DCWA: A DISTRIBUTED COLLABORATIVE WRITING AID

Technical Report 94-04

K. H. Chang, Y. Gong, A. R. Khan, R. Vuppula
and A. W. Wear

Department of Computer Science and Engineering
Auburn University
Auburn University, Alabama 36849-5347

February 9, 1994

DCWA: A Distributed Collaborative Writing Aid

K. H. Chang, Y. Gong, A. R. Khan, R. Vuppula, A. W. Wear

Department of Computer Science and Engineering
Auburn University
Auburn, AL 36849

Abstract

Existing computerized writing tools are mostly designed for access by one user where participants must work in an interleaved fashion to prevent inconsistency. The need for collaborative writing tools has just been recognized in recent years. In this paper, the focus is on the design of a real-time platform-independent collaborative writing prototype, called DCWA. This prototype can be used to improve productivity in many application areas such as academic writing, programming, and administration. No attempt has been made in this work to address all issues related to collaborative writing since it is believed that those should be answered at some later time. Observations of the use of DCWA, including its strength and weakness, are also discussed.

1 Introduction

In a conventional problem solving environment, computers normally provide passive support to a task team. That is, while computers store, analyze, retrieve, and present information to the users, the decision process that involves cooperative interactions among the team members is left for the users. As the pressure of improving productivity continues to grow, computer technologies would have to provide active support as well. This means new technologies must be developed to provide active coordination in the cooperative process and to overcome the geographic distribution of team members [1]. This demand leads to the recent research and development of *computer supported cooperative work* (CSCW). The collection of hardware and software that supports CSCW and the like is also known as *groupware* [2].

The definition and requirements of CSCW can be expressed by the term itself. "Computer supported" indicates the extensive application of computer related technologies like computers and networks. "Cooperative work" indicates that a system must provide an environment suitable for a group of people to accomplish a common goal [3]. Detailed requirement analysis for effective CSCW has been reported in [3, 4, 5, 6].

The prototype presented in this paper describes a system supporting distributed collaborative writing. It facilitates a group of authors an environment to write articles. As stated in [6], there are very few systems supporting collaborative writing. Therefore the experience is premature to provide solid guidelines for system development. It is suggested that "Don't rush to invent systems for collaborative writing until the process is better understood" [6]. However, some requirements for collaborative writing have been suggested in the literature [5, 7]. These include

1. Provision of communication and exchange of information among authors.
2. Mechanisms for sharing information to allow cooperation.
3. Management of inputs and outputs for users.
4. Mechanism for "floor" or "role" control for the team members.

It is not the intention of this prototype to provide a full range of service for collaborative writing. Instead, it serves as an exploratory platform to examine and discover features and require-

ments. It provides many important CSCW application features, including coordination, provision of interaction and discussion, real-time, structures, and WYSIWIS (what you see is what I see).

2 Related Systems

Many systems supporting CSCW actively incorperate artificial intelligence and planning techniques. These systems first collect information from the participants and suggest the appropriate courses of actions to take. The SYBIL system [8] collects and coordinates options about a specific design topic from members of a hardware platform project team, and suggests them goals and subgoals. The Office Works system [9] arranges meeting for a group of participants by checking each participant's schedule.

CSCW can be accomplished through multimedia communication channels. The MERMAID [10] and the SPIN [11] systems are prototypes that provide real-time conferencing environments for geographically distributed participants by using synchronous textual, audio, and video communications. One of the drawbacks associated with these systems is that the cost is far beyond what most users can afford. A simplified version of conferencing service is to use extended textual bulletin boards [7]. In addition to providing conferencing service, the COGNOTER system [12] also collects and organizes ideas from its participants for discussion. It is one of the piloting systems that provides the WYSIWIS capability.

A fundamental requirement for CSCW is the provision of information/message exchange among the participants. It has been suggested that messages should be organized in a structured way to achieve efficiency [13]. That is, protocols must be designed to discriminate between cooperation information and those of little relevance, e.g., the cognitive filtering in Information Lens [14] and the semi-automatic agents in Object Lens [15]. The second requirement is the provision of mechanisms for information sharing and cooperation [7]. Most CSCW environments stress the need for the distinction of public and private information. However, as described in [16], information to be used in cooperative work needs to be taken out the limit of "personal wall". It was suggested that all users should have common access to all information.

In the field of collaborative writing, the collaboration among coauthors can be classified into two categories: *shared mind* and *division of labor* [6]. In the first category, multiple authors edit

the same file at their own locations and a coordinator integrate the contributions. In the latter category, a task is divided into mutually independent subtasks for coauthors. A merging section then takes place to integrate the subtasks. These classifications are parallel to the asynchronous and the synchronous collaboration. In the asynchronous approach, users do not work at the same time; while in the synchronous approach, users work at the same time on the same file.

