DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors

Michael J. Knister and Atul Prakash

Software Systems Research Laboratory
Department of EECS
U. of Michigan, Ann Arbor, MI 48109

ABSTRACT

The purpose of our project is to provide toolkits for building applications that support collaboration between people in distributed environments. In this paper, we describe one such toolkit, called DistEdit, that can be used to build interactive group editors for distributed environments. This toolkit has the ability to support different editors simultaneously and provides a high degree of fault-tolerance against machine crashes. To evaluate the toolkit, we modified two editors to make use of the toolkit. The resulting editors allow users to take turns at making changes while other users observe the changes as they occur. We give an evaluation of the toolkit based on the development and use of these editors.

Keywords: groupware, collaboration technology, group editors, distributed systems.

INTRODUCTION

Computers are now commonplace in work environments and are influencing the way people interact. Examples of computer-supported interaction mechanisms include electronic mail, newsgroups, and distributed file systems. These have opened up new ways to interact, but all support only non-interactive styles of communication. There are also "talk" programs that are more interactive but are generally restricted to two users and allow only very simple forms of interaction, such as the exchange of messages in different windows. In recent years, there has been a growing interest in providing collaboration tools to support more closely coupled interactions among a group of people [Abde88, Halo89, Elwa89, Stef87]. Our focus in this paper is on one type of collaboration tools, group editors, that allow several people to jointly edit a shared document in a distributed environment.

One difficulty in building collaboration systems is that they require solutions to problems in distributed concurrency control, fault-tolerance, user-interfaces, psychology, human factors, and software design [Elli88b]. Our goal is to remove most of the concerns of distributed concurrency control and fault tolerance by providing toolkits which can be used to build collaboration tools.

This paper describes a toolkit, called DistEdit, which provides a set of primitives that can be used to add collaboration support to editors without having to deal with distributed systems issues, such as communication protocols and fault-tolerance. The primitives provided by the toolkit are generic enough to support editors with different user interfaces. We have tested our approach by modifying two editors, MicroEmacs and GNU Emacs, to make use of DistEdit. The resulting group editors allow users to take turns at making changes while other users observe the changes concurrently. This paper describes the requirements considered, related work, the operation of the test system, the design of the toolkit, and the modifications needed to adapt an editor to group editing using the toolkit. We give the results from our use of the toolkit and discuss tradeoffs that went into some of the design decisions. Finally, we discuss some issues for future work.

REQUIREMENTS OF THE DISTEDIT TOOLKIT

The requirements of the DistEdit toolkit are quite simple:

Multiple-user collaboration: editors built using the DistEdit toolkit should allow users to collaboratively edit text files without being in physical proximity.

Reasonable performance: communications protocols used within DistEdit should give a consistent view of files to all users with reasonably low delay so that group editing is not an inconvenience.

Compatibility with multiple, existing editors: it should be possible to use different editors in a single group session. People usually have their own favorite editors and it would be desirable not to force them to use a different editor when participating in a group session.

Fault-tolerance: the group session should continue to run smoothly despite machine crashes and people leaving or joining the group.

Hiding of communication protocols: adapting an editor to group editing using DistEdit should not require knowledge of distributed systems issues, such as communication protocols.

RELATED WORK

GROVE [Elli88] and ShrEdit [Cogn90] are examples of editors that are especially designed to support group editing. DistEdit, unlike these systems, is not an editor but a toolkit that can be used to adapt single-user editors to the task of group editing. Using the DistEdit toolkit, it is possible to use different editors, which could possibly be running on different types of hardware, in a single group session. For instance, a single group session could potentially allow one user to use *Emacs*, another to use *vi* (both running on terminals), and yet another to use *xedit* (requiring a workstation running X), with only minor modifications required to the code of each editor for using them in a group editing session.

Using DistEdit with different editors as front-ends is similar in some ways to the configurable user interfaces described by Lantz in [Lant86]. Lantz proposes creating customizable user interfaces by making the user interface a separate, easily replaceable module, which interfaces with a single back-end. Similarly, different editors act as a replaceable front-ends for DistEdit. However, the modular front-ends in [Lant86] are meant to be fairly small and easy to develop. DistEdit, in contrast, is actually much smaller than the front-ends it supports.

