Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements

Chengzheng Sun
School of Computing and Information Technology
Griffith University
Brisbane, Qld 4111, Australia
scz@cit.gu.edu.au
http://www.cit.gu.edu.au/~scz

Clarence (Skip) Ellis
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430, USA
skip@colorado.edu
http://www.cs.colorado.edu/~skip/Home.html

ABSTRACT

Real-time group editors allow a group of users to view and edit the same document at the same time from geographically dispersed sites connected by communication networks. Consistency maintenance is one of the most significant challenges in the design and implementation of these types of systems. Research on real-time group editors in the past decade has invented an innovative technique for consistency maintenance, called operational transformation. This paper presents an integrative review of the evolution of operational transformation techniques, with the goals of identifying the major issues, algorithms, achievements, and remaining challenges. In addition, this paper contributes a new optimized generic operational transformation control algorithm.

Keywords

Consistency maintenance, operational transformation, convergence, causality preservation, intention preservation, group editors, groupware, distributed computing.

INTRODUCTION

Real-time group editors allow a group of users to view and edit the same text/graphic/image/multimedia document at the same time from geographically dispersed sites connected by communication networks. These types of groupware systems are not only very useful tools in the areas of CSCW [5], but also serve excellent vehicles for exploring a range of fundamental and challenging issues facing the designers of real-time groupware systems in general. One such issue is consistency maintenance of shared documents under the constraints of short response time, and support for free and concurrent editing in distributed environments [17].

Research on real-time group editors in the past decade has invented an innovative technique for consistency maintenance, under the name of operational transformation, which was pioneered by the GROVE (GRoup Outline Viewing Editor) system in 1989 [3]. Since then, several research groups have independently extended the operational transformation technique in their design and implementation of these types of systems. Major representatives in this area include the REDUCE (REal-time Distributed Unconstrained Cooperative Editing) system [14, 15, 16, 17], the Jupiter system [11], and the adOPTed algorithm [13]. This paper will present an integrative review of the evolution of operational transformation techniques, with the goals of identifying the major issues, algorithms, achievements, and remaining challenges. In addition, this paper will contribute a new optimized generic operational transformation control algorithm. This paper will focus exclusively on transformation-based consistency maintenance algorithms. For discussion of alternative consistency maintenance techniques, such as turn-taking, locking, serialization, and transactions, the reader is refereed to [5, 7, 8, 10, 17].

The rest of this paper is organized as follows: First, some basic concepts and terminologies are introduced. Then, the operational transformation algorithm in the GROVE system is reviewed to see where the original work was started and what problems were left unsolved. Next, the problems with the original GROVE transformation algorithm are analyzed, and three different approaches to solving them are discussed one by one, including the REDUCE approach, the Jupiter approach, and the adOPTed approach. Furthermore, a new optimized generic operational transformation control algorithm is proposed. Finally, the paper is concluded with a summary of the major achievements so far and remaining challenges for future research.

PRELIMINARIES

In this section, some basic concepts and terminologies are introduced. Following Lamport [9], we define a causal (partial) ordering relation on operations in terms of their generation and execution sequences as follows.

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 $O_x \rightarrow O_b$. □

Definition 2: Dependent and independent operations
Given any two operations O_a and O_b. (1) O_b is *dependent* on O_a *iff* $O_a \rightarrow O_b$. (2) O_a and O_b are *independent* (or *concurrent*), expressed as $O_a \parallel O_b$, *iff* neither $O_a \rightarrow O_b$, nor $O_b \rightarrow O_a$. □

To illustrate, consider a real-time group editing session with three sites, as shown in the time-space graph of Figure 1. There are four editing operations in this scenario: operation O_1 generated at site 0, operations O_2 and O_3 generated at site 1, and operation O_4 generated at site 2. It is assumed in this scenario that an operation is executed immediately at

Appeared in **Proc. of 1998 ACM Conference on Computer-Supported Cooperative Work,** *Seattle, USA, Nov.14-18, 1998, pp. 59-68.*

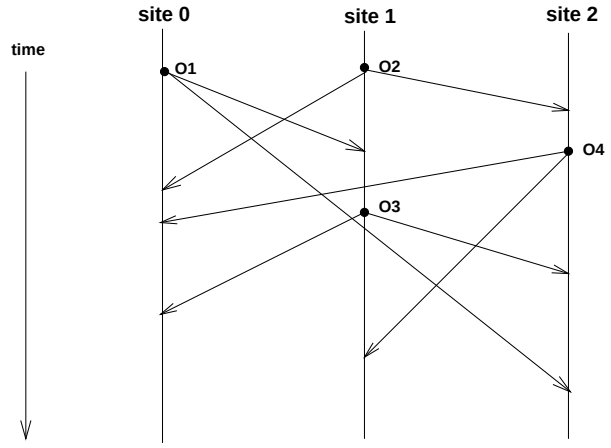


Fig. 1. A scenario of a real-time group editing session.

the local site, then propagated to remote sites and executed there upon their arrival. The arrows in the graph represent the propagation of operations from the local site to remote sites. Each vertical line in the graph represents the activities performed by the corresponding site. At site 1, for example, O_2 is executed first, followed by O_1, O_3, and O_4.

According to Definitions 1 and 2, there are three pairs of dependent operations in this scenario: $O_1 \rightarrow O_3$, $O_2 \rightarrow O_3$, and $O_2 \rightarrow O_4$ because the execution of O_1 happens before the generation of O_3, the generation of O_2 happens before the generation of O_3, and the execution of O_2 happens before the generation of O_4. Moreover, there are three pairs of independent operations in this scenario: $O_1 \parallel O_2$, $O_1 \parallel O_4$, and $O_3 \parallel O_4$ because for any pair, neither operation's execution happens before the other operation's generation. As will be seen in the following discussion, several fundamental inconsistency problems are embedded in this scenario. Moreover, the seemingly simple independence relationship among operations in this scenario is actually quite intricate, and has given significant technical challenges to the design of correct operational transformation algorithms [17].