An experimental system providing asynchronous collaborative writing for a small group has been reported in [4]. The system uses email as the communication media and the text is represented in the ASCII format. The advantage of this approach is that the impact of the geographical locations and the computer variation among the team members is kept minimum. However, since only email is used, referencing comments or changes to existing files may be difficult. An interesting observation made by the developer was that the success of collaborative writing depends heavily on the mutual confidence and trust among the team members.

The principle of hypertext technology is to manage a structure of text, graphics, and other media in a network of nodes. Quilt [17] is a hypertext system that provides coauthoring services. The users are identified as either coauthors or commenters. It allows all users to comment on a document. However, only certain privileged users, i.e., the coauthors, can modify the document. The ForComment system [9] provides a group editing environment. It allows up to 16 reviewers to comment on a documents. However, only the original author can actually modify the document.

The rIBIS [18] and the SEPIA [19] are synchronous hypertext systems providing various levels of collaboration modes. In the independent mode, users may work on their own tasks without interfering each other. In the loosely-coupled mode, users may share certain public information while working on their own tasks. In the tightly-coupled mode, users will share the same view, and resources, e.g., mouse and file, are strictly controlled to avoid conflicts. The major improvement of SEPIA over rIBIS includes the automatic mode switching and the use of composite nodes. SEPIA also provides audio communication channel between participants.

The DCWA described in this paper addresses many issues identified in the literature. It can be utilized in either asynchronous or synchronous mode. The composite node concept of [19] has been adopted to organize the file structure. A tree-like hierarchical structure is used to define the non-overlapping working spaces among users and guarantees a conflict free collaborating environment. This eliminates the requirement of the traditional "floor control" mechanism. The

communication among users is accomplished through a real-time textual "talk" window, which is different from the audio channels provided in most systems [11, 19]. A textual message is less intrusive and would simply "stay" on the screen. It makes referencing to a discussed subject easier.

3 Design of the DCWA

This section contains a description of important choices that are made in design of the DCWA. The focus herein will be on the rationales behind these choices since we believe the DCWA is not "just another program" in the CSCW field; but rather, it includes many novel design principles that, as explained in this section, deserve special attention in the future endeavor. In the next section, these design choices will be examined together with specific applications that may potentially benefit from the DCWA tool. Some preliminary experiences with the DCWA are also discussed there.

In this section, we start with an overview of the DCWA, introducing the "asynchronous communicating processes" architecture as the overall organizational framework for the prototype. Then the design of individual processes is explained in detail.

3.1 Overview

DCWA is an ASCII text based distributed writing tool implemented using *de facto* standard programming facilities, including BSD4.3 Unix, the TCP/IP protocols, the OSF's Motif toolkits set, and the ANSI C. Portability is, therefore, automatically maintained. Among the many capabilities, the following are the major features of this software.

- *Maintaining a unique version of the document.*
  Problems associated with many existing collaborative writing tools which create separate versions of a document (or parts of the document) are eliminated. Users can always access and edit the most up-to-date version of the file. While they all see the same document, many identical copies of the file can actually be stored in separate networked file servers (NSFs), depending upon the locations of the users. Users in different local area networks can work together through the DCWA across the Internet. Note that some existing CSCW writing tools rely on emails to cope with the problem generated from distributed heterogeneous machines.

- *Structured file representation.*

  Cooperating users are provided capability to specify logical structures for the document. The logical structure currently supported is a hierarchical, tree-like structure with leaf nodes containing the actual contents of the file. The intermediate nodes then function as indicators for the scopes of the users. The information pertinent to the logical structure is stored separately from the text file so that the continuous text file can be used for any other purposes. Some other software tools store the textual and structural information together. Our choice of storing these two types of information separately is with future expansion in mind. For example, in a software development environment, users may rely on the "makefile" to define the logical structure for the document (i.e., a large program). It is well-known that the program structure encoded in a "makefile" may possess non-tree structures. Therefore, separate storage of the two types of information will allow a document structure that is compatible with an early version of the DCWA to be upgraded to a newer version easily. The upgrading can be done, for example, by modifying the logical structure file(s) only, without having to touch the text file(s).

- *WYSIWIS in a distributed environment.*

  In addition to an editing window for a user's own work, the user is also provided with another window to view the work spaces of others. The graphical representation of the logical structure tracks the location of each user in the entire document. Changes generated by one user are transmitted immediately to others to render a real-time effect.

