

Using Familiar Single-User Editors for Collaborative Editing

Du Li, Rui Li, Yingwei Yu, and Yi Yang
 Department of Computer Science
 Texas A&M University
 College Station, TX 77843-3112 USA
 Contact Email: lidu@cs.tamu.edu

ABSTRACT

A number of real-time group editors have been developed as vehicles for investigating various technical issues in computer-supported cooperative work (CSCW). Excellent as they are in serving academic purposes, those research prototypes are not (and will probably not be) widely used by ordinary users for “serious” group editing activities. The reason is primarily that they are generally not as powerful or usable as single-user editors that people are already familiar with. In this paper, we propose a novel approach which converts existing single-user editors to group editors without modifying their source code. As a result, different editors such as Word and GVim can be shared by a group of distributed users to edit the same document simultaneously. Users can use familiar single-user editors for collaborative editing which are allowed to be heterogeneous. Towards this end we have been working on a research project called *intelligent collaboration transparency* or ICT to address related technical issues. Our work is novel compared to existing application-sharing systems especially in that it is able to interoperate heterogeneous single-user applications. We report our latest progress on this project.

Keywords

Groupware, CSCW, Group Editing, Application Sharing, Collaboration Transparency, Heterogeneity, Interoperation

1 INTRODUCTION

Real-time group editing has been attracting significant and lasting research interests in the CSCW field, to name but a few, Dewan et al [7], Ellis et al [10], Knister and Prakash [16], and Sun et al [25]. Many researchers have been using group editors as an effective vehicle for investigating a variety of research issues that are common to a range of collaborative applications [7]. As a result, a number of group editors have been developed since the birth of CSCW, e.g., Grove [10] and Reduce [27]. In general, group editors are built in a collaboration-aware manner, either *ad hoc* (e.g. Reduce [27]) or with the help of some groupware toolkit (e.g. DistEdit [16]).

Excellent as those group editors are in serving academic purposes, no evidence has been reported on their actual use in “serious” group editing tasks, e.g., a group of geographically separated people edit a document together at the same time. According to Grudin [14], research projects usually have purposes other than producing something useful, e.g., to explore some interesting technical problem. As a result, those prototype group editors generally lag behind in features and compatibility with well-accepted single-user word processors such as the Microsoft Word. As noted by Crowley et al [6] and Grudin [14], groupware features are often used less frequently than features supporting individual activities. Having to learn new interfaces for a sporadic cooperative task will discourage many users. Therefore, as a result, ordinary users are not motivated to abandon their favorite word processors to use a coauthorship application [14].

An alternative approach to addressing this problem is called *collaboration transparency*, in which existing single-user applications are shared for cooperative work without modifying their source code. The relative merits of this approach have led to a plethora of application-sharing systems, for example, Rapport [2], MMConf [6], Dialogo [19], XTV [1], SharedX [11], and Microsoft Netmeeting. Generally those systems deliberately avoid assuming any application-specific knowledge in the application-sharing system [19] so that a generic system can be used to share *any* single-user applications.

However, we argue that existing application-sharing systems are not sufficient in supporting such intellectual activities as group editing, for the following reasons. First, according to Begole et al [4], Bholal et al [5], and Ellis et al [10], group editing usually demands high local responsiveness and independent work among collaborators. Unfortunately, as revealed in previous work (e.g. [1][6][4][19]), existing centralized application-sharing systems usually suffer from performance problems and are not able to provide sufficient responsiveness at the Internet scale, while both replicated and centralized systems generally use turn-taking mechanisms to ensure consistency, which allow for limited concurrent

work. Secondly, while existing work enables group editing using single-user editors, it has the major drawback of forcing all participants in the same collaboration to use the same editor [17]. According to Greenberg [12], a number of non-technical issues affect the effectiveness and acceptance of application-sharing systems. One becomes a second-class citizen in the group if one's favorite editor is not chosen for sharing. Forcing all users to use the same application in cooperative work is often counterproductive and results in group resistance [13].

In this paper we present a novel application-sharing approach to developing collaborative editing systems. Our approach, as termed "intelligent collaboration transparency" or ICT, allows people (e.g. coauthors of the same paper) to use their favorite single-user editors that are allowed to be heterogeneous (e.g. MS Word and GVim) for group editing tasks. It adopts a replicated architecture in which an ICT agent is interposed between the editors to be shared and their respective windowing systems at each site. The ICT agents are able to execute application semantics such as how different input streams are mutually translated and how consistency between different editors are achieved. When different editors or other single-user applications are introduced, we only need to load the corresponding components that formalize the application knowledge. Compared to previous work in application sharing, ICT is novel in that it is able to share and interoperate *heterogeneous* single-user applications. We show further how important features such as concurrent work, synchronization, spontaneous interaction, and late-joining, which are difficult to achieve in traditional application-sharing systems [19], become more tractable and efficient by exploring application semantics.

The rest of this paper is organized as follows. The next section presents our technical approach to sharing and interoperating single-user editors. This is followed by a discussion of some system problems and a brief account of application scenarios. Related work is then compared next. We conclude this paper with a summary of technical contributions of the presented work and future directions.

