition, it informs participants if any actions they take are likely to affect the other participants (such as clearing the design surface). It is interesting to note that although our participants reported that TelePICTIVE did enhance the collaborative process compared with paper PICTIVE, they reported at the same time that it limited communication. Participants reported that the lack of ability of any participant to take control over the design session enhanced the collaborative process. On the other hand, participants found that limited communication made it relatively difficult to coordinate the group’s work. This could be improved, however, through enhanced communication facilities (audio and video links) as well as through additional computer mediation.

Coordinating simultaneous access to shared design surface

Another interesting issue in designing TelePICTIVE has been how to coordinate simultaneous access to a shared design surface. It is difficult to determine how much the actions of a participant should be allowed to affect the work of other participants. If there is no control, users could inadvertently interfere with each other’s work. If there is too much control, cooperative work could be inhibited. This issue has been explored before (Greenberg, 1990), and there are research and commercial examples of both extremes. We believe that the solution lies somewhere in the middle. Some objects should be ‘locked’ while being manipulated by a participant so that the other participants cannot interfere with that object until the first participant is finished with it. This is the default in the current implementation.

On the other hand, sometimes multiple users should be allowed to edit the properties of a single object cooperatively. For example, multiple users could edit different sections of a text item at the same time, or one user could change its size while another user is changing its colour. We believe that users should be given some control over the level of locking of shared objects in a multi-user interface. Our participants generally agreed that the system made it easy to avoid conflicts between participants, but did not necessarily agree that the system made it easy to coordinate work. The version of the prototype that was used, however, did not provide any control over locking (all objects were automatically locked during use). We plan to implement some level of control over locking and then further study any problems participants have in coordinating work.

Privacy

Although it is important that certain actions by a TelePICTIVE participant be synchronously related to all other participants (such as a change to the design), sometimes it may be important for certain actions to be private to a particular user. For example, an inexperienced user may be unsure of a proposed change, and may want to play with some ideas before presenting them as changes to the design.

In the current version of TelePICTIVE, this user would be able to turn off access to his or her pointer by other participants, and could edit the properties of an object on an options window (off the design surface). However, as soon as any change was applied to an object on the design surface, all other users would see it. There is no space for the user to design 'privately'. Of course, it is possible to add private design surfaces to TelePICTIVE, but this raises the problem of what would happen when different participants made inconsistent changes in their private spaces and attempted to apply these changes to the shared design. Our experiences with PICTIVE have shown the importance of carefully limiting the extent to which participants can branch into separate, parallel design sessions (Muller, 1992). This argues against providing too much 'private' design surface in a cooperative design environment. Our experience with users so far has shown that privacy is not a major concern. In fact, our users reported that their collaboration was hampered by any component of TelePICTIVE that currently is individual to each user, such as options windows (Figure 13.1(g)). This suggests that all components of TelePICTIVE should be shared by default, though some may have the option of being displayed privately on an individual view.

13.6 TELEPICTIVE ARCHITECTURE

As mentioned previously, the Rendezvous system provides the infrastructure for TelePICTIVE. The Rendezvous system architecture is based on the abstraction-link-view paradigm (Hill, 1992), in which a centralized abstraction object is shared among all users, each of whom has a separate view object, connected to the abstraction via a link object (Figure 13.2). The views provide displays of the shared data from the abstraction specialized for the user of that view by a set of constraints in the link. In the current Rendezvous system architecture, the abstraction and each view run as separate processes, but all run on the same physical processor. Communication between different machines occurs at the level of the window system protocol.

The Rendezvous system has a sophisticated constraint system for maintaining dependencies between objects as well as between the slots of a single object. When a TelePICTIVE session is started a TP-Abstract-Root object is created as the root of the TelePICTIVE abstract object tree (see Figure 13.3 for the abstract object tree corresponding to the view in Figure 13.1). The purpose of this object is to hold state about the session as a whole, and to organize the tree of abstract objects. The children of the abstract root object are the abstract Canvas, the abstract UserInfo, and the abstract Used Objects Box. The abstract Canvas represents the abstraction of the shared design surface. Any child of the abstract Canvas represents an object in the design. For example, the abstract window and its children (as in Figure 13.3) represent the window object in the view as shown in Figure 13.1(d), (e) and (f). The abstract UserInfo represents information

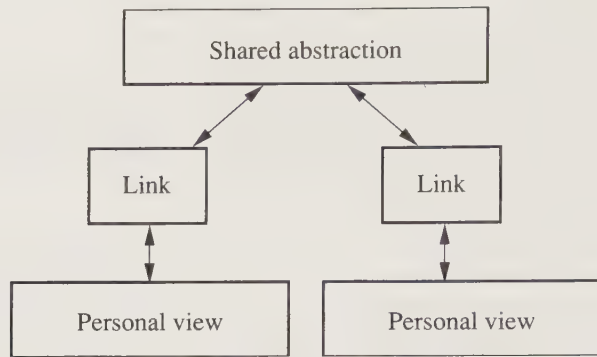


Figure 13.2 The abstraction-link-view paradigm.

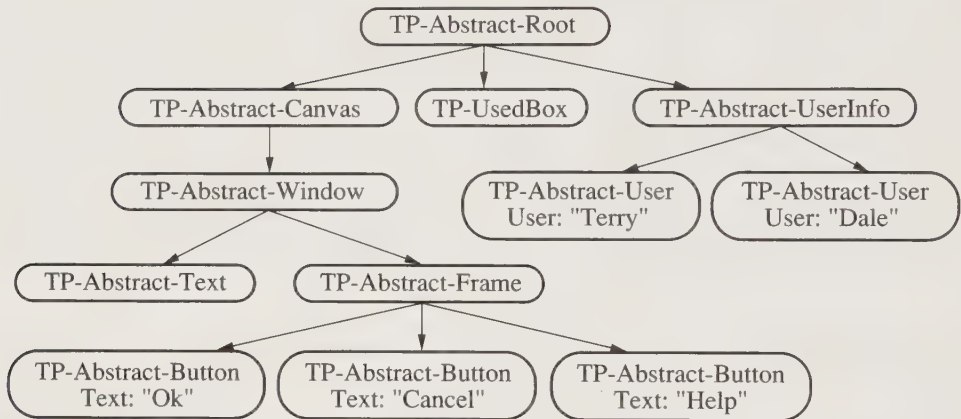


Figure 13.3 Abstract object tree.

