Issues in the Design of a Toolkit for Supporting Multiple Group Editors *

Michael Knister and Atul Prakash
Software Systems Research Laboratory
Department of EECS
U. of Michigan, Ann Arbor, MI 48109

April 2, 1993

Abstract

A great interest has developed in recent years in building tools that allow people to collaborate on work without the need for physical proximity. One such class of tools, group editors, allows collaborators to view and edit a shared document simultaneously from their workstations. Building group editors, however, requires solving non-trivial problems such as providing adequate response time for edit operations and yet ensuring consistency with concurrent updates, and providing adequate per-user undo facilities. We have implemented a toolkit, called DistEdit, for building new interactive group editors and for converting existing single-user editors into group editors with minimal changes to their code. The toolkit allows different users to use their favorite editors (e.g., Xedit, Gnu Emacs) to edit a shared file and observe each others' changes as they occur. The toolkit provides fine-grain concurrency control, fault-tolerance, synchronization of views, and support for per-user undo. We describe the detailed design and implementation of the DistEdit toolkit and report our experiences in converting several editors, including Gnu Emacs and Xedit, to group editors using the toolkit.

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

1 Introduction

Computers are now commonplace in work environments and have had great influence on the way in which 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 mostly non-interactive styles of communication. There also exist *talk* programs that

*To appear in *Computing Systems*, The Journal of the USENIX Association.

are more interactive but usually are restricted to two users and only allow exchange of messages in different windows. In recent years, there has been a growing interest in developing the technology further to provide support for more closely-coupled interactions [1, 7, 11, 19]. Our focus in this paper is on one type of collaboration tool, group editors, that allows several people to jointly edit a shared document in a distributed environment.

One difficulty in building collaboration systems is that they require solutions to non-trivial problems in distributed concurrency control, fault-tolerance, user-interfaces, psychology, human factors, and software design [6]. The goal of our project is to remove most of the concerns of distributed concurrency control and fault tolerance by providing a library of primitives that can be used to build collaboration tools.

This paper describes the design issues we faced during the development of the DistEdit toolkit. DistEdit provides a set of primitives that can be used to add collaboration support to existing text editors with minimal changes to their code as well as ease development of new group editors. The toolkit takes care of many non-trivial issues, such as concurrency control, consistency of views, history lists for the purpose of per-user undo [17], and fault-tolerance. The primitives provided by the toolkit are generic enough to support different text editors in the same group environment. We have tested our approach by modifying two medium-size editors, MicroEmacs and Xedit (20,000 and 16,000 lines of code), and one large-size editor, Gnu Emacs (75,000 lines of code), to make use of DistEdit. The resulting group editors allow users to make changes concurrently to the same document and to observe changes of others as the editing is in progress.

GROVE [4] and ShrEdit [13] are examples of editors that are designed specifically to support group editing. DistEdit, unlike these systems, is not an editor but a toolkit that can be used to build new group editors and adapt existing single-user editors to the task of group editing. Using the DistEdit toolkit, it is possible to use different editors in a single group session. For instance, while jointly editing a single document, DistEdit can allow one user to use Gnu Emacs, another to use MicroEmacs (on a terminal), and yet another to use Xedit (requiring a workstation running X), with only minor modifications required to the code of each editor. Furthermore, as far as we are aware, proper undo facilities are lacking in the other group editors; they only allow users to undo the globally last editing actions, but not just their own actions. DistEdit specifically addresses the problem of per-user undo in group editors, making the facility available to all editors built using the toolkit.

MACE [15], another group editor, is structured to make it easy to integrate different editors into a collaborative environment by replacing only a few modules. At present, however, only one editor interface, based on the Athena text widget, is supported. We believe that the following design decisions in MACE may make it difficult to integrate other editors: (a) to integrate a new editor in MACE requires one to implement a module that provides conversion between keystroke commands and a canonical form understood by all editors [15] – a task that we believe may prove difficult for sophisticated editors such as Emacs with a large number of keystroke commands; and (b) MACE is based on a different model of user/editor interaction than is found in single-user

editors. It requires that a user explicitly lock the region to be updated and allows the undo of an operation only if the lock was not released since the operation was done.

The design of the DistEdit toolkit provides a very high degree of fault-tolerance. A simple way to design group editors, and one which is used in several group editors and collaboration tools, is to use the *client-server* model with a centralized server. The server 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-based editors maintain a copy of the state of the editor buffer for each user. The communication protocols and algorithms used by the toolkit ensure mutual consistency between the buffers, even in the presence of failures.

Different approaches to groupware toolkits can be seen in LIZA [9], GroupKit [18], Rendezvous [16], and Suite [3]. LIZA provides a high-level collection of tools to support sending messages, indicating moods of participants, giving slide shows, and monitoring the group. Both Rendezvous and GroupKit provide generic facilities for doing conference management, sharing of windows, implementing various floor control policies, and basic access control. The Suite system provides facilities for implementing both loosely-coupled and closely-coupled synchronization of views in groupware applications. Some of the facilities in DistEdit, related to session management, use ideas similar to those provided in these other systems; however, DistEdit is much more focused on one particular class of group applications, group text editors, and specifically addresses issues related to adequate response time, fine-grain concurrency control, fault-tolerance, per-user undo, and multiple-editor support.

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 [10], Quilt [8], and Prep [14]. 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.

An earlier version of the DistEdit toolkit is described in [12]. Several major features have since been added. Unlike the earlier version, the present version allows several users to edit the same file simultaneously in a single session, provides support for locking of regions, allows users to undo the globally last as well as their own last actions, handles transaction-like operations that require several updates to complete (such as globally replacing a string), and provides a window to monitor and control the group session. This paper goes into the details of design tradeoffs we faced in implementing these additional facilities as well as describes the experience we have had since then in porting existing editors to use the toolkit.

This paper is organized as follows. Section 2 describes the requirements considered in the

design of the toolkit. Section 3 gives an example of a group session with two editors built using the toolkit. Section 4 describes the software architecture of DistEdit-based editors. Section 5 discusses the basic modifications needed to adapt an existing editor to group editing using the toolkit. Section 6 describes the library interface provided by the toolkit to an editor and the issues in the design of the library interface. Section 7 discusses the implementation and efficiency issues in the design of the communication layer that implements the library calls. Section 8 describes our experiences in adapting several existing editors to group use and using them. Finally, Section 9 presents concluding remarks and some issues for future work.

2 Goals of the DistEdit Toolkit

The goals of the DistEdit toolkit are as follows:

Multiple-user collaboration: editors built using the DistEdit toolkit should allow users to edit text files jointly without physical proximity. All users should have a consistent view of the editor buffer and should be able to see other users' changes as they occur.