THE GROVE APPROACH

To achieve good responsiveness and avoid a single-point of failure in the system, a replicated architecture has been adopted by GROVE: the shared documents are replicated at the local storage of each participating site. An (update) operation is executed on the local replica of the shared document immediately after its generation, then broadcast to remote sites for execution (after some delay and transformation).

Divergence and causality-violation problems

Suppose remote operations are executed upon their arrival and in their *original form*, two inconsistency problems which may occur in a concurrent editing session have been identified in GROVE: one is divergence, and the other is causality-violation.

For example, consider the scenario shown in Fig. 1. The four operations arrive and are executed in the following orders: O_1, O_2, O_4, and O_3 at site 0; O_2, O_1, O_3, and O_4 at site 1; and O_2, O_4, O_3, and O_1 at site 2. If operations are not commutative, final editing results would not be identical among cooperating sites. This problem is called *divergence*. Clearly, the divergence problem should be prohibited for applications where the consistency of the final results is required.

Moreover, since each cooperating site generates and broadcasts operations without synchronization, operations may arrive and be executed in an order different from their natural causal order. As shown in Fig. 1, operation O_3 is generated after the arrival of O_1 at site 1, so $O_3 \rightarrow O_1$. However, since O_3 arrives before O_1 at site 2, the execution of O_3 before O_1 may result in an undefined operation O_3, which refers to a nonexistent context to be created by O_1, or a confused user at site 2, who observes the *effect* in O_3 before observing the *cause* in O_1. This problem is called *causality-violation*. Out of causal order execution should be prohibited for applications where a synchronized interaction among multiple users is required.

Consistency correctness criteria

Based on the identification of the two inconsistency problems, the GROVE consistency correctness criteria were defined by the following two properties:

1. **Convergence property**: copies of the shared document are identical at all sites at quiescence (i.e., all generated operations have been executed at all sites).
2. **Precedence property**: if one operations O_a causally precedes another operation O_b, then at each site the execution of O_a happens before the execution of O_b.

In search of a solution where the only constraint on execution order is the causal ordering among operations, GROVE invented the late well-known distributed OPerational Transformation (dOPT) algorithm. GROVE's solution consists of two components: one is the state-vector timestamping scheme for ensuring the precedence property, and the other is the dOPT algorithm for ensuring the convergence property. The basic idea of the dOPT algorithm is that when an operation satisfies the precedence condition for execution, it is transformed against independent operations in the Log (which saves all executed operations in the order of their execution) in such a way that executions of the same set of properly transformed independent operations in different orders produce identical document states, thus ensuring the convergence property.

A transformation property

To ensure convergence, the dOPT algorithm requires the transformation function T to satisfy the following condition: For any two independent operations O_a and O_b, suppose that $O'_a = T(O_a, O_b)$, and $O'_b = T(O_b, O_a)$, it must be that

$$O_a \circ O'_b \equiv O_b \circ O'_a$$

where "$\equiv$" means the two sequences of operations $O_a \circ O'_b$ and $O_b \circ O'_a$ are *equivalent* in the sense that when applied on the same input document state they produce the same output document state.

In addition to the above formally specified condition, GROVE also recognized there were some circumstances, in which the transformation function should achieve an effect which is *non-serializable*. For example, suppose O_a and O_b are two independent (character-wise) *delete* operations referring to the same position, then T must ensure only one character is eventually deleted no matter in which order O_a and O_b are executed. This non-serializable effect is, however, not captured by the above formal condition for T.

A sketch of the dOPT algorithm

The transformation function T relies on the semantics of the editing operations and hence is application-dependent. The dOPT algorithm, however, is generic and takes care of selecting operations for transformation and determining the transformation order. The basic control structure of the dOPT algorithm is simple: Given a causally ready operation O, the dOPT algorithm scans the Log to transform O against any operation in the Log which is independent of O; then the transformed O, denoted as EO (i.e., the execution form of O), is executed and saved in the Log. The dOPT algorithm is sketched below.

```
dOPT(O, Log) {
   EO = O;
   for (i = 1; i <= n; i++) {
       if (Log[i] || O)
       then EO = T(EO, Log[i]);
   }
   Execute EO;
   Append EO at the end of the Log;
}
```

An unsolved dOPT puzzle

In [3] (Fig. 4 in Section 6: Discussion of Correctness), one scenario was identified, where the dOPT algorithm could not ensure convergence. This scenario[1] is re-displayed in Fig. 2.

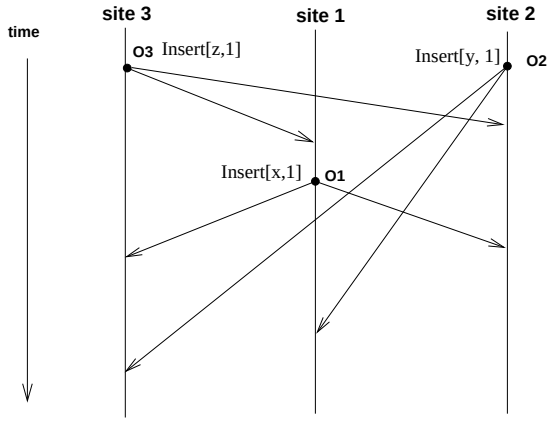


Fig. 2. The mixed priority example, in which the dOPT algorithm failed to ensure convergence.

Suppose the GROVE transformation function uses the following *priority rule*: when two insert operations have the same position parameter, the position of the operation with a *lower* priority (i.e., smaller site identifier) will be shifted[2]. According to the generic dOPT algorithm and the application-dependent transformation function in [3], the operation transformation and the final document states at the three sites are as follows (assume the initial document is empty).

