Group Editor Using Graphical Operational Transformation

Zameer Fatima, Ankit Agarwal, Gaurav Gupta and Mayank Sharma

1. Lecturer, MAIT, 2,3,4. MAIT, GGSIPU

1zameer.fatima@gmail.com, 2ankit.agarwal63@yahoo.com, 3guptagaurav01@gmail.com and 4mayankmait26@gmail.com

ABSTRACT

Collaborative editing enables multiple users who reside remotely to share and edit some documents at the same time. It is fundamentally based on operational transformation which adjusts the position of operation according to the transformed execution order. For a last decade, many researches have been performed in this area and the correctness and possibility of operational transformation have been proved.

In this paper, a novel operation transformation framework is developed to overcome the weakness of existing system. Based on a concept called “operation effects relation,” in which two criteria are defined, causality preservation and operation effects relation preservation and establishes a novel operation transformation framework for developing OT algorithms and proving their correctness for graphical data sharing and editing. In the existing system there is a novel solution for editing and sharing of textual data only. But this paper proves the correctness of textual as well as graphical data sharing and editing using graphical operational transformation algorithm.

KEYWORDS

Graphical Operational transformation algorithm, G.O.T algorithm.

1. INTRODUCTION

Collaborative editing system is which enables multiple users who reside remotely to share and edit some text / graphical / multimedia documents. This is an application area of Computer-supported cooperative work and it has been known for enhancing the efficiency and productivity of jobs and solving the physical limitation with virtualization.

For collaborative editing, the system should ensure the consistency of the shared documents and supports the rapid processing of documents editing requests. In other words, it should be the same as one user edits some documents alone. To satisfying these kinds of real limitation, various novel algorithms were proposed such as locking, operational transformation.

In the locking approach writing documents at the specific part is allowed only for one user at one moment and reading documents is allowed for any user. Therefore, it cannot support real collaborative environment which can provide multiple users to edit some documents at the same time. Even though some locking algorithms try to support collaborative editing with using small-part locking, it cannot support totally real-time collaboration.

As for operational transformation, it ensures that the effect of executing a group of concurrent operations is the same as if the operations were executed in the same total order at all sites. By transforming the order of operation effectively, it ensures the consistency of the shared documents. Besides, it ensures the short response time for multiple user’s request because operational transformation is based on the replica-based approach.

In spite of the novelty of operational transformation, there are some blind sides. It is not proposed without any consideration of human-centric viewpoint and practical usage of collaborative editor. As for technical viewpoint, as the size of document and the number of users grows, the number of invoked operation increase. Therefore, the complexity of operational transformation is bigger and the scalability is limited due to the huge size of operational transformation. In the case of human-centric viewpoint, the most important part of the shared document is at current caret. However, to achieve the consistency of replica-based shared document system with the existing operational transformation, it is enforced to process all invoked operations from other users but unrelated to current caret.

Through the deliberate experience, we are capable of figuring out this problem. We have solved this problem by borrowing up the famous concepts like priority queue and cache in computer system. In other words, multiple users have their own history buffer for operational transformation and accept instantly the invoked operation related to their current caret in the shared document and postpone other operation from low-priority buffer. It is a kind of classified history buffer management system which conform the priority of operation for transformation.

In the following section, in order to describe operational transformation, the basic requirements for a good GRAPHICAL OT algorithm is introduced. Next, section 3 describes G.O.T algorithm. Section 4 explains how to implement the architecture and algorithm of collaborative editor. This paper concludes with a discussion of the implications of this work to the development of collaborative computing, and proposals for future work.

2. DESIGN REQUIREMENTS

In this section we proposed and defined some of the underlying design requirements that we felt are important for a real-time
conferencing application. We do not believe this list is complete, nor every item be necessary for every groupware application. However, we do feel the requirements listed must apply to a wide range of real time conferencing applications, and that they provide a reasonable starting point for discussion. Requirements listed in the first section are human-centered, and those in the second are programmer-centered.

