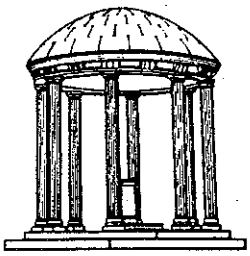
Synchronous and Asynchronous Collaboration: A Program of Research

TR90-024

May, 1990

Peter Calingaert, Hussein Abdel-Wahab
Gregory Bollella, Sheng-Uei Guan
F. Donelson Smith, John B. Smith



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

This paper has been submitted for publication elsewhere. As a courtesy to the intended publisher, please do not make further copies without the permission of the authors.
UNC is an Equal Opportunity/Affirmative Action Institution.

SYNCHRONOUS and ASYNCHRONOUS COLLABORATION: A PROGRAM of RESEARCH

Peter Calingaert, Hussein Abdel-Wahab, Gregory Bollella,
Sheng-Uei Guan, F. Donelson Smith, and John B. Smith

Department of Computer Science
University of North Carolina at Chapel Hill

ABSTRACT

Our work in CSCW encompasses both synchronous (real-time) and asynchronous forms of collaboration. For synchronous collaborations, we have demonstrated visual workspaces, shared remotely, to support both document editing on character-mode displays and execution of graphic primitives on bit-mapped displays. We have developed for operating systems a number of extensions to enhance their support for synchronous collaboration. We have also initiated related research in cognitive modes for both synchronous and asynchronous collaboration. Our results provide a coherent framework in which to build a theory of cooperative intellectual work. We have applied the results of that theory to designing system development tools to support different CSCW applications, as well as methods and tools for analyzing groups working with our systems. In conjunction with the research on shared visual workspaces, the theory also provides a framework for integrating a broad research program in CSCW.

1 INTRODUCTION

For more than 20 years, our department has been implementing systems for intelligence amplification. Our first reported study supported business decision making [1] and our longest continuing project has been the application of interactive computer graphics to assist a biochemist elucidating the structure of a complex protein molecule [2]. A more recent development has been the Writing Environment, a hypertext-based system designed to support authors of scientific, technical, and other expository documents [3]. Crucial to our success has been the selection of driving problems whose solutions have been of significance not merely to us as tool builders, but also to professionals in other disciplines for whom the problems are real indeed.

An integral part of our efforts has been the design of environments for the human user. We have used the systems we built to study their human users engaged in complex intellectual tasks. These studies have drawn on methods that range from collecting informal comments by users to formal analyses of machine-recorded protocols using cognitive grammars. As computers and communication have become more intertwined, and with great strides in the development of distributed systems, it has become natural for us to extend the foregoing efforts in the direction of support for multiple professionals working in collaboration.

An ideal plan for research in computer-supported cooperative work (CSCW) might focus on three components: study of the human element, development of applications, and refinement of the technological infrastructure to support the first two. Rather than address these components serially, we have instead adopted a cyclic developmental framework, tackling all three fronts in parallel. We have also elected, with some exceptions, to build proof-of-concept demonstrations rather than full implementations.

Technology advances in computer and communication systems — our primary infrastructure — now offer us the support we need for distributed, synchronous applications. We have developed shared workspace tools for several forms of collaboration involving a few professionals, any two of whom may be remote from each other. These include original creation of a document and its subsequent editing, exploration of problems modeled by a spreadsheet, debugging of a computer program, and planning of treatment decisions by an oncologist or surgeon looking at the same picture as a radiologist. We will continue to develop these and other synchronous forms of collaboration tools; but we see new opportunities also for more advanced asynchronous tools, particularly in the form of large hypertext databases that can support groups working cooperatively on the same large, interconnected, conceptual artifact.

Technological support for CSCW is provided by communication systems, data representation in multiple media, data bases and hypertexts, distribution facilities (remote procedure calls, naming, authentication), and operating systems. Our basic strategy in developing CSCW tools has been to extend single-user tools to multi-user tools. Thus, while we emphasize the work we have completed in buiding multi-user tools, we also describe work-in-progress in which we are currently extending single-user tools.