about the participants in the current session. Its children represent information about individual participants in the current TelePICTIVE session. Finally, the abstract Used Objects Box represents information about objects which have been removed from the design surface. When a view is started for a TelePICTIVE session, each of the existing abstract objects becomes linked to a new corresponding view object which is displayed on the new participant's terminal. In addition, a new abstract User object is created which represents information about the new user, and which is linked to each of the existing views. This way, the new participant sees what every other participant is looking at as soon as he or she joins the session, and each of the other participants is provided with information about the new user. Also, non-shared objects, such as the menus and scroll bars are created for the new view (Figure 13.1(j)(k)). Since non-shared objects are not linked to other participants, the new participant can then independently select menu choices or scroll through his or her view of the canvas without affecting any of the other participants. When a participant moves an object onto the shared design surface from the Palette area (Figure 13.1(c)), the abstract version of that object is created and made a child

of the abstract Canvas. This causes a corresponding view object to be placed in each participant's view. Since each of the GUI widgets that participants are able to manipulate in a TelePICTIVE session are represented by a unique Rendezvous view object, the widgets can be given unique, customizable behaviour appropriate to their type. For example, text items are editable, and their properties, such as font and the editing commands they accept, can be controlled. In addition, any participant has the ability to edit the properties of either an individual object or an entire class of objects (see Figure 13.1(g) for an example of an option editing window). Any participant can save a TelePICTIVE design session at any time. Saving a design session essentially involves saving the tree of abstract object instances currently present in the abstract work space. This is done by writing Rendezvous system code which recreates these objects upon being loaded. This approach has the advantage of making the save file generic and readable, and also compilable for the sake of efficiency. Individual objects can be removed from the design surface and saved by placing them in the Used Objects Box (Figure 13.1(i)).

13.7 FUTURE PLANS FOR TELEPICTIVE-MEGA

TelePICTIVE is envisioned as the central component of a longer term project called the multimedia environment for GUI authorship (MEGA). MEGA will be a more complete environment for designing GUIs, which will provide a wide variety of support and information relevant to the GUI design process. The first extension we plan to make to TelePICTIVE involves the multimedia component. As mentioned previously, we plan to integrate audio and video communication into TelePICTIVE not only to enhance the communication between participants, but to produce a more complete on-line record of the design. In addition, audio and video are likely to be a component of the GUI being designed, as more and more software systems are moving towards multimedia interfaces. As also mentioned previously, participants in a TelePICTIVE design session will be able to actively build upon the work of previous designers by using a repository of previous designs. This design repository will form a component of MEGA. Another component of the MEGA system will be the on-line GUI style guide, which will not only provide information relevant to the design process, but will provide intelligent assistance as described in Section 13.4.2. Other components of the MEGA environment will include repositories of usability data and GUI widget code. These repositories will be indexed to TelePICTIVE object types, so that a user designing a menu, for example, will have access to usability data relevant to menus, as well as the working code which implements menus.

13.8 CONCLUSION

Participatory design is an important technique in the design of usable graphical user interfaces. However, although there are many tools designed to assist in the GUI development process, few support participatory design. TelePICTIVE is unique in that it is intended to support designers with diverse expertise in the collaborative design of GUIs. It is also designed to be the central component of a multimedia system that will provide intelligent assistance to the design process. Although it is only an experimental prototype, TelePICTIVE has already proven valuable in testing ideas involving how to build

collaborative GUI design environments, and in demonstrating how a CSCW tool might allow the participatory design of GUIs to be accomplished over wide geographical distances. We plan to have GUI designers begin to use the TelePICTIVE prototype as part of the design process for real applications, and hope that through the use of TelePICTIVE they will begin to quantify the benefits of participatory design as well as the benefits of using a computerized GUI design tool.

ACKNOWLEDGEMENTS

We would especially like to thank Danny Wildman and Ellen White for assisting in the design of TelePICTIVE by taking part in our PICTIVE sessions. In addition, we would like to thank John Patterson and Danny Wildman for their comments on the manuscript.

REFERENCES

- Barth, P. S. (1986) An object-oriented approach to graphical interfaces, *ACM Transactions on Graphics*, **5**(2).
- Bier, E. A. and Freeman, S. (1991) MMM: a user interface architecture for shared editors on a single screen, in *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology*, Hilton Head SC, ACM Press, pp. 79–85.
- Bjerknes, G., Ehn, P. and Kyng, M. (eds.) (1987) *Computers and Democracy: A Scandinavian Challenge*, Gower, Brookfield VT.
- Bly, S. A. and Minneman, S. L. (1990) Commune: a shared drawing surface, in *Proceedings of the Conference on Office Information Systems*, Cambridge MA, ACM Press, pp. 184–92.
- Bodker, S., Ehn, P., Knudsen, J., Kyng, M. and Madsen, K. (1988) Computer support for cooperative design, in *CSCW 88: Proceedings of the Conference on Computer-Supported Cooperative Work*, Portland OR, ACM Press, pp. 377–93.
- Douglas, S., Doerry, E. and Novick, D. (1990) QUICK: a user-interface design kit for non-programmers, in *Proceedings of the Third Annual Symposium on User Interface Software and Technology*, Snowbird UT, ACM Press, pp. 47–56.
- Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1990) Design and use of a group editor, in Cockton, G. (ed.), *Engineering for Human-Computer Interaction*, North-Holland, Amsterdam, pp. 13–25.
- Goodwin, N. C. (1987) Functionality and usability, *Communications of the ACM*, **30**, pp. 229–33.
- Gould, J. D. and Lewis, C. (1985) Designing for usability: key principles and what designers think, *Communications of the ACM*, **28**(3), pp. 300–11.
- Greenbaum, J. and Kyng, M. (1991) *Design at Work*, Erlbaum, Hillsdale NJ.
- Greenberg, S. (1990) Sharing views and interactions with single user applications, in *Proceedings of COIS*, Cambridge MA, ACM Press, pp. 227–37.
- Hill, R. (1992) The abstraction-link-view paradigm: using constraints to connect user interfaces to applications, in *Striking a Balance: Proceedings of CHI '92*, ACM Press.
- Linton, M. A. (1989) Composing user interfaces with InterViews, *IEEE Computer*, February.
- Mantei, M. (1988) Capturing the capture concepts: a case study in the design of computer-supported meeting environments, in *CSCW '88: Proceedings of the Conference on Computer-Supported Cooperative Work*, Portland OR, ACM Press, pp. 257–70.
- Muller, M. J. (1991) PICTIVE – an exploration in participatory design, in *Reaching Through Technology: CHI '91 Conference Proceedings*, ACM Press, pp. 225–31.
- Muller, M. J. (1992) Retrospective on a year of participatory design, using the PICTIVE technique, in *Striking a Balance: Proceedings of CHI '91*, ACM Press.
- Myers, B. et al. (1989) The Garnet toolkit reference manuals: support for highly interactive, graphical user interfaces in lisp, *Technical Report CMU-CS-89-196*, CMU, Pittsburgh PA.
- Namioka, A. and Schuler, D. (1992) *Participatory Design: Principles and Practices*, Erlbaum, Hillsdale NJ.

