Distributed Concurrency Control in Real-time Cooperative Editing Systems

C. Sun[+] Y. Yang[‡] Y. Zhang[†] D. Chen[+]

[+]School of Computing & Information Technology
Griffith University, Brisbane, Qld 4111, Australia

[‡] School of Computing & Mathematics
Deakin University, Geelong, Vic 3217, Australia

[†]Department of Mathematics & Computing
The University of Southern Queensland, Toowoomba, Qld 4350, Australia

Email: {C.Sun, D.Chen}@cit.gu.edu.au, yun@dstc.edu.au, yan@usq.edu.au

Abstract

Efficient distributed concurrency control is one of the most significant challenges in building real-time cooperative editing systems. In this paper, we focus on the definition and verification of an integrated set of distributed concurrency control schemes based on a novel consistency model for solving three inconsistency problems: divergence, causality-violation, and intention-violation in real-time cooperative editing systems.

Keywords: concurrency control, consistency models, cooperative editing, distributed computing, CSCW.

1 Introduction

Cooperative editing systems are very useful and intensively used tools in the rapidly expanding area of CSCW (Computer Supported Cooperative Work) [5, 8, 10, 11, 15, 16]. A cooperative editing system allows multiple users to view and edit a shared document simultaneously from different sites, which are connected by a communication network. Each site typically contains an user interface for generating editing operations and displaying shared documents, a local storage for saving documents being edited locally, and computing and communicating facilities for executing, synchronizing, and propagating editing operations. A

Appeared in the Proc. of *Asian Computing Science Conference*, **LNCS, #1179, Springer-Verlag, Singapore, pp.84-95, Dec. 1996.**

real-time cooperative editing system is characterized by a short response time, which is the time necessary for the actions of one user to be reflected by his/her own interface, and a short notification time, which is the time necessary for one user's action to be propagated to the remaining users' interfaces. To facilitate free and natural information flow among cooperating sites, it is desirable for a real-time cooperative editing system to support unconstrained cooperative editing, i.e., all sites are allowed to edit the same document at any location and at any time, as advocated in [5, 10, 18, 20]. One of the most significant challenges in designing and implementing real-time cooperative editing systems is efficient concurrency control under the constraints of a short response time, a short notification time, and support for unconstrained cooperative editing in a distributed environment.

1.1 Three major inconsistency problems

Concurrency control is needed for the synchronization and coordination of concurrent activities to maintain consistency of a real-time cooperative editing system in the presence of concurrent operations issued by multiple cooperating users and nondeterministic propagation latency caused by the underlying communication network. To illustrate, consider an operation propagation scenario in a real-time cooperative text editing system with 3 sites as shown in Figure 1. Operation O_1 is generated at site 0, and operations O_2 and O_3 are generated at site 1. Without concurrency control, operations would be executed locally when they are generated, then broadcast to remote sites and executed there in its *original form* upon their arrival. Three major inconsistency problems manifest themselves in this scenario:

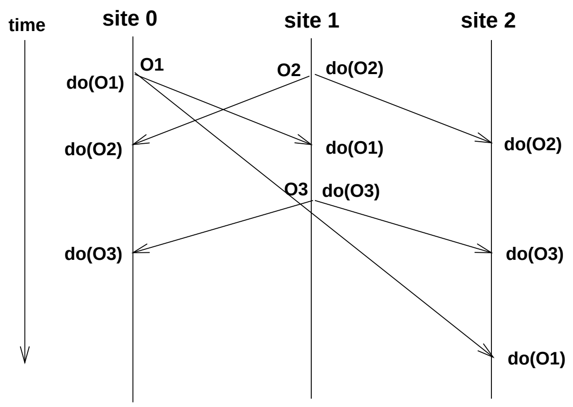

Figure 1: A scenario of generating, propagating, and executing operations in a real-time cooperative editing system without concurrency control.

First, due to concurrent generation of operations and nondeterministic communication latency, operations may arrive and be executed at different sites in different orders, resulting in *divergent final results*. As shown in Figure 1, the three operations in this scenario arrive and are executed in the following orders: O_1, O_2, and O_3 at site 0; O_2, O_1, and O_3 at site 1; and O_2, O_3, and O_1 at site 2. Unless operations are commutative (which is generally not the case), final editing results would not be identical among cooperating sites. Apparently, divergent final results should be prohibited for applications where the consistency of the final results is required, such as real-time cooperative software design and documentation systems.