2 SHARING HETEROGENEOUS EDITORS

We define the concept of *application family* as a category of single-user applications that are functionally equivalent or compatible. For example, MS Word, GVim, and Emacs belong to the same application family since they are used to edit documents. In practice, however, people prefer different editing styles. When writing papers, some prefer using MS Word which integrates editing and formatting so that what you see is what you get. Some others are more used to the latex style in which a document is typed in using GVim and Emacs where the formattings are not directly visible until compiled

by latex and then viewed using a separate tool (e.g. GhostView). Although the styles are different, those applications are text editors and are considered compatible in functionality. For the scope of this paper, we do not consider the editing of figures and complicated formattings which are available in Word but not in GVim. Those advanced features are generally available in neither pure single-user text editors nor existing multi-user text editors. Supporting pure text editing already makes our work comparable to the state-of-the-art group text editors such as Reduce [27]. The major difference and novelty is that we allow users to use their familiar single-user editors for collaborative editing.

Consider a scenario in which a group of distributed users use single-user editors e.g. MS Word and GVim to edit a textual document collaboratively. To synchronize, inputs to each editor are broadcast to all sites and somehow executed there. If all editors are the same, then it is equivalent to a replicated application-sharing system, in which simply replay input events, as discussed in Lauwers et al [19], would suffice. However, in presence of heterogeneity, a verbatim replay of the same input stream at all sites does not make sense at all. The same semantic operation, e.g. to insert a character, is often implemented by different input sequences in different editors. Therefore an intelligent mechanism must be present in the application-sharing infrastructure to "understand" the user operations in one application first and then translate them into equivalent input sequences before they are replayed to other editors. Apparently this cannot be achieved without semantic knowledge about the editors and their operations.

In the subsections that follow, we explain how exactly our system has been implemented to share and interoperate heterogeneous single-user editors (MS Word and GVim) that belong to the same family of text editors. Specifically, we will give an overall picture of the system, including system architecture; event capture, replay, and translation; consistency maintenance and concurrent independent work support; session control and awareness control leveraging existing file systems; spontaneous interaction and late-comer accommodation; and mixed fonts editing so that the main feature of MS Word can be kept.

2.1 System Architecture

Heterogeneous application sharing necessarily implies a distributed architecture. Figure 1 shows the ICT system architecture. An ICT agent is replicated and interposed at each site between the single-user applications (GVim and MS Word) and the windowing system (MS Windows). We assume a reliable communication bus which connects all the ICT agents in a group editing session for them to exchange messages. The ICT agent is divided into a number of communicating components.

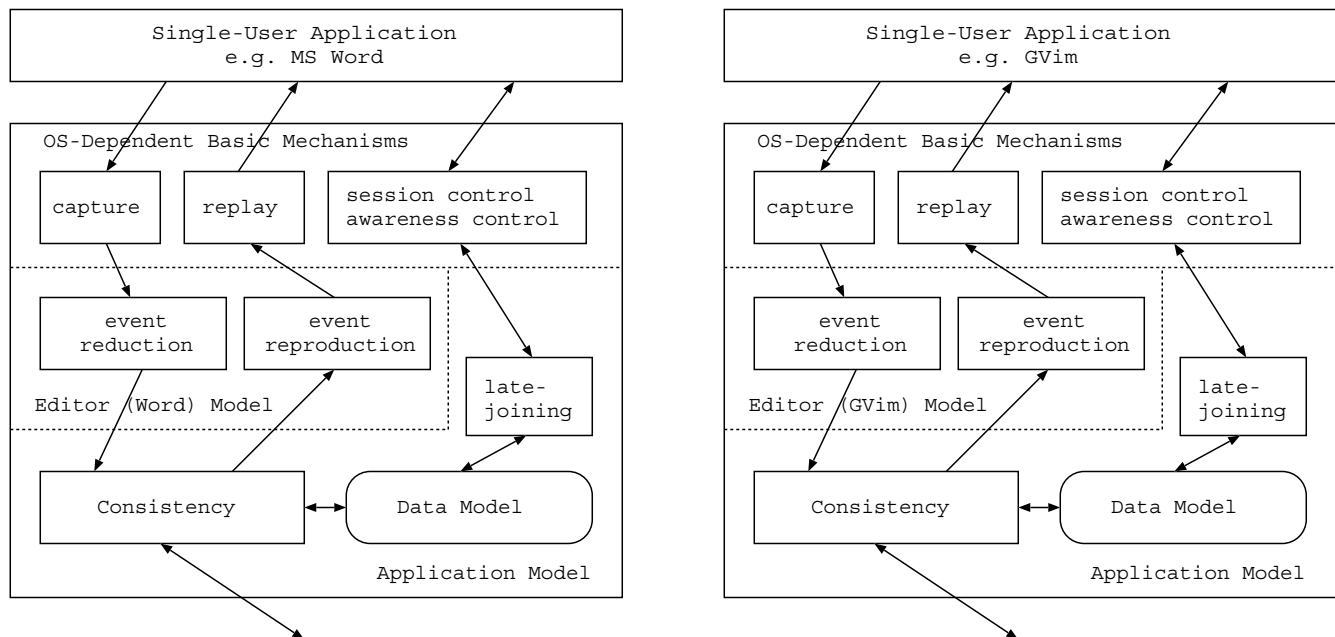


Figure 1: ICT System architecture: An ICT agent runs at each site to interoperate heterogeneous single-user editors.