- Olson, G. M. (1990) Technology support for collaborative workgroups, *Technical Report*, University of Michigan, June 1990. Annual progress report for 1989-90 to the National Science Foundation, Grant No. IRI-8902930.
- Patterson, J. F., Hill, R. D., Rohall, S. L. and Meeks, W. S. (1990) RendezVous: an architecture for synchronous multi-user applications, in *Proceedings of CSCW '90*, Los Angeles CA, ACM Press, pp. 317-28.
- Rowe, A., Lawrence, J. A., Konstan, B. C., Smith, S., Seitz, S. and Liu, C. (1991) The PICASSO application framework, in *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology*, Hilton Head SC, ACM Press, pp. 95-105.
- Scheifler, R. W. and Gettys, J. (1986) The X-Window system, *ACM Transactions on Graphics*, **5**(2).
- Steele J., G. L. (1990) *Common LISP: The Language*, 2nd edn., Digital Press.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. (1987) Beyond the chalkboard: computer support for collaboration and problem solving in meetings, *Communications of the ACM*, **30**(1), pp. 32-47.

INDEX

- abstract documents, 160–161
 - abstraction-view-link model (AVL), 237, *see also*
 - separation of view from abstractions
 - access control, 110
 - additive operations, *see* concurrency control
 - annotations, 99, 128, 143, 148–149, 159, 167–170, 180, 216, *see also* gesturing
 - anonymity, 68, 194, 211
 - Apple Macintosh, 108, 129, 163, 184, 194
 - Appletalk, 50, 56, 109–110, *see also* communication networks
 - application programming interface (API), 202, 206
 - architectures:
 - centralized, 3, 24, 31, 40, 50, 51, 52, 54, 60, 128, 137–140, 167, 180, 200–201, 237
 - drafter-mirror, 103, 106, 110, 112
 - graphics kernel, 28–29, 40, 50, 52, 57–60, 128, 137–140, 145–146, 160–166, 216–223, 237–238
 - hybrid 50–51, 53, 60
 - logical layering 158–173
 - multi-depth displays, 136
 - multitasking, 135
 - replicated, 3, 24, 26, 29, 40, 50–52, 54, 60, 110, 128, 137–140, 146–148, 167–170, 180, 200–202
 - replicated and centralized trade-offs, 50, 137–140, 145, 167
 - run-time, 128, 146–148
 - SLICE model, 128, 157–173
 - slicing, 158–173
 - to support software development, 182, 184
 - Argnoter, 157, *see also* brainstorming systems
 - ARKola simulation system, 100, 168
 - arrival of participants, 19, 36, 40, 50, 52–55, 60, 128, 131, 144–145, 147–151, 168, 180, 201, 231
 - Aspects, 59, 69, 99, 144, 157, *see also* drawing systems, writing systems
 - attention switching, 104, 115, 119, 122, 144
 - audio links, 3, 9, 31, 41, 45–46, 65, 68, 85, 88, 99, 145, 158, 189, 194, 213–214, 223, 233, 236
 - audio quality, 182–183, 187, 189–191
 - audio records, *see* capture of meetings
 - auditory icons, 156, 162, 167–168
 - author identity, 162
 - awareness:
 - echoing, 161, 163–164, 166, 168
 - gestalt views, 48, 71, 164–166
 - localization, 161, 164–166, 168
 - of other's actions, 128, 158, 163–167, 224, 231, 235, *see also* telepointers
 - of other's location in the drawing, 90, 99–123, 218, 223–224, 233
 - radar views, 48, 71, 164–166
 - telepointers, *see* telepointers
 - through animation, 163
 - through sound, 90, 167–168, *see* auditory icons
 - through view slaving, 48, 72, 90, 166, 213–214, 218–219, 223–224, *see also* scrolling
- bandwidth, 31, 56, 60, 63, 69–70, 78, 181, 186, 229, *see also* communication network
 - being there, *see* telepresence
 - BoardNoter, *see* drawing systems
 - body language, 37, 103–123
 - bottlenecks:
 - network, 50, 56, 60, 63, 69–70, 225
 - processor, 50, 56, 69–70, 78
 - brainstorming, 41, 49, 88, 181, 209
 - brainstorming systems:
 - Argnoter, 157
 - Cognoter, 60, 99, 157
 - Graphical Brainstorming, 72
 - Graphical Issue Analyzer, 67, 71–72
 - business factors, 179–197
 - business opportunities, 181
- callbacks, 136, 152
 - camera positioning, 97, 100, 104, 108
 - capture of meeting:
 - audio records, 93
 - playback, 93, 162, *see also* history of interactions
 - search, 91–93
 - storing images, 110, 111, 112, 132, 180
 - through video, 82, 85, 91–92, 192, 234, 239
 - thumbnail sketches, 111
 - capturing the meeting process, 43, 82–83, 91–93, 108, 160–163, 180, 195, 216, 239
 - caricatures, 45
 - CaveCat, 100–101, *see also* media space systems
 - CaveDraw, 60, 99, 131, 157, 162, 171, *see also* drawing systems
 - centralized architectures, 3, 24, 31, 40, 50, 51, 52, 54, 60, 128, 138–140, 168, 180, 200–201, 237, *see also* architectures
 - chroma-keying, 110, 112
 - ClearBoard, 4, 49, 96–125, 130, *see also* drawing systems
 - ClearFace, 97, *see also* drawing systems
 - CLOS object system, 230
 - cloth erasers, 104, *see also* input devices
 - Codraw, 202, *see also* drawing systems