- *Real-time communication and interaction among users.*

  Network access and communication is taken care of by an independent process. By using the specialized process, network access can be served with the best effort. (Perhaps this is the best one can do over the current Internet.) In addition to the regular channel for carrying textual changes, an independent channel is also maintained for users to conduct discussions on their work.

Due to the presence of the logical structure and the possibility of partial views of users over the document, many problems associated with traditional database systems exist, e.g., managing the consistency of the file, editing the logical structure, assigning views to each user, error recovery

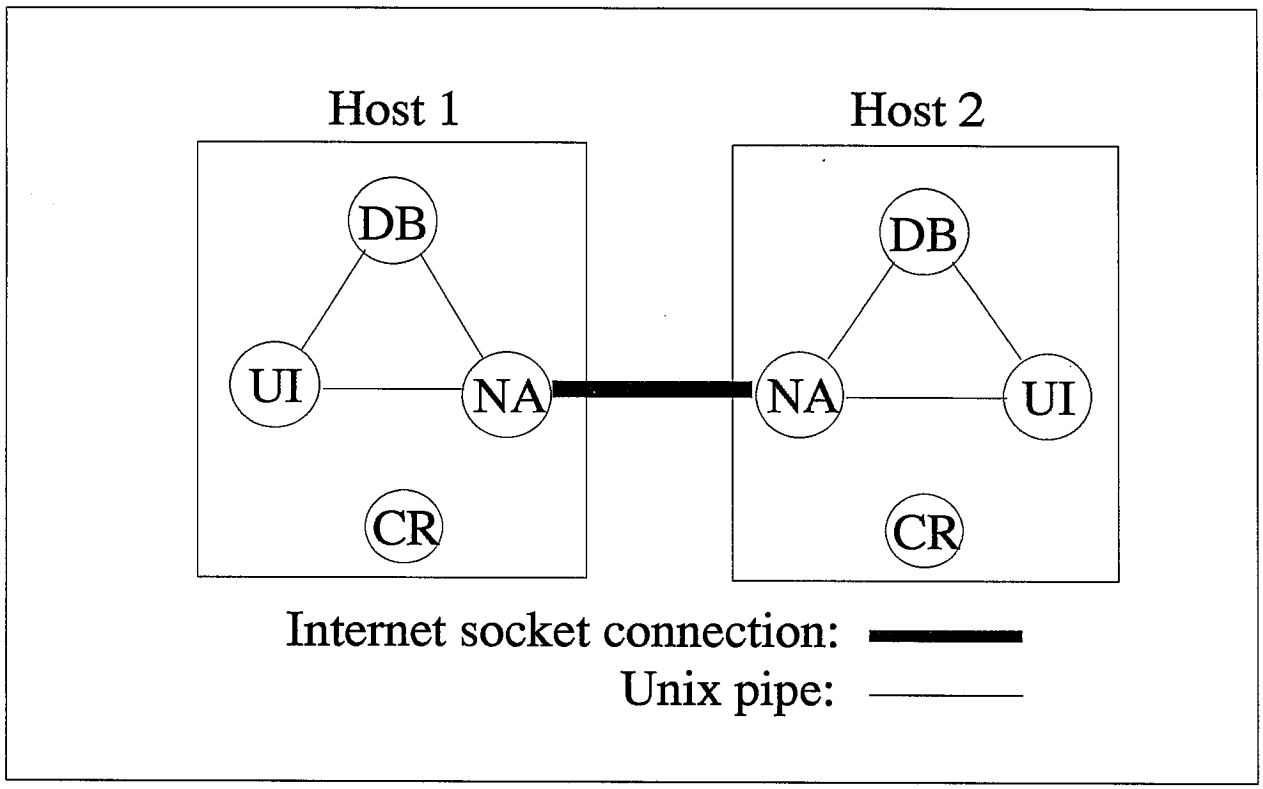

Figure 1: The Communicating Processes Architecture

and rollback, searching in the text or in the logical structure, etc. In fact, almost all existing CSCW writing tools incorporate some database functions. A large number of them even directly utilize commercial database systems. In the design of the DCWA, even though an Oracle 6 system is running at our site, we resisted the temptation to directly utilize the existing database systems mainly because it conflicts with our goal of allowing users in different file systems to work together. In addition, unlike other software systems, database systems have widely diverse interfaces for application programs. Therefore, a decision is made to build a specialized database function within the DCWA.

The overall organization of the software is based on a *communicating processes architecture* (CPA). This architecture is depicted in Figure 1 where the DCWAs on two machines are shown communicating with each other using their network access (NA) processes. On each host, two other processes (the database process (DB) and the user interface process (UI)) are also created. The three independent processes on each machine communicate with each other using unidirectional

pipes. The purposes of these three processes are immediately clear from their names. Their details will be described in the next three subsections. In addition to these three functional processes, a controller process (CR) also exists on each machine. Most of the time, the CR process is self-suspended. It is primarily used for tearing down the connection (when a user issues a "quit" request) or reporting errors to the remote host.