As shown in Figure 1, some components are operating system dependent mechanisms, such as event capture and replay, session control and awareness control. Some components are common to the application family, such as those to support consistency maintenance and late-joining. Some are specific to particular editors, such as rules for event reduction and reproduction. The working of those components is discussed briefly next and then in more detail in the subsequent subsections.

At each site, the event capture mechanism intercepts user inputs (e.g. keyboard or mouse events) to the editor being shared. The event replay mechanism, on the other hand, inserts a given sequence of events to the message queue of its target application so that the events are executed by the application as if they are input by the user. In absence of heterogeneity, simply capturing inputs to one replica of an application and replaying to another would basically emulate a conventional replicated application sharing system.

To address heterogeneous application sharing, however, application knowledge must be supplied to enable semantic translation of events. As such, the event reduction component helps the agent to “understand” the meaning of a user operation by reducing the input event sequence into a higher level description. The event reproduction component, on the other hand, translates the higher level description from one editor to a corresponding input sequence to another application. The event replay mechanism then replays the reproduced sequence to that application.

To maintain consistency among applications at all sites, a collaboration specific concurrency control component must be supplied. For example, if only one user is allowed to generate input at any time, then the concurrency control component will tell the agents of all non-active users to block local input and only the active user’s input is allowed to propagate. This is the so-called floor control in most existing application-sharing systems, e.g. MMConf [6] and Dialogo [20]. An optimistic policy, for example, operational transformation [24], on the other hand, may always allow the local operations to be executed immediately once generated and then strives to keep a consistent state across all sites.

Spontaneous sharing and late-joining have long been recognized as a key feature of fluid collaboration (e.g. [9]). Users do not have to foresee the sharing. They start with their favorite single-user applications as usual to perform individual work. When the need for sharing emerges, they activate the application sharing infrastructure and indicate which application(s) to share and for which task(s). A sharing session is triggered, for example, by a user through the session control interface, or by the event that one user opens a shared file that is already being edited by another user. If all users start with the same shared data state, then consistency maintenance in the rest of the session will rely on the event translation (reduction and reproduction) and concurrency control components.

If a user joins a running editing session late, the current state of the shared data must be transferred to the new user. In conventional systems, late-comer ac-

commodation is not efficient in general [19]. For example, event replay is time-consuming and direct state transfer demands special features from the application or the operating system, which are not always available. However, in ICT, application semantics can be explored in supplying a late-joining component to accommodate late comers more efficiently, as will be illustrated later in this paper.

Independent views, concurrent work, and group awareness are desirable features for effective collaboration and can be provided by relevant mechanisms. For example, users view and manipulate the shared data in ways implemented by their favorite editors. They work on different parts of the shared data independently as long as it is permitted by the concurrency control component. Group awareness mechanisms (e.g. telecarets, telepointers, and radar views) can be directed by the awareness component to promote mutual awareness. For example, a radar view may be generated at each site to show the position of each user relative to the shared data. As a basis, the users' editing or pointing positions must be tracked to make sense which part of the shared data each user is working on.

Groupware applications often share an information space and operate collaboratively on it [7]. Different single-user applications usually manipulate data in different ways. A common model of the shared data and operations is an important and necessary basis atop which other components are built, especially for heterogeneous application sharing. For example, a linear string is usually used in group editors [10]. The model must be designed so that the basic operations are efficient in manipulating the data and are easy to translate between the operations of any single-user application in question.

2.2 Event Capture, Replay, and Translation

MS Windows systems provide APIs to set user-provided hook functions which can be used to intercept user inputs to given application windows. There are also APIs for user programs to simulate user actions. Those APIs are the basis on which the event capture and replay mechanisms are implemented in ICT. Our hook function for capturing user inputs returns immediately after posting the intercepted message to another component of the ICT agent. The user application is not blocked or delayed by the ICT agent's event processing. Fast local response has been identified as an important principle in interactive systems [5].

Input events captured are low-level. For example, a user pressing 'A' on the keyboard generates an event sequence which includes "KEYDOWN A", "CHAR A", and "KEYUP A" in MS Windows. If the window systems and applications are the same, replaying those

event sequences at different sites would conceptually emulate replicated application-sharing systems such as Dialogo [19].

In the presence of heterogeneous applications, it becomes more complicated. For example, to insert a character 'A' at the current caret position in Word, the user simply types 'A'. However, to do the same thing in GVim, the user typically has to switch to input mode by typing a sequence, 'i', 'A', and then return to the command mode by pressing the "Esc" key. So if the same input sequence to one application is replayed to another, different application states will likely result. To solve this problem, the ICT agent first translates the event sequence into a high-level description of the user operation by an application-dependent event reduction component. This semantic operation is then delivered to the ICT agents of cooperating editors, where it is translated to an equivalent input sequence by an event reproduction component. Application semantics must be formalized in the event reduction and reproduction components.