The design of DistEdit toolkit provides a very high degree of fault-tolerance. A simple way to design group editors, and one which has been used in many editors and collaboration tools, is to use the *client-server* model. There is one server that is responsible for maintaining the state of the editor buffer. Mutual consistency between users' views can then be ensured by requiring all the editing commands to go through the shared server. This design, although simple, is vulnerable to a server crash. Furthermore, even if only one person is editing a file, the updates still have to go through the server, making the editing slow. In contrast DistEdit maintains a copy of the state of the editor buffer for each user. Reliable atomic broadcast protocols [Birm87] are used to ensure mutual consistency between the buffers, even in presence of failures.

A different approach to a groupware toolkit can be seen in LIZA [Gibb88]. LIZA provides a high-level collection of tools for rapid groupware prototyping. These high-level tools provide support for sending messages, indicating moods, editing collaboratively, giving slide shows, and monitoring the group. With this framework and basic set of tools, LIZA is primarily intended to test new ideas in collaboration quickly

and easily. DistEdit is a much lower-level, application-specific toolkit. It is designed to explore in-depth the possibilities of group editing in a setting already familiar to users.

A closely related class of collaboration systems are those that support more asynchronous or non-real time styles of interaction. Examples are editors such as CES [Grie76] and Quilt [Fish88]. These editors allow users to work on the same document but typically on different sections and at different times. As a result, interactions are over a much longer duration, even up to several days. Many of the issues of fault tolerance and real time propagation of updates are not important in such systems. The DistEdit toolkit concentrates on providing more closely coupled real time interaction.

EXAMPLE OF AN EDITOR: DISTEDIT-BASED MICROEMACS

This section describes, from the user's point of view, the operation of one test system, MicroEmacs [Lawr89] modified to use DistEdit. Operation of GNU Emacs is similar, except more powerful editing operations are available. In general, each participant in a group editing session could use a different editor, provided each editor has been been modified to use the DistEdit toolkit.

Operation of the MicroEmacs editor changes very little from its normal use without the DistEdit system. A user starts the editor by giving the program name, followed by the name of a file to edit. As the editor runs for the first user, everything works just as it normally would; the only noticeable difference is an indicator on a status line showing that this user is the 'master' (as explained below) (see Figure 1).

When an additional user runs the program for the same file name, the operation differs considerably. The text loaded into the edit buffer will be that of the first user, complete with all the changes the first user has made. The first user is the 'master', with complete editing capabilities; the second user is an 'observer', with no editing capabilities. As an observer, the second user will see every change and cursor movement made by the master; the observer's cursor is in 'lock-step' with the master's cursor at this point.

The observer cannot perform any operations which change the text; if attempted, such operations simply have no effect. He can, however, save files, change the window size, and issue any editor commands which do not alter the text.

Should an observer desire to move his cursor independently, he may invoke the 'lock-switch' command (bound to a key sequence) which frees his cursor from dependence upon the master's cursor. In this 'free-cursor' mode, the observer may move the cursor anywhere he wishes, but he still cannot make changes; all changes the master makes are still simultaneously updated (and visible if the observer's cursor is in proximity to the change). The lock-switch and free-cursor modes may be toggled at will by an observer.

At all times, at most one user is the master; all others are observers. The master may, at any time, relinquish control by invoking the 'control-switch' command. The status line of all editors is updated to indicate that there is no master. Once there is no master, any observer may take control and become master by using the control-switch command; the first user to issue a control-switch becomes the master. The status lines are then updated to indicate master/observer status, and the new master operates as described above. When there is no master, no user can perform any changes. Presumably, a transfer of control is initiated by a conversation on an outside line, such as a telephone, or perhaps an on-line talk session.

Any number of users may participate in an edit session. Users may join and leave a session at any time without affecting the other users. A failure on one user's machine simply results in that user leaving the session. Should the master leave the session or experience a failure, the other users will observe a no-master status and be allowed to take control. The text being edited can only be lost if all users leave the session and none of them has saved it locally.

All users in a session will always have identical text buffers. Each user may have a slightly different view, depending on individual window size. When in lock-step, the cursor of an observer moves within a window to always be in the same logical position in the text (same line and column), though perhaps not at the same physical screen location.

A key point is that for each supported editor, the usual editing commands will continue to work as they normally do. The only changes from the user's view are the addition of lock-step and control-switch commands, and the new master/observer and lock-step/free-cursor status indicators.