2 SHARED VISUAL WORKSPACES

Whereas collaborators may spend a great deal of time working independently, many activities require periods of close interaction — searching, questioning, proposing, reviewing, experimenting, agreeing. We believe that such episodes will be most productive if all participants simultaneously have full access to their shared computer-stored materials. Furthermore, it should be convenient to begin and sustain these collaborative activities without elaborate scheduling or, better yet, without leaving one's office. Shared visual workspaces provide one form of computer support for these activities.

2.1 Human-Computer Interfaces

Shared visual workspaces are abstractions that denote a collection of objects (*e.g.* documents, images, programs) and the tools used to view and change them. Each participant in a collaboration can have the same visual representation of the workspace(s) and the same facilities for operating on the objects. The design issues for shared visual workspaces have been discussed extensively elsewhere [4,5,6] — we review them here to show where our work fits in this design space.

The leading decision is how closely the collaboration environment will be integrated with an existing computer environment. We agree with the prevailing view that great advantage is gained from making familiar programs readily available to participants. In most cases these tools have been written for a single user — they assume a single input source and present a single view (most programs useful in collaborative environments will be interactive). Since it would be impractical to modify even the most-used tools, it is necessary to provide "adapters" that allow any single-user tool to be used unchanged in multi-user environments. Such adaptation can be accomplished by interposing, between the tool and its users, "agent" processes that present a single input stream to the tool and replicate its output to multiple viewers.

Next we choose between a single activation of the workspace and multiple replicas. Whereas the single-workspace model is simpler, and synchronization of the user views much easier to accomplish, it may result in poor interactive performance. Processing times in the agent and network latency for output are likely bottlenecks. The replicated-workspace alternative may improve performance but at a cost of complexity in protocols for reliably delivering duplicate input to all copies. If some input is not duplicated at all copies, it may be very difficult to resynchronize the states of the replicated tool instances (and data objects). An equally serious problem is duplicating the total execution environment for tools at all sites, particularly when extensive customization is common. Most of our work has been based on the single-workspace model.

How is the single input stream to be shared among multiple users? Single-user tools are not likely to provide adequate mechanisms to recover from the effects of the contention that arises if any user can generate input at any time. We use "token-passing" protocols implemented in the agent processes to switch the single input stream among users. A user holding the token must be given an adequate time quantum to complete useful work, but, for fairness to other participants, will be forced to yield after a brief additional grace period. If several users want the token, their requests will be queued and granted in FIFO order.

We must also consider whether each user should see exactly the same view of the tool output (the "what you see is what I see" — WYSIWIS — model). Although exactly duplicated views

are difficult to achieve on heterogeneous displays, we believe this is the best strategy for adapting existing tools. We also assume that most computer-supported collaborations will involve voice communications (generally using telephone system facilities such as conference calling and speaker-phones). Because such facilities are not always available, we also provide a "talk" facility that allows users to coordinate their activities using keyboards and displays for messages.

Some additional design decisions that are reflected in our systems are (a) provision for dynamic addition or removal of users participating in a shared visual workspace, (b) ability of users to participate in multiple shared workspaces concurrently, and (c) shared workspaces that include views from multiple tools.

The same design decisions discussed here must also be considered in explicitly creating new multi-user collaboration tools (perhaps with different conclusions). In particular, a replication strategy may prove fruitful if made an integral part of the design, achieving both performance and reliability. Multiple input streams (with contention) and user-tailored views of shared objects may improve the "look and feel" of many collaborative activities.

2.2 Implementation Examples

2.2.1 Remote Shared Textual Workspaces under Unix

Our objective here is to provide the large community of Unix users linked by the rapidly expanding Internet with a general-purpose facility that effectively converts any single-user software tool into one that can be used for real-time collaboration. We have implemented a prototype [6] and tested it over distances up to 180 miles. Objects are restricted to text files and tools can accept input only from keyboards. Although any tool that accepts input only from keyboards may be used, the one most frequently used is the text editor **vi**. The system response time to user commands depends, of course, on the network traffic and loads of the participants' sites. For users located within the same LAN the response is acceptable and comparable to the case where the tools are used individually. This system, like the others we have built, adheres to the single-tool, single-workspace model.