- Coed, 200, *see also* writing systems
- Coex, 176, 198–208, *see also* groupware toolkits
- Cognoter, 60, 99, 157, *see also* brainstorming systems
- CoLab, 130, 231, *see also* electronic meeting systems
- collaboration-aware groupware, 51, 130–131, 133, 142
- collaboration-transparent groupware, 51, 65–66, 99, 131, 145
- Commune, 59, 67, 99, 100, 130, 157, *see also* drawing systems
- communication latency, 56, 63, 79
- communication links:
 - audio, 3, 9, 31, 41, 45–46, 65, 68, 85, 88, 99, 145, 158, 189, 194, 213–214, 223, 233, 236
 - video, 3, 9, 25, 41, 60, 65–66, 99, 145, 158, 194, 213–214, 236, *see also* drawing systems: ClearBoard, ClearFace, LookingGlass, TeamWorkstation, VideoCom, VideoDraw, VideoWhiteboard
- communication network:
 - AppleTalk, 50, 56, 109–110
 - Ethernet, 74
 - groupware requirements, 29, 146, 186
 - groupware-specific protocols, 29, 52, 57–59, 72, 128, 131, 140, *see also* open protocols
 - ISDN, 3, 6, 11, 13, 97, 187
 - local area networks, 11, 13, 50, 56, 63, 72–79, 83, 140, 186–187, 209, 213, 215
 - modems, 180, 182, 186–187
 - multicasting, 57, 72–79, 146, 176, 201
 - need for many protocols, 186
 - Network Interface Object, 72
 - Novell, 72, 74, 186
 - sockets and TCP/IP, 29, 52, 56, 136, 147, 186, 205
 - telephones, 28, 41, 85, 100, 145, 176, 180, 186–187
 - wide area networks, 63, 140
 - X windows protocol, 129–130
- commutative operations, 170–171, *see also* concurrency control
- computer supported cooperative work, 5, 63, 129, 142, 180, 209–210
- computer-controlled video environments, *see* media space
- concurrency control:
 - general, 128, 138, 142, 146, 156, 168, 176, 182, 201, 215
 - additive operations, 31, 170–171
 - commutative operations, 170–171
 - immutable objects, 29, 32
 - interface to users, 47–48
 - locking, 31, 47, 52–53, 57–59, 90, 146, 161, 163, 166, 177, 215, 218, 231, 236, *see also* turntaking
 - logical clocks, 170–171
 - need for, 31, 47, 50, 52, 89, 140, 146
 - optimistic approaches, 47
 - serialization, 50, 52, 53, 170–171, 201
 - social protocol, 29, 33–34, 48, 89–90
 - time stamps 92, 170–171
 - transformations, 169–171
- concurrent data access, 214
- concurrent design, 5
- concurrent engineering, 5
- conference rooms, 180, 182–184, 189–192, 211
- Conference Toolkit, 146–147, *see also* groupware toolkits
- consistency, 212, *see also* concurrency control
- contact facilitation, *see* initiating groupware sessions
- control-display compatibility, 67, 103
- ConversationBoard, 131, *see also* drawing systems
- Costed, 202, *see* writing systems
- coupling, 48, 57–59, 67, 72, 146, 161, 163–164, 166, 167, 168, 176, 199, 205, 215
- cross platform implementation, 194, 206
- Cruiser, 99, *see also* media space systems
- CSCW, 5, 63, 129, 142, 180, 209–210
- cultural differences, 15, 18, 64, 121–122, 133, 187, 196, 203
- cursor wars, 66
- data conferencing, 38, 45, 181
- departures of participants, 53–55, 144, 147–149, 168
- design principles:
 - for a clearboard metaphor, 103, *see also* ClearBoard
 - for shared drawing, 3, 11, 24, 30–31, 41–43, 46, 59, 81, 93, 143–145, 156–158, 180, 189–196
- Design/OA programming interface, 169
- designated scribe, 88, 195
- desktop conferencing, 184–185, 211
- desktop video systems, *see* media space systems
- diagram markers, 223
- Dialogo, 99, *see also* window-sharing systems
- digitizing tablets, 4, 11, 17, 35, 46, 59, 67, 83, 86, 96, 103, 108, 112, *see also* input devices
- direct manipulation, 156–157, 160, 163–164, 166–167
- DisLayers system, 122
- distance education, 176, 180–183, 188–190
- DistEdit, 169–170, 206, *see also* groupware toolkits
- distributed meeting rooms, 119
- distributed systems: management of multiple processes, 128, 145, 147–149
- drafter-mirror architecture, 103, 106, 110, 112
- drawing:
 - artefacts, 41, 46, 86–87, 104, 110
 - conventional view, 42, 88–89,
 - freehand, 18, 24, 28, 64–65, 82, 103, 110, 132, 142, 149–150, 153–154
 - information storage, 42–46, 65, 82, 88, 91–93, 108, 132, *see also* persistent sessions, capture of meeting
 - object-based, 3, 28, 32, 47, 142, 146–147, 160–165, 176, 209, 215–221, 229, *see also* graphics kernel
 - orientation, 41, 46, 102–103, 122, 212
 - ownership of marks, 110
 - private, 35, 42, 47, 51, 87–88, 133, 161, 171, 237
 - role in small group design, 6, 41–42, 86–87
 - size, 46–47, 49, 88, 97, 104, 110, 132, 160, 216
 - stroke-based, 26, 28, 85, 92
 - temporal aspects, *see* history of interactions
 - to express ideas, 42–43, 63–67, 86–87, 143, 235
 - transparent image layers, 110, 158–159
- drawing systems:
 - Aspects, 59, 69, 99, 144, 157
 - BoardNoter, 59, 65, 157, 214
 - CaveDraw, 60, 99, 131, 157, 162, 171

drawing systems (*continued*)

- ClearBoard, 4, 49, 96–125, 130
 - ClearFace, 97
 - Codraw, 202
 - Commune, 59, 67, 99, 100, 130, 157
 - Conversation Board, 131
 - FreeStyle, 144
 - Group Teleconferencing Systems, 60
 - GroupDesign, 128, 156–173
 - GroupDraw, 37–62, 67, 99, 146
 - GroupGraphics, 176, 209–227
 - GroupKit example, 152–153
 - GroupSketch, 37–62, 66, 70, 99, 152, 157, 214
 - GroupSketchPad, 67, 70, 72
 - LookingGlass, 102, 215
 - Mblink, 214
 - MMM, 67, 146, 157, 163
 - Muge, 60, 67, 71–72, 209–210, 215
 - Rococo, 3, 5–23, 49, 103
 - Shared ARC, 71
 - Shdr, 56, 60, 67, 69
 - Smart 2000, 176, 179–197
 - Team Workstation, 59, 66, 96–125, 215
 - TeamPaint, 99, 110–112
 - TelePictive, 177–179, 228–241
 - Tivoli, 3, 24–36, 99
 - VideoCom, 112
 - VideoDraw, 59, 66, 100, 104, 214, 215
 - VideoWhiteboard, 59, 66, 102, 130
 - We-Met, 4, 81–95, 99, 167
 - Wscrawl, 51, 59, 60, 67, 99, 128, 129–141
 - Xerox LiveBoard, 24
 - XGroupSketch, 37–62, 67, 71
 - XSketch, 59, 131
- ease of use, 183, 187, 224, 229, 232–233, 235–236
- echoing, 161, 163–164, 166, 168, *see also* awareness
- electronic meeting systems:
- CoLab, 130, 231
 - general, 63–64, 142, *see also* group decision support systems
 - GroupSystems, 49, 209, 210, 211, 215
 - TeamFocus, 210
 - VisionQuest, 210, 215
- electronic whiteboards, 3, 8, 24, 28, 59, 65, 128, 129, 131, 180, 183, 189, 193
- Eoops object oriented language, 199, 202
- erasing, 26, 43, 45, 59, 108, 110, 132
- Esc system, *see* Gnu Emacs
- Ethernet, 75
- ethnography, 41, 64, *see also* studies
- expressing ideas, 42–43, 63–67, 86–87, 143, 235
- eye contact, 4, 37, 49, 96–97, 99, 101–103, 112, 115–122
- facial expressions, 99, *see also* body language
- facilitation, 41, 88, 91, 144, 180–181, 188, 190, 193, 224, *see also* social protocols
- facsimile transmission, 65
- familiarity with existing tools, 187
- feedback, 160, 163, 166
- file management, 36, 108, 112, 216