The second inconsistency problem is not concerned with the final results of an editing session, but the process of an editing session. Due to nondeterministic communication latency, operations may be executed out of their natural *causal-effect* order. As shown in Figure 1, operation O_3 is generated after the arrival of O_1 at site 1, the editing effects of O_1 on the shared document have been seen by the user at site 1 at the time when O_3 is generated. Therefore, O_3 may be *dependent* on O_1. However, since O_3 arrives (and is executed) before O_1 at site 2, the user at site 2 could be confused by observing the *effect* in O_3 before observing the *cause* in O_1. Out-of causal order execution, called *causality violation*, should be prohibited for applications where a synchronized interaction among multiple users is required, such as electronic brainstorming meeting in which a group of people simultaneously generate and record ideas in the same file [10].

The third inconsistency problem is caused by concurrent generation of independent operations and is concerned about the consistency between the *intended effect* of an operation at the time of its generation and the *actual effect* at the time of its execution. As shown in Figure 1, operation O_1 is generated at site 0 without any knowledge of O_2 generated at site 1, so O_1 is *independent* of O_2, and vice versa. At site 0, O_2 is executed on a document state (i.e., the contents of the document) which has been changed by the preceding execution of O_1 (the reverse execution order occurs at site 1). Therefore, the subsequent execution of O_2 may refer to an incorrect position in the new document state, resulting in an editing effect which is different from the *intended effect* of O_2. For example, assume the shared document initially contains the following sequence of characters: "ABCDE". Suppose O_1 intends to insert string "12" at location 1, i.e., between "A" and "BCDE"; and O_2 intends to delete the two characters starting from location 2, i.e., "CD". After the execution of these two operations, the intention-preserved result (at all sites) should be: "A12BE". However, the actual result at site 0, obtained by executing O_1 followed by executing O_2, would be: "A1CDE". Even if the system is able to ensure that all sites have an identical result, but this identical result violates the intention of O_1 since the character "2", which was intended to be inserted, is missing in the final text, and also violates the intention of O_2 since characters "CD", which were intended to be deleted, are still present in the final text. Apparently, *intention violation* should be prohibited in cooperative editing systems which allow multiple users to concurrently edit the same document.

From the above discussion, we know that the absence of concurrency control in a real-time cooperative editing system could lead to inconsistencies in the final results of shared documents, as well as in the cooperating users' mental model about what is actually going on in a cooperative editing session. It should be noted that the above identified three inconsistency problems are *independent* in the sense that the existence of any one or two of them does not necessarily imply the existence of another one or two.

1.2 Alternative approaches

The study of effective and efficient distributed concurrency control in real-time cooperative editing systems has been a hot topic for researchers and designers in both CSCW and distributed computing systems [8, 17, 18]. Some representative alternative approaches are discussed below.

Turn-taking protocols: only one user at a time has the "token/floor" to edit the shared document [5]. Access to the token may be controlled by internal technical protocols (implemented by software) or through external social protocols (followed by cooperating users). The main problem with this approach is that it is ill-suited for systems with high parallelism and inhibits unconstrained cooperative editing.

Lock-based protocols: an object (e.g., a word, a line, or a section, etc.) is first locked before it is updated, so only one user at a time is able to update an object [5]. The main problems with locking include restriction in supporting unconstrained cooperative editing, degradation in response time due to the overhead of requesting and releasing locks, and the danger of deadlock.

Transaction-based protocols: traditional ACID (Atomicity, Consistency, Isolation, and Durability) transaction mechanisms have been a well-researched topic in database systems, and have allowed for successful concurrency control in non-real-time CSCW systems, such as CES [9] and Quilt [7, 13]. For real-time CSCW systems, ACID mechanisms are not well-suited due to the inhibition of unconstrained cooperative editing and degradation in response and notification time caused by the complexity of implementing ACID mechanisms in distributed environments.

Serialization-based protocols: operations are executed in the same total order at all sites, so the final results will be identical at all sites. This can be implemented by means of the distributed algorithm based on logical clocks [12], or by using a centralized coordinator, i.e., all operations are first sent to the coordinator, which then broadcasts them to all sites in sequence. However, in addition to the problem of single point of failure and a performance bottleneck in the case of using a centralized coordinator, the serialization-based protocols have two main problems. First, responsiveness is lost since operations are not executed when generated but after some delay, and notification is delayed for the same reason. Secondly, an operation may refer to a state which has been changed by the preceding executions of other independent operations, leading to a result which is inconsistent with the intended effect of operations.