No special-purpose resources are required to use our system. For example, collaborators may use any ASCII terminal. No modification or restriction to the use of these tools is imposed; the exact commands of the tools as described in their respective manuals can be used during a collaborative session. Communication between agents is based on the 4.3BSD stream sockets in the Internet domain, and agents use the **select** system call to multiplex input from several sources. To let tools read from and write to other processes rather than a terminal, a "pseudo-terminal" device is used between the tool and these processes.

2.2.2 Remote Shared Graphical Workspaces under the X Window System

We have built an X11-based multi-user tool, called "CoDraw", for drawing figures and graphs as well as for coordination and communication among geographically distributed collaborators [7]. It can be used at workstations running the X11 servers and connected to the Internet. In one of the sessions, CoDraw was running on a VAX/780 with three participants using Sun 3/50, DEC3100, and Sun/Sparc workstations.

CoDraw creates for each participant two windows: Workspace and Chat. The Workspace is used to compose a figure from standard geometrical shapes (*e.g.* circles, rectangles, and lines), freehand writing, and arbitrary bitmap images. The Chat window is used mainly for posting messages among the participants. Our system allows a user to develop part of a figure off-line using his or her preferred single-user tool, then insert it into the workspace for others to see and include in the final version of the collaboratively constructed figure. The Chat window is structured into zones, one for each participant. All participants can simultaneously type messages in their respective zones to discuss what is being constructed in the Workspace window.

In our original textual-workspace project, we were able effectively to convert a traditional single-user text-based tool without graphics or mouse input, such as vi, into a multi-user tool for a group of remote users. With the networking capabilities of the X Window System, we are in the process of implementing a similar system to convert any X11-client program into one that can be shared and used by a group of remote collaborators.

2.2.3 Remote Shared Textual Workspaces under OS/2

We are currently building a system that facilitates the use of a single-user text editor as multi-user text editor for the OS/2 operating system. Our design is similar in concept to the Unix-based system presented previously but differs in implementation. We again chose the single-tool single-workspace model because of the vastly different environments that can exist on personal computers attached to a LAN. One fundamental difference between Unix and OS/2 led to our design. Under Unix all I/O passes through the file system and is hence redirectable, but under OS/2 the suggested interface to the screen, the Video Subsystem, is below the file system and therefore not redirectable. There exist interfaces to the screen under OS/2 that *are* redirectable, but our future plans include accommodating various tools and we choose to intercept the screen output at the Video Subsystem, through which all screen output passes.

OS/2 provides a mechanism that allows a programmer to redefine the Video Subsystem calls. The redefinition may include a directive that forces the call to be made normally after the code of the redefinition executes. Our redefinition of the calls simply packs the arguments and data and sends them to the participating machines, where the call is made to the Video Subsystem of the remote machine. The effect is to reproduce the local screen exactly at all of the remote machines. Keyboard input is managed by using OS/2 Character Device Monitors. The mechanism provided by OS/2 for redefining system calls is potentially very powerful, essentially providing an application developer the ability to tailor the operating system to the application. In our case, however, that mechanism imposed some limitations, notably restrictions on available stack space.

2.3 Operating-System Support

In this section we review the development of protocols to support collaboration groups whose composition may vary dynamically and the provision of operating-system support for jointly-owned objects. Much of the work has been reported in greater detail in the dissertation by Guan [8].

2.3.1 Protocols Supporting Dynamic Groups

We have designed a dynamic group structure to allow users to join or leave dynamically, to send or receive messages, and to share their views. Any user is free to create a dynamic group by specifying its name, mode (public, closed, secret), and a list of participants. A user joins a group by issuing join_group (group_name) and leaves it by issuing leave_group (group_name). The call list_groupname () returns a list of all existing groups. A user can obtain further information about a group by issuing list_group (group_name). The call close_group (group_name) marks a dynamic group entry as closed: no more participants will be accepted. Several further calls are provided for active users of a group to share views or messages.