2.3 Concurrent Work and Consistency

To maintain consistent states among functionally equivalent editors, we assume a shared data model. As in other (collaboration-aware) group editing systems such as Grove [10] and Reduce [27], we model the shared data as a linear character string. The most significant editing operations are insert and delete. We assume no knowledge about or control over the internal data structures of the shared applications under the general blackbox assumption of application-sharing systems. It is critical and challenging in ICT to keep consistency between the logical view (data model) and the physical view (application display and its internal data) at each site and to keep consistency between logic views across different sites.

The consistency problem is two-fold. First, ICT must track the caret position in each editor. Its corresponding position in the logical data model must be updated accordingly. This is the basis to maintain consistent logical and physical views. The problem is aggravated if characters in the editor are allowed to be in mixed fonts. In Li and Li [21] we present a simple machine learning algorithm to address the caret positioning problem. When the cursor is moved onto the face of a character bitmap, the built-in strategy of an editor adjusts it either before or after the character. The strategy is font-sensitive and differs between editors. Our algorithm discovers the caret positioning strategy of each editor for every font only by observation. The strategy is formalized in a knowledge base that is loaded by the ICT agent as an editor is invoked.

Second, the logical views must be consistent across dif-

ferent sites. To support concurrent work instead of the strict turn-taking protocols in traditional application-sharing systems [6] [19], we implemented the operational transformation algorithm GOTO by Sun et al [24] for concurrency control. This algorithm has proved effective for consistency maintenance in (collaboration-aware) group editors [25, 27]. In GOTO, local editing operations are always executed immediately. Remote operations are “transformed” before execution. No locking or backtracking is necessary. Thus multiple coauthors can work on their parts independently and concurrently without the need to take turns in editing and without being frequently blocked by collaborators.

Given these two levels of consistency, collaborators in ICT use their familiar editors such as GVim and Word to perform group editing. Each editor’s input sequences are reduced into semantic operations such as “insert character ‘A’ at position X” and “delete one character at position Y”. These operations are executed on the local data model immediately and then propagated to remote sites. A remote operation is transformed by the concurrency control component and then executed on the shared data model. The transformed operation is translated into an equivalent input sequence to the local editor by its event reproduction component. Then the input sequence takes effect on the target editor (its display and internal data structure) through the event replay mechanism.

Ideally, local and remote operations should not interfere with each other so that users can input independently and concurrently. Upon the arrival of each remote operation, we reflect it on the screen immediately and then move the caret back to the local user’s original input position. To reduce messaging traffic and the number of screen flickers incurred by event replay, we have implemented the string-wise operation transformation as proposed by Sun et al [25], in which consecutive operations can be merged. The users are able set appropriate timing and granularity policies for propagating local operations and rendering remote operations [22]. However, operation transformation algorithms assume editors start with a consistent initial state. Therefore in situations where spontaneous interactions and late-joining are to be supported, the editors’ must be brought to the same state first, as discussed in the next two subsections.

2.4 Session Control and Awareness

Now the problem is how to let the potential collaborators know of each other and decide which documents they want to edit together. We thus implemented session control and awareness mechanisms as shown in Figure 2, which leverage the Windows Explorer with some extra work. Certainly we did not assume access or modification to its source code.

MS Windows systems implement SMB as the default file sharing protocol. Each file in Windows is either a local file that is not shared, a local file that has been opened for sharing, or a shared file residing on a remote system. When a file is opened, an API function can be called to determine whether its parent directory is being shared and to retrieve the server address if it is. A file under a shared directory will be potentially browsed or edited by other users. Thus a shared directory provides a rendezvous for cooperative work.

Most existing operating systems to our knowledge do not coordinate concurrent accesses to files as groupware. If a file is opened simultaneously by several users, their writes will overlap each other and the user processes are not notified by other users’ changes. Approaches to modify the file (SMB or NFS) server, e.g. as in Placeless Documents [17], are only able to capture document-level operations such as open, close, and write, which is not sufficient to implement finer-grained, character- or string-level notifications for synchronous collaboration.

We address this problem by the following method. The ICT agent as shown in Figure 1 runs at each site as the user logs on. If the user opens any directory for sharing, an ICT daemon is started to manage sharing of files under that directory. A small application launcher is installed to replace the default one for certain file types, as shown in Figure 2. Currently as we are only able to interoperate MS Word (Wordpad and Notepad) and GVim, we assume the files of interest are with a “.txt” suffix. A supplementary scheme is later in this section to keep the formatting information of MS Word.

If a file is not under a shared directory, its default application is opened when the file icon is double-clicked. If a file is likely to be shared, our launcher will only invoke the registered applications (which we currently support) and at the same time tell the local ICT agent to notify the ICT daemon running on the server machine of that file. Since all clients (ICT agents) are registered with the ICT daemon, all parties will be notified by the daemon of the joining of a new collaborator: his login name, host address, the time his editor is launched, and the application he is using to edit the file.