At site 3, O_3 first inserts "z" into the document[3]. When O_1 arrives, it inserts "x" in front of "z" to get a document with "xz". Finally, when O_2 arrives, since $O_2 \parallel O_3$ and $O_2 \parallel O_1$, it is first transformed against O_3 and becomes $O_2' = Insert[y, 2]$ due to its lower priority than O_3; then it is transformed against O_1 and becomes $O_2'' = Insert[y, 3]$. After the execution of O_2'', the document contains "xzy"[4]. At site 1, the process of operation transformation and the final result are the same as that at site 3. At site 2, O_2 first inserts "y" into the document. When O_3 arrives, it has to be transformed against O_2 since $O_3 \parallel O_2$, but no change has been made to O_3 due to its higher priority than O_2. After the execution of O_3, the document contains "zy". Finally, when O_1 arrives, it has to be transformed against O_2 since $O_1 \parallel O_2$. The transformation of O_1 against O_2 will produce $O_1' = Insert[x, 2]$ due to its lower priority than O_2. O_1' does not need to be transformed against O_3 since $O_3 \rightarrow O_1$. After the execution of O_1', the document contains "zxy", which is not identical to "xzy" at sites 3 and 1.

The problem illustrated in Fig. 2 is fundamental to the correctness of operational transformation approach. As correctly pointed out in [3], this problem could not be fixed by simply reversing the priority rule, since this patch works in this case but fails in other rather similar cases. In search of a correct solution to this problem, the simple-minded priority scheme (using a single site identifier) was thought to be root of the problem, thus a sophisticated (and complicated) priority scheme (using a list of site identifiers) was proposed in [3]. This new priority scheme did not prove to be successful in solving the problem, thus leaving one unsolved puzzle to the groupware research community.

The innovative idea of maintaining consistency by operational transformation, as well as the unsolved dOPT puzzle, has been a major inspiration and stimulation to a number of research groups in the area of real-time groupware systems. In fact, several research groups [1, 11, 13, 17], have independently re-discovered that the dOPT algorithm did not work whenever an operation is concurrent with two or more dependent operations, and different approaches have been proposed to fix it. In the following sections, three alternative approaches will be discussed, including the REDUCE approach [14, 15, 17] using an *1-dimensional* data structure for keeping track of executed operations, the Jupiter approach [11] using a *2-dimensional* data structure for maintaining executed operations, and the adOPTed approach [13] using a *N-dimensional* data structure (where N is number of cooperating sites in the system) for maintaining executed operations.

THE REDUCE APPROACH

REDUCE follows GROVE in adopting a fully distributed and replicated system architecture. A linear *History Buffer* (HB), which is the same as the Log in GROVE, is used to keep track of all executed operations. In addition, a garbage collection scheme was devised to remove useless operations from the HB [17].

The intention-violation problem

Apart from divergence and causality-violation problems, one special kind of inconsistency problem – *intention-violation* – has been identified in REDUCE [14].

[1] In fact, the scenarios in Fig. 2 can be obtained by removing O_2 from the scenario illustrated in Fig. 1.

[2] It should be noted that this priority rule is actually opposite to the one used in the definition of transformation function T_{11} in [3]. This change is necessary to correctly illustrate the problem the GROVE designers really intended to illustrate.

[3] In this paper, the sequence of characters in a text document are referred to (or addressed) from 1 to the end of the document.

[4] It should be pointed out that this result is different from what was presented in [3].

To illustrate, consider the two independent operations O_1 and O_2 in the scenario shown in Fig. 1. At site 0, O_2 is executed on a document state which has been changed by the preceding execution of O_1. Therefore, the subsequent execution of O_2 may refer to an incorrect position in the new document state, and result in an editing effect different from the O_2's *intention*, which is defined as the editing effect which could be achieved by applying O_2 on the document state from which O_2 was generated [14].

For example, assume the shared document initially contains the following sequence of characters: "ABCDE". Suppose $O_1 = Insert$["12", 2], which intends to insert string "12" at position 2, i.e., between "A" and "BCDE"; and $O_2 = Delete[2,3]$, which intends to delete the two characters starting from position 3, 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", which clearly 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. A serialization protocol might be used to ensure that all sites execute O_1 and O_2 in the same order to get an identical result "A1CDE", but this identical result is still inconsistent with the intentions of both O_1 and O_2.

It is important to recognize that intention violation is an inconsistency problem of a different nature from the divergence problem. The essential difference between divergence and intention violation is that the former can always be resolved by a serialization protocol, but the latter cannot be fixed by any serialization protocol if operations were always executed in their original forms.

A consistency model

Due to the distinction of the intention-violation problem from the divergence problem, one additional consistency correctness criteria – intention-preservation – was proposed in REDUCE [14]. The REDUCE correctness criteria for consistency maintenance has been defined in the form of a consistency model as follows.

Definition 3: A consistency model
A cooperative editing system is consistent if it always maintains the following properties:

1. **Convergence**: when the same set of operations have been executed at all sites, all copies of the shared document 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 operation O, the effects of executing O at all sites are the same as the intention of O, and the effect of executing O does not change the effects of independent operations.

□

To support the three properties of the consistency model, REDUCE adopted the same state-vector timestamping scheme as that in GROVE for achieving causality-preservation (or precedence in GROVE's terminology). With the distinction of intention-preservation from convergence, two separate schemes were devised for supporting these two different properties: an *undo/do/redo* scheme for achieving convergence, and an operational transformation algorithm for achieving intention-preservation.