A. Human-centered design requirements

Supporting multi-user actions over a visual work surface. We believe that there are several general but critical activities that people do over a shared work surface, regardless of its contents. Two of these, gesturing and annotation, are described below.

a) Provide support for gesturing. Researchers at Xerox PARC studied the use of conventional drawing surfaces by small groups [26]. A critical finding was that participants frequently gesture over the drawing surface: to enact ideas, to signal turn-taking, to focus the attention of the group, and to reference objects on the surface. Several recent computer systems emulating group drawing surfaces support gesturing with multiple cursors appearing on all displays, and their usability studies confirm the ubiquity of gesturing [13,2,15]. We believe that gesturing can enhance communication in many diverse types of conferences, and should be supported at the application level.

b) Provide support for graphical annotation. The Xerox studies also noticed many instances of annotations made to existing drawings, serving both as gestures (eg underlining text while saying “this one here”) and as meta-level notes. Several systems now incorporate graphical annotations of their objects. FREESTYLE users, for example, can verbally and graphically annotate bitmap snapshots; the results can be mailed to others who can then play back the transcript[9]. Both PROOFMARKS in vmacs, and the commercial MARKUP application allow comments and markup symbols to be added to written documents [18]. As with gesturing, we believe that real-time group graphical annotation over a work surface is useful in many situations and should be supported by the application. Structuring group processes during a meeting Some researchers believe that groupware should impose a social model of interaction on the group. This is an explicit attempt, based on management theory, to provide methods for keeping the group on task, enforcing roles and commitments, and making the group more efficient and productive. There is certainly controversy between those who believe that social protocol should be determined only by the group members (eg, by the software and somewhere in between). We believe that some group process primitives should be provided by the application, accommodating groupware that wishes to control meeting structure. The list below discusses only a few group process requirements.

c) Provide various floor control policies. Floor control or turn-taking mechanisms provide a way to mediate access to shared work items. Lauwers[19] and Greenberg[14] recommend that systems should “support a broad range of [floor control] policies” to suit the users' needs. Systems such as SHAREDX and ASPECTS (from Group Technologies) support a few different policies, while SHARE[14] strives to provide complete flexibility. Floor control can be important in many situations, such as shared screens allowing only serial interaction, or systems following strict interaction models, such as a teacher controlling which students can access the work surface.

d) Support different registration methods. Another part of group process controls who is allowed to join the meeting. For some meetings, anyone may be allowed to join. For others only a select group can participate, or perhaps new users must be “sponsored” by an existing user. Sometimes more spontaneous creation of conferences is desired while other situations require a central facilitator to handle registration[24]. Applications should provide the flexibility to support any reasonable registration process.

e) Support latecomers to the conference. A consequence of spontaneous conferences is that all users may not join the conference at the start. Provisions should exist allowing newcomers to join at any time, as well as allowing existing members to leave. Strategies must also be supported to assist the newcomers in “getting up to speed.” This may involve simply sending the current conference state to the new user[15] or providing summary information on how the conference has progressed over its lifetime. Integration with conventional ways of doing work. Groupware should not impose a barrier between “individual” and “group” ways of working. For example, the system should provide group members with ready access to their individual work, and allow them to import it to a conference. Additionally, all normal communication channels (eg telephone, email) should be readily available.

f) Integrate other forms of communication. Voice communication is an important factor in most conferences[4] and, given the ubiquity of telephones, we have assumed that a voice channel is available. While many real-time conferencing systems assume a voice channel is present, they do not explicitly support creating voice links. Ideally, there should be a mechanism in the conferencing system to establish voice conferencing — perhaps automatically when a data conference is started. Similar arguments hold for other channels that may be available such as video links.

g) Allow use of single-user applications. Soon, most computer applications may be designed to support multiple users. Unfortunately, most of today's applications support single users only. There are several reasons why single-user programs should be available in multi-user conference settings. Groupware counterparts to single-user programs may not exist; a person's work may be accessible only through a particular application; people are skilled on particular applications. Conference users should be able to view and interact with single-user systems through shared screen or shared windows eg SHARE[12] and SHAREDX[10]. The application should provide shared windows or the means to incorporate other shared window systems.
B. Programmer-centered design requirements

Technical support of multiple and distributed processes. Most groupware, especially for geographically distributed conferences, require an architecture where multiple (perhaps distributed) processes can communicate with each other. While most operating systems provide process control and inter-process communication, the programmer's job of initiating, maintaining and tearing down processes and their communication channels is a tedious one. As well, state information about sessions may need to survive beyond the lifetime of a single process or meeting.