Views can be shared through a WYSIWIS mode in the same group, where a participant allows the standard output of his or her process to be shared. Assuming a cooperative environment, any user process in the same dynamic group may enter the mode. A group thus defines a secure and cooperating environment like a conference: outsiders are not able to enter; insiders are free to enter any WYSIWIS in the group.

The prototype implementation is based on 4.3BSD Unix. A C-language library has been built as the programmers' interface and a system-server daemon installed. The library is compiled with a user program that calls the proposed interface. The library code shares the same address space with the user program in execution.

2.3.2 Operating-System Support for Jointly-Owned Objects

We have encountered some problems due to fundamental shortcomings of Unix (and most operating systems) regarding group collaboration. For example, there is no such concept as "jointly-owned object". Usually an object is created and owned by a single user, and others may access the object by asking the super-user to add them to the owner group. We have developed three concepts to avoid reliance on such a rigid and static approach.

2.3.2.1 Multi-User Processes

We have extended the ownership of a process from one to several users. The multi-user process runs under the union of its owners' privilege domains and provides a multi-user terminal interface to the joint owners. The creator makes an existing process multi-user by issuing allow_join with a list of users who may join. The creator then issues wait_join when ready to accept participants. When a joining process issues join_proc, standard input, output, and error channels are created in the multi-user process for this user. An object or a process created during the execution of the multi-user process will generally be owned by the joint-owners.

2.3.2.2 Shared Capability Lists

These are shared by multiple users, where each participant may post a capability to his private object and allow others to share access. A process can create a shared capability list (C-list), returning a key for further access. To *share* the shared C-list, the process provides this key to other processes. To *use* a shared C-list, a process presents the system with the key. A process can revoke capabilities at any time.

2.3.2.3 Conditionally Jointly-Owned Objects

The use of multi-user tools makes the existence of jointly-owned objects a necessity: a participant who joins a multi-user tool written by others knows that the user agent executed in his or her name is not a Trojan horse if the multi-user tool is jointly owned by all the participants.

We have generalized the concept of jointly-owned to conditionally jointly-owned objects [9]. These can be accessed, or their protection state changed, only by subsets of the owners that satisfy stated conditions. We have focussed in particular on quorum- and authority-based objects. We have shown that conditionally jointly-owned objects can be used to resolve the conflicts that may arise among joint owners.

2.4 Futures

Clearly more experimentation (with real collaborators using a series of prototypes) will be needed to grow our systems toward maturity. We will first attack issues in extending the breadth of use — how can interoperability be achieved for users who would collaborate but are stymied by heterogeneity in hardware (including displays), operating systems, and networks? While far from being a complete solution, the X Window System seems to offer many desirable properties. The most appealing aspects are (a) clean separation of function between client applications and a server for managing user interface resources (display, keyboard, mouse), and (b) ability to interconnect distributed applications and user interfaces over networks. It is also widely available, its components interoperate in many heterogeneous environments, and it can be readily used to support multi-user collaborations. We are now investigating how our current and future tools can best make use of the X Window System.

We are also investigating whether additional media for communicating can enhance the quality of collaboration. Live video is an intriguing example. If, without leaving our offices, we could participate in a collaboration using shared visual workspaces and see (as well as hear) our colleagues, would the character of our interactions be enhanced? Would the addition of sight and sound make the process more appealing (and more frequently and effectively used)? Or would some opposite effect ensue, perhaps a preference for the anonymity of characters or lines traced on a display?

A secondary but important series of issues concerns the form of presentation for voice and video. Should display real estate be used for video or should there be an independent monitor? Who should appear on-screen at any time — a single speaker or some subset of the participants? Should voice and video be integrated with the workstation to allow more flexible control by the collaboration software? As networks become faster and compression technology becomes more powerful, integration of sight and sound directly in the collaboration environment may provide a powerful tool.