filter functions, 169

fine-grained interaction, 30, 34, 65, 133, 213

flat panel displays, 108

flicker, 115

floor control, *see* turn-taking

focus switching, 104, 115, 119, 122, 144

freehand drawing, 18, 24, 28, 64–65, 82, 103, 110, 132, 142, 149–150, 153–154

FreeStyle, 144, *see also* drawing systemsGarnet, 230, *see also* toolkits

gaze awareness, 4, 49, 67, 88, 96–123

GDSS, *see* group decision support systemsgestalt views, 48, 71, 164–166, *see also* awarenessgesture recognition, 26, 195, *see also* input devices

gesturing, 4, 26, 34, 42–43, 45, 63–70, 86–87, 97, 99, 104, 107, 110–112, 119, 128, 143–144, 160, 167, 187, 213–214, 224

gIBIS, 216, *see also* hypertext systems, brainstorming systemsglass board metaphors, 103, *see also* ClearBoard

Gnu Emacs, 199

Graphical Brainstorming system, 72, *see also* brainstorming systemsGraphical Issue Analyzer, 67, 71–72, *see also* brainstorming systems

graphical overlays, 149–150, 156–168

graphical user interface design, 177–179, 228–241

graphics kernel, 28–29, 40, 50, 52, 57–60, 128, 138–140, 146–147, 161–167, 216–223, 237–238

group-aware groupware, 51, 130–131, 133, 142

group decision support systems, 63–64, 146, 122, 215, *see also* group support systems

group familiarity, 191

group membership, 45, 70, 82, 161–162, 181, 188, 214

group size, 3, 48, 63–64, 70–71, 119, 140, 176, 180, 186–187, 192, 222–224

group support systems, 63–64, 142, *see also* group decision support systemsGroup Teleconferencing Systems, 60, *see also* drawing systemsGroupDesign, 128, 156–173, *see also* drawing systemsGroupDraw, 37–62, 67, 99, 146, *see also* drawing systemsGroupGraphics, 176, 209–227, *see also* drawing systemsGroupKit, 37, 40, 47, 50, 53, 59, 60, 142–155, 160, 169–171, 204, 206, *see also* groupware toolkitsGroupSketch, 37–62, 66, 70, 99, 152, 157, 214, *see also* drawing systemsGroupSketchPad, 67, 70, 72, *see also* drawing systemsGroupSystems, 49, 209, 210, 211, 215, *see also* electronic meeting systems

groupware definition, 99, 157

groupware for: distributed meetings, 5, 24, 83, 130–131, 147, 157, 179, 211

groupware for:

drawing, *see* drawing systems

face-to-face meetings, 8, 26, 48, 59, 81, 147, 181, 199

small group design, 6, 112, 161–162

structured diagrams, 156, 169–171

voting, 8

- groupware toolkits:
 - Coex, 176, 198–208
 - commercialization, 198–208
 - Conference Toolkit, 146–147
 - DistEdit, 169–170, 206
 - general, 40, 128, 156–158, 198–208
 - GroupKit, 37, 40, 47, 50, 53, 59, 60, 142–155, 160, 169–171, 204, 206
 - Liza, 146, 169
 - MMConf, 146, 150–151, 169–170
 - Rendezvous, 147, 169, 228–230, 238–239
- Grove, 99, 131, 157, 162–163, 169, *see also* writing systems
- Grow, 230, *see* toolkits
- half-silvered glass and mirrors, 9, 67, 102–109, 112, 121, *see also* video tunnels
- history of interactions, 4, 29, 83, 87, 91–93, 162–163, 202, 230–231
- horizontal displays, 59, 67, 100, 113
- human factors of shared drawing, 37, 41, 46, 133, 185, 189–196, 211–215, *see also* design principles for shared drawing
- hybrid architectures, 50–51, 53, 60
- Hydra, 121, *see also* media space systems
- hypertext systems:
 - glBIS, 215, *see also* brainstorming systems
 - Neptune, 214
 - Notecards, 214
 - Sepia, 214
- IIIF, 215, *see also* media space systems
- image formats:
 - BMP, 221
 - PCX, 221
 - Pict, 112
 - postscript, 138
 - X window dump (xwd), 138
 - X Windows bitmap, 138
- immutable objects, 29, 32, *see also* concurrency control
- implementation:
 - cross platform, 194, 206
 - of fine-grained interactions, 34
 - on personal computers, 4, 28, 50, 56, 63–64, 72–79, 176–197, 209–226
 - performance issues, 130, 133, 136, 181, 186, 235
 - technical considerations, 28, 37, 50–60, 128, 145–147, 156–161, 182, 199–200, 205, 213–215
 - telepointers, 55–57, 63–80, 128, 149–150, 160, 167, 187, 218, *see also* telepointers
 - user considerations, 31, 37, 143–145
- importing images, 110, 128, 132, 138, 189, 221
- industry partners, 185–186
- information search and retrieval, 91–93
- information storage, 42–46, 65, 82, 88, 91–93, 108, 132, *see also* persistent sessions, capture of meeting
- initiating groupware sessions, 19, 36, 40, 50, 52–55, 60, 128, 131, 144–145, 147–151, 168, 180, 201, 231
- input devices:
 - cloth erasers, 104
 - coloured pens, 96, 104–105, 108
 - gesture recognition, 26, 195
 - keyboard, 26, 86, 184
 - liveboards, 11, 17, 24
 - mouse, 26, 44, 46, 103, 160, 180, 184, 216, 233
 - palm-sized, 26
 - pen-based computers, 26, 35, 196, 216
 - stylus and digitizing tablets, 4, 11, 17, 35, 46, 59, 67, 83, 86, 96, 103, 108, 112
 - touch screens, 179, 180, 182–184
 - wireless, 26, 83, 122
- intentions of other participants, 34–35
- inter-process communications, 145
- international markets, 187, *see also* cultural differences
- interpersonal distance, 121–122
- interpersonal space, 96–97, 99, 101, 121
- InterViews, 147, 160, 202, 205, 230, *see also* toolkits
- ISDN, 3, 6, 11, 13, 97, 187, *see also* communication networks
- Isis, 169, *see also* toolkits
- Jot, 200, *see also* writing systems
- keyboard, 26, 86, 184, *see also* input devices
- latecomers to a conference, 145
- LCD displays, 108, 112, 114–115, 179, 180
- learning, 46, 102, 156, 176, 180, 183
- liquid crystal displays, 108, 112, 114–115, 179, 180
- listing, *see* writing as part of a drawing
- liveboards, 11, 17, 24, *see also* input devices, screen size
- Liza, 146, 169, *see* groupware toolkits
- local area networks, 11, 13, 50, 56, 63, 72–79, 83, 140, 186–187, 209, 213, 215, *see also* communication networks
- localization, 161, 164–166, 168, *see also* awareness
- location, 90, 99–123, 218, 223–224, 233, *see also* awareness
- locking, 31, 47, 52–53, 57–59, 90, 146, 161, 163, 166, 177, 215, 218, 231, 236, *see also* concurrency control, turntaking
- logical clocks, 170–171, *see also* concurrency control
- logical layering architecture, 158–173, *see also* architectures
- look and feel, 164
- looking glass metaphor, 96, 97, *see also* ClearBoard
- LookingGlass, 102, 215, *see also* drawing systems
- loosely-coupled, *see* coupling
- Lotus Notes, 198, 205, 210
- marketing, 186, 196–197, 202–203, 206–207, 209–210, 215
- MarkUp system, 144
- Mblink, 214, *see also* drawing systems
- MediaSpace system, 99, *see also* media space systems
- media spaces:
 - general, 102, *see also* media space systems