There are many reasons for separating the functions of the DCWA into three independent processes.

1. In many occasions, the tasks associated with the three processes should be parallelized. For example, when a host is recording what has happened in the remote host, the user interface should still be able to interact with its user. This is achieved by the concurrent DB and UI processes. Similarly, when the UI process is interacting with the DB process to record local modifications to the document, messages from remote host should still be received. This concurrency is achieved by a separate NA process.

2. One can easily discover that a CSCW writing environment involves many concurrent events, such as the ones explained above. We believe that many more concurrent events will be encountered as the tool becomes more sophisticated in the future. As such, an arrangement in the software for possible future expansion must be planned. Our strategy is to build entire software with the CPA so that new processes can be added with minimum modification to the existing code.

3. As long as all processes maintain the same protocol, the details of all processes become independent. This allows the partition and integration of efforts among our team members much more easily.

3.2 Communications

As the result of the CPA design, communication among all processes becomes important. Since the progresses of the processes are asynchronous, they must exchange messages to ensure that the entire system functions logically. The design of all processes follows an event-driven principle as shown in Figure 2.

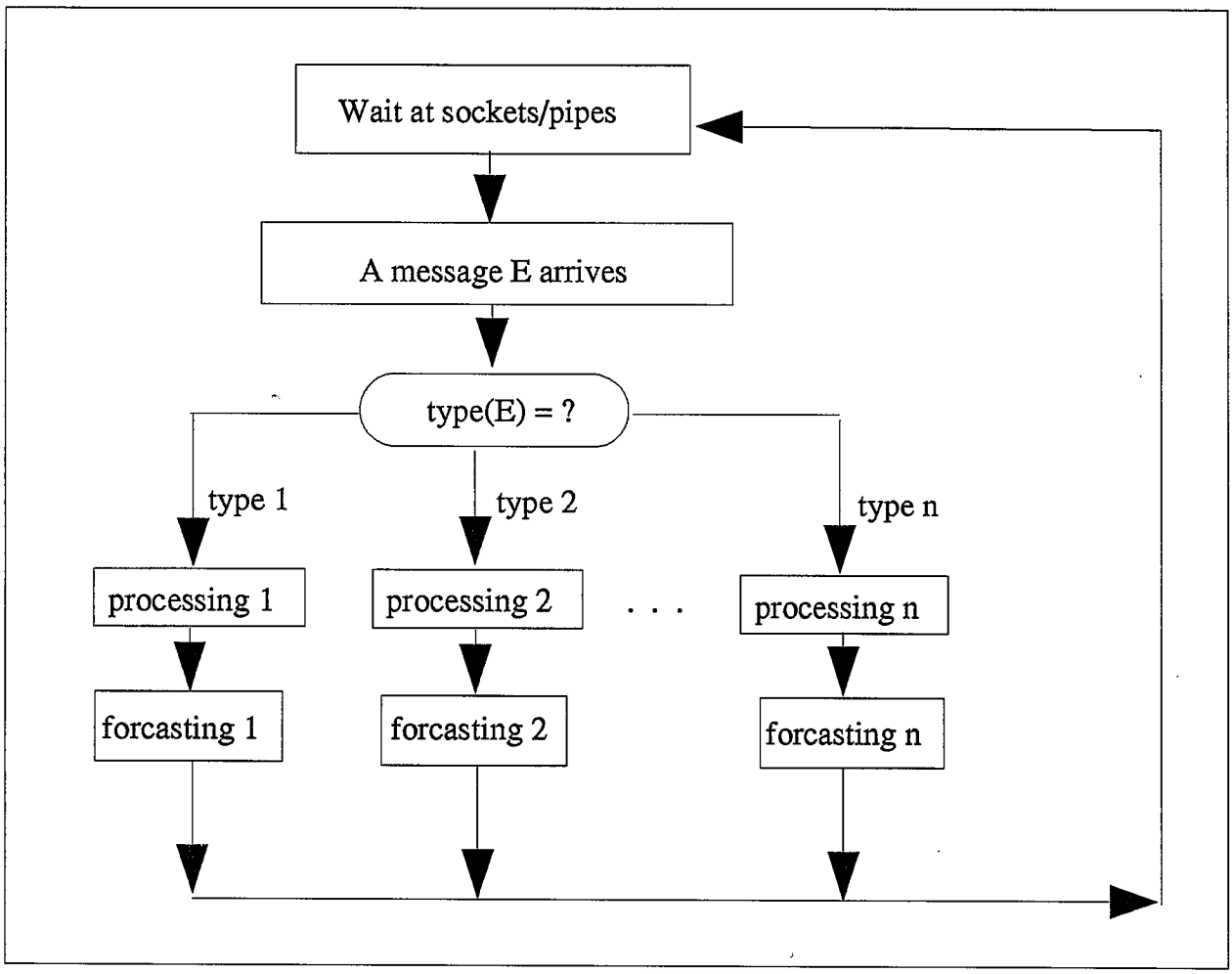

Figure 2: The Event-Driven Design of Processes

Each process suspends itself if it does not receive any messages from other processes. Upon reception of a message, it wakes up to provide the appropriate services. Since all processes follow this algorithm, what differentiate a given process from others are the message types and the required functions corresponding to each message type. This event-driven strategy make the DCWA an open-ended system in the sense that additional features may be provided by adding new message types and functions into the relevant processes. The overall architecture, as well as the details of unrelated portion of the software can be kept intact. As an example, a list of messages sent from the NA process and the corresponding functions is shown in Appendix A.

In addition to the interprocess communications after the software has started to run, there

is also an equally important problem as to how the software can be initiated. Since users are geographically separated apart, a mechanism must be designed to allow them initiate the DCWA in a totally asynchronous manner, i.e., they do not have to rely on another medium (e.g., telephone, email, or face-to-face communications, etc.) to negotiate a schedule to start the tool altogether. Any user must be able to initiate the software any time without the knowledge of the work schedules of his/her partners. If, however, a user find himself/herself to be the only user of the software, then the software should gracefully degenerates to a single user editing tool.