Use of familiar editors: users should not have to change to a different editor in order to collaborate. Since people usually have their own favorite editors, this implies that it should be possible to use different editors in a single group session. It also means that adapting an editor to use the toolkit should not require removing functionality from the editor.

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.

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

Easy adaptation: adapting an editor to use DistEdit should require minimal effort, and no knowledge of distributed systems issues should be needed.

Support for multiple paradigms of interaction: group editors which use DistEdit should be applicable to different types of tasks. For instance, (a) all users could edit, in parallel, different parts of the document or (b) only one user could edit while others observe the changes. The facilities provided by the toolkit should allow editors to be built that support a variety of interaction paradigms.

3 Usage of Editors Based on DistEdit

This section describes, from the users' point of view, the operation of editors based on the DistEdit toolkit. Consider two authors working jointly on a paper who would like to work in parallel on different aspects of the paper from the workstations in their offices. Sometimes, they also might need to interact very closely with each other to come to an agreement on how to rephrase a paragraph. Without a group editor, such a task can be difficult, requiring a great deal of verbal synchronization to avoid editing the same file at the same time. Brainstorming over changes to a paragraph in real time, as one is making changes, is not possible – that has to be done either by getting together or by sending the paragraph back and forth until an agreement is reached.

Use of group editors based on the toolkit can facilitate such a task. The two authors can edit at independent times with their favorite DistEdit-based editor. If they happen to be editing the same file at the same time, a joint editing session is established. DistEdit ensures that they will have consistent views of the file being edited at all times. Each user, however, may edit or view a different portion of the document and could have a different window size.

Figure 1 shows such a situation. One author is using the Xedit editor to edit a file, and the other author is using the Gnu Emacs editor to edit the same file, where both editors have been modified to use the facilities of DistEdit. For the convenience of generating the figure for this paper, displays of both editors are shown on the same screen, although they could be running on different workstations. We assume that the authors can communicate via a telephone or additional shared windows to discuss the joint work.

The operation of each editor changes very little from its normal use without the DistEdit system. A user invokes his editor on the file to be edited in exactly the same manner as he would with a single-user editor. The only visible difference is the addition of a *DE-session window*. In the present DistEdit design, this window consists of two subwindows: (1) a *status window*, which shows the file being edited, the local user's name, and the names of the users currently participating in the editing session; and (2) a *control window*, which allows a user to control the group-specific behavior of his editor.

One noticeable change from single-user editing is the presence of *automatic locks*. When a user does any editing, a temporary lock is acquired automatically on the portion of the document to be modified. Automatic locks are used for concurrency control to prevent two users from simultaneously trying to edit at exactly the same place in the document.

DistEdit also provides *explicit locks*. These locks allow a user to deliberately lock the region on which work is to be done, assuring that no other users can alter that region until the lock is released. Explicit locks are invoked either through the control window or through commands which can optionally be added to the editor.

A user can hold any number of locks at one time and can set time-out periods for both automatic and explicit locks through the control window. The time-out for automatic locks is usually a few seconds while the time-out for explicit locks is much longer, minutes or hours.

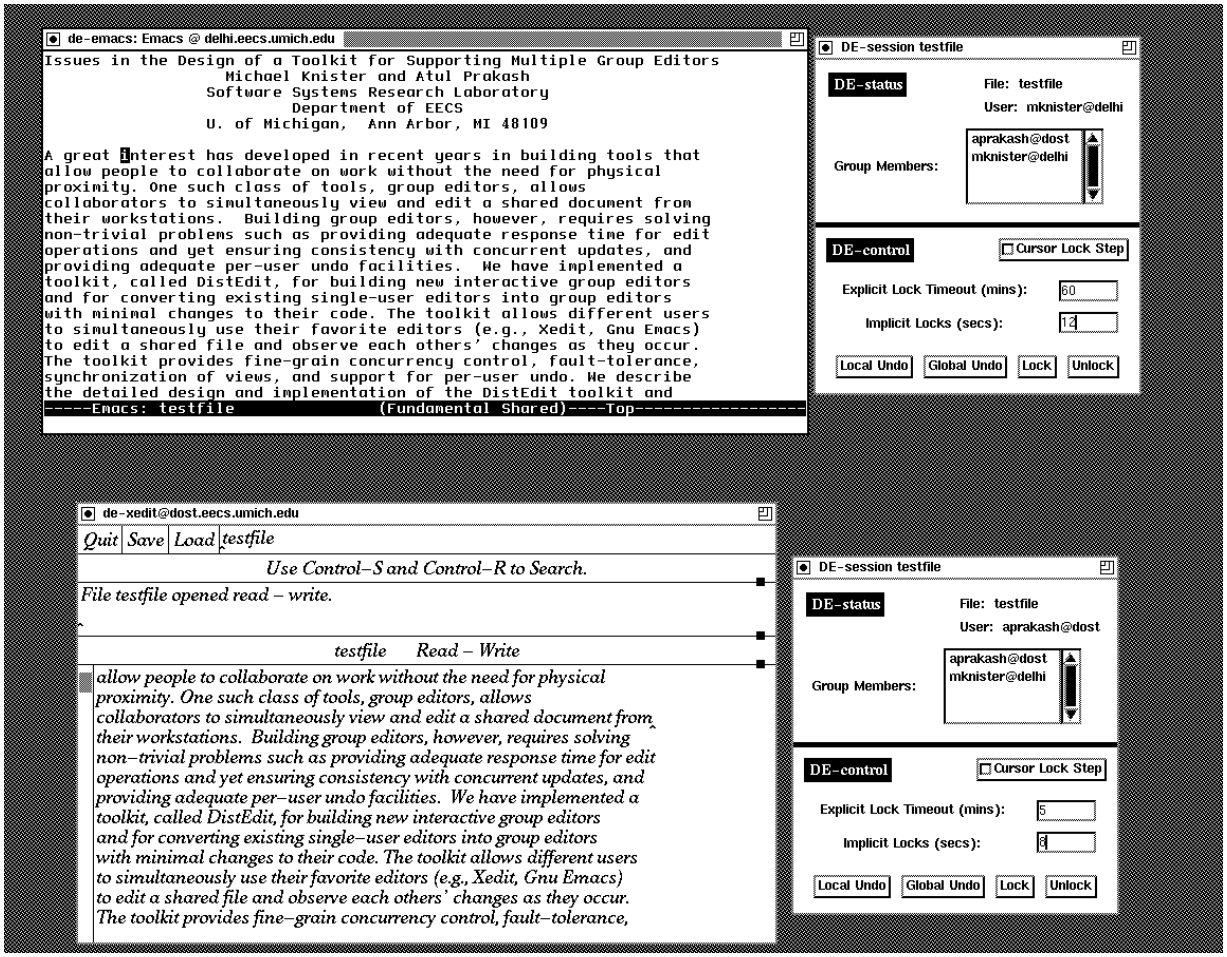

Figure 1: Sample screen display showing DistEdit versions of GNU Emacs and Xedit

DistEdit supplies a simple facility for synchronizing the cursors of different users via the *lock-step* option (see the control window in Figure 1). This option, when invoked, makes the cursor of the user's editor effectively join the cursor of all other users who have also selected the lock-step option. In the present design, no one owns the cursor. When any user with the lock-step mode enabled moves his cursor, all users with the lock-step mode enabled will see their cursors move. If, however, one user in the lock-step mode acquires a lock on the position of the shared cursor (for instance, by starting to edit), then the ability to move the shared cursor is disabled for all other users. If only one user has selected the lock-step mode, the option has no effect.