- media spaces (*continued*)
 - integrated with drawing systems, 100, 101, *see also*
 - ClearBoard, LookingGlass,
 - TeamWorkstation, VideoDraw,
 - VideoWhiteboard
- media space systems:
 - CaveCat, 100–101
 - Cruiser, 99
 - Hydra, 121
 - IIIF, 215
 - Media Space, 99
 - Mermaid, 101
 - PMTC, 101
 - Sepia-IPSI, 99–100, 131
 - Video Windows, 99
- mediating conversation, 42, 44, 65, 70–71, 82, 89, 144, 176, 188, 190, 193, 219, *see also* social protocols, process support
- meeting appliances, 183
- meetings, 41, *see* small group design
- meetings: decision support, 64, *see also* group decision support systems
- Mermaid, 99, *see also* media space systems
- MetaDesign system, 169
- methodologies for studying drawing activity, 12, 85, 91–92, 116, 119, 192
- Microsoft Windows, 56, 72–79, 176, 179, 180, 182, 184, 190, 194, 210, 216
- mirror-reversing of video images, 102, 104, 106
- MMConf, 146, 150–151, 169–170, *see also* groupware toolkits
- MMM, 67, 146, 157, 163, *see also* drawing systems
- mode-switching, 4, 26, 44–45, 47, 69, 132, 214
- modems, 180, 182, 186–187, *see also* communication networks
- moderator, 41, 88, 91, 144, 180–181, 188, 190, 193, 224, *see also* social protocols
- Motif, 128, 129–140, 164, *see also* toolkits
- mouse, 26, 44, 46, 103, 160, 180, 184, 216, 233, *see also* input devices
- MS-Dos, 182, 216
- Muge, 60, 67, 71–72, 209–210, 215, *see also* drawing systems
- multi-depth colour displays, 137
- multi-point conferencing, 119, 147, 176, 180, 186
- multicasting, 57, 72–79, 146, 176, 201
- multimedia conferencing, 181, 215, 239, *see also* media spaces, teledata
- multiple cursors, *see* telepointers
- multitasking, 136
- Neptune, 214, *see also* hypertext systems
- network bottlenecks, 50, 56, 60, 63, 69–70, 225
- Network Interface Object, 72, *see also* communication networks
- network keyboard, 181
- NextStep, 164, *see also* toolkits
- NLS, 130
- non-verbal communication channels, 37, 103–123
- Notecards, 214, *see also* hypertext systems
- notes, *see* writing as part of a drawing
- Novell, 72, 74, 186, *see also* communication networks
- object based drawing, 3, 28, 32, 47, 142, 146–147, 160–165, 176, 209, 215–221, 229, *see also* graphics kernel
- Oneko system, 68
- open protocols, 128, 150–152
- OpenLook, 164, *see also* toolkits
- optimistic concurrency control, 47, *see also* concurrency control
- orientation of drawing, 41, 46, 102–103, 122, 212
- overhead projector, 99
- ownership of marks, 110
- pages, 35, 82, 84, 89, 110, 161, 180, 223
- participatory design, 177, 228, 229–230, 233, 235–236, 239
- pen-based computers, 26, 35, 196, 216, *see also* input devices
- persistent sessions, 145–146
- perspective distortion, 104
- Persuasion system, 180
- Pictive, 228–230, 234–236, *see also* participatory design, TelePictive
- playback of meeting, 93, 162, *see also* history of interactions, capture of meeting
- PMTC, 101, *see also* media space systems
- point and quiver, 56, 63, 69–70, 77, *see also* telepointers
- pointing, *see* gesturing
- PowerPoint system, 180
- presentations, 180–181, 194, 222
- privacy, 35, 42, 88, 177, 237, *see also* workspace: private
- private workspace, *see* workspace
- process gains and losses, 86–91
- process support, 82, 144–145, 150–151
- processor bottlenecks, 50, 56, 69–70, 78
- product development, 179–197, 198–208, 209–226
- products:
 - commercialization, 182, 189–196, 198–208, 209–226
 - for group drawing, *see* GroupDesign,
 - GroupGraphics, Smart 2000, Tivoli, Xerox LiveBoard, Wscrawl
- programming language choice, 203
- projection
 - front, 183
 - rear, 108, 176, 182
- ProofMarks system, 144
- proximity, 42, 46
- Quick toolkit, 230, *see also* toolkits
- radar views, 48, 71, 164–166, *see also* awareness
- registration, *see* initiating groupware sessions
- remote procedure calls, 149
- remote training, 181
- Rendezvous toolkit, 147, 169, 228–230, 238–239, *see also* groupware toolkits
- replicated architectures, 3, 24, 26, 29, 40, 50–52, 54, 60, 110, 128, 138–140, 147–149, 168–171, 180, 200–202
- response time, 31, 168, 170, 225
- rIBIS, 67, 215