As shown in Figure 2, several awareness mechanisms are provided for the user to be aware of sharing. First, the file icon is changed by the ICT agent after the ICT daemon indicates that the file is being opened (edited) by another user. For example, files a-e.txt as listed are of the same type. But a.txt and d.txt icons are augmented by a human head and a number “1”, indicating that they are currently being edited by one user respectively. Similarly the icon of file b.txt is augmented by a human and a number “2”. When the number of concurrent editors goes beyond two, e.g. file c.txt, the icon will

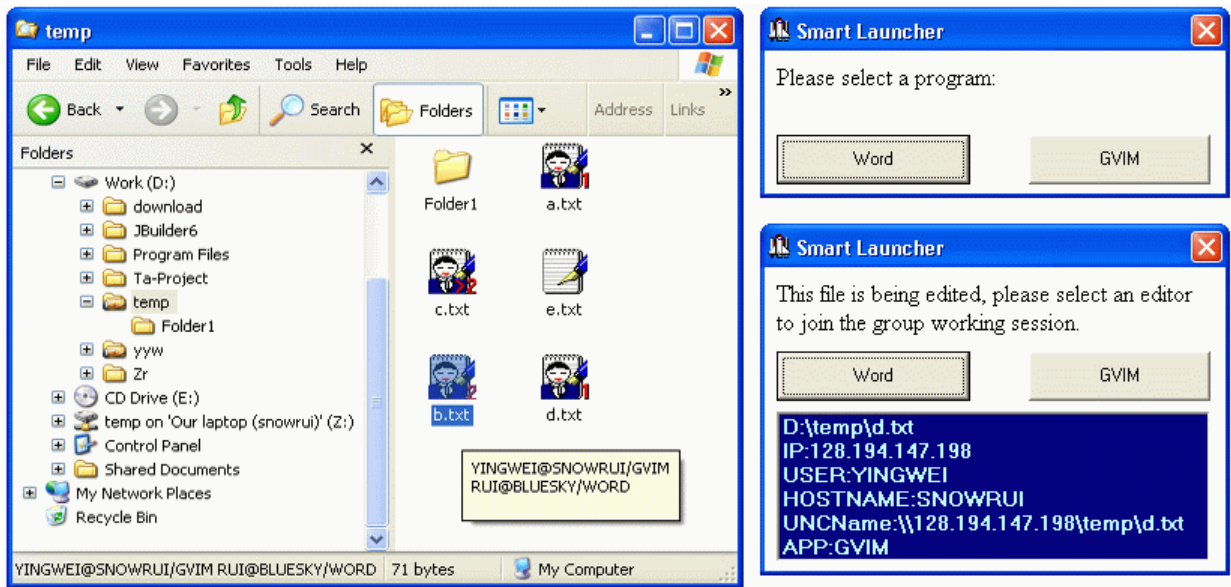


Figure 2: Leveraging the Windows Explorer for session and awareness control.

be augmented by “>2”. Second, the status bar in Windows Explorer displays a summary of the collaborators’ information when a file is pointed to or selected. Third, an extended infotip is displayed when a file has been pointed to for a few seconds. Fourth, when the smart launcher is activated, it prompts the user that the file is being edited and displays all the collaborators’ information.

2.5 Spontaneous Sharing and Late-Joining

Spontaneous sharing is implemented as follows. If the user is the first one to edit a shared file, the ICT agents and the daemon only collaborate to propagate awareness information. But the user’s editing actions will not be monitored by his ICT agent. The editor (GVim or Word) loads the file from the SMB server and everything goes as normal in Windows.

However, when a second user attempts to join an ongoing editing by another user, a major reconfiguration is triggered. The first user’s ICT agent will be instructed to capture the current state of his editor and transfer it over to the ICT daemon, which in turn transfers the content to the second user (the late-comer). Then the second user’s agent will generate a command to load this updated content of the shared file into his editor, to replace the old content which was originally obtained from the SMB server when the editor was launched. Because the SMB server is not aware of any updates in the file until an explicit write, the second editor’s original content is already out of date. At this point, both editors’ states are synchronized. The ICT agents of both parties are then instructed (by the ICT daemon) to capture events from the two editors respectively. The

concurrency control algorithm (GOTO) now begins to execute.

The ICT daemon replicates the concurrency control and the shared data model from the ICT agent, as shown in Figure 1. All ICT agents communicate with each other through the ICT daemon. User actions are translated into semantic operations on the shared data model within each ICT agent. Then the semantic operations are sent to the ICT daemon, executed there, and then propagated to other ICT agents. Presumably all replicas of the data model are synchronized by the dOPT algorithm. When other later comers join the ongoing group editing session, the latest state can be obtained from the ICT daemon’s data model.