Several interaction paradigms can be supported using the above basic facilities. For instance, if the group wants to interact in a manner where only one user is allowed to edit while others observe changes, then all users can select the lock-step mode, and the user who will make changes can explicitly lock the entire document (so that others are prevented from editing). On the other hand, if the members of the group want to do concurrent, independent editing, then they could unselect the lock-step mode and avoid explicitly locking large regions.

Any number of users may participate in an edit session. Users may join and leave a session at any time without affecting other users. A failure on one user's machine simply results in that user leaving the session. Should a user leave the session or experience a failure, the other users will observe the change in the status window. The text being edited can be lost only if all users leave the session and none of them have saved it.

A key point is that when multiple editors are being supported, usual editing commands will continue to work for each editor. The only changes in a user's view are the appearance of the status and control windows that specifically have to do with controlling a group session.

4 Software Architecture of DistEdit-based Editors

The high-level structure of a typical single-user text 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 the appropriate output.

The structure of editors modified to use the DistEdit toolkit is shown in Figure 3. As shown, DistEdit-based group editors use a fully replicated architecture, with each editor maintaining a copy of the buffer state. The buffer representations do not have to be identical in each editor; however modifications to the buffer have to be done using a few standardized update primitives provide by DistEdit. Each editor's text update routines are mapped to calls on the *DistEdit primitives* for text update. Those DistEdit primitives first check for possible region overlap with updates of other users. If no region overlap exists, the primitives do the operation locally first and then distribute them over the network, using the ISIS communication package[2], to all the editors. DistEdit-

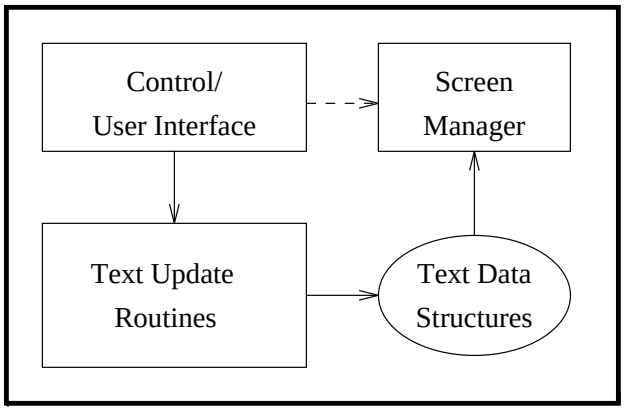

Figure 2: Typical structure of a single-user editor.

provided internal routines then map the received DistEdit primitives back to calls on the standard *access primitives*, which are provided by each editor to update and access the editor's buffer state. The DistEdit primitives and the access primitives are discussed in Section 6.

Using a replicated architecture with different buffer representations is crucial for several reasons. First, DistEdit is designed to support multiple, different editors. Different editors typically use different data structures to represent the editor buffer; forcing a common buffer representation on all the buffers (as in a centralized scheme) would have required us to rewrite in all the editors the substantial code that directly accesses the buffer for display and navigation. Second, we wanted to ensure a fast local response time to all operations done by a user. Using a centralized server to maintain the buffer state would have required going over the network for all updates or display operations. Finally, a centralized architecture would not have been fault-tolerant, a key goal in our design.

The DistEdit system is designed as a modular toolkit which can easily be applied to various visual editors. The only desirable feature in an editor being adapted 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 poor editor design. The editor need not localize the routines which read from the text buffer.

5 Modifications Required in an Editor

The following are the basic modifications needed to convert a single-user text editor to into a group editor using DistEdit:

- The editor code which *directly* modifies the text buffer must be mapped to calls on the DistEdit text update primitives to achieve the same change. The editor routines which