To achieve asynchronous start of the software, our solution is to implement a daemon process to be placed in the background of all machines which may potentially use the DCWA. All daemon processes recognize each other with a well-known Internet socket address. In addition, all daemon processes maintain the information about the ongoing connections in the system. A startup program is provided to a user who is just "walking in". This startup program wakes up the local daemon process using a message. A protocol is then executed between the daemon process and the startup program to examine the status of the connection which the user would like to join. In case his/her colleagues have not entered the DCWA, the daemon process informs the startup program; subsequently, the startup program forks the four processes shown in Figure 1, and the user works alone. At the same time, the information about the existence of this user is transmitted to the destination machine(s) where the colleagues of this user may be located. Therefore, at a later time, when a collaborator enters the system, the connection can immediately be set up by the daemon process, and the real multiple user collaboration begins.

3.3 The Database Process

As of this writing, functions designed for the database process have not all been implemented since one of the main concerns of this first version of the prototype is to experiment with the framework for the software (i.e., the CPA design concept) and to build the ground-work for future expansion. Furthermore, the experience from implementation showed that a comprehensive database could quickly make the prototype exceedingly complex.[1] Given this fact, it was decided that the implementation of the complete set of functions for the database process should be separated into several

[1]The DB processes form a specialized distributed database system. No commercial product can satisfy all of our needs.

stages, i.e., versions of the prototype. In the first version of the prototype, we managed to eliminate many of the database related problems by selecting some reasonable simplifying assumptions.

In this subsection, instead of only discussing about those simplifying assumptions, the focus is placed on important functions that we feel should eventually be implemented in the database process. In addition, we also discuss how far the current version of the prototype is from these goals.

In general, the database process should provide backup and recovery service, functions to allow a user to define node attribute and attribute inheritance, search facility, definition of views for different users, sharing and conflict resolution, and finally, run-time modification of the logical structure. Some details of these functions are discussed below.

- *Backup and recovery.*

  The database process currently provides full service for recording local and remote users' work areas and logical structures of the document. However, only the most up-to-date version of the text is stored. Rollback, as is a common capability in all commercial database systems, is not supported. As such, users are not provided with the capability to undo a previous destructive action.

- *Node attributes.*

  We currently only allow the user to assign a label to each node in the logical structure. This is also the design choice in may other similar software tools. However, the information that can be carried by a label is quite limited. In the later stages of the development, users will be able to assign arbitrary attributes to each node. For example, in an academic writing application, a node may correspond to a segment of text which mentions about a certain chemical compound. The user may desire to record this fact in the corresponding logical node as a quick reminder for the future. More importantly, the attributes can be used by the search function to gain immediate insight to the *semantical* structure of the document, as opposed to the organizational structure reflected by the current logical structure.