To achieve convergence, a total ordering relationship "$\Rightarrow$" among operations is defined [14]. However, operations are allowed to be executed in any order as long as their causality is preserved. When a new operation O is causally-ready for execution, (1) **undo** operations in the *HB* which totally *follow* O to restore the document to the state before their execution; (2) **do** O; and finally (3) **redo** all operations that were undone from the *HB*. It should be noted that the *undo/redo* operations involved in this scheme are *internal* operations, rather than *external* operations initiated from the user interface [12]. Therefore, the undo/do/redo scheme should be implemented in such a way that only the final result (instead of the intermediate ones) produced at the end of the undo/do/redo process is reflected on the user interface.

Transformation pre-/post-conditions

Since transformation functions in REDUCE are not responsible for ensuring convergence, they are not required to satisfy the same condition as in GROVE. In REDUCE, when operation O_a is transformed against operation O_b, it is required that the effect of the transformed operation O'_a on the document state that contains the impact of O_b should be the same as the effect of O_a on the document state that does not contain the impact of O_b. This type of transformation is called *Inclusion Transformation* (IT), since it transforms an operation O_a against another operation O_b in such a way that the impact of O_b is effectively included. The GROVE transformation functions can be regarded as a kind of inclusion transformation. Most importantly, it was recognized that the correctness of this inclusion transformation relies on the condition that both O_a and O_b are defined on the same document state [17], so their parameters are comparable and can be used to derive a proper adjustment to O_a. Failing to recognize and to ensure this condition is the root of the unsolved dOPT puzzle.

In search of a correct and sophisticated solution to intention-preservation, REDUCE introduced another type of transformation, called *Exclusion Transformation* (ET), which transforms O_a against another operation O_b in such a way that the impact of O_b is effectively excluded from O_a [17]. For example, O_4 and O_1 are independent operations but generated from different documents states, as shown in Fig. 1. When O_4 arrives at site 0, it is incorrect to simply transform O_4 against O_1. Instead, exclusion transformation should be applied on O_4 against its causally preceding operation O_2 to produce O'_4 in such a way that O_2's impact on O_4 is excluded. Consequently, O'_4 effectively shares the same document state with O_1, and then can be applied with the inclusion transformation against O_1.

To capture the required relationship between operations for *correct* transformation, the notion of operation *context* is introduced. The context of a document state is the sequence of operations executed on the initial document state to arrive at the current document state. Given an operation O, the *definition context* of O, denoted as $DC(O)$, is the context of the document state on which O is defined; and the *execution context* of O, denoted as $EC(O)$, is the context of the document state on which O is to be executed. The inten-

tion of an operation can be preserved if its definition context matches its execution context, i.e., $DC(O) = EC(O)$.

REDUCE uses two primitive transformation functions – $IT(O_a, O_b)$ and $ET(O_a, O_b)$ – to make an operation's definition context equivalent to its execution context. For specifying pre-/post-conditions of the transformation functions, two context-based relations are defined below (Note: a context is expressed as an operation list in the rest of the paper).

Definition 4: Context equivalent relation " $\sqcup$ "
Given two operations O_a and O_b, O_a and O_b are *context-equivalent*, i.e., $O_a \sqcup O_b$, *iff* $DC(O_a) = DC(O_b)$. □

Definition 5: Context preceding relation "$\mapsto$"
Given two operations O_a and O_b, O_a is *context preceding* O_b, i.e., $O_a \mapsto O_b$, *iff* $DC(O_b) = DC(O_a) + [O_a]$ (where "+" expresses the concatenation of two lists). □

With the context-based relations, the pre-/post-conditions of the two transformation functions are specified as follows.

Specification 1: $IT(O_a, O_b) : O'_a$
1. Precondition for input parameters: $O_a \sqcup O_b$.
2. Postcondition for output: $O_b \mapsto O'_a$, and the effect of O'_a in $DC(O'_a)$ is the same as the effect of O_a in $DC(O_a)$.

□

Specification 2: $ET(O_a, O_b) : O'_a$
1. Precondition for input parameters: $O_b \mapsto O_a$.
2. Postcondition for output: $O_b \sqcup O'_a$, and the effect of O'_a in $DC(O'_a)$ is the same as the effect of O_a in $DC(O_a)$.

□

The design of a pair of IT/ET functions for string-wise operations, which satisfy the specified post-conditions, can be found in [16, 17].

The GOT control algorithm

To ensure transformation pre-conditions a Generic Operational Transformation (GOT) control algorithm has been devised [17]. Taking a causally-ready operation O and its execution context $EC(O)$ (i.e., the current contents of the HB) as input parameters, the GOT control algorithm uses the IT/ET functions to transform O into EO (the execution form of O) such that $DC(EO) = EC(O)$.

Three cases have been distinguished and handled differently in the GOT control algorithm, as illustrated in Fig. 3. In this example, we assume $EC(O) = HB = [EO_1, EO_2, EO_3]$.

Case 1 : All operations in $EC(O)$ are causally preceding O. It must be that $DC(O) = EC(O)$, so that $EO = O$ (no transformation is performed).

Case 2 : Operations causally preceding O are listed in $EC(O)$ before operations independent of O. Since $EO_1 \rightarrow O$, $EO_2 \parallel O$, and $EO_3 \parallel O$, by transforming O against EO_2 and EO_3 in sequence, we get EO such that $DC(EO) = EC(O)$.

Case 3 : At least one causally-preceding operation is positioned after an independent operation in $EC(O)$. This is the case that the dOPT algorithm failed to handle correctly. Since $EO_1 \rightarrow O$, $EO_2 \parallel O$, and $EO_3 \rightarrow O$, it must be that $DC(O) = [EO_1, EO'_3]$, where EO'_3 is the original form of EO_3 when O was generated. Transforming O directly against any operation in $EO(O)$ would violate the pre-conditions for IT/ET functions. The strategy taken by the GOT algorithm is as follows: (1) apply exclusion transformation on EO_3 against EO_2 (both EO_3 and EO_2 are available in $EO(O)$) to obtain EO'_3, (2) apply exclusion transformation on O against EO'_3 to get an intermediate O'; and finally (3) apply inclusion transformation on O' against EO_2 and EO_3 in sequence, we get EO such that $DC(EO) = EC(O)$.