h) Provide processes for basic conference management. Groupware applications must oversee all conference management, which include activities such as participant registration, initiation and teardown of meeting processes, communications, and so on. Groupware applications have placed much emphasis in providing capabilities for conference management e.g. LIZA[11], CONFERENCE TOOLKIT[3], and MMCONF[5]. We too believe that the basic run-time infrastructure for conference management must be supplied by the application.

i) Provide a robust communications infrastructure. Any groupware application must provide the communications facilities on which to build conferencing components. At the very least, it must be possible for any process to send messages to specific processes owned by conference users, and it is preferable if a multi-cast facility is available to broadcast a single message to all. Of course, the communications demands depends heavily on the way the process architecture is determined. The trade-offs between centralized and replicated architectures are well-documented, with centralized architectures simplifying concurrency control and replicated architectures being more efficient and robust to machine failure.

j) Provide support for persistent sessions. Often computer conferences spans more than a single session, for example decision support meetings [24]. It is desirable to maintain session state information over the full duration of the conference. There should exist a general mechanism whereby conference objects can be made persistent. Technical support of a graphics model. A visual work surface should require graphical and textual primitives. Yet shared graphics require several capabilities that are not present in single user systems.

k) Provide primitives to a shared graphics library. Many groupware applications require graphical library primitives for creating multi-user objects such as shared lines, rectangles, circles and text. Greenberg, Roseman et al's discussion of GROUPDRAW[15] describes technical issues of a shared object-oriented drawing package, and provides their design of an abstract drawing object that can be sub-classed into concrete objects such as shared lines. Similarly, the fine-grained editing of simple graphics and text objects in Bier and Freeman's MMM system gives insight into how shared objects should behave[1]. Similar extendible graphics libraries should be provided by applications, so that programmers can easily create shared interactive graphical objects on the display.

l) Provide object concurrency control. Many groupware conferencing systems support access to some type of shared object, be it structured graphics or a text buffer. Concurrency control is often needed to mediate access to the object, for example, two people trying to manipulate the same point on a line. In fact several concurrency schemes have already been implemented in groupware applications[18,11]. Concurrency can be achieved through simple locking, transaction mechanisms, or numerous other schemes[8]. In addition, the degree of concurrency and access to shared objects can be specified through the notion of flexible coupling[6].

m) Separate the view of an object from its underlying representation. Many single-user graphical systems separate the properties of an object from its view on the screen. Patterson argues that this separation is critical in groupware[22], and that abstractions should be used to create an interface-independent representation of data. As a consequence, users can have multiple perspectives on the same data.

3. GRAPHICAL OT ALGORITHM

Collaborative editor could be possible effectively in the presence of operational transformation. With an algorithmic viewpoint, a few assumptions should be premised as following.

1. Optimistic Content Delivery - The invoked operations are ultimately delivered at each site even though its arrival may be delayed.

2. Operation Generation Model - Every operation is processed as following five steps.

   - Generated at one site
   - Broadcasted to other sites
   - Rejected by other sites and recorded into executed buffer
   - Executed on other sites
   - Recorded to history buffer at each site

3. State vector timestamping - Every operation contains its own state vector which is recorded at the moment of generation.

4. Total ordering relation - Given two operations 01 and 02, generated at sites i and j and times-tamped by SVo, and SVo2, respectively, then 01 => 02, iff it conforms Definition4.

5. Operation Composition - Every operation is composed as follows.

Op := (OPtype, OPposition, OPtimestate-vector, OpSite-index)

Point 1 describes that no operation missing happens in GOT. In other words, network environment in GOT supports atomic reliable broadcasting and the loss of operation delivery could not possible. In the case of point 2, its life-cycle model of operation in GOT. This model is assumed to discuss operational transformation effectively.

Both of point 3 and 4 is devised for causality preservation. As we mentioned before, time-stamping vector may be a nice choice for devise an algorithm and implement a program. Finally, point 5 states the composition of operation which isused in GOT. OTtype is ooperation like insert, delete. And oPosition is a position of operation. OPtimestate-vector is a
time-stamp vector which contains the originating site’s state vector. 0Psite-index is a site’s identifier of operation.