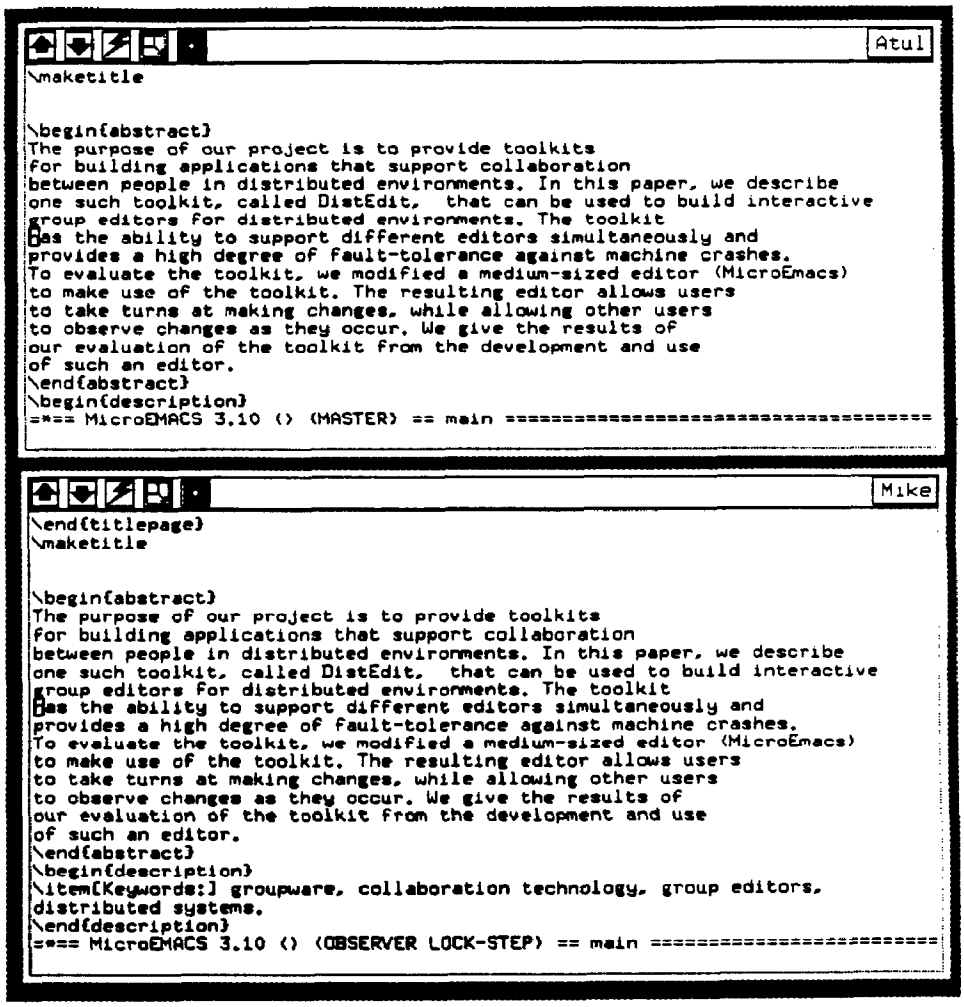

Figure 1. A master window and an observer window in a group session of the DistEdit-based MicroEmacs group editor. Typically, these windows would be on different workstations. Note the status line at the bottom of each window indicating who has control of the session.

DESIGN OF THE DISTEDIT TOOLKIT

The DistEdit system is designed as a modular toolkit which can be easily applied to various visual editors. The only requirement is that the editor have a small number of subroutines which directly modify the text buffer. If this is not the case, the editor will require extensive modifications; this is probably an indication of a poor editor design. The editor need not localize the routines which read from the text buffer.

DistEdit Primitives

A DistEdit primitive is an operation which can be invoked from the code in an editor. A primitive is initiated by a single function call from the editor, but can proceed through several layers of code and across a network, before reaching the location where it is carried out. In this sense, a DistEdit primitive is somewhat like a remote procedure call, except that it could result in many remote executions.

There are three types of primitives provided by DistEdit: (1) update primitives; (2) a cursor position notification primitive; and (3) control primitives. DistEdit provides three update primitives to perform all text update operations, one cursor position notification primitive for broadcasting changes in the cursor position, and two control primitives for transferring status and control information.

Update primitives and the cursor position notification primitive may be issued only by the master editor. Control primitives may be used by an observer or a master editor.

Update Primitives

The update primitives are *insert string*, *delete string*, and *replace string*. These primitives are capable of performing any desired operation on the text. A call to one of these primitives from the master effectively updates the text buffer of every editor in the group. All update primitives operate from the current cursor location; in fact, when a primitive is called, it looks up the current cursor location of the caller to determine where the change will occur. The insert primitive takes a string to be inserted as its operand. The delete primitive takes a count of the number of characters to be deleted. The replace primitive takes a string which will replace existing text.