Since the shared data model is but a linear character string, late-comer accommodation can be extremely efficient in ICT. To capture the current state of an editor is equivalent to making a copy of the data model as a text string from its ICT agent. To load the data model into an editor is equivalent to pasting the text string into the editor. Copy-paste is just a command implemented on any popular editor (GVim or Word) which is easily reproduced by the ICT agent. Pasting a string is significantly more efficient than replaying the operation history, which effectively inserts the string character by character. Li and Li [21] presents additional approaches to late-comer accommodation in ICT, which are in general more efficient than traditional approaches [19].

2.6 Keeping Special Formattings in MS Word

Unlike GVim which is a pure text editor, MS Word is a formatting tool at the same time. Different fonts can be mixed together in a Word document. To support group

editing with Word and GVim, we must support mixed fonts editing, which has the following complications.

First, the shared data should be modeled as a link list instead of a string. Each node of the link should include information such as the font type and size of the character. From this information the height and width of each character can be obtained by calling window APIs. When the cursor position is changed, the engine calculates on the face of which character in which line the cursor is positioned and then retrieves from the knowledge base the rule to determine the caret position.

Second, mixed fonts editing complicates late-comer accommodation. If fonts are not considered, maintaining one consistent copy of the shared data is sufficient. With mixed fonts, a new problem arises if the users do not want to lose the formatting information. Our current strategy is to use auxiliary files to keep formatting information pertinent to each type of involved editors, in addition to the common data structure. At the same time an operation log is maintained.

Consider the following scenario for example: Users A, B, and C edit the same document at the same time. A and B use GVim and C uses MS Word. The common string is maintained in file `x.txt` which is “almost” up-to-date (with the editing buffer). An auxiliary file `x.doc` maintains the formattings of user C. Now C quits but the group editing continues. (So ICT stops updating file `x.doc`.) After a period of time C joins with MS Word again. Certainly C wishes to start with his previous state in `x.doc` and catch up by replaying the part of the operations since he left. If the operation replaying takes too long and `x.doc` is relatively small, C may be given an option to catch up using the current content of `x.txt` or the editing buffer instead, which means a loss of his previous formattings. By comparison, a strategy that only considers operation log replay may take too long to accommodate a late comer. A strategy that only transmits the common data structure, on the other hand, will lose the formattings.

3 DISCUSSION AND APPLICATION SCENARIOS

Scalability is not of our concern at least for the present time. An ICT daemon process monitors a shared directory and corresponds to a SMB server. An ICT agent corresponds to a SMB client. When multiple files are being shared simultaneously, those processes can duplicate themselves so that one suite of processes (daemon, agent, editor) matches one shared file. Every host can run the agent (client) and the daemon (server) at the same time in a peer-to-peer fashion. Given the rich resources of today’s PCs or workstations, no host in ICT will likely become a performance bottleneck. Even in situations where many users share the same directory, it is not likely that a big number of them will edit the

same file at the same time.

The reason why synchronous group editing rarely happens can be multi-fold. It may be that off-the-shelf editors (e.g. MS Word and GVim) do not support synchronous group editing and that the document management systems (e.g. Samba and NFS) provide insufficient support. It could also be that the state-of-the-art group editors (e.g. Reduce[27]) are not as easy to use as familiar single-user products. However, it is true that many people need to write documents together while social protocols play the main coordination role. With ICT, we hope to make coauthoring a much easier job and wish to attract more end users to evaluate group editing technologies such as the operation transformation algorithms [24] and ICT itself.

The application knowledge in ICT can contain nontrivial semantic information for more intelligent event translation between different editors. For example, when two users use Word and GVim, respectively, to coauthor a paper, ICT can do more than verbatim operation translation. Inserting an image in the Word version of the paper can be translated into a sequence of graphics commands to include an `.eps` file that is converted from the `.jpeg` or `.gif` image in the latex version which is being edited in GVim. A citation in the latex file, on the other hand, is translated into an equivalent operation of referencing in Word after the inclusion of the bibliography item. Of course such semantic translation may only make sense at certain points of the editing process, e.g., after a citation operation is detected to have been completed. As noted by Baecker et al [3], an actual authoring process can intermingle a number of synchronous and asynchronous collaboration protocols. However, support of such sophisticated semantic translations and protocols goes beyond the scope of this paper. We are currently investigating into those problems and will report our achievements in future publications.

In addition to its more obvious application in sharing familiar single-user editors for cooperative editing, we wish to explore other application areas of ICT. The UbiData project at University of Florida [18] recently reveals an interesting application of ICT: On computing devices with limited capabilities, such as PDAs, the user may not be able to manipulate full MS Word documents. If an ICT-like mechanism is available, then the user can use a simple text editor running on his PDA to manipulate a Word document maintained by his desktop PC, for example, when he is traveling.

4 RELATED WORK

Most existing application-sharing systems are built on X Windows and adopt a centralized architecture such as XTV [1] and SharedX [11]. All known work with an element of heterogeneity has been on centralized ap-

plication sharing atop heterogeneous window systems. In general they are not able to share and interoperate heterogeneous applications, as compared to ICT. Greenberg [12] raised the issue of distributing application output to incompatible displays. A centralized view manager was proposed to have the application write to a “virtual terminal” from where the output stream is translated so that the same view appears across different terminals. Wolf et al [26] discussed how to share a centrally-executed application on different graphics and window systems. Translation of application output, user input, and window management was addressed. Alternative strategies of distributing various system components are discussed for optimal performance.