- *Search facility*

  The search facility allows a user to specify (arbitrarily complex) queries about the document. The SQL would be a good candidate for this query interface. So far, few people have realized

the definite need of the search function in CSCW writing since in the case of single user writing, the writer tends to know, and almost always remember, every detail of the document. In the CSCW writing, especially in case of "division of labor" CSCW writing, each writer only concentrates on his/her own part. Repeated mentioning of the same point becomes highly possible. An automatic search facility would help collaborators gain knowledge about what others have written without having to read the actual document.

- *Definition of views*

  The first version of the DCWA makes the entire logical structure available to all users. In reality, however, not all users are interested in the entire document until the final stage of the writing process. Therefore, users must be able to specify his/her own logical view.

- *Sharing and conflict resolution*

  Sharing in the first prototype is handled manually by users through locking and unlocking nodes in the logical structure. A node is locked automatically when it is selected by a user, and unlocked if the user selects another node. Conceivably, users must discuss among themselves about who should monopoly a given node. In the next version, complete sharing and automatic conflict resolution will be implemented which allows users to work without knowing the whereabout of others.

- *Run-time modification of the logical structure*

  Quite possibly, the entire document is assigned a primitive logical structure at the very beginning. This logical structure needs be refined during the writing process. Therefore, users must be provided with capability to define new nodes or delete unreasonable node during the writing. Again, an automatic conflict resolution function must be implemented in case two or more users create conflicting logical structures in the run time.

3.4 The User Interface

The main window of the graphical user interface (GUI) of the DCWA is shown in Appendix B. Six subwindows are provided in this GUI. They are defined as follows. (In the following, the local user is referred to as User 1 and the remote user is the User 2.)

1. *User 1's Working Area:* is the subwindow located in the upper left corner of the GUI. This area contains the text that can be edited by User 1.

2. *User 1's Viewing Area:* is the subwindow located in the lower left corner. In this window, User 1 can see, in real-time, changes made by the other user.

3. *Global Text Area:* is the subwindow located in the upper right corner, i.e., the one with a darker background in the figure in Appendix B. This window is for User 1 to view (but not to edit) the *entire* document without restriction.

4. *Global Logical Structure Area:* is the largest subwindow in the middle of the GUI. This subwindow is designed to hold the entire logical structure of the document. As the size of the structure increases, this area may become crowded and selections using the pointing device may become less accurate. Therefore, the next subwindow is designed to cope with this problem.

5. *User 1's Zoom Area:* is the subwindow below the global structure area. In this subwindow, User 1 may view and select from a part of the global structure.

6. *User 2's Zoom Area:* the last window in the GUI, i.e., the one in the lower right corner. This subwindow is a reflection of the User 2's Zoom Area. As such, User 1 is prohibited from selecting nodes in this region.

The functionality of the user interface can be described in four aspects, file structure and access, basic editing capabilities, view for other user's working space, and interactive talk service.

A file is graphically represented by a tree structure with a look-and-feel similar to the file cabinet in the MS-DOS shell. An example is shown in the Global Logical Structure Area in Appendix B. The scope of a node is defined using a simple rectangular box. A user may assign a label to each node. If desired, a leaf node can be divided into subnodes; and adjacent sibling leaf nodes (corresponding to blocks of actual texts) can be merged. The scopes of neighboring nodes are not allowed to overlap. This constraint is required to guarantee the nonconflicting working spaces for all users. A user may select a leaf node to work on that is not currently edited by other user. After a node is selected, the node on the tree structure indicates this fact by changing its color on all participating hosts. The file structure is normally defined in the early stage of writing. If a file

is not divided, only one user can edit it at any given time because of the strict scoping rule in this current version. In order to avoid confusion, modification to the logical file structure is allowed when only one user is working on the file.

When a user selects a leaf node, the contents of the node is retrieved and displayed in the user's text window, i.e., the User 1's Working Area defined earlier. Basic editing capabilities provided in the Motif toolkit are utilized in the user's text subwindow. At the end of an editing session, the system interacts with the user to either save or discard the changes. If the changes are to be saved, other users working on the file are queried to reach a consensus for saving. In this version of the prototype, the User 1's Viewing Area is updated every 15 seconds, administered by a Unix alarm clock. In general, the DCWA faithfully displays what the other user actually sees, i.e., the WYSIWIS property.

Furthermore, in the current version, we allow only pair-wise connections, i.e., a user may have multiple connections to other users, but no connection involves more than two users. Because only two users are admitted, there is no confusion as to who is "the other user". In future version of the DCWA, multiple user connections will be supported and the work is currently ongoing.

Finally, users can also communicate with each other using a "talk" window which is popped up when a user presses the "Talk" button in the command bar (see Appendix B). The talk window provides the similar function with the Unix "talk" command, and is, therefore, not shown here. For example, the "talk" window consists of two portions. The top half displays the messages typed by the remote user and the lower portion provides the message editing area for the local user. The "talk" communication is implemented on a channel independent of the textual channel to achieve fast response.