Line breaks are assumed to be single characters in the DistEdit model. All text update primitives can handle line breaks. Any newline characters within an *insert string* primitive create line breaks, and any characters replaced or deleted may be line breaks.

Cursor Position Notification Primitive

A *cursor position notification* primitive is provided to allow the master editor to inform the other editors of its current cursor position, so that they may synchronize their cursors if they are in lock-step. The cursor position notifications are ignored by an editor that is in the free-cursor mode. This is quite different from text update notifications, which must always be carried out in all editors to ensure consistency.

Control Primitives

Two control primitives are provided: *control switching* and *lock-step toggling*.

The control switching primitive allows a transfer of master status to another user; when a master performs a control switch, he gives up control and becomes an observer; when an observer performs a control switch, he becomes master if there is no current master. There is also a provision for an observer to forcefully become a master, even if the previous master does not relinquish control. This is to allow for the situation where the user of the master editor is not at his desk, and other users in the session want to continue the editing. These two control primitives are broadcast to all other users.

Lock-step toggling is purely a local primitive, which simply informs DistEdit of a desired change in lock-step status (synchronizing the observer's cursor with the master's cursor). The control primitives were chosen simply to support the desired functionality of transferring control and synchronizing cursors.

At present, simultaneous updates from different editors in a session is not supported by the primitives. See the Future Work section for a discussion of some of the issues in providing such an extension.

Underlying Communication Software

ISIS [Birm89], a toolkit for programming distributed applications, was chosen as our communications package because of its elegant broadcast facilities, its error handling, and its lightweight process system.

The broadcast facilities of ISIS remove any need for DistEdit to deal with low level communications; no messages are lost and all broadcasts are guaranteed to arrive in the same order to all the participants in the group.

Users may enter a group session any time. Whenever a user enters a session, the editor state is transferred from one of the current users to the editor for the new user. ISIS provides mechanisms for notification of new users and fault-tolerant communication protocols for state transfer.

Machine crashes may occur at any time, and users can leave the group at any time, as long as there is at least one remaining user. We have designed the toolkit under the assumption that each editor in the session will be maintaining its own state. The ISIS system ensures that either all the active participants receive a broadcast, or none of them do, when the sending site fails in the middle of the broadcast. Therefore, machine crashes or users leaving the session still leave other users in a mutually consistent state.

The lightweight process system in ISIS allows broadcasts to be received while waiting for keyboard input; also, signals, such as a group member failing, can be handled by triggering a process.

DISTEDIT/EDITOR INTERFACE

This section describes the interface between an editor and the DistEdit toolkit. The assumed structure of an unmodified editor is shown in Figure 2. A user interface and control section waits for input; when input is received, it is translated into a set of calls which move the cursor or update the text. These text update routines modify the data structures which contain the text. The results are displayed by a screen manager, which reads from the text structures and displays appropriate output.

The basic functioning of a distributed group editor is as follows (see Figure 3): an editor's text update routines are mapped to calls on the DistEdit update primitives; if the calls on the update primitives are from the master, the DistEdit toolkit takes the received DistEdit primitive calls and, using ISIS, distributes them to all the editors; each editor then maps the received DistEdit primitives back to the original text update routines.

Notice from Figure 3 that all updates to a text buffer have to first go through the network, even for the master site. In our present version of DistEdit, because there is only one master, the updates could have been carried out directly on the master's buffer without waiting to receive the broadcast. We decided not to make updates locally before sending a broadcast for two reasons. First, treating the master and observers identically from the point of view of buffer updates allows us to keep the DistEdit code simple. Second, going through the network first will allow us to more easily extend our toolkit to support simultaneous updates from multiple sites. The atomic broadcast communication protocols used in ISIS can ensure that all the sites see the updates in the same sequence. The performance of the master editor has been satisfactory so far. Should the performance have proved to be unsatisfactory, we would have considered modifications to our design to immediately update the master's local copy, without going through ISIS.

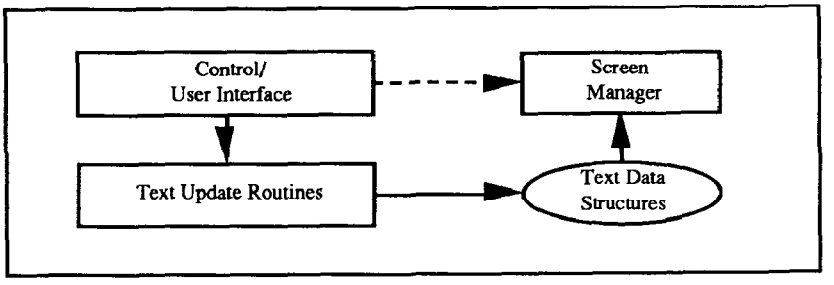

Figure 2. Typical Structure of a single-user editor.