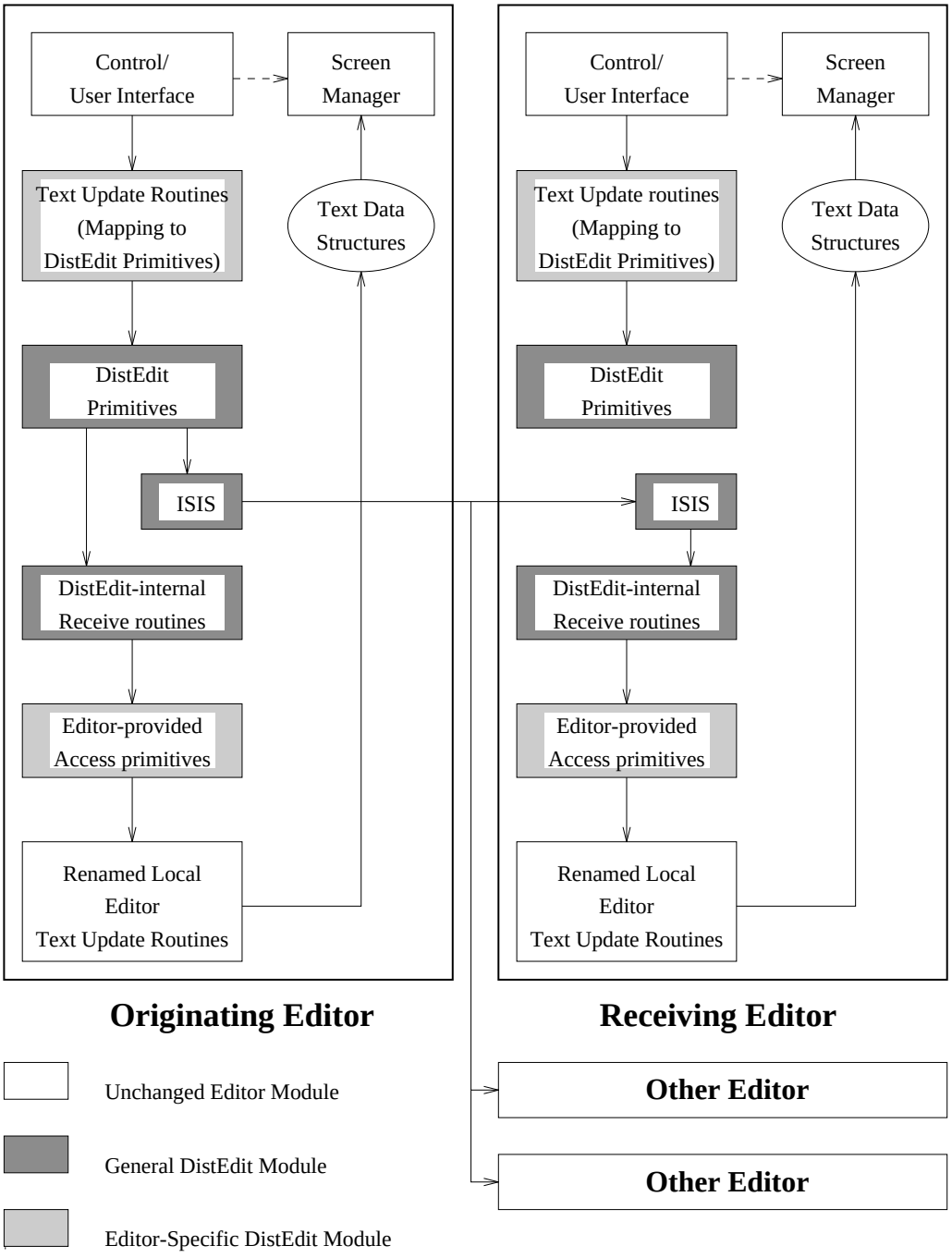

Figure 3: Structure of an Editor built using the DistEdit toolkit

indirectly modify the text buffer by calling other routines normally do not require changes (see Section 7.3 for one case which does require changes). Ideally, to reduce the work in writing code 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 one to ten functions.

- Each editor must provide a common set of access routines. The access routines map the DistEdit text update primitives back to the editor's original buffer-modifying routines. As we will see in Section 6, DistEdit uses very few text update primitives. Little work is therefore required here.

- Each editor also must provide a common set of access routines for DistEdit to retrieve text from the editor buffer, to move and query the cursor, and to control when the screen is updated. Any calls which perform screen updates must be removed from the editor's text-update routines.

- The editor's input handling routines must be modified to call DistEdit code when communication packets arrive from other editors.

- The editor optionally may provide user-interface code to support DistEdit capabilities such as different kinds of undo in a group environment, explicit locking of regions, and notifications of users leaving or joining a group session. This code may be shared by different editors, as exemplified by the use of similar DE-session windows by both Xedit and Gnu Emacs in Figure 1.

- Undo code must be disabled in the editor. Instead, the editor's undo routine should call the DistEdit group undo primitives.

- The editor must call DistEdit primitives for the toolkit initialization and for opening files.

All aspects of the system that deal with concurrency control, group undo, locking, 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 implementation, the editor's actual text update routines are renamed, typically by prefixing a string such as "real_"; they become the bottom layer in Figure 3 and remain the only routines which actually modify the text. No modifications are required to these routines except 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 DistEdit text update primitives.

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 also are simplified, as the general DistEdit code is not intermixed with editor code and can be replaced separately.

6 Editor/DistEdit Interface

This section describes the interface between an editor and DistEdit. This interface consists of *DistEdit primitives* provided by the DistEdit library and of *access primitives* supplied by each editor to allow DistEdit to retrieve and update the editor's state. The two types of primitives are listed in Tables 1 and 2.

A DistEdit primitive is an operation provided by DistEdit that can be invoked from the code in an editor. A single call to a DistEdit primitive can result in messages being sent across the network and code being invoked in numerous other editors. In this sense, a DistEdit primitive is somewhat like a remote procedure call, except that it could result in numerous remote executions, all hidden from the caller of the primitive.

An access primitive is an operation that is callable from DistEdit to retrieve or update the state of an editor. The routines implementing the access primitives are created separately for each editor which is to use DistEdit. Most of the effort required to adapt an editor to use DistEdit is spent creating these access routines. Very little original editor code must be modified.

The selection of DistEdit primitives was a critical design decision in DistEdit. The types of primitives include: *editing primitives*, *locking primitives*, *notification primitives*, and *control primitives*. Most important among these are the editing primitives, those that make changes to the document. The issues in the selection of these primitives are elaborated next.

6.1 Editing Primitives

Editing primitives are those that modify the document state. In single-user editors, whether to treat an operation as a primitive or as a sequence of other more primitive operations is dictated primarily by efficiency concerns. For instance, in a single-user editor, an *IndentParagraph* operation may be treated as a primitive operation rather than composed as a sequence of *InsertChar* and *DeleteChar* operations if it is simpler and more efficient to indent a paragraph by directly accessing the document buffer than by calling *InsertChar* and *DeleteChar* operations on the buffer. In a group editor, on the other hand, there are many other factors that need to be considered:

Heterogeneity Issues: Every editor built using the toolkit has to be prepared to support all the primitive operations. Thus, if it is expected that the toolkit will be used in an environment with different users using different editors in the same group session, all the editors need to have routines mapping the same set of primitives to updates on the buffer. Thus, if *IndentParagraph* were made a primitive and one wanted to support both Xedit and Emacs using

Table 1: **DistEdit Library Primitives**

de_init()	Initialize the toolkit
de_open(filename) de_close()	Join a group editing session, or create a new one Leave a group editing session
de_insert(position,text,length) de_delete(position, length)	Basic operations to modify buffers through DistEdit, including automatic locking and concurrency control
de_lock_region(region, timeout) de_unlock_region(lock id) de_lock_info(position, lock_info)	Explicitly lock a region; optional to use. Explicitly unlock a region; optional to use. Supplies information on lock id, its range, and ownership
de_local_undo() de_global_undo() de_reset_undo()	Executes the per-user history undo function Executes the global history undo function Switches out of history undo mode
de_setting_set(setting-name, value) de_setting_get(setting-name)	Set/retrieve settings such as lock timeouts, user name, debugging flag.
de_run()	Allow DistEdit to carry out its processing
de_fds() de_select()	Event-loop blocking utilities to know when de_run should be called
de_notify_set(join/leave/lock change/cursor change)	Requests that DistEdit inform the editor of changes in group status, locks, and cursor positions by calling loc_notify

Table 2: **Editor-provided Access Primitives**

loc_insert(position,text,length) loc_delete(position, length)	Basic operations which allow DistEdit to modify a buffer; must provide a mapping into editor's internal buffer-modifying operations.
loc_cursor_set(position)	Allows DistEdit to move the cursor
loc_cursor_get(position)	Supplies DistEdit with current cursor position
loc_display_update()	Allows DistEdit to control when the display is updated
loc_text_get_start(position, length) loc_text_get_next()	Prepares to return portion of editor's buffer data in chunks Returns next chunk
loc_notify(status report)	Notifies the editor of changes in group status, if desired

DistEdit, both editors would need to be able to understand the *IndentParagraph* command. Clearly, the amount of work required in modifying the editors can be large if the number of primitives is large and heterogeneity is to be supported.