Optimistic execution protocols: operations are allowed to be executed upon their arrival without regard to their ordering. If out-of-order execution occurs, the problem can be repaired by undoing the executed out-of-order operations, and then redoing them after executing the new one [14], or by transforming the new out-of-order operation so that its effect is the same as if it had arrived in order [4, 20]. This optimistic approach allows unconstrained cooperative editing, and hence has very good responsiveness and notification. But the existing proposals in this category suffer from the problems of intention violation and possibly causality violation if care is not taken.

To the best of our knowledge, none of the existing approaches adequately addresses all the three inconsistency problems in real-time cooperative editing systems under the constraints of a short response time, a short notification time, and support for unconstrained cooperative editing in a distributed environment. In this paper, a novel approach to concurrency control in real-time distributed unconstrained cooperative editing systems is proposed to solve the three major inconsistency problems in an integrated way.

The rest of the paper is organized as follows: First, a consistency model with three properties – convergence, causality-preservation, and intention-preservation – is introduced as a framework for solving the three inconsistency problems in Section 2. Thereafter, the basic mechanisms, upon which various concurrency control schemes in our approach are based, will be introduced in Section 3. Then, an integrated set of concurrency control schemes for supporting the consistency model, including the convergence scheme, the causality-preserving scheme, and the intention-preserving scheme are defined and verified in Sections 4, 5, and 6, respectively. Finally, the conclusions and future work are given in Section 7.

2 A consistency model

In [18], we have proposed a consistency model, which effectively specifies what assurance a cooperative editing system may give to its users and what properties the underlying concurrency control schemes need to support. In this section, the definitions related to this model are introduced. Following Lamport [12], we first define a causal (partial) ordering relation on operations in terms of operation generation and execution sequences.

Definition 1 *Causal ordering relation "$\rightarrow$"*
Given two operations O_a and O_b, generated at sites i and j, then $O_a \rightarrow O_b$, *iff*: (1) $i = j$ and the generation of O_a *happened before* the generation of O_b, or (2) $i \neq j$ and the execution of O_a at site j *happened before* the generation of O_b, or (3) there exists an operation O_x, such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$.

With the definition of causal ordering relation, given any two operations O_a and O_b, O_b is said to be *dependent* on O_a *iff* $O_a \rightarrow O_b$. We also say that O_a causally precedes O_b, or O_b causally follows O_a. O_a and O_b are said to be *independent* (or *concurrent*) *iff* neither $O_a \rightarrow O_b$, nor $O_b \rightarrow O_a$, expressed as $O_a \parallel O_b$. In the following discussion, the term of *editing session* is used to mean an invocation of the cooperative editing system. It can last an hour or two but may be much shorter or longer. An editing session is in a *quiescent state* if and only if all operations have been executed at all sites, i.e., there are no operations in transit or waiting for execution by any site.

Definition 2 *Intention of an operation*

Given an operation O, the intention of operation O is an execution effect which could be achieved by applying O on the shared document state observed by the user at the time of issuing O.

Definition 3 *A consistency model*

A cooperative editing system is said to be consistent if it always maintains the following properties: (1) **Convergence** : when an editing session is at quiescence, the copies of the shared documents at all sites are *identical.* (2) **Causality-preservation**: for any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is executed before O_b at all sites. (3) **Intention-preservation**: for any pair of operations O_a and O_b, if $O_a \parallel O_b$, the execution of O_a and O_b in either order at any site preserves the intention of each operation.

In essence, the *convergence* property ensures the consistency of the final results *at the end* of a cooperative editing session; the *causality-preservation* property ensures the consistency of the execution orders of *dependent* operations *during* a whole cooperative editing session; and the intention-preservation property ensures that for a user at any site, (1) what he or she edits (locally) will be what he or she sees on the local screen and what all cooperating users see on remote screens (regardless the nondeterministic propagation delay of local operations to remote sites), and (2) what cooperating users are concurrently editing have no effect on what he or she is editing.

3 Basic mechanisms

In this section, some basic mechanisms are introduced, on which all concurrency control schemes in our approach will be based. In the following discussion, we assume that the number of cooperating sites in the system is a constant N, and the N sites are identified by integers $0, ..., N-1$, respectively. Under this assumption, we postpone our concern about important issues, such as dynamical group membership management, site crashes and recovery.