The reason for incorporating a conferencing channel within the DCWA is also with the future development in mind. It is planned that in the next phase of the experimental development, synchronization of multiple time-correlated channels will be investigated. To successfully overcome this problem is one of the key issues to the multimedia CSCW software.

4 Observations

One of the concerns of the DCWA is its speed of response when a host is heavily populated or the file becomes large. This is a concern because the overall framework of the software (the CPA) relies on message exchanges among processes. The actual use of the prototype verified that this is not a serious issue. The prototype demonstrated consistently fast response to both local users and remote processes.

Furthermore, many simplifications made in the database process do not seem to create any problem either. The software shows that the database functions listed in Section 3.3 can be completely optional. However, we believe this should be attributed to the fact that the current version of the prototype only allows two users per connection. Users tend to follow each other's work closely. We expect that in an environment where more than two users share one connection, the existing implementation would not be satisfactory. The software, in particular, the database process, must be modified according to our discussion in Section 3.3.

The talk channel in the current version helps the users greatly in negotiating their scopes of work. Once a user occupies a region in the document, the user can continue to monoply it for as long as he/she desires. Other users are prevented from modifying the region although they can view its contents. We feel that in the case of multiple users per connection, negotiation among more than two users is likely to generate many problems, including slowing down everyone's work. In general, arrangement of work areas should be completely automated. Rules for conflict resolution must be specified and agreed upon by all users before the software starts to run.

As shown in Appendix B (a snapshot of the DCWA window containing a case of programming application), nodes in the logical structure are labeled by words corresponding to the function names in the C program. One annoying experience is that both collaborators must know the details of the code written by others in order to use the code appropriately. Different from reading a paragraph of an article, the tediousness of the code often gives rise to the need for a reader to consult with the writer directly for clarification. In this case, the reader often wishes he/she were not reading the code itself, but instead, a natural language elaboration of the code! This is an example which demonstrates the need for some *summarizing information* about the actual text. This information is precisely the *attribute* that we discussed in Section 3.3.

5 Conclusion and Future Work

In this paper, the design of a prototype CSCW writing tool, called DCWA, is introduced. A software architecture, referred to as "communicating processes architecture" (CPA), is used to organize all parts of the software. Under the CPA, some important modules are allowed to operate independently and concurrently so that the overall responsiveness of the software is improved. However, we point out that this concurrent processes approach should not be over-used. It is very tempting to carry the idea of the CPA throughout the design, resulting in a large number of communicating processes time-shared on a single CPU. In such a case, the responsiveness of the software would most likely be hindered by the overhead spent in communication among processes. In our first version of the prototype, only three processes are allowed to run concurrently (the user interface, database, and network access processes); but these three functions have covered most functional capabilities expected of a CSCW writing aid. Experimentation showed that the software performs satisfactorily without noticeable delay.

The work reported in this paper is still ongoing. The experience gained from the design and implementation of the first version of the prototype will no doubt contribute to the work toward the second version in the near future. In our second version, emphasis will be placed on the implementation of a sophisticated distributed database system (c.f. Section 3.3). In a distributed CSCW writing environment, users' awareness of the entire document is blurred by the "division of labor". A database containing information about the semantical structure of the document will greatly reduce the users' burden in case they need to quickly perceive what others have written. We believe that design of distributed database for CSCW is one of the key issues in this area.

References

[1] J. Cuena and A. Garcia-Serrano. "Intelligent Computer Support". *Cooperation among Organizations — The Potential of Computer Supported Cooperative Work*, pages 72–102, Power, R.J.D. (Ed.), Germany: Springer-Verlag, 1993.

[2] J. N. Orr. "Graphics and Groupware: Increasing Intimacy through Broadening Bandwidth". In *Groupware'92*, pages 73–76, Morgan Kaufmann Publishers, 1992.

[3] R. J. D. (Ed.) Power. *Cooperation among Organizations — The Potential of Computer Supported Cooperative Work*. Germany: Springer-Verlag, 1993.

[4] D. Diaper. "Small-Scale Collaborative Writing Using Electronic Mail". *CSCW in Practice: An Introduction and Case Studies*, pages 69–92, Diaper and C. Sanger (Eds.), Germany: Springer-Verlag, 1993.

[5] B. Hewitt and G. N. Gilbert. "Groupware Interface". *CSCW in Practice: An Introduction and Case Studies*, pages 31–38, Diaper and C. Sanger (Eds.), Germany: Springer-Verlag, 1993.