Communication and Processing Requirements: Communication and processing requirements will usually go down if a complex operation is made a primitive operation, rather than mapped to a sequence of more basic primitives. For instance, it probably would be cheaper to transmit an *IndentParagraph* command, rather than the sequence of *InsertChar* and *DeleteChar* commands to which it might map.

Support for Undo: Undo implementation is much more complex in a group editor [17] than in a single-user editor. In particular, the ability to reverse and resequence operations is needed for all the primitive operations. It is much easier to provide this capability if the set of primitive operations is small. Thus, to simplify the implementation of undo, it is better to implement the *IndentParagraph* operation as a sequence of more basic primitives.

Reordering of Operations: Even if undo is not supported in a group editor, there may be reasons to keep the primitives restricted to a small set. For instance, if a scheme such as that in [5] is used to ensure consistency, functions to resequence operations are required (T_{op} matrix in [5]). Defining such functions is much easier if the set of primitives is small.

In DistEdit, we chose to support a very simple and general model of text editing. A text buffer (document) is considered to be a single string of characters with a cursor pointing somewhere in

that string. Characters in the text are referenced by their offset from the beginning of the text, and line breaks are treated as 'newline' characters. This simple model is compatible with almost any visual text editor, allowing DistEdit to work with many different editors.

There are two basic DistEdit editing primitives which modify the buffer: *de_insert(position, string, length)* and *de_delete(position, no. of characters)*. A group editor using DistEdit must use these two primitives to carry out all editing operations. All other editing operations which an editor provides must be mapped into one or more *de_insert* and *de_delete* calls.

For each editor, corresponding access primitives *loc_insert* and *loc_delete* must be provided to allow DistEdit to change the editor's buffer. These routines are called when editing operations are received from other editors (see Figure 3).

These two text update primitives were chosen for their generality, simplicity, and conciseness. They are general enough to perform any desired editing operation, given that line breaks are handled properly. They are quite simple to understand and implement in any editor. Finally, they are quite concise; only two are required. This reduces the effort required to implement the primitives when modifying an editor to use DistEdit; supporting more primitives is simply more work.

In DistEdit, only the text buffer and cursor position are shared. Cut buffers, bookmarks, and many other features found in some editors are outside the scope of DistEdit; they remain strictly local to each user. These features appeared to be of questionable value for sharing and are not even provided by all editors. Thus, we chose to keep them local to each editor, rather than to add them to DistEdit at this time.

6.2 Locking Primitives

DistEdit-based group editors use an automatic locking scheme to manage concurrent access to the document. Before any *de_insert* or *de_delete* is carried out, DistEdit acquires the smallest possible lock for the affected region, ensuring that multiple users do not modify the same area simultaneously. This form of locking is hidden from the editor; the toolkit takes care of everything. Locks automatically time out after a chosen delay. Implementation of the locking strategy is described in further detail in Section 7.2.

One implication of locking is that a *de_insert* or *de_delete* may fail due to an inability to lock the appropriate part of the document. This can cause problems for an editor which internally assumes that all text-modifying routines will succeed. If this is the case, the transaction feature described in Section 7.3 can be used to avoid failures from becoming visible to the editor.

DistEdit also supports explicit locking, where a user deliberately selects and locks a region of the document. The primitives *de_lock_region* and *de_unlock_region* can be called for this purpose. Use of these primitives is optional; an editor can choose to use only automatic locking.

Each lock is associated with a contiguous region of text covering at least one character; a user can change a region only if he owns a lock covering the entire region. Inserting a string requires obtaining a lock on the character which precedes the point of insert. Deleting a string requires a

lock covering the characters of the string. To permit inserts at the beginning of a buffer, the buffer is considered by DistEdit to have an imaginary, undeletable character marking the beginning of the buffer.

As insertions or deletions are performed within the region, the associated lock expands or shrinks accordingly. A lock is deleted automatically if it shrinks to size zero.

Regions covered by locks do not overlap. When a user requests a lock for a region that already contains locks belonging to him, all those locks are merged into a new lock. The new lock covers the smallest contiguous region that contains the previous locks and the requested lock.

At the toolkit level, time-outs can be independently set for each lock. At the user level, our DE-session window currently allows two time-out values to be set by each user, one for all automatic locks and one for all explicit locks.

6.3 Notification Primitives

DistEdit allows an editor to take action based on events occurring in other editors. An editor can use the *de_notify_set* primitive to request that DistEdit inform it of any changes in the cursor positions, group status, and locks held by group members. Based on this information, the editor could highlight locked regions or supply other features. Using these primitives is optional.

To provide cursor notification to other users, DistEdit calls the editor access primitive *loc_cursor_get* to determine the current cursor location. The other types of notifications are supported by monitoring calls to primitives such as *de_insert*, *de_delete*, *de_open*, and *de_close*.

6.4 Control Primitives

DistEdit supplies control primitives to manage group membership, DistEdit settings, and to control when DistEdit runs. It also requires that access routines be supplied by each editor to control the editor's cursor and display updates, and to retrieve the contents of the text buffer.

The *de_open* primitive searches for an existing group session for a particular file. If it finds one, it transfers the text buffer from that group to the new user, automatically bypassing the file on disk. If no session exists, it loads the file and creates a group session with a single member. The *de_close* primitive is used to leave a group editing session. See Section 7.5 for details of file access.

DistEdit maintains a number of settings which can be accessed with the *de_setting_set* and *de_setting_get* primitives. Settings include the name of the user and lock timeout periods for both automatic and explicit locks.

To allow DistEdit to process activity from other users, the editor's event loop must be modified to let DistEdit run when the editor is not busy. The *de_run* primitive is then called from the editor's main event loop; this primitive causes DistEdit to process all network packets and update the editor's buffer with any recent changes. The input blocking of the editor's normal keyboard/window

system can be supplemented with the *de_select* and *de_fds* primitives to determine when DistEdit needs to run.

7 Implementation of DistEdit Primitives

This section describes the implementation of key DistEdit primitives listed in Table 1. When a DistEdit primitive is invoked, locks must be acquired if necessary, and messages have to be sent out to other editors to update their state. Because users can edit concurrently, and updates are performed locally first, editors potentially could process updates in different orders. Care must be taken to ensure that resulting state is identical in all editors and matches the expectations of the users. The following subsections address this issue and other implementation issues.