Definition 4 *The local operations execution scheme*

An operation is executed at the local site immediately after its generation.

Obviously, the above scheme not only gives good responsiveness but also ensures all operations are executed at their local sites in the order of their generation. To capture the causal relationship among all operations in the system, a timestamping scheme based on a data structure State Vector (SV) is defined.

Definition 5 *The state vector based timestamping scheme*

Each site maintains a state vector with N components, one per site. Consider the state vector SV^k at site k: (1) Initially, $SV^k[i] := 0$, for all $i \in \{0, ..., N-1\}$. (2) **Updating Rule 1 (UR1):** After executing a local operation, the kth component of SV^k is updated as follows: $SV^k[k] := SV^k[k]+1$. (3) **Timestamping Rule (TR):** After executing a local operation and updating SV^k according to **UR1**, the local operation is timestamped by the current value of SV^k and and broadcast to all remote sites. (4) **Updating Rule 2 (UR2):** After executing a remote operation O with a timestamp SV_O, SV^k is updated as follows: $SV^k[i] := max(SV^k[i], SV_O[i])$, for all $i \in \{0, ..., N-1\}$.

The following theorem states a well-known basic property of the vector clock (state vector) based timestamping scheme [1, 6].

Theorem 1 *Given two operations O_a and O_b, timestamped by SV_{O_a} and SV_{O_b}, respectively, $O_a \rightarrow O_b$ iff (1) for all $i \in \{0, ..., N-1\}$, $SV_{O_a}[i] \leq SV_{O_b}[i]$, and (2) there exists one j, such that $SV_{O_a}[j] < SV_{O_b}[j]$.*

4 The convergence scheme

The basic ideas of ensuring the convergence property are as follows: (1) Define a total ordering relation "$\Rightarrow$" among all operations based on their state-vector timestamps and their site identifiers. (2) Maintain a *history buffer* for executed operations at each site. (3) When an operation arrives out-of the total ordering, all totally preceding operations in the history buffer are first undone, then the newly arrived operation is executed, and finally the undone operations are redone.

Definition 6 *total ordering relation "$\Rightarrow$"*

Given two operations O_a and O_b, generated at sites i and j and timestamped by SV_{O_a} and SV_{O_b}, respectively, then $O_a \Rightarrow O_b$, *iff*: (1) $sum(SV_{O_a}) < sum(SV_{O_b})$, or (2) $sum(SV_{O_a}) = sum(SV_{O_b})$ and $i < j$, where $sum(SV) = \sum_{i=0}^{N-1} SV[i]$.

Theorem 2 *The total ordering relation "$\Rightarrow$" is consistent with the causal ordering relation "$\rightarrow$" in the sense that if $O_a \rightarrow O_b$, then $O_a \Rightarrow O_b$.*

Proof: Suppose O_a is timestamped by SV_{O_a}, O_b is timestamped by SV_{O_b}. If $O_a \rightarrow O_b$, then there must be $sum(SV_{O_a}) < sum(SV_{O_b})$ according to Theorem 1. From $sum(SV_{O_a}) < sum(SV_{O_b})$, we have $O_a \Rightarrow O_b$ according to Definition 6.

Definition 7 *History Buffer (HB)*

Each site maintains a history buffer to keep track of all operations executed locally. Initially, $HB[i] := \phi$, for all $i \in \{0, ..., MAX-1\}$, where MAX is maximum number of entries in *HB*. After an operation is executed, it is saved in a proper location in HB, so that all buffered operations in HB are sorted in the order of "$\Rightarrow$," i.e. for any $i \in \{0, ..., MAX-2\}$, if $HB[i] \neq \phi$ and $HB[i+1] \neq \phi$, then $HB[i] \Rightarrow HB[i+1]$.

Definition 8 *The undo/do/redo scheme*

To execute a new (local/remote) operation O_{new}, the following steps are executed: (1) **Undo** all operations in the *HB* which totally *follow* O_{new} to restore the document's state before their execution. (2) **Do** O_{new} and save it in the *HB*. (3) **Redo** all undone operations in the *HB*.