- Roberts' rules of order, 82
- robustness, 50, 53, 168, 181–182, 200, 210, 224–225
- Rococo, 3, 5–23, 49, 103, *see also* drawing systems
- room size, 191
- run-time architectures, 128, 147–149
- Sasse, 157, 162, 164, 166, *see also* writing systems
- scanners, 49, 189
- screen size, 46, 49, 88, 97, 108, 110, 119, 166, 179, 180, 182, *see also* drawing size
- scribe, 88, 195
- scroll wars, 48, 90
- scrolling, 4, 45, 47–48, 51, 72, 82, 84, 88–90, 132, 160, 166, 168, 218–219, *see also* view slaving
- seamlessness, 4, 47–49, 96–123, 133, 145
- separation of view from abstractions, 146, 158–166, 237–238
- Sepia, 214, *see also* hypertext systems
- Sepia-IPSI, 99–100, 131, *see also* media space systems
- serialization, 50, 52, 53, 170–171, 201, *see also* concurrency control
- session management, *see* initiating groupware sessions
- Share, 144–145, *see also* window-sharing systems
- Shared ARC, 71, *see also* drawing systems
- shared goals, 7
- shared manipulation, 128, 156, 157–159,
- shared task space, 100, 158
- shared understanding, 88, 158
- shared workspace, *see* workspace
- SharedX, 131, 133, 144, 145, 205, *see also* window-sharing systems
- Shdr, 56, 60, 67, 69, *see also* drawing systems
- ShrEdit, 67, 99, *see also* writing systems
- simultaneous user activity on a shared workspace, 11, 31, 42, 44–45, 50, 85, 87, 133, 189, 211, 231
- SLICE model, 128, 158–173
- slicing, 158–173
- small group design:
 - description of design process, 5, 41, 42, 64–65, 228–233
 - intelligent assistance, 230, 234
 - of graphical interfaces, 177–179, 228–241
 - over large distances, 5, 7, 189, 230, 233, 240
 - problems when using groupware, 13, 18, 65, 89–90
 - sample tasks, 20, 85, 116, 192, 234
 - through groupware, 6, 7, 42, 81, 112–113, 228–233
- Smart 2000, 176, 179–197, *see also* drawing systems
- smoothing, 56, 63, 69–70
- snapshots, 43
- social protocols:
 - embedded into system, 3, 44, 64, 89, 91, 144–145, 176, 188, 190, 193, 212, 236
 - for concurrency control, 29, 33–34, 48, 89–90, *see also* concurrency control
 - mediated through drawing orientation, 41
- sockets and TCP/IP, 29, 52, 56, 136, 147, 186, 205, *see also* communication networks
- split screens, 47
- standards, 195
- storing images, 110, 111, 112, 132, 180, *see also* capture of meetings
- stroke-based drawing, 26, 28, 85, 92
- studies:
 - co-located groups, 48, 83, 85–86, 102–103
 - constraining the communications channel, 9
 - distance-separated groups, 5, 85–86, 104, 108, 112, 115–119, 133, 234
 - encoding the meeting, 119, *see* capture of meeting
 - ethnographic, 41, 64
 - meetings over large geographic distances, 5, 189
 - meetings over large time zone differences, 5, 15, 19
 - methodologies, 12, 85, 91–92, 116, 119, 192
 - of design work, 5, 8, 41–42, 49, 64–65, 81–95, 115–119, 234–235, *see also* small group design
 - of face to face meetings, 8, 11, 40–42, 48, 81, 86
 - use of electronic drawing systems, 3, 4, 5, 17, 45–46, 49, 104, 108, 112, 115–119
- stylus and digitizing tablets, 4, 11, 17, 35, 46, 59, 67, 83, 86, 96, 103, 108, 112, *see also* input devices
- Sun, 184, 194
- syntactic/semantic model of interaction, 156
- TCP/IP, 29, 52, 56, 136, 147, 186, 205, *see also* communication networks
- Team Workstation, 59, 66, 96–125, 215, *see also* drawing systems
- TeamFocus, 210, *see also* electronic meeting systems
- TeamPaint, 99, 110–112, *see also* drawing systems
- teleconferencing, 65, 131, 206
- teledata, 38, 45, 181
- telephone lines, 28, 41, 85, 100, 145, 176, 180, 186–187, *see also* communication networks, modems
- TelePictive, 177–179, 228–241, *see also* drawing systems
- telepointers:
 - appearance, 66, 68, 78, 233
 - finding them on the display, 68, 78
 - for large groups, 70–71, 189
 - general, 4, 11, 35, 40, 43–45, 49–50, 55–57, 59–60, 63–80, 87, 110, 130, 138, 149–150, 160, 167, 187, 204, 214, 218–219, 224, 231, 233
 - implementation, 55–57, 63–80, 128, 149–150, 160, 167, 187, 218
 - in relaxed WYSIWIS, 68, 71–72
 - lifetime on the display, 69–70, 78
 - ownership, 68, 78, 233
 - performance on video displays, 77
 - size, 66, 68, 70, 78
 - transmission time, 72
- teleportation, 166, *see also* view slaving
- telepresence, 37, 45, 48, 67, 97, 99, 115, 123, 212
- temporal sampling, 93
- thumbnail sketches, 111, *see also* capture of meetings
- tight coupling, *see* coupling
- Timbuktu, 66, 99, 131, 167, *see also* window-sharing systems
- time stamps, 92, 170–171, *see also* concurrency control
- Tivoli, 3, 24–36, 99, *see also* drawing systems
- toolkits, 40, 130, 133, 136, 142–143, 176, 198–208
 - Garnet, 230
 - groupware, *see* groupware toolkits
 - Grow, 230
 - InterViews, 147, 160, 202, 205, 230