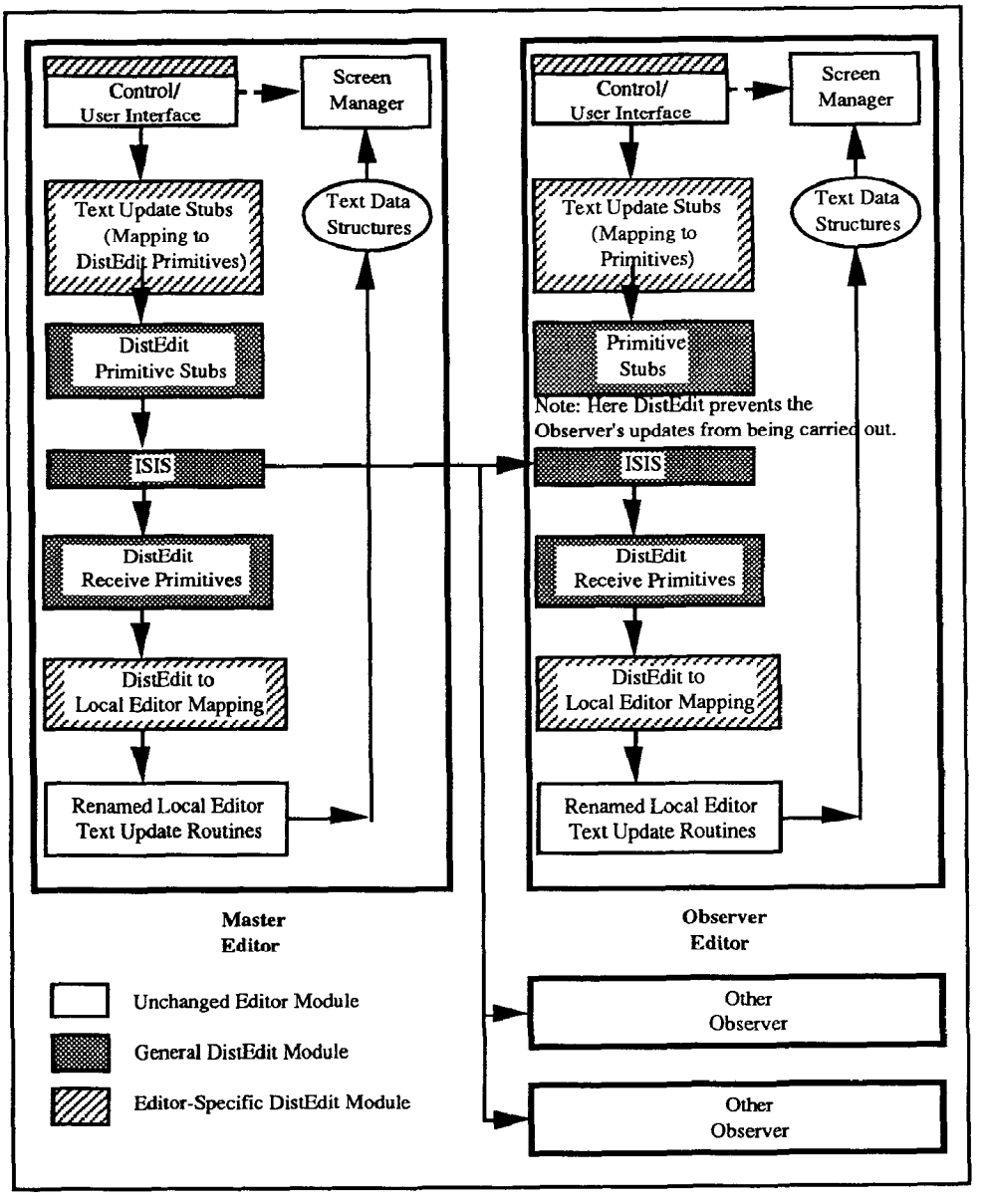

Figure 3. Structure of a group editor using the DistEdit toolkit.

MODIFICATIONS REQUIRED IN AN EDITOR

Based on the interface described in the previous section, the following code modifications are needed to use an ordinary editor in a DistEdit-based group editing session.

- All text update operations on the editor buffer must be mapped to calls to the three DistEdit update primitives.