One assumption made by the above scheme is that all operations in the cooperative editing system are *reversible* [16]. It is required that buffered operations should contain enough information in order to be undone and redone. Moreover, a distributed garbage collection scheme can be incorporated to remove the operations in *HB* which are not needed any more by the undo/do/redo scheme [17]. It should be noted that the *undo* operation involved in the undo/do/redo scheme is an *internal* operation, rather than an *external* operation initiated from the user interface [14]. Therefore, only the final result of an invocation of the undo/do/redo scheme should be reflected on the user interface in a single flushing operation, rather than flushing all intermediate results produced by individual internal steps. The following theorem establishes that the undo/do/redo scheme is a convergence scheme which ensures the convergence property of the consistency model.

Theorem 3 *At a session's quiescence, copies of the shared documents are guaranteed to be identical at all sites if the undo/do/redo scheme is used.*

Proof: When all operations have been executed, the editing effect will be the same as if all operations were executed in the total order "$\Rightarrow$" at all sites according to the *undo/do/redo* scheme (Definition 8). Therefore, copies of the shared documents at all sites must be identical at a session's quiescence.

5 The causality-preserving scheme

Althhough the convergence scheme ensures the consistency of the final results *at the end* of a cooperative editing session, it does not address the issue of consistency maintenance *during* a cooperative editing session, which is very important for providing users with a logical and consistent mental model about what is actually going on in a cooperative editing session. To enforce the execution order constrained by the causal relationship among operations, the remote operations execution scheme is introduced below.

Definition 9 *The remote operations execution scheme*

Operations are selectively delayed at remote sites according to the following rules: Let O_i be an operation generated at a remote site i and timestamped by SV_{O_i}. O_i is not allowed to be executed at site k ($k \neq i$) until (1) $SV_{O_i}[i] = SV^k[i] + 1$, and (2) $SV_{O_i}[j] \leq SV^k[j]$, foror all $j \in \{0, 1, ..., N-1\}$ and $j \neq i$.

The first condition ensures that O_i must be the next operation in sequence from site i, so no operations originated at site i have been missed by site k. The second condition ensures that all operations originated at other sites and executed at site i before the generation of O_i must have been executed at site k. Altogether these two conditions ensure that all operations which causally precede O_i have been executed at site k. The following lemmas and theorems establish that the remote execution scheme, integrated with the basic mechanisms defined in Section 3, is a causality-preserving scheme.

Lemma 1 *Operations generated at any site i, $i \in \{0, 1, ..., N-1\}$, are executed at all sites in the order of their generation.*

Proof: It follows directly from the local operations execution scheme (Definition 4) and the remote operations execution scheme (Definition 9).

Lemma 2 *At site k, after executing an operation originated from site i, the ith component of SV^k will be incremented by 1, i.e., $SV^k[i] := SV^k[i] + 1$, and no other component of SV_k will be changed.*

Proof: It follows directly from the fact that the execution of a local operation (i.e., $i = k$) will increment $SV^k[k]$ and only $SV^k[k]$ by 1 according to **UR1** in Definition 5, and the execution of a remote operation (i.e., $i \neq k$) will increment $SV^k[i]$ and only $SV^k[i]$ by 1 according to the two conditions in Definition 9 and **UR2** in Definition 5,

Lemma 3 *For any site k, when $SV^k[i] = m$, $i \in \{0, 1, ..., N-1\}$, the first m operations generated at site i must have been executed at site k.*

Proof: It follows directly from Lemma 1 and Lemma 2.

Theorem 4 *Given any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is guaranteed to be executed before O_b at all sites.*

Proof: Suppose O_a and O_b were generated at sites i and j, respectively. (1) In case that $i = j$, O_a must be generated before the generation of O_b according to Definition 1. Therefore, O_a must be executed before O_b at all sites according to Lemma 1. (2) In case that $i \neq j$, let SV_{O_a} be the timestamp of O_a and SV_{O_b} be the timestamp of O_b. We have $SV_{O_a}[i] \leq SV_{O_b}[i]$ according to Theorem 1. (a) At site j, $SV_{O_b}[i] = SV^j[i]$ at the time of generating and timestamping O_b according to **TR** in Definition 5. From $SV_{O_a}[i] \leq SV_{O_b}[i] = SV^j[i]$, we know that O_a must have been executed at site j at the time of generating O_b according to Lemma 3. (b) At any other site $k \neq j$, for O_b to be executed, there must be $SV_{O_b}[i] \leq SV^k[i]$ according to the second condition in Definition 9. From $SV_{O_a}[i] \leq SV_{O_b}[i] = SV^k[i]$, we know that O_a must have been executed at site k at the time of executing O_b at site k according to Lemma 3.