4. BUILDING AN EXAMPLE APPLICATION
The steps include specifying the application-specific graphical presentation and interaction, initializing objects to send and receive application messages, selecting an application Conference object, and initializing a Coordinator object. As an illustration, we described the construction of a simple multi-user freehand sketching program using the multiple cursor overlay. The interface is similar to GROUPSKETCH[13] multiple cursors are always visible, any user can draw at any time, and fine-grained actions are immediately visible on all displays.

A. Graphical presentation
The application needs one or more INTER-VIEWS glyphs to manage the graphical presentation and user interaction aspects of the interface, as well as any internal data structures. For a freehand sketching application, this involves creating a glyph holding a bitmap, and providing tools (pencil and eraser) that respond to mouse events for changing the bitmap. To incorporate the cursor overlay, the bitmap glyph is “composed within” the cursor glyph.

Designing this part of the sketching application is comparable to designing a single user version of the group application. There are some general conventions that are helpful to follow, such as separating event handling (cause) and the result of events (effect) into different routines. This facilitates use of common routines for local and remote invocations.

B. Messaging objects
Writer and Reader objects are used to send and receive application-specific messages. The routines in Writer objects are invoked as a result of local actions, for example, transmitting coordinates of a drawn line segment to the other replicated applications. Callbacks in the Reader objects interpret these messages, usually calling routines in the graphical presentation object to handle requests. In the example, the Reader instructs the sketchpad glyph to draw the line specified in the message. The standard objects must be initialized to include the required callbacks.

C. Coordinator
The Coordinator connects the registration mechanism (via the Registrar Client) to the application Conference objects running as separate processes. Available conference types must be specified to the Coordinator, using the standard X resource mechanisms (ie..XDefaults).

D. Application Conference
The application Conference maintains communications channels with other distributed applications. In our example, the generic Group application conference has sufficient functionality. The main program instantiates this object, and “attaches” to it the bitmap glyph described earlier, so that the glyph can send and receive messages. The Conference is notified when users join or leave. Routines in the base Conference class manage the low-level socket connections between users. However, other classes may be notified when new users join or leave, to manipulate application data structures maintained for each conference user. For example, the cursor overlay uses this information to add or remove cursors as users join or leave.

FUTURE WORK
The work presented here should be seen as an initial attempt to formalize the design and implementation of general groupware conferencing applications. The design requirements emphasize the important abstractions needed in real-time CSCW applications, and provide a basis for generalizing existing application or toolkit features. The three strategies presented — a run-time process and communication architecture, overlays and flexible policies — should be seen as general strategies that can be used to implement certain design features. It is expected that further design principles and strategies will evolve.

GROUP EDITER has proven to be a flexible platform for testing our ideas. We have already built prototype drawing programs and shared terminals, and we will be constructing more elaborate and robust applications shortly. Currently, several of the design principles have not yet been embodied in the application. Our immediate plans are to address the concurrency control issues, building a layer of support for generic shared graphical objects, that follow ideas presented in[15]. Ideally, a framework for building domain-specific group graphical editors could be created, drawn from the ideas in Unidraw[19].

Some other work has focussed on blurring the distinction between synchronous and asynchronous groupware, by providing a system that determines the appropriate means of conferencing (synchronous or asynchronous) and makes available various communication channels (text, voice, video). The choices offered the user depend on environmental information (ie who is currently around, what communication channels are available). The registration mechanisms in GROUP EDITER provide the flexibility to implement such a scheme.

REFERENCES


David Sun, Steven Xia, Chengzheng Sun, and David Chen, "Operational Transformation for Collaborative Word Processing", Proceedings of International Conference on Computer Supported Cooperative Work, November, 2004


C.A Ellis and S.J. Gibbs, "Concurrency control in groupware systems", Proceeding of ACM SIGMOD conference on Management of Data, June, 1989

M.Knister and A. Prakash, "Issues in the design of a toolkit for supporting multiple group editors", Journal of the Usenix Association, page 135-166, 1993


Greenberg, S., Roseman, M., Webster, D., and Bohnet, R. Human and technical factors of distributed group drawing tools. In Interacting with Computers (Special Edition on CSCW, in press).


Computer-Supported Cooperative Work (CSCW ’90),
Oct. 7–10 1990.