- The three DistEdit update primitives must be mapped back to the editor's text update routines. Ideally for these mappings, all of the editor's text update operations (all routines which directly update the text buffer) should be contained within a small set of routines, perhaps five to fifteen functions.

- Calls to the DistEdit control primitives must be inserted in the user-interface section of the editor to allow the user to execute the lock-step and control-switch commands and to show the user the master/observer and lock-step/free-cursor status.

- Access routines must be provided to let DistEdit move and query the cursor and read the lines in the text buffer. A routine to read the lines in the text buffer is needed so that DistEdit can transfer the state of the text buffer to a joining user.

- A routine that can be called from within DistEdit must be provided to display the changes to the text buffer on the screen.

All aspects of the system that deal with communication protocols, multiple processes, and fault-tolerance are hidden within the DistEdit toolkit. All the above changes are such that they can be carried out without knowing anything about distributed programming.

In our actual implementation, the editor's actual text update routines are renamed, typically by prefixing a string such as 'real_'; they become the bottom layer in the diagram and remain the only routines which actually modify the text. No modifications should be required for these routines except for the renaming.

Replacing the renamed update routines are a set of stub update routines. These stubs map the update routines which the editor uses to calls to the three DistEdit primitives: *insert string*, *delete string*, and *replace string*.

The DistEdit primitives called by the update routine stubs are actually stubs, too, which simply forward their arguments through ISIS, along with the cursor location at which the operation is to occur.

The DistEdit primitives stub layer thus sends the primitives through ISIS to all the members of the editing group, including the sending member. At the receiving end of the ISIS broadcasts are the DistEdit receive primitives. These routines do some cursor adjustment and call the local DistEdit primitives.

The local DistEdit primitives layer maps the DistEdit primitives back to the editor's commands; thus, this layer makes calls to the original (renamed) update routines of the editor.

The layering, as shown in Figure 3, was chosen to isolate the general-purpose DistEdit layers from the editor-dependent layers. This allows an editor to be adapted to use DistEdit by supplying only editor-dependent code; the DistEdit code need not change. Updates to the DistEdit code are also simplified, as the general DistEdit code is not intermixed with editor code and can be replaced separately.

RESULTS

We successfully converted the MicroEmacs and GNU Emacs editors to distributed group editors using the DistEdit toolkit. Both editors can be used in the same group session.

Very little change was required to original editor code, and the performance was quite acceptable.

Change Required to MicroEmacs and GNU Emacs

Amount of Change

Modifying MicroEmacs to use DistEdit required adding nine lines of C code and changing six lines within the body of the original program. An additional file containing mapping and support functions specifically for MicroEmacs required approximately sixty lines of code.

Modifying GNU Emacs to use DistEdit require adding ten lines and changing twelve lines within the body of the original program. An additional support file specifically for GNU Emacs required approximately three-hundred lines of code.

We consider the changes quite minimal. The bulk of the added code is isolated in a separate files, outside of the normal editor code.

Difficulty of Change

Modifying the editors involved locating and modifying three areas in the code: 1) the buffer-modification routines, 2) the top-level user interface, and 3) the input blocking point.

Locating the buffer-modification routines, those routines which are called to directly update the text being edited, was quite easy. These were clearly marked and contained in a separate files. Mapping them to DistEdit primitives was easy because of a strong similarity to DistEdit primitives. For instance, each editor had routines to insert and delete strings and characters, which mapped directly to the DistEdit *Insert String* and *Delete String* primitives.

Locating the top-level user interface to add DistEdit-specific commands was also quite easy. Both editors were designed to be easily extensible.

Locating the input blocking point was the most challenging task. This is the point at which the editor blocks waiting for input from the user. The input routines had to be modified to accept DistEdit primitives broadcast through ISIS while waiting for keyboard or mouse input. Locating and modifying this portion of the code was difficult because of numerous levels of input abstraction and machine-dependencies.

Performance

We ran the editors using Dec3100 workstations, each with 8 MB of main memory, connected via an Ethernet. The performance was reasonable even with several users. The master editor generally suffered no significant slow-down. The observing editors sometimes experienced jumpy cursor motion, presumably because broadcasts did not always take the same time to travel. There was initially some concern that the ISIS communication package would be too slow to keep up with a typist. Should this have been the case, some caching would have been added to prevent updates from being sent for every keystroke.

DISCUSSION

Multiple-Editor Interaction

The two editors work well together in a group session, but operations and views of the text differ between editors. Some examples of this can be seen in window displays and text marks. Different editors can also complement each others' capabilities.