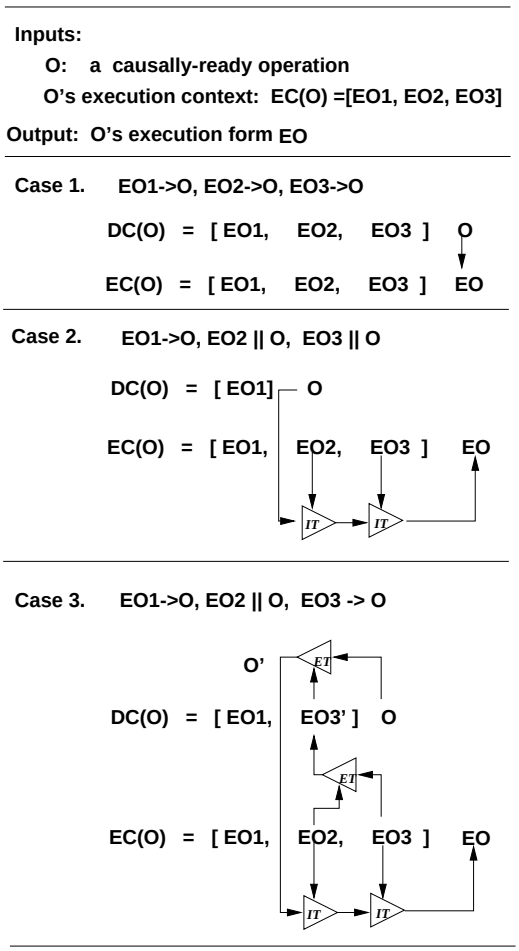


Fig. 3. Three cases analysis and handling by the GOT control algorithm

To describe the GOT algorithm, a few notations need to be introduced: Given a list of operations L, $L[i, j]$ denotes a sublist of L containing the operations from EO_i to EO_j inclusively; and L^{-1} denotes the reverse of L. $LIT(O, L)/LET(O, L)$ is used to denote the application of IT/ET function on operation O against a list of operations in L in sequence from left to right.

Algorithm 1: GOT(O, L): EO
O: a causally-ready operation
L: the list of operations $[EO_1, EO_2, ..., EO_m]$ in $EC(O)$.
EO: the execution form of O.

1. Scan $L[1, m]$ from left to right to find the first operation EO_k such that $EO_k \parallel O$. If no such an operation is found, then **return** $EO := O$.
2. Otherwise, scan $L[k + 1, m]$ to find operations causally

preceding O. If no single such operation is found, then **return** $EO := LIT(O, L[k,m])$.

3. Otherwise, let $L_1 = [EO_{c_1}, ..., EO_{c_r}]$ be the list of operations in $L[k,m]$ which are causally preceding O.
   (a) Get $L'_1 = [EO'_{c_1}, ..., EO'_{c_r}]$ as follows:
      i. $EO'_{c_1} := LET(EO_{c_1}, L[k, c_1 - 1]^{-1})$.
      ii. For $2 \leq i \leq r$,
         $O_t := LET(EO_{c_i}, L[k, c_i - 1]^{-1})$;
         $EO'_{c_i} := LIT(O_t, [EO'_{c_1}, ..., EO'_{c_{i-1}}])$.
   (b) $O' := LET(O, L'^{-1}_1)$.
   (c) **return** $EO := LIT(O', L[k,m])$.

□

It can be shown that the pre-conditions required by the transformation functions are always guaranteed by the GOT control algorithm. Therefore, if the post-conditions are always ensured by the transformation functions, then the GOT control algorithm will transform O into EO, so that the execution of EO on $EC(O)$ will preserve the intention of O. To achieves both intention-preservation and convergence, the GOT control algorithm has been integrated with the undo/do/redo scheme to form an undo/transform-do/transform-redo scheme [17].

A solution to the dOPT puzzle

In this section, we show how REDUCE solves the dOPT puzzle. We assume, without losing generality, the total ordering relationship "⇒" among the three operations in Fig. 2 is: $O_3 \Rightarrow O_1 \Rightarrow O_2$. Also, we assume the the REDUCE transformation function $IT(O_a, O_b)$ uses the following *shifting rule*: if both O_a and O_b are insertions and have the same position parameter, the position of O_a will be shifted. This shifting rule is consistent with the priority rule used in GROVE.

Under REDUCE, the operation transformation and the final document states (i.e., "xzy") at sites 3 and 1 are the same as they are under GROVE. The situation at site 2, however, is different: O_2 first inserts "y" into the document. Next, when O_3 arrives, O_2 has to be undone since $O_3 \Rightarrow O_2$. Then O_3 is executed as is, and O_2 is inclusively transformed against O_3 (according to $O_3 \parallel O_2$ and Case 2 in the GOT algorithm) to become $O'_2 = Insert[y, 2]$ according to the shifting rule. After the execution of both O_3 and O'_2, the document contains "zy". Finally, when O_1 arrives, O'_2 has to be undone since $O_1 \Rightarrow O'_2$. Then O_1 is executed as is (since $O_3 \rightarrow O_1$), and O'_2 is inclusively transformed against O_1 (according to $O_2 \parallel O_1$ and Case 2 in the GOT function) to become $O''_2 = Insert[y, 3]$. After the execution of O''_2, the document contains "xzy", which is identical to the document state at sites 3 and 1, and the intentions of all three operations are preserved. In this particular example, the exclusion transformation is not used, but in more complex scenarios, such as the one shown in Fig. 1, exclusion transformation is needed (see [17]).