7.1 Underlying Communication Software

ISIS [2], 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. A globally ordered broadcast is available which guarantees that messages 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 new user's editor. ISIS provides mechanisms for notification of new users and fault-tolerant communication protocols to facilitate 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, events, such as a group member failing, can be handled by triggering a lightweight process.

7.2 Dealing with Concurrent Updates

Unlike the earlier version of DistEdit described in [12], the current version of the DistEdit toolkit provides support for concurrent updates. To keep response time low, any update is performed locally first and then broadcast to other sites. It is well known that concurrent updates can

Table 3: **DistEdit Protocol Messages**

LOCK(previous lock id, offset, length, last_lock_id)	attempts to acquire a lock; globally ordered
UNLOCK(lock id)	owner of a lock releases the lock; globally ordered
INSERT(lock id, offset, string, undo_info)	owner of given lock inserts text within locked region
DELETE(lock id, offset, length, undo_info)	owner of given lock deletes text within locked region
CURSOR(lock id, offset)	notifies of a user's cursor position
JOIN(member id)	ISIS informs of a new group member
LEAVE(member id)	ISIS informs of a group member leaving
STATE_TRANSFER(buffer contents, lock table)	transfers state of the current buffer and state of the locks to the new UNLOCKed (JOINing) group member
INFO(member info)	member informs group of user name and host name

lead to inconsistencies in the buffer state at various sites [5]. We use an efficient locking-based solution which requires locks to be acquired only at the start of an insert/delete but not during an insert/delete. For instance, if a user starts to insert a sequence of characters, there is a slight network delay in acquiring a lock prior to the insert of the first character, but after that inserts proceed at the speed of the local editor. Another reasonable alternative would have been to use the somewhat more complex scheme suggested in [5], which does not require locks but does require messages to contain version vectors and requires messages to be processed against a command log.

Table 3 shows the messages in the protocol which DistEdit uses to maintain consistency among editors.

7.2.1 Insert/Delete Primitive Processing

All changes to a text buffer begin with a call to the DistEdit *de_insert* or *de_delete* primitive. DistEdit consults a lock table to determine whether the user has a lock covering the affected region. If DistEdit finds a lock, it immediately updates the local editor's buffer (for a quick response time), and broadcasts an INSERT or DELETE message to the group.

If the lock table indicates that the user does not have a lock covering the affected region, DistEdit first attempts to acquire a lock, as described in Section 7.2.3. This process requires several network messages and can cause some delay. If the lock attempt succeeds, the INSERT or DELETE message proceeds. If the lock fails, the primitive fails, and the transaction mechanism of Section 7.3 may

be used to recover.

7.2.2 Lock-relative Messages

INSERT and DELETE messages do not contain the absolute positions where the operations are to occur, since these positions may be different for other users depending on messages in transit. For example, suppose users A and B are working on a document. User A is working at the top, and has the first 50 characters locked; user B has the following 50 characters locked. Suppose, at the same time, both users insert a character at the beginning of their locked regions, user A at position 1, user B at position 51. Both immediately update their local buffers and send out an INSERT message. A, upon receiving B's message, must perform performs B's operation at position 52, not 51, because A inserted a character earlier in the document that B did not know about when the message was sent.

To avoid this address shifting problem, INSERT and DELETE messages contain character positions relative to the beginning of the locked region to be changed. In the above example, B's INSERT message would indicate that the operation is to occur at offset 0 from the starting position of the lock held by B. All locks are given globally unique identifiers so that they can be referenced in such messages.

Because only the owner of a lock can make changes within a locked region, the owner immediately can perform any change within the region, and INSERT/DELETE messages sent to other editors need only be ordered relative to the sender. This type of message can be sent directly to the recipients, with no need for a central point for routing. INSERT and DELETE messages are therefore very fast.

7.2.3 Lock Acquisition

For locking to ensure consistency between the buffers of different editors, the editors must have strictly consistent views of the locks. DistEdit maintains a table for each group member which contains every existing lock. Thus, the lock table, like the buffer text, is replicated for every group member.

The LOCK and UNLOCK messages use the ISIS globally ordered broadcast mechanism to achieve consistency in the lock tables. When a lock is requested for a given region, DistEdit broadcasts a LOCK message containing the request to the entire group. ISIS guarantees that all LOCK/UNLOCK messages are received in the identical order by every member. If the region in an incoming LOCK message is not already locked, the lock request is granted and a new entry is created in the lock table. If the region overlaps an existing entry in the lock table, the lock is refused unless the lock is owned by the same user. After sending a LOCK message, the sender waits to receive the message back (globally ordered) to determine whether the request succeeded or failed. Because every group member receives the same messages in the same order, from the

same starting state, the lock tables are always identical. Each editor independently uses the order of arrival to assign the lock a globally consistent, unique identifier.

Only the owner of a lock can release a lock using the UNLOCK message. This ensures that the owner is not planning to send any further INSERT or DELETE messages based on that lock. DistEdit automatically sends an UNLOCK message when a lock has not been used within a user-defined period. If the owner crashes or exits without releasing his locks, other editors receive notifications through ISIS about a member failure, and they release all the locks belonging to that owner (such notifications are hidden from the editor code – the toolkit code handles the notifications and releasing of locks). ISIS guarantees that all members of the group have a consistent view of member failures.

Locks, like INSERT and DELETE messages, use a lock-relative addressing scheme. DistEdit converts the starting absolute address of a lock request to an offset from the *end* of the nearest prior lock in the table. Using absolute offsets in a lock acquisition request would have presented problems in the following scenario: suppose user A requested a lock at position 50 and, at the same time, user B inserted one character at position 1. Furthermore, assume that the insert operation is received at other editors before the LOCK message. In such a case, the lock really should be granted at position 51, not 50. By making the lock request contain the offset from the end of the previous lock, we avoid the need for such adjustments in positions.

Using a reference to the prior lock in a new LOCK message, however, can present two problems. First, the referenced lock could have been deleted by the time the new LOCK message is received. Second, another LOCK message with the same reference from a different user, and perhaps even the subsequent UNLOCK message, could be received, invalidating the relative address. Both problems are fairly rare – we had to try editing operations several times to create the right timing. The solution therefore adopted in DistEdit is to have the lock request fail when such situations are detected. Detecting them is accomplished easily by including the identity of the last received lock in the LOCK message. Another possible solution would have been to log LOCK and UNLOCK messages and use that log to adjust the LOCK messages based on the state when they were sent.

A single LOCK message can contain any number of lock requests. The entire set of locks in the message is granted or rejected as a group. Thus, if an operation requires a large number of changes throughout a document, all necessary locks can be acquired at once to assure that the operation will succeed as a single transaction. This facility is used by the transaction mechanism that is described next.