Hao et al [15] allows the user to dynamically group applications into various contexts and perform simultaneous semantic operations. A semantic operation can be defined by the user to group a sequence of low-level input events. When a semantic operation is performed, it is translated into different input sequences that are replayed on all the applications in the same context. This work resembles ours in the translation of semantic operations into application-specific input sequences. However, it serves different purposes and falls short in generality and flexibility.

Placeless Documents [17] inadvertently supports heterogeneous application sharing to some degree. It implements a middleware layer that sits between the single-user applications and the document repositories. All document-level operations performed by the applications, including read, write, move, and delete, are monitored. The middleware maintains active properties which are user-supplied code segments. When documents are accessed, active properties are triggered by the middleware to make notifications, to perform the operations as proposed, or to veto them. In this way, workflow processes are implemented which are transparent to the single-user tools used to access the documents, without the need to introduce a new workflow tool. Since the operations monitored are document-level, it does not make any difference to the middleware from which applications they are generated. So heterogeneous single-user applications are allowed. Because the operations of interest are not application-specific, it does not seem necessary for the active properties (which implement collaboration semantics) to include application semantics. However, the operations monitored in Placeless Documents are coarse-grained and insufficient to implement tightly-coupled, synchronous collaboration, which our approach and other application-sharing systems are able to support.

There have been numerous research efforts in interoperating heterogeneous groupware systems and applications. For example, the DISCIPLE and Manifold frame-

work by Marsic [23] is built on Java and XML and provides some features to handle heterogeneity. However, its purpose is to match the disparities in computing and communication capabilities of the participants, not to transparently share and interoperate heterogeneous single-user applications.

Dewan and Sharma [8] addressed the problem of inter-operating collaboration-aware groupware systems along the dimensions of architecture, concurrency control policy, and coupling policy. It proposed to build a bridge (or “dummy user”) between two systems that are to be interoperated. The bridge is programmed so that it is able to communicate with both sides and perform protocols translation. It has been noted in Dewan and Sharma [8] that it may not be always possible to inter-operate two systems whose collaboration protocols are radically incompatible. However, their work assumes source code and internal knowledge of the groupware applications to be interoperated, which is different from the strict blackbox assumption we make in our work.

DistEdit [16] is a groupware toolkit which provides a library of functions so that existing single-user editors can be converted to group editors. In this way, heterogeneous editing styles (e.g. MS Word and GVim) can be accommodated in the same group editor that results. However, our approach is fundamentally different: In DistEdit, the source code of the original single-user editors must be obtained and extended with the provided collaboration-aware functions, which is not always possible in practice. Our approach, on the other hand, strives to convert single-user editors in a collaboration-transparent way, i.e., heterogeneous single-user editors are shared and interoperated without assuming modification to their source code.

Our earlier work [21] attempts to establish the general ICT framework with the focus on generality while many important details must be left out. This paper differs significantly from Li and Li [21], particularly in its group editing orientation and introducing our latest development of key mechanisms for session control, awareness control, and spontaneous sharing.

5 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we motivated our investigation into a novel application-sharing approach which is able to share and interoperate familiar single-user editors for collaborative editing activities. Compared to the state-of-the-art group editors such as Reduce [27], coauthors using our system are able to bring in their favorite editors that are allowed to be heterogeneous, which presumably can increase group and individual productivity. Compared to existing application-sharing systems such as XTV [1] and Dialogo [19], our work is uniquely able to interoperate heterogeneous single-user applications and

is more effective in supporting concurrent work and later-comer accommodation. Besides, our approach is general because we work strictly under the blackbox assumption of application sharing systems, i.e., we do not assume access or modification to the source code.

This paper discussed our prototype system in appropriate detail, including the system architecture, the event capture and replay mechanisms, the support of concurrent work and spontaneous sharing, and the session control and awareness mechanisms that are built atop traditional file system tools without modification to their source code. Currently our system runs on MS Windows platforms and is able to interoperate popular editors such as GVim and MS Word for pure text editing. Due to the tremendous amount of system hacking required in this exploratory prototyping process, we are not able to handle figures editing and complicated formattings yet. We plan to address those problems and at the same time generalize our work to other types of applications and other popular operating systems (e.g. Linux) as well in future publications. However, the presented work in its current form is already sufficient in serving the purposes of this paper and as a proof of concepts. More importantly, it makes significant technical contributions as argued above.

According to Dewan et al [7], a wide variety of groupware applications can be generalized as multi-user editors. Each user interacts with the application interface which generates text, graphics, and/or multimedia editing commands to change the state of the shared data objects. Towards this end, the ICT framework as presented in this paper has the potential to be extended to share and interoperate any single-user applications that the collaborators are familiar with. We also plan to explore this potential in future work.

ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation CAREER award #0133871. We thank the anonymous reviewers for their insightful and constructive comments.