6 The intention-preserving scheme

With the convergence scheme and the causality-preserving scheme only, there is no guarantee that the totally and causally ordered execution effect is consistent with the intention of independent operations. The intention-violation problem could not be fixed by re-scheduling the execution of operations without changing the format of operations. Therefore, the basic idea of the intention-preserving scheme is to transform an operation before its execution to compensate the changes made to the document state by the preceding execution of independent operations.

The intention-preserving scheme consists of two parts: (1) A generic part, which relies only on the causal ordering relation "$\rightarrow$" and the total ordering relation "$\Rightarrow$" among operations, and determines which operations need to be transformed against which. (2) An application-dependent part, which relies on semantics information of the operations involved, and does the real operation transformation. In this paper, our discussion will be confined to a brief description of the generic part of the intention-preserving scheme, which could be applied to different systems with different sets of primitive editing operators. An intention-preserving transformation algorithm for primitive operations in a cooperative text editing system can be found in [19].

In the following discussion, O_{new} is used to denote a new incoming operation. To select operations for transformation, the causal ordering relation "$\rightarrow$" is used to divide all operations in the *HB* into two groups: (1) One consists of operations which causally precede O_{new}. O_{new} needs not be transformed against them since their execution effects must have been seen and taken into account at the moment when O_{new} was generated, and their execution order has been enforced by the causality-preserving scheme. (2) The other consists of operations which are independent of O_{new}. This group can be further divided into two subgroups by means of the total ordering relation "$\Rightarrow$": (a) One subgroup consists of operations which totally precede O_{new}. O_{new} needs to be transformed against all of them (repeatedly) since the execution effects of them were not seen at the moment when O_{new} was generated and the execution of them may have changed the state of the shared document. (b) The other subgroup consists operations which totally follow O_{new}. This group of operations need to be undone first. Then, before redoing them, they need to be transformed against O'_{new} (the transformed O_{new}) since the execution effect of O'_{new} was not seen at the moment when they were generated. Based on the above analysis, we have devised the following scheme which integrates the intention-preserving scheme with the *undo/do/redo* scheme:

Definition 10 *The undo/transform-do/transform-redo scheme*

When a new incoming operation O_{new} has passed the remote operation execution conditions test successfully, the following steps are executed: (1) **Undo** all operations in the *HB* which totally *follow* O_{new} to restore the document's state before their execution. (2) **Transform** O_{new} against all the totally *preceding* and *independent* operations in the *HB*, then **do** the transformed O_{new} and insert it into the *HB*. (3) **Transform** all undone operations against the transformed O_{new} one by one, and then **redo** them one by one.

If all independent operations in a session were generated out of the same document state, then the *transform-do* step and the *transform-redo* step could be easily accomplished by applying the transformation algorithm in [19]. When, however, some independent operations are generated from *incompatible* (i.e., non-identical) document states, which is very common in a highly current cooperative editing environment, then the situation becomes much more complicated. The main trouble is that not only does operation O_{new} need be transformed against preceding independent operations, but also may those preceding independent operations need first be transformed against other operations which causally precedes O_{new}. The detailed treatment of the undo/transform-do/transform-redo scheme is beyond the scope and page limitation of this paper and will be reported in a forthcoming paper.

7 Conclusions

We have discussed a novel approach to distributed concurrency control in real-time cooperative editing systems, under the constraints of a short response time, a short notification time, and support for unconstrained cooperative editing. Major contributions of our approach include: (1) Identification and classification of the three independent inconsistency problems, i.e., divergence, causality-violation, and intention-violation. (2) Definition of a consistency model with three properties, i.e., convergence, causality-preservation, and intention-preservation. (3) Design and theoretical verification of an integrated set of schemes, which maintain the three properties required by the proposed consistency model.

In addition to the theoretical verification of the concurrency control algorithms, which is the focus of this paper, software simulation has been conducted to investigate the implementability of these algorithms [2]. Moreover, we have found that the three properties of the proposed consistency model is not only desirable in the text cooperative editing domain but also in the graphics cooperative editing domain [3]. We have started work toward removing the restriction of fixed number of cooperating sites in each session and incorporating crash recovery and group-awareness supporting mechanisms into the system. Based on these results, an Internet-based prototype REDUCE (REal-time Distributed Unconstrained Cooperative Editing) system is being built using Java to explore system design and implementation issues and to gain usage experience.