DistEdit does not give identical views of the text to all users; it simply guarantees that the text is always consistent. Display aspects are determined by the particular editor. One way the views can differ considerably is with the treatment of long lines. GNU Emacs (by default) wraps long lines to the next line of the screen; MicroEmacs uses horizontal scrolling. Window sizes, which are chosen by the users, can also differ considerably, perhaps giving a different view of the same text to each user.

One very useful aspect of using different editors is that the functions of multiple editors can be interspersed to complement one another. For instance, the operations performed by a MicroEmacs (which has no *undo* function) user can be undone by a GNU Emacs (which does have an *undo* function) user after switching control. Similarly, a GNU Emacs user, using the GNU Emacs C-mode, can take control and re-format C code created by a MicroEmacs user. In this way, the strengths of different editors and different users can be combined to provide more power than is available with any one particular editor.

Communication Layering

There are several levels at which communication between different editors in the session could have been carried out. We decided to provide some basic update and control primitives for communication. The low-level operations of each editor are translated into calls to these primitives.

As an alternative, we could have intercepted the user's key inputs directly and broadcasted them to all the users. This alternative scheme has two problems. First, it becomes difficult to implement if the editors are allowed to be different in the same group session — all the editors would have to modified to understand key inputs from every other editor. Second, even with the simplification of requiring identical editors in the session, the extensions to allow concurrent updates become non-trivial. Key inputs from different users cannot be simply intermixed in one stream — that could lead to unexpected behavior. Solutions that simply merge inputs from multiple users into one input stream (as in some X-based tools) would not work in a group editor because semantics of operations are ignored.

Choice of Primitives

In choosing communication primitives, we had several alternatives. We chose to provide some very basic primitives, independent of any particular editor. Instead, a more comprehensive and powerful set of primitives could have been chosen. For instance, one alternative would have been to choose our communication primitives as the available functions in one particular editor, say GNU Emacs. Such more powerful primitives might include operations such as global replacement or block indentation, rather than just insertion, deletion, and replacement of strings.

Three factors influenced our decision. First, if one chooses a more powerful set (as in GNU Emacs) of basic primitives, then code has to be added to translate this powerful set of primitives into the calls on the update routines of every new editor added to the system. With the current set-up, the primitives are simple, and it is relatively easy to map the primitives of any given editor to these simple primitives and from these simple primitives back to any editor's primitives.

Second, there may be a performance penalty in choosing a very powerful set of basic primitives if most of them are not going to be used very often. It will probably increase the bandwidth needs of the system by requiring longer message formats. We expect most of the editor operations to be simple insert and deletes and therefore chose those as our primitives.

Third, dealing with concurrent updates may become complicated if editor primitives are too many and complex. As noted in [Elli89], an algorithm is needed to resolve conflicts between every pair of operations provided by the editor. In our case, using simpler editor primitives is likely to make the task of resolving concurrent updates easier.

One issue that arises because of simpler primitives is that what a user perceives to be one command may actually translate to multiple calls on basic primitives. An obvious example of that is commands to carry out global search and replacement of strings. If concurrent updates are allowed, it is not clear whether such a command should be treated as an atomic transaction or whether it is acceptable to allow other conflicting editing operations to be executed between the calls to the basic primitives. A similar issue arises with *undo* commands. In the single-user editor, an *undo* operation unambiguously undoes the last operation. In a group editor, especially with concurrent updates, it is possible that another user has carried out a conflicting operation that makes it difficult to define the effect of *undo*.

A closely related issue which also results from the use of simple primitives is that semantics of some high-level operations can change or be lost. For instance, a search and replacement operation might normally be undone with a single command. However, when the replacement is split into many separate primitive calls for DistEdit, the undo mechanism could treat each replacement as a separate operation, requiring many undo commands instead of just one to undo the changes made during the search and replacement. We plan to explore these issues in the next version of the editor toolkit.

FUTURE WORK

A number of extensions to the DistEdit toolkit are being explored, including allowing concurrent updates, support for more complex documents, and naming and security issues.

The most obvious extension of DistEdit is to allow multiple users to concurrently update a file. Such an extension requires solving several problems. One problem is maintaining consistency between all the users' buffers. If one user inserts a word at the beginning of the document at the same time as another user, the system must at least ensure that both users view the same resulting text. Resolving conflicting updates is another major challenge. For instance, one user may insert a character in the middle of a line of text which another user is attempting to delete; such conflicting operations must be reconciled.