REFERENCES

1. H. Abdel-Wahab and M. Feit. XTV: A framework for sharing x window clients in remote synchronous collaboration. In *Proceedings of IEEE Tricomm '91*, pages 159–167, Chapel Hill, NC, April 1991.
2. S. R. Ahuja, J. Ensor, and D. Horn. The rapport multimedia conferencing system. In *Proceedings of ACM Conference on Office Information Systems*, pages 1–8, Palo Alto, March 1988.
3. Ronald M. Baecker, Dimitrios Nastos, Ilona R. Posner, and Kelly L. Mawby. The user-centered iterative design of collaborative writing software. In *Proceedings of InterCHI'93*, pages 309–405, April 1993.
4. James "Bo" Begole, Mary Beth Rosson, and Clifford A. Shaffer. Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. *ACM Transactions on Computer-Human Interaction*, 6(2):95–132, June 1999.
5. Sumeer Bhola, Guruduth Banavar, and Mustaque Ahamad. Responsiveness and consistency trade-offs in interactive groupware. In *Proceedings of ACM CSCW'98 Conference*, pages 79–88, Seattle, November 1998.
6. Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of ACM CSCW'90 Conference on Computer-Supported Cooperative Work*, pages 329–342, Los Angeles, California, 1990.
7. Prasun Dewan, Rajiv Choudhary, and Honghai Shen. An editing-based characterization of the design space of collaborative applications. *Journal of Organizational Computing*, 4(3):219–240, 1994.
8. Prasun Dewan and Anshu Sharma. An experiment in interoperating heterogeneous collaborative systems. In *Proceedings of European CSCW Conference*, pages 371–391, 1999.
9. W. Keith Edwards. Policies and roles in collaborative applications. In *ACM CSCW'96 Proceedings*, pages 11–20, 1996.
10. Clarence A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *ACM SIGMOD'89 proceedings*, pages 399–407, Portland Oregon, 1989.
11. Daniel Garfinkel, Bruce Welti, and Thomas Yip. HP SharedX: A tool for real-time collaboration. *Hewlett-Packard Journal*, 45(4):23–36, April 1994.
12. Saul Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of ACM Conference on office information systems*, pages 227–237, 1990.
13. Jonathan Grudin. Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW'88)*, pages 85–93, Portland, Oregon, September 1988.
14. Jonathan Grudin. Eight challenges for groupware developers. *Communications of the ACM*, 37(1):92–105, 1994.

15. Ming C. Hao, Alan H. Karp, and Daniel Garfinkel. Collaborative computing: A multi-client multi-server environment. In *Proceedings of ACM Conference on Organizational Computing Systems*, pages 206–213, Milpitas, CA, 1995.
16. Michael J. Knister and Atul Prakash. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of ACM CSCW'90 Conference on Computer Supported Cooperative Work*, pages 343–355, Los Angeles, California, October 1990.
17. Anthony LaMarca, W. Keith Edwards, Paul Dourish, John Lamping, Ian Smith, and Jim Thornton. Taking the work out of workflow: Mechanisms for document-centered collaboration. In *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work (ECSCW'99)*, pages 1–20, Copenhagen, Denmark, September 1999.
18. Mike Lanham, Ajay Kang, Joachim Hammer, and Abdelsalam Helal. Ubidata: A blend of event- and data-based synchronization methods to support ubiquitous data access in mobile environments. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS 2002)*, 2002.
19. J. Chris Lauwers, Thomas A. Joseph, Keith A. Lantz, and Allyn L. Romanow. Replicated architectures for shared window systems: A critique. In *Proceedings of ACM OIS'90 Conference on Organization Information Systems*, pages 249–260, 1990.
20. J. Chris Lauwers and Keith A. Lantz. Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, pages 303–311, 1990.
21. Du Li and Rui Li. Transparent sharing and inter-operation of heterogeneous single-user applications. In *Proceedings of the ACM CSCW'02 Conference on Computer-Supported Cooperative Work*, November 2002. To appear.
22. Du Li, Chengzheng Sun, Limin Zhou, and Richard R. Muntz. Operation propagation in real-time group editors. *IEEE Multimedia Special Issue on Multimedia Computer Supported Cooperative Work*, 7(4):55–61, 2000.
23. Ivan Marsic. An architecture for heterogeneous groupware applications. In *Proceedings of the 23rd IEEE/ACM International Conference on Software Engineering (ICSE 2001)*, pages 475–484, Toronto, Canada, May 2001.
24. Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of ACM CSCW'98 Conference*, pages 59–68, Seattle, Washington, December 1998.
25. Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, March 1998.
26. Klaus H. Wolf, Konrad Froitzheim, and Peter Schulthess. Multimedia application sharing in a heterogeneous environment. In *ACM multimedia'95 conference*, pages 57–64, San Francisco, CA, November 1995.
27. Yun Yang, Chengzheng Sun, Yanchun Zhang, and Xiaohua Jia. Real-time cooperative editing on the internet. *IEEE Internet Computing*, 4(3):18–25, 2000.