7.3 Transactions

Since DistEdit only provides two basic editing operations, *de_insert* and *de_delete*, any other editing operations to be provided by an editor must be mapped to a sequence of these operations. An important issue then is whether to treat that sequence of lower-level operations as a single atomic (indivisible) action or as a sequence of independent operations.

In single-user editors, treating a group of simple operations as one larger, user-level operation is important primarily for implementing undo; a user, upon doing an undo operation, usually expects all the changes associated with the last single user-level action to be undone, rather than just some of them.

In a group editor, the issue becomes important because of two assumptions that are embedded in the code of most existing editors. First, editors assume that operations like *insert* and *delete* always succeed; with DistEdit, operations can fail if they cannot acquire the needed locks. Second, they assume that nothing will change the buffer in the middle of a user-level operation.

We faced the above issue in handling the *replace-string* command in Gnu Emacs. The *replace-string* command is implemented as a Lisp function that searches for the specified string, stores its position, replaces the string at that position with a new string, and then continues the search from the stored position added to the length of the replacement string. The function assumes that after the replace, the cursor has moved to the end of the replaced string. If the replace fails, as it could due to locks, or if intervening updates are received, the algorithm could replace the wrong string or start the next search from the wrong location.

Our goal was to deal with the above two problems but not to require changes to the substantial code that implements multi-operation actions in existing editors. We accomplish this goal by using a *delayed* lock acquisition strategy for providing atomicity. The beginning and end of a transaction are determined using the *de_run* primitive. The editor calls this primitive each time it waits for input. This is considered to mark the beginning or end of a transaction. All activity in between calls to *de_run* is considered a transaction. For example, say a user initiates an *IndentParagraph* function from the event loop. This function will generate a lengthy sequence of *de_insert* and *de_delete* operations before returning to the user for the next command. The entire sequence between calls for user input is considered to be one transaction.

During a transaction, DistEdit records each *de_insert* and *de_delete*, and executes each one on the local buffer. Because every command is executed, the primitives never fail, and any non-failure assumptions built into the editor are satisfied. During the transaction, no changes are displayed on the editor screen. When the transaction ends, DistEdit uses a single LOCK message to attempt to acquire all locks required for the transaction that are not already held. If the lock request is granted, the transaction succeeds; the local display is updated and the changes are broadcast to other users. If the lock request fails, the transaction fails, and the DistEdit undo mechanism is used to roll back all the changes. In this case, the user or the editors never see that the changes actually occurred.

The above strategy of relying on calls to *de_run* to determine transaction boundaries does not quite work for interactive commands such as a *query-replace* in Emacs, which asks the user whether to make each individual replacement. In that case, *de_run* will get called whenever user is asked for input and each set of changes between prompts will be considered to be a separate transaction, which we find acceptable. The problem is that if a replace operation fails due to the inability to acquire a lock, any non-failure assumptions built into the *query-replace* routine will not be satisfied.

Such high-level routines that interact with the user simply have to be rewritten to make failure assumptions. Fortunately, most editors have only few such routines.

Transactions using delayed lock acquisition incur a negligible penalty for basic editing operations and actually enhance the performance of complex operations. No additional messages are ever required; however the operations may have to be undone locally if the transaction fails – a rare occurrence in practice. For complex operations, transactions effectively batch the lock request messages into a single message, reducing overhead.

The transaction mechanism can be enabled or disabled through the *de_setting_set* primitive. For editors which make non-failure assumptions, such as Gnu Emacs, we recommend always using the transaction facility, since the overhead is low.

7.4 Undo

Implementation of undo is more complex in group editors because a per-user undo facility is needed that undoes a user's last action rather than the last action seen by the editor. In [17], we proposed a general framework for undoing actions in collaborative systems. The framework takes into account the possibility of conflicts between different users' actions that may prevent a normal undo. The framework also allows selection of actions to undo based on who performed them, where they occurred, or any other appropriate criterion. DistEdit provides a per-user undo facility using those ideas that allows users to undo just their own changes. It also provides a global undo facility that allows users to undo globally last actions irrespective of who executed them.

To implement per-user undo, each editor maintains a history list which contains all prior insert/delete operations. Each operation is tagged with the identity of the user who performed it. When the *de_local_undo* primitive is invoked, DistEdit looks through the history list and finds the last operation that was done by the user. DistEdit then attempts to undo the operation by *shifting* the operation to the end of the history list using a sequence of *transpose* operations [17]. If the operation is shifted successfully (i.e., no conflicts with later changes of other users), the operation is undone by executing its inverse operation. To update the state of other editors, the inverse operation is broadcast to other editors. For more details of this strategy, see [17].

7.5 File Management

Several problems arise from sharing document files. First, when a user requests a file be opened for editing, DistEdit must determine whether anyone else is editing that file and, if so, load from the active group session rather than from the file. Second, a user should not be allowed greater editing access rights using DistEdit than the file system would allow. Third, care must be taken should several users attempt to save a shared file at the same time.

In determining whether several users wish to edit the same particular file, it is not possible simply to examine the path names of the files; because of network file systems, a file can be

potentially referenced by different paths.

When the *de_open* primitive is called to open a file, DistEdit searches in the directory containing the file for an auxiliary file of the same name without the path prefix, but prefixed by '.de.'. For example, when opening the file */u/aprakash/docs/testfile*, DistEdit will search for the auxiliary file */u/aprakash/docs/.de.testfile*. This auxiliary file contains a unique identifier to be used as the ISIS group name for the particular file. If no such file exists, DistEdit creates it so other users will be able to join the session. If a file is a soft link, the link is resolved before applying the above procedure.

If a file has multiple hard links to it, the above procedure may fail; one could deal with that by using an alternative, more complex, approach of looking up machine name, device name, and i-node number of the file being edited, and using their combination as the unique id.

After obtaining the unique identifier, DistEdit instructs ISIS to join or create a group session. If no session exists (as indicated by ISIS), DistEdit loads the editor's buffer from the file. Otherwise, DistEdit employs the ISIS state transfer mechanism to obtain the current state of the buffer and the lock table from a member of the group.

DistEdit provides read-only/update editing rights based on a user's access rights to a file. A user who has only read-only permission to a file is not allowed to make changes using a DistEdit group editor. In such a case, the *de_open* primitive sets an internal read-only flag. When this read-only flag is set, the *de_insert* and *de_delete* primitives always fail, just as if a lock could not be acquired, thus preventing the user from making changes through the group editing session. Such a user, however, may still observe the changes of others and use all other DistEdit facilities.

We have found that the normal file save routines of editors work quite well with DistEdit. There is, however, a potential problem. If several group members were to save different versions (due to network message latency) at approximately the same time, the resulting file could be different than any user's version of the file. One approach that could be used to solve this problem would be for DistEdit to provide a locking mechanism for file saves. The editor requesting the save would have to acquire this lock before the save and release it afterward.