3 ASYNCHRONOUS COLLABORATION: MODE-BASED SYSTEMS

Many intellectual tasks can be described as the efforts of an individual, a group, or a collection of groups to create and refine concepts. Our view is that people working together usually share some (often complex) conceptual artifact — a book manuscript, a patient's medical records, a musical score, a manual of procurement procedures, or a computer program. Much of collaboration is concerned with creating the conceptual structure, reaching a shared understanding of it, and agreeing on changes to it. Indeed, Fred Brooks has identified the creation of complex conceptual structures as the essential difficulty inherent in large software engineering projects [10]; the same is true for many other fields. Since computers most often provide the tools of choice for representing, storing, and manipulating concepts, the notion of extending the same tools for augmenting collaborations is appealing.

To address this issue, we have focused our research on the following questions: How do people go about developing structures of ideas? What computer tools can help them? How can we tell for sure that we are helping them?

3.1 Cognitive Modes

To answer the question of how people develop structures of ideas, we are building a theory for understanding and describing intellectual tasks in terms of the cognitive modes and strategies they use. Intuitively, a cognitive mode is a particular way of thinking engaged for the purpose of accomplishing some part of an intellectual task. Examples for the task of writing include brainstorming, planning, drafting, and several forms of editing. More formally, a cognitive mode is an interdependent combination of a goal, a particular form of intellectual product built or modified in the mode, the particular cognitive processes used, and the rules and constraints that determine behavior within that mode. People use modes within strategies that guide them in moving back and forth among modes.

3.2 Mode-Based Computer Tools

To answer the question of how computer tools can help us build conceptual structures, we have used this theory to guide the design and development of several intelligence-amplifying systems. For example, we have built a mode-based Writing Environment (WE) in which four system modes correspond with six cognitive modes used by writers (two of the system modes each serve two cognitive modes). Thus, system architecture matches cognitive architecture.

We are currently extending both our theory of modes and our tools for building mode-based systems from single-user to multi-user tasks. To understand how groups work together, we are developing a theory of modes of activity that focuses on the conceptual artifacts that groups build over the duration of a project. These artifacts may be tangible, in the form of design notes, diagrams, minutes of meetings, drafts of documents, *etc.* But they may also be intangible, in the form of a shared body of knowledge or a common understanding of the goals for a project. Focussing on these products provides a basis from which to observe the actions of a group, the processes operating within the group, and the goals and constraints that guide their behavior. This theory will guide us in building computer tools to support the different modes of activity engaged by groups.

One such tool is a user interface management system, called MoDE, especially designed for building mode-based systems [11]. Using it, the developer can construct an interface for a system by selecting components from a library of objects and adapting them through direct manipulation. By incorporating the communications platform described earlier for shared working environments, MoDE will provide a general tool for building many different application systems, all of which can also support synchronous collaboration in the form of modes shared among groups of users.

We have used the concept of mode in developing three systems. These include the Writing Environment (WE) described earlier, a software development environment, and a set of tools for managing, analyzing, and displaying machine-recorded protocols. The software development environment is inherently collaborative in its conception, since it was intended to support groups designing and maintaining intermediate-sized systems (*ca.* 50,000 lines of code). Consequently, part of this project is developing a general hypertext graph server to support groups building a single large, but coherent product — in this case, system code together with documentation and drawings. Since this software development environment includes support for written documents, we are using that effort to extend our work on writing from single to multiple users.

3.3 Analysis of User Activity

To answer the third question — how do we know that our theory and systems are helpful — we are studying users' cognitive behavior while they work with our systems. To do so, we have developed new protocol analysis tools and methods. Each of our systems — in fact, any system built with MoDE — can produce a detailed transcript, or protocol, that describes a user's actions. We use these transcripts to replay a session — in real time, sped up, or manually stepped through. We have also built a cognitive grammar in the form of an expert system to analyze protocols, and we have developed several display and analysis tools. These tools are being used in various studies and to refine our theory of modes. We are also extending our tools and techniques in order to combine several different protocols. When our cooperative-work application systems are completed, we will use the extended tools to record, analyze, and display protocols for groups of individuals working cooperatively within a networked computer environment and to refine the theory of modes of activity.