- toolkits (*continued*)
 - Isis, 169
 - Motif, 128, 129–140, 164
 - NextStep, 164
 - OpenLook, 164
 - Quick, 230
 - Unidraw, 153
 - Xlib, 134
 - Xt intrinsics, 136
 - Xtv, 160
- topic closure, 89
- touch screens, 179, 180, 182–184, *see also* input devices
- trade-offs in replicated and centralized architectures, 50, 138–140, 146, 168, *see also* centralized architectures, replicated architectures
- training of users, 187, 205, 229, *see also* ease of use, learning
- transformations, 169–171, *see also* concurrency control
- translucent overlays, *see* video overlays
- transparent windows, *see* graphical overlays
- travel, 191
- turn-taking, 42, 63–65, 67, 70–71, 82, 85, 87, 128, 131, 133, 143–144, 149, 151, 157, 167, 169, 193, 213, 215, 223–224, 236
- undo, 3, 29, 32, 36, 59, 164, 170–171, 217
- Unidraw, 153, *see also* toolkits
- Unix, 26, 50, 72, 133, 136, 147, 205, 210
- update delay, 56
- updating new arrivals to a groupware session, 30, 52
- Vconf, 99, *see also* window-sharing systems
- versioning, 36
- video links, 3, 9, 25, 41, 60, 65–66, 99, 145, 158, 194, 213–214, 236, *see also* drawing systems: ClearBoard, ClearFace, LookingGlass, TeamWorkstation, VideoCom, VideoDraw, VideoWhiteboard
- video phone, 99–100
- Video Windows, 99, *see also* media space systems
- video:
 - camera positioning, 97, 100, 104, 108
 - conferencing, 99–100, 123, *see* media space
 - fusion, 49, 66, 110–115
 - mirror-reversing, 102, 104, 106
 - overlays, 49, 59, 97, 104–110, 121
 - picture in picture, 97–98, 101
 - polarizing filters, 104, 108–109, 112, 121
 - printers, 108
 - projection, *see* projection
 - resolution, 108
 - scalability, 108, 121
 - to capture meeting information, 82, 85, 91–92, 192, 234, 239, *see also* capture of meeting
 - to support gesturing, 49, 96–123
 - tunnels, 9, 49, 100, *see also* video links
- VideoCom, 112, *see also* drawing systems
- VideoWhiteboard, 59, 66, 102, 130, *see also* drawing systems
- view slaving, 48, 72, 90, 166, 213–214, 218–219, 223–224, *see also* scrolling
- view-sharing, 51, 65–66, 99, 131, 145, *see also* window sharing systems
- viewpoints, 83, *see* history of interactions
- viewports, 71–72, *see also* what you see is what I see, view slaving
- virtual reality, 102, 121
- VisionQuest, 210, 215, *see also* electronic meeting systems
- Vmacs system, 144
- voting, 8
- wall-size displays, 119, 223
- We-Met, 4, 81–95, 99, 167, *see also* drawing systems
- what you see is what I see
 - description, 7, 65, 130, 157, 209–210, 213
 - relaxed, 4, 45, 47–48, 51, 68, 71–72, 82, 90, 143, 157, 204, 214, 216, *see also* view-slaving
 - strict, 4, 42, 44, 65, 71, 108, 133, 143, 214
- what you see is what you get, 157
- whiteboards, 41, 87–89, 99, 101, 140, 157, 193, *see also* electronic whiteboards, workspace
- window-sharing systems
 - Dialogo, 99
 - Farallon Timbuktu, 66, 99, 131, 167
 - NLS, 130
 - Share, 144–145
 - SharedX, 131, 133, 144, 145, 205
 - Vconf, 99
 - XVT, 205
- work surface, 143, *see also* drawing, workspace
- workspace, *see also* drawing
 - private, 35, 42, 47, 51, 87–88, 133, 161, 171, 237
 - shared, 3, 38, 41–42, 65–66, 87, 97, 99–100, 112, 133, 212
- writing as part of the drawing, 42, 44–45, 64–65, 86–87
- writing systems, *see also* hypertext systems
 - Aspects, 59, 69, 99, 144, 157
 - Coed, 200
 - Costed, 202
 - Grove, 99, 131, 157, 162–163, 169
 - Jot, 200
 - rIBIS, 67, 215
 - Sasse, 157, 162, 164, 166
 - ShrEdit, 67, 99
- Wscrawl, 51, 59, 60, 67, 99, 128, 129–141, *see also* drawing systems
- X windows:
 - as an implementation platform, 11, 26, 50–51, 128, 129–140, 204, 230–231
 - description, 134
 - protocols, 129–130
- Xerox LiveBoard, 24, *see also* drawing systems
- Xeyes, 68
- XGroupSketch, 37–62, 67, 71, *see also* drawing systems
- Xlib, 134, *see also* toolkits
- Xsketch, 59, 131, *see also* drawing systems
- Xt intrinsics, 136, *see also* toolkits
- Xtv, 160, *see also* toolkits
- XVT, 205, *see also* window sharing systems
- zooming, 72

GROUPWARE FOR REAL-TIME DRAWING

Computers and modern networks are making it possible for teams to work together in real-time through groupware. People can now share not only their voice and video image, but their work as well: documents, notes, presentations, and drawings.

This book describes the design, implementation and commercialization of conferencing systems that allow people to collaborate in real-time over a shared drawing surface. Its chapters, written by some of the world's top researchers in this area, take traditional small group meeting practices and show how they can be supported and enhanced in both electronic meeting rooms for face-to-face interaction, and desktop conferencing systems for geographically separated groups. It presents solutions to conceptual groupware design problems, the technology used to implement the systems, and studies of system use in practice. The book also explores how research prototypes can be translated into viable commercial products and the kinds of new applications that can be built on top of them.

Suitable for professional software designers and programmers, researchers, students and anyone else interested in groupware products, this book provides:

- An unrivalled and comprehensive look at the design, implementation and use of state of the art technology to support human communication and coordination through real-time collaborative drawing.
- Details on all aspects of collaborative working with reference to drawing. Specific areas covered are design issues, implementation specifics, practical applications of the technology studies of the use of groupware and the marketing and deployment of this new technology.

THE AUTHORS

Saul Greenberg is an associate professor in computer science at the University of Calgary, and an active researcher in computer supported cooperative work (CSCW) and human computer interaction (HCI). He is in charge of the GroupLab project, which investigates the design and development of groupware systems and toolkits.

Stephen Hayne recently joined Arizona State University West as an assistant professor in management information systems. Dr Hayne's papers have appeared in the *Journal of Information and Management*, the *Journal of Computer Supported Collaborative Work*, *Journal of Systems and Software*, *Journal of Management Information Systems*, and others. Much of his research is rooted in the desire to use innovative technology to solve real business problems. Currently, he has implemented tools in graphical environments to assist groups in communication and decision making, i.e. shared drawing, group brainstorming, concurrent issue surfacing and consolidation.

Roy Rada is a professor of computer science at the University of Liverpool and has been leading a team designing groupware tools, particularly collaborative authoring tools, for ten years. He has previously published with McGraw-Hill a book entitled *Hypertext: from text to expertext*.

McGRAW-HILL BOOK COMPANY

ISBN 0-07-707899-3



9 780077 078997 >

KP-378-239