THE JUPITER APPROACH

The Jupiter collaboration system was developed at Xerox PARC [11]. Since Jupiter has already had a central server for maintaining the states of objects (e.g., White-board, text documents, etc.) in the shared persistent virtual world, it is natural to use this central server for supporting consistency maintenance of shared objects as well. The Jupiter consistency maintenance algorithm was derived from the dOPT algorithm. The most interesting part of the Jupiter approach is the adaptation of the dOPT optimistic algorithm to an environment with multiple replicated clients sites *plus* one centralized server site.

In Jupiter, the shared documents are replicated at all cooperating client sites, which is the same as in GROVE. The difference is that the shared documents are also maintained at the central server and communications happen only between a client and the server (i.e., a 2-way communication). When an updating operation is generated at a client site, it is immediately executed at the local client site (for fast response to user actions), and then propagated to the central server. The server first transforms the incoming operation if necessary, then executes the transformed operation on its copy of the shared document, and finally broadcasts the transformed operation to all other client sites. Upon receiving an operation propagated from the central server, a client site may transform this operation if necessary, and then executes it on the local copy of the document. This star-like topology of communication eliminates the concern for ensuring causality (i.e., causality-violation never occurs). It also substantially simplifies the operational transformation control algorithm.

To achieve convergence, the Jupiter transformation function is required to satisfy the same property as that required by the dOPT algorithm. However, Jupiter uses a 2-dimensional *state space* graph, instead of a linear Log/HB, to keep track of all possible operation transformation paths to guide the selection of right operations for transformation. The Jupiter algorithm ensures that any pair of operations involved in a transformation must have originated from the same starting state in the state space graph, which is essentially the same as ensuring the *context equivalent* precondition by the GOT algorithm in the REDUCE approach. Therefore, the Jupiter algorithm is able to correct the dOPT algorithm under the condition that only 2-way communications are allowed in the system. An alternative approach to correcting the dOPT algorithm for the 2-way communication special case can be found in [1].

THE ADOPTED APPROACH

The adOPTed algorithm adopted the same correctness criteria from GROVE for consistency maintenance: convergence and precedence (i.e., causality-preservation). It also followed GROVE in using a fully distributed and replicated architecture. What is different in the adOPTed algorithm is that it requires an additional property for transformation functions to satisfy. Given two operations O_a and O_b, let $O'_a = T(O_a, O_b)$, and $O'_b = T(O_b, O_a)$, the transformation function T is required to possess the following two properties:

Transformation Property 1 (TP1) :

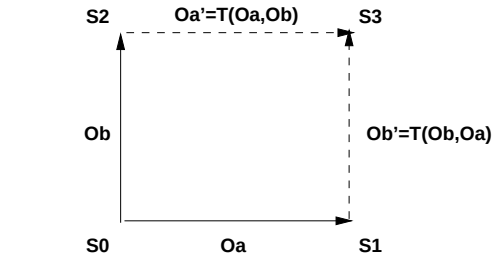
$$O_a \circ O'_b \equiv O_b \circ O'_a$$

Transformation Property 2 (TP2) : For any O,

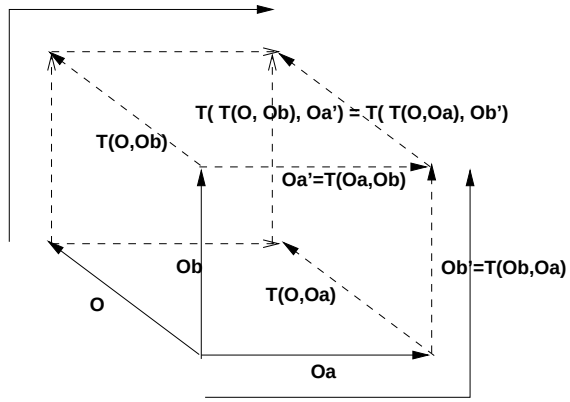
$$T(T(O, O_a), O'_b) = T(T(O, O_b), O'_a)$$

TP1 is the same as that required by the dOPT algorithm and the Jupiter algorithm, but TP2 is new in the adOPTed algorithm. TP2 ensures that the transformation of operation O along different paths will yield the same resulting operation. These two properties can be illustrated by using a

directed graph, called *interaction model* [13], as shown in Figure 4.



(a) Transformation property 1: : Oa o Ob' ≡ Ob o Oa'



(b) Transformation property 2: T(T(O, Oa), Ob') = T(T(O,Ob), Oa')

Fig. 4. Interaction model illustration of transformation properties.

The vertices of the interaction model graph are labeled by document states, and the edges are labeled by operations. For example, the four vertices of the square in Figure 4-(a) are labeled by four document states: S_0, S_1, S_2, and S_3, respectively; the two solid edges are labeled by two original operations: O_a and O_b, respectively; and the other two dashed edges are labeled by two transformed operations: $O'_a = T(O_a, O_b)$, and $O'_b = T(O_b, O_a)$, respectively. Essentially, TP1 ensures the unique vertices labeling, whereas TP2 ensures the unique edge labeling in the interaction model graph. It has been shown in [13] that TP1 and TP2 are the necessary and sufficient conditions for ensuring convergence in systems which allow N-way communication (where N is the number of cooperating sites).

The adOPTed algorithm used an N-dimensional interaction model graph to keeps track of all valid paths of operation transformations. The N-dimensional interaction model graph can be viewed as a generalization of the 2-dimensional state space in the Jupiter algorithm, and it also plays the same role in guiding the selection of the right path and right operations for transformation. The adOPTed algorithm ensures that any pair of operations involved in a transformation are defined on the same document state.