4 CONCLUSION

Our work in CSCW encompasses both synchronous (real-time) and asynchronous forms of collaboration. For synchronous collaborations, we have demonstrated visual workspaces, shared remotely, to support both document editing on character-mode displays and execution of graphic primitives on bit-mapped displays. We have developed for operating systems a number of extensions to enhance their support for synchronous collaboration. Among these are protocols for the formation, modification, and management of dynamically changing groups of computer users; multi-user processes; and conditionally jointly-owned objects with an associated model of protection.

Our research in cognitive modes provides a coherent framework in which to build a theory of cooperative intellectual work, system development tools to support different CSCW applications, as well as methods and tools for analyzing groups working with our systems. In conjunction with the research on shared visual workspaces, it also provides a framework for integrating a broad research program in CSCW.

We are continuing by adding more applications of shared workspaces, extending WE to multiple authors, installing shared workspaces on more architectural platforms, and studying the effects of such developments as very high bandwidths and integrated sight and sound. We hope thus to continue to contribute to the development of a National Collaboratory [12].

REFERENCES

[1] Jan Stuart Prokop. An Investigation of the Effects of Computer Graphics on Executive Decision Making in an Inventory Control Environment. Ph.D. dissertation, University of North Carolina at Chapel Hill (1969).

[2] William Vaughn Wright. An Interactive Computer Graphic System for Molecular Studies. Ph.D. dissertation, University of North Carolina at Chapel Hill (1972).

[3] J. B. Smith, S. F. Weiss, G. J. Ferguson, J. D. Bolter, M. Lansman, and D. V. Beard. WE: A Writing Environment for Professionals. *Proceedings, National Computer Conference '87*: 725–736 (1987).

[4] Keith A. Lantz. An Experiment in Integrated Multimedia Conferencing. *Proceedings, Conference on Computer-Supported Cooperative Work*: 267–275. Austin, TX (December 1986).

[5] J. R. Ensor, S. R. Ahuja, D. N. Horn, and S. E. Lucco. The Rapport Multimedia Conferencing System — A Software Overview. *Proceedings, 2nd IEEE Conference on Computer Workstations*: 52–58. Santa Clara, CA (March 1988).

[6] Hussein M. Abdel-Wahab, Sheng-Uei Guan, and Jay Nievergelt. Shared Workspaces for Group Collaboration: An Experiment using Internet and UNIX Interprocess Communications. *IEEE Communications*, **26**(11): 10–16 (November 1988).

[7] Hussein Abdel-Wahab. CoDraw: An X-Window Multiuser Tool for Collaborative Drawing. Submitted to *IEEE Software*.

[8] Sheng-Uei Guan. A Model, Architecture, and Operating-System Support for Shared Workspace Cooperation. Ph.D. dissertation, University of North Carolina at Chapel Hill (1989).

[9] Sheng-Uei Guan, Hussein Abdel-Wahab, and Peter Calingaert. Operating System Support and Protection Model for Jointly-Owned Objects. Submitted to *The Journal of Systems and Software* (1990).

[10] Frederick P. Brooks, Jr. No Silver Bullet — Essence and Accidents of Software Engineering. In *Information Processing 86*: 1069–1076; Elsevier, Amsterdam (1986). Reprinted in *Computer* **20**(4): 10–19 (April 1987).

[11] Yen-Ping Shan. MoDE (Mode Development Environment). To appear in *IEEE Software* 7(3) (May 1990).

[12] Joshua Lederberg and Keith Uncapher (eds.). Towards a National Collaboratory: Report of an Invitational Workshop at the Rockefeller University, March 17–18, 1989. Distributed by the National Science Foundation.