8 Experience in Using DistEdit

8.1 Adaptation Effort

The amount of effort required to adapt editors we have used is fairly low. Table 4 shows the total size of three editors, the lines of code added to support DistEdit, and the number of lines in the body of the editor code which had to be changed.

To give an idea of time required, the editor most recently adapted, Xedit, required about four hours for the conversion. GNU Emacs, which was adapted gradually as DistEdit developed, required much more effort because of its complex, multi-platform input code, and because we were

Table 4: **Changes Required to Adapt Editors for DistEdit**

Editor	Total Editor LOC	LOC Added	LOC Changed
GNU Emacs	75,000	450	5
MicroEmacs	20,000	60	6
Xedit	16,000	250	30

solving problems related to concurrent updates, transactions, and group undo while working on the GNU Emacs conversion.

We considered adapting the *vi* editor to use DistEdit but concluded it would be too difficult. The *vi* buffer modification routines were spread across the entire system, and much of the code was designed originally for *ex* command-line interaction, a style which is not amenable to group editing. We have examined Elvis, a functional clone of *vi* and believe it can be adapted to use DistEdit.

8.2 Usage Experience with the Editors

We have used the editors developed using the toolkit internally within our group as they have been evolving. Our experience indicates that people are more likely to use a group editor consistently if they do not lose functionality in switching from the corresponding single-user editor. The earlier versions of DistEdit-based Gnu Emacs, our favorite editor, did not behave as expected as they did not provide per-user undo. Furthermore, they did not deal properly with multi-operation actions. The current version of DistEdit-based Gnu Emacs addresses these problems using the techniques outlined in this paper and has been found to be much more satisfactory. For results of usage studies with group editors in general, see [6].

Work still needs to be done for support of shared editing of multiple buffers – at present, editing multiple files in a shared manner requires opening one editor per file, as the DistEdit library primitives allow only one file to be edited in the shared mode.

Performance of the editors built using the toolkit has been very satisfactory. We have used them primarily in local area environments. Local updates are done with no noticeable delay. The updates on other editors usually appear immediately. However, if the network is congested, updates messages are sometimes batched (by ISIS) and after a small delay, several updates are seen in rapid succession.

9 Conclusions

The DistEdit toolkit has allowed the creation of editors which provide a seamless transition between individual and group editing. These editors, which include Gnu Emacs and Xedit, provide a familiar

and powerful work environment in a group setting, simplifying parallel work as well as facilitating close coordination. Adapting editors to group use with the toolkit has required surprisingly little effort. The toolkit provides support for all necessary distributed systems protocols, concurrency control, fault-tolerance, and synchronization. It also supplies powerful additional features, such as undo, to all editors which use it.

The original motivation behind the use of simple, standard primitives for communications in DistEdit was to facilitate migration of existing editors to group use. It has, however, turned out that the design is also appropriate for group editors where only a single editor interface is supported. Choosing more complex primitives simply would mean more work in implementing transactions and the group undo facilities.

The current version of the toolkit only provides global and per-user history undo. In [17], we suggested that other undo methods, such as region-undo and time-based undo, also might be useful in group environments. We plan to enhance the toolkit to provide support for these other types of undo facilities.

We also plan to work on building toolkits similar to DistEdit for other types of documents such as rich text and graphics. Supporting such document types will require defining the semantics of their primitive operations so that services such as group undo and concurrency control can be supported.

10 Acknowledgements

The authors would like to thank the referees for their valuable comments. This work reported here has been supported by the National Science Foundation under the grant number IRI-9216848 and by a fellowship from AT&T Bell Laboratories.

References

[1] 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*, pages 10–16, Nov. 1988.

[2] K. Birman et al. *The ISIS System Manual, Version 2.0*, April 1990.

[3] P. Dewan and R. Choudhary. A flexible and high-level framework for implementing multi-user user interfaces. *ACM Transactions on Information Systems*, 10(4):345–380, October 1992.

[4] C. Ellis, S.J. Gibbs, and G. Rein. Design and use of a group editor. In G. Cockton, editor, *Engineering for Human-Computer Interaction*, pages 13–25. North-Holland, Amsterdam, September 1988.

[5] C. Ellis, S.J. Gibbs, and G. Rein. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD '89 Conference on Management of Data*, pages 399–407. ACM Press, 1989.

[6] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, pages 38–51, January 1991.

[7] M. Elwart-Keys, D. Halonen, M. Horton, R. Kass, and P. Scott. User interface requirements for face to face groupware. Technical Report CMI-89-020, Center for Machine Intelligence, Ann Arbor, MI, December 1989.

[8] R. Fish, R. Kraut, M. Leland, and M. Cohen. Quilt: A collaborative tool for cooperative writing. In *Proceedings of ACM SIGOIS Conference*, pages 30–37, 1988.

[9] S.J. Gibbs. LIZA: An extensible groupware toolkit. In *Proc. of the ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 29–35, April 1989.

[10] I. Grief, R. Seliger, and W. Weihl. Atomic data abstractions in a distributed collaborative editing system. In *Proc. of the 13th Annual Symposium on Principles of Programming Languages*, pages 160–172, 1976.

[11] D. Halonen, M. Horton, R. Kass, and P. Scott. Shared hardware: A novel technology for computer support of face to face meetings. Technical Report CMI-89-015, Center for Machine Intelligence, Ann Arbor, MI, November 1989.

[12] M. Knister and A. Prakash. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 343–355, Los Angeles, California, October 1990.

[13] L. McGuffin and G. M. Olson. ShrEdit: A shared electronic workspace. Technical Report CSMIL Technical Report No. 45, The University of Michigan, Ann Arbor, 1992.

[14] C.M. Neuwirth, D.S. Kaufer, R. Chandhok, and J.H. Morris. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 183–195, Los Angeles, California, October 1990.

[15] R.E. Newman-Wolfe and H. K. Pelimuhandiram. MACE: A fine-grained concurrent editor. In *Proceedings of the ACM/IEEE Conference on Organizational Computing Systems (COCS 91)*, pages 240–254, Atlanta, Georgia, November 1991.

[16] J.F. Patterson, R.D. Hill, S.L. Rohall, and W.S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 317–328, Los Angeles, California, October 1990.

[17] A. Prakash and M. Knister. Undoing actions in collaborative work. In *Proceedings of the Fourth Conference on Computer-Supported Cooperative Work*, pages 273–280, Toronto, Canada, October 1992.

[18] M. Roseman and S. Greenberg. GroupKit: A groupware toolkit for building real-time conferencing appliations. In *Proceedings of the Fourth Conference on Computer-Supported Cooperative Work*, pages 43–50, Toronto, Canada, October 1992.

[19] 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*, 30(1):32–47, Jan. 1987.