[6] M. Sharples. "Adding a Little Structure to Collaborative Writing". *CSCW in Practice: An Introduction and Case Studies*, pages 51–68, Diaper and C. Sanger (Eds.), Germany: Springer-Verlag, 1993.

[7] T. Rodden. "Technological Support for Cooperation". *CSCW in Practice: An Introduction and Case Studies*, pages 1–22, 1993.

[8] J. Lee. "SIBYL: A Tool for Managing Design Rationale". In *Proc. of the Conference on the Computer Support Cooperative Work*, Los Angeles, CA, 1990.

[9] S. Opper. "A Groupware Tool Box". *BYTE*, pages 275–282, Dec. 1988.

[10] K. Watabe, S. Sakada, K. Maeno, H. Fukuoka, and T. Ohomri. "Distributed Multi-party Desktop Conferencing: MERMAID". In *Proc. of the Conference on the Computer Support Cooperative Work*, Los Angeles, CA, 1990.

[11] L. G. Palmer and R. S. Palmer. "DECSPIN: A Networked Desktop Videoconferencing Application". *Digital Technical Journal*, Vol. 5(No. 2):pp.65–76., Spring 1993.

[12] M. Stefik, G. Foster, D. Babrow, K. Kahn, S. Lanning, and L. Suchman. "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving In Meeting". *Communications of the ACM*, Vol.30(No.1):pp.32–47, Jan. 1987.

[13] P. V. Rangan. "Managing Multimedia Collaboration". In *Groupware'92*, pages 418–420, Morgan Kaufmann Publishers, 1992.

[14] T. W. Malone, K. R. Grant, and F. A. Turbak. "The Information Lens: an Intelligent System for Information Sharing in Organization". In *Proc. of the ACM Human Factors in Computing Systems Conference*, pages 1–8, Boston, MA, 1986.

[15] T. W. Malone and K. Lai. "Object Lens: A Spreadsheet for Cooperative Work". In *Proc. of the Conference on Computer Supported Cooperative Work*, Portland, OR, 1988.

[16] J. Brooke. "User Interface for CSCW Systems". *CSCW in Practice: An Introduction and Case Studies*, pages 72–102, Diaper and C. Sanger (Eds.), Germany: Springer-Verlag, 1993.

[17] R. S. Fish, R. E. Kraut, M.D. Leland, and M. Cohen. "Quilt: A Collaborative Tool for Cooperative Writing". In *Proc. of the Conference on Office Information Systems*, pages 30–37, Palo Alto, CA, 1988.

[18] G. L. Rein and C. A. Ellis. "rIBIS: A Real-Time Group Hypertext System". *Int. Journal of Man-Machine Studies*, pages 339–367, Aug. 1993.

[19] J. M. Haake and B. Wilson. "Supporting Collaborative Writing of Hyperdocuments in SEPIA". In *Proc. of the 1992 CSCW Conference*, pages 138–146, Nov. 1992.

A Representative Messages Sent from the NA Process

In the following table, subscript 1 is used to label the processes belonging to local host; and 2 is used to label the processes on the remote host. The sender process of the messages listed below is always the NA_1 process. The column with title "Code" lists the constant name defined in the program. The actual code that is passed over the channel is an integer.

Recipient	Code	Purpose
NA_2	DISP_UPDATE	Update remote user area
NA_2	FILENAME	File name from local host
NA_2	USER_AREA	Local host defined scope
UI_1	DISP_UPDATE	Update local user area
DB_1	FILENAME	File name from remote host
DB_1	USER_AREA	Remote host defined scope
NA_2	SAVE_QUIT	Local host quit

B A Typical DCWA User Interface

The following window dump shows the typical user interface of the DCWA tool. It also shows an example use of the software in a software development application. The logical structure is derived mainly from partitions of major functions of the program the users are working on.

```
/*_________________ M A I N _________________
*/


ed2main(int argc, char *argv[])
{

int pid ;
char buf[BUFSIZ] ;

   toplevel = XtVaAppInitialize(&app, "Ed3test",
        NULL, 0, &argc, argv,NULL, NULL);

   main_w = XtVaCreateWidget("Main_w",
        xmMainWindowWidgetClass, toplevel, NULL);

   goand_create_menubar();
   goand_create_mainrc();
   goand_create_messg_win() ;
   goand_create_rowcols();
   goand_create_textwidgets();
   goand_create_drawwidgets();
   goand_create__draw_win_popups();
   goand_create_selecttext_widget();
/* create_test_button();
*/

      do_icon_stuff();
      do_cursor_stuff();
      getcolors();
      manage_all();
      register_piperead_funcs();


     XtRealizeWidget(toplevel);


    XtAppMainLoop(app);


}
```