Using the adOPTed algorithm and the same transformation function T_{11} from GROVE, the solution to the dOPT puzzle can be illustrated in Figure 5. At sites 3 and 1, the operation transformation and execution follow the same path: O_3 and O_1 are executed as is, but O_2 is transformed against O_3 and O_1 in sequence, resulting in $O'_2 = Ins[y, 2]$, then $O''_2 = Ins[y, 3]$ (In the meantime, the adOPTed algorithm also produces $O'_3 = Ins[z, 1]$, and $O'_1 = Ins[x, 1]$, which are of no use at sites 3 and 1). The execution of O_3, O_1, and O''_2 in sequence results in the final document state "xzy". At site 2, a different path in the interaction model graph is taken. First, O_2 is executed as is. When O_3 arrives, it is transformed against O_2 to become $O'_3 = Ins[z, 1]$. Meanwhile, the adOPTed algorithm also transforms O_2 against O_3 to produce $O'_2 = Ins[y, 2]$, and both O'_3 and O'_2 are maintained at proper positions in the interaction model graph. When O_1 arrives, the adOPTed algorithm searches the interaction model graph to find the right operation O'_2 (instead of O_2, which was used in the dOPT algorithm) for transformation to get $O'_1 = Ins[x, 1]$. In the meantime, the adOPTed algorithm also produces and maintains $O''_2 = Ins[y, 3]$ at site 2 (O''_2 is of no use in this example). The execution of O_2, O'_3, and O'_1 in sequence results in an identical document state "xzy".

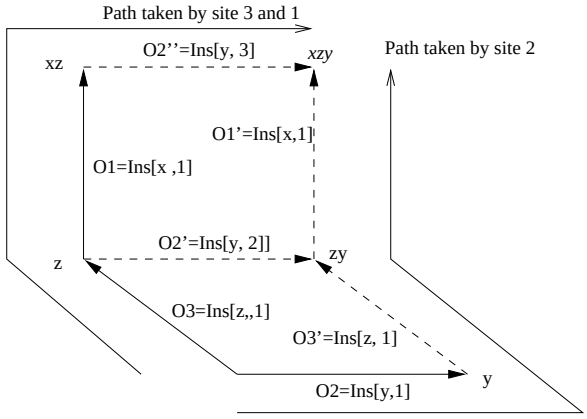


Fig. 5. The adOPTed solution to the dOPT puzzle

AN OPTIMIZED ALGORITHM: GOTO

Without requiring TP1 and TP2, the GOT control algorithm, integrated with the undo/do/redo scheme [17], is the only known solution for achieving both intention-preservation and convergence. An interesting question is: what could the GOT algorithm achieve if TP1 and TP2 are satisfied by IT/ET functions? In this section, we will answer this question and propose a new optimized GOT control algorithm.

To take advantage of the two additional post-conditions TP1 and TP2, we modify the original context-based relations in Definitions 4 and 5 as follows: replace the *equal* sign "=" with the *equivalence* sign "≡". Obviously, the equal relation "=" between operation contexts is a special case of the equivalence relation "≡". With this generalization of context-based relations and the extension of pre-/post-conditions for IT/ET functions, we found that the original GOT control algorithm can ensure both intention-preservation and convergence, without integrating with the undo/do/redo scheme or using a multi-dimensional graph. The verification of this claim can follow similar reasonings as used in [13], which is, however, beyond the scope of this paper.

Moreover, the two additional post-conditions TP1 and TP2 can be employed to optimize the GOT control algorithm

by reducing the number of IT/ET transformations. The optimized algorithm, named as GOTO (GOT Optimized), resembles the original GOT algorithm in handling the first and the second cases (see Fig. 3). For the third case, the handling is different. In addition to performing transformations on the definition context of O, we also perform transformations on the execution context of O to make the two contexts equivalent. This can be achieved by executing the following two steps:

1. Transform execution context $EC(O)$ into such an *equivalent* $EC(O)'$ that all operations causally preceding O are positioned before independent operations in $EC(O)'$. Let $EC(O)' = EC(O)'.left + EC(O)'.right$, where $EC(O)'.left$ is the sublist of causally preceding operations, and $EC(O)'.right$ is the sublist of independent operations.
2. Apply the inclusion transformation on O against the list of independent operations in $EC(O)'.right$. The transformation pre-condition is satisfied because $EC(O)'.left \equiv DC(O)$.

The question now is: how to transform $EC(O)$ into such an equivalent $EC(O)'$?

By using IT and ET functions, the *Transpose* function is defined to transform and swap two operations in an execution context.

Function 1: $Transpose(O_a, O_b) : O'_b, O'_a$
{
  $O'_b := ET(O_b, O_a)$;
  $O'_a := IT(O_a, O'_b)$;
  return (O'_b, O'_a);
}

The pre-condition for O_a and O_b is: $O_a \mapsto O_b$. The post-condition for O'_a and O'_b is: $O'_b \mapsto O'_a$. Based on the Transpose function, function $LTranspose(L)$ is defined, which transforms and circularly shifts the list of operations in L.

Procedure 1: $LTranspose(L)$
{
  for (i = $|L|$; $i > 1$; i- -)
    $(L[i-1], L[i]) := Transpose(L[i-1], L[i])$;
}

According to TP1 and TP2, and the definition of *Transpose*, it must be that $L \equiv L'$, where L' is the list of operations after calling $LTranspose(L)$.

As an example, the handling of case 3 by the GOTO algorithm is shown in Fig. 6. In this example, we can transpose EO_2 and EO_3 in $EC(O)$ by calling $Transpose(EO_2, EO_3)$, so that an equivalent execution context $EC(O)' = [EO_1, EO'_3, EO'_2]$ can be obtained. Then, since $DC(O) \equiv [EO_1, EO'_3]$, we can apply an inclusion transformation on O against EO'_2 to get EO, such that $DC(EO) \equiv EC(O)'$. To transform O into EO in this example, three IT/ET transformations (one Transpose function costs one IT and one ET transformations) are needed under the GOTO control algorithm, whereas four IT/ET transformations are needed under the GOT control algorithm.