Acknowledgements

The authors wish to thank Rodney Topor for his advice and comments, which helped improve our work on the consistency model. The authors are very grateful to the anonymous referees for their valuable comments.

References

[1] K. Birman, A. Schiper, and P. Stephenson: "Lightweight causal and atomic group multicast, " *ACM Trans. on Comp. Sys.* 9(3), pp. 272-314, Aug. 1991.

[2] D. Chen and C. Sun:"The design and implementation of a software simulation system for real-time cooperative editing systems," To appear in *Proc. of the Third Australasian Conference on Parallel and Real-Time Systems,* Brisbane, Sept 30 - Oct. 1, 1996.

[3] D Chen, and C. Sun: "Consistency Maintenance and Conflict Resolution in Real-time Cooperative Graphics Editing Systems," In *Proc. of the Inaugural Australian Computer-Supported Cooperative Work Symposium,* Brisbane, August 1996, pp.31-37.

[4] C. A. Ellis and S. J. Gibbs: "Concurrency control in groupware systems," In *Proc. of ACM SIGMOD Conference on Management of Data*, pp.399-407, 1989.

[5] C. A. Ellis, S. J. Gibbs, and G. L. Rein: "Groupware: some issues and experiences," *CACM 34(1)*, pp.39-58, Jan. 1991.

[6] G. Fidge:"Timestamps in message-passing systems that preserve the partial ordering," *Proc. of the 11th Australian Computer Science Conference,* pp. 56-66, 1988.

[7] R. Fish, R. Kraut, M. Leland, and M. Cohen:"Quilt: a collaborative tool for cooperative writing," In *Proc. of the Conference on Office Information Systems,* pp. 30-37, Mar. 1988.

[8] S. Greenberg, D. Marwood:"Real time groupware as a distributed system: concurrency control and its effect on the interface," In *Proc. of ACM Conference on Computer Supported Cooperative Work,* pp. 207–217, Nov. 1994.

[9] I. Grief, R. Seliger, and W. Weihl:"Atomic data abstractions in a distributed collaborative editing system," In *Proc. 13th Annual Symposium on Principles of Programming Languages*, pp.160-172, 1986.

[10] C. M. Hymes and G. M. Olson:"Unblocking brainstorming through the use of a simple group editor," In *Proc. ACM Conference on Computer Supported Cooperative Work,* pp. 99-106, Nov. 1992.

[11] M. Knister and A. Prakash:"DistEdit: a distributed toolkit for supporting multiple group editors," In *Proc. of ACM Conference on Computer Supported Cooperative Work,* pp. 343-355, Oct. 1990.

[12] L. Lamport: "Time, clocks, and the ordering of events in a distributed system," *CACM 21(7)*, pp.558-565, 1978.

[13] M. Leland, R. Fish, and R. Kraut:"Collaborative document production using Quilt," in *Proc. of ACM Conference on Computer Supported Cooperative Work,* pp. 206-215, Sept. 1988.

[14] A. Prakash, M. Knister: "A framework for undoing actions in collaborative systems," *ACM Trans. on Computer-Human Interaction*, 4(1), pp.295-330, 1994.

[15] T. Rodden: "A survey of CSCW systems," *Interacting with computers – the interdisciplinary journal of human-computer interaction*, 3(3), pp.319–353, Dec. 1991.

[16] S. Sarin, and I. Grief: "Computer-based real-time conference," *Computer,* 18(10), pp.33-45, 1985.

[17] C. Sun, Y. Zhang, and Y. Yang: "Distributed synchronization of group operations in cooperative editing environments," In *Proc. of the Second International Conference on Concurrent Engineering*, pp.279-290, Washington DC, August 1995.

[18] C. Sun, Y. Yang, Y. Zhang, and D. Chen: "A consistency model and supporting schemes in real-time cooperative editing systems," *Proc. of the 19th Australian Computer Science Conference,* Melbourne, pp.582-591, Jan. 1996.

[19] C. Sun, Y. Yang, Y. Zhang, and D. Chen: "An intention preserving transformation algorithm for operations in cooperative text editing systems," In *Proc. of the Third International Conference on Concurrent Engineering*, pp.16-23, Toronto, August 1996.

[20] Y. Zhang and Y. Yang: "On operation synchronization in cooperative editing environments," In *IFIP Transactions A-54,* pp.635-644, May 1994.