One approach to solving concurrency problems would be to support locking of regions. A user can only make updates if he hold locks on the region of the update. For instance, if one user were modifying a line, that line will could be locked, preventing another user from deleting it. Selecting policy for locking will require experimentation with the user interface.

Another approach to solving the concurrency problems would be not to use locks but to carry out the updates in any globally consistent order. Since the users can see the results of changes immediately, they will be in a position to correct the conflict by talking to each other and to agreeing to a more social protocol.

Allowing concurrent updates also could have an impact on performance if the updates from different sites have to be executed in the same sequence on all the sites. ISIS provides atomic broadcast protocols that guarantee such sequencing. However, if the performance of ISIS protocols is not satisfactory for the application of concurrent group editing, the design of efficient application-specific protocols may be required. For a discussion of some of the issues in the design of special protocols for group editing, see [Elli89].

Another extension being explored is the application of DistEdit to documents more complex than simple text, such as word processing documents. Such documents include formatting such as fonts, margins, and spacing, which require a more complex model of a document than that currently used by DistEdit.

A final area of exploration is how to deal with naming and security of documents. Keeping group documents consistent across heterogeneous systems requires reconciling various naming mechanisms. Keeping documents consistent after a group-editing session

is also desirable. Finally, since the document is being shared across a network, security must be considered to prevent unauthorized users from reading or interfering with the group's work.

ACKNOWLEDGEMENTS

We would like to thank David Halonen and Lola Killey for valuable input on our work. The facilities for this work are in part supported by the NSF grant CCR-8909674.

REFERENCES

[Abde88] H. M. Abdel-Wahab, S. Guan, and J. Nievergelt, "Shared Workspaces for Group Collaboration: An Experiment using Internet and Unix Inter-process Communication," **IEEE Communications Magazine**, Nov. 1988, pp. 10-16.

[Birm87] K.P. Birman and T.A. Joseph, "Reliable Communication in the Presence of Failures," **ACM Transactions on Computer Systems**, February 1987, 47-76.

[Birm89] K. Birman, R. Cooper, T. Joseph, K. Kane, and F. Schmuck, **The ISIS System Manual**, June 19, 1989.

[Cogn90] Cognitive Science and Machine Intelligence Lab, U. of Michigan, Ann Arbor, "ShrEdit 1.0: A Shared Editor for the Apple Macintosh, User's Guide and Technical Description".

[Elli88] C. Ellis, S.J. Gibbs, and G. Rein, "Design and Use of a Group Editor," **Report STP-263-88**, MCC Software Technology Program, Sept. 1988.

[Elli88b] C.A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: The Research and Development Issues," **Report STP-414-88**, MCC Software Technology Program, Dec. 1988.

[Elli89] C.A. Ellis and S.J. Gibbs, "Concurrency Control in Groupware Systems," **Report STP-417-88**, MCC Software Technology Program, 1988. Also in **Proc. of ACM SIGMOD**, 1989.

[Elwa89] M. Elwart-Keys, D. Halonen, M. Horton, R. Kass, and P. Scott, "User Interface Requirements for Face to Face Groupware," **Report CMI-89-020**, Center for Machine Intelligence, Ann Arbor, MI, Dec. 89.

[Fish88] R. Fish, R. Kraut, and M. Leland, and M. Cohen, "Quilt: A Collaborative Tool for Cooperative Writing," **Proceedings of ACM SIGOIS Conference**, 1988, 30-37.

[Gibb88] S. J. Gibbs, "LIZA: An Extensible Groupware Toolkit", **Report STP-042-88**, MCC Software Technology Program, 1988.

[Grie76] I. Grief, R. Seliger, and W. Weihl, "Atomic Data Abstractions in a Distributed Collaborative Editing System," **Proc. of the 13th Annual Symposium on Principles of Programming Languages**, 1976, 160-172.

[Halo89] D. Halonen, M. Horton, R. Kass, and P. Scott, "Shared Hardware: A Novel Technology for Computer Support of Face to Face Meetings," **Report CMI-89-015**, Center for Machine Intelligence, Ann Arbor, MI, Nov. 89.

[Lant86] K. A. Lantz, "An Architecture for Configurable User Interfaces," **Foundation for Human-Computer Communication**, K. Hopper and I.A. Newman (eds), North-Holland, 1986, pp. 257-275.

[Lawr89] D. M. Lawrence and B. Straight, **MicroEmacs Full Screen Text Editor Reference Manual**, version 3.10, March 1989.

[Stef87] M. Stefik, G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman, "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings," **Communications of the ACM**, Vol. 30, No. 1, Jan. 1987, 32-47.