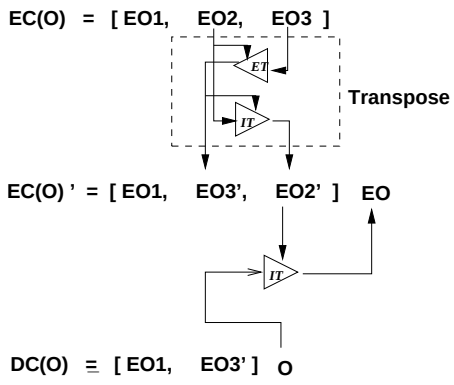


Fig. 6. The handling of mixed independent and dependent operations by the GOTO control algorithm

Algorithm 2: GOTO(O, L): EO
O: a causally-ready operation
L: the list of operations $[EO_1, EO_2, ..., EO_m]$ in $EC(O)$.
EO: the execution form of O.

1. Scan $L[1, m]$ from left to right to find the first operation EO_k such that $EO_k \parallel O$. If no such an operation is found, then **return** $EO := O$.
2. Otherwise, scan $L[k+1, m]$ to find operations causally preceding O. If no single such operation is found, then **return** $EO := LIT(O, L[k, m])$.
3. Otherwise, let $L_1 = [EO_{c_1}, ..., EO_{c_r}]$ be the list of operations in $L[k, m]$ which are causally preceding O.
   (a) For $1 \le i \le r$:
       $LTranspose(L[k+i-1, c_i])$;
   (b) **return** $EO := LIT(O, L[k+r, m])$.

□

It can be shown that the pre-conditions required by the transformation functions are always guaranteed by the GOTO control algorithm. Therefore, if the post-conditions, including TP1 and TP2, are always ensured by the transformation functions, then the GOTO control algorithm will transform O into EO, so that the execution of EO on $EC(O)$ will preserve the intention of O and ensure convergence.

CONCLUSIONS AND FUTURE DIRECTIONS

Many people have experiences of using various editors. Not so many people have recognized that there would exist many interesting research issues in an editor when used in a real-time collaborative context. Even less people have come to learn that some research issues in real-time group editors, such as consistency maintenance, would be so challenging that a decade exploration would not be enough to exhaust their research potential. In this paper, we have reviewed a number of major operational transformation algorithms for consistency maintenance in real-time group editors, including the dOPT algorithm, the GOT algorithm, the Jupiter algorithm, and the adOPTed algorithm, and have proposed a new optimized transformation control algorithm – the GOTO algorithm. In this concluding section, we summarize the major achievements in the past decade on the transformation-based

consistency maintenance techniques and point out the major open issues for further exploration.

Major achievements

Three inconsistency problems – divergence, causality-violation, and intention-violation – have been identified and explored. Particularly, the non-serializable intention violation problem has been distinguished from the serializable divergence problem. Corresponding to these three problems, consistency correctness criteria consist of three properties: convergence, causality-preservation, and intention-preservation. It is useful to integrate these three properties in a consistency model, which effectively specifies what consistency has been promised to the system users and what properties must be supported by the underlying system algorithms.

The discovery of the necessary transformation pre-conditions has been a significant step toward the design of correct transformation control algorithms. The notion of operation context is very useful in capturing the required relationship between operations for correct transformation. Alternative approaches to ensuring transformation pre-conditions include the GOT/GOTO control algorithms working on an 1-dimensional history buffer, the Jupiter algorithm working on a 2-dimensional state space graph, and the adOPTed algorithm working on a N-dimensional interaction model graph.

Two types of transformation functions have been proposed: inclusion and exclusion transformations. For algorithms that use a multi-dimensional data structure to keep track of operations in their original, intermediate, and executed forms, such as the Jupiter and adOPTed algorithms, only inclusion transformation is needed. For algorithms that use an 1-dimensional history buffer to save operations in their executed form only, such as the GOT and GOTO algorithms, apart from inclusion transformation, exclusion transformation is needed to recover operations' original and intermediate forms from their executed forms.

The identification of proper transformation post-conditions has played a crucial role in the design of both the generic transformation control algorithms and application dependent transformation functions. By requiring context-based post-conditions, the GOT control algorithm can achieve intention-preservation. The context-based post-conditions, however, do not capture the conditions for ensuring convergence, so the GOT control algorithm must be integrated with an undo/do/redo scheme to achieve convergence. In essence, undo/redo can also be viewed as a kind of transformation, which is performed directly on the document states rather than on the operations. By requiring TP1 only, the Jupiter algorithm can achieve convergence in systems which are restricted to 2-way communication. By requiring both TP1 and TP2, the adOPTed algorithm achieves convergence in systems which allow N-way communication. Neither TP1 nor TP2, however, captures the conditions for ensuring intention-preservation, so intention-preservation has been *implicitly* handled by transformation functions in the dOPT algorithm, the Jupiter algorithm, and the adOPTed algorithm. By requiring both TP1 and TP2, in addition to the context-based post-conditions, the GOT control algorithm alone is able to achieve both intention-preservation and convergence. By performing transformations on both definition and execution contexts, the GOTO algorithm is able to optimize the GOT algorithm by reducing the number of transformations.

Open issues and future directions

The correctness of the whole operational transformation scheme relies on the satisfaction of both transformation pre-conditions and post-conditions. Lots of work have been done on the design of correct generic transformation control algorithms to ensure transformation pre-conditions. However, not much work has been done on the design of application-dependent transformation functions which could really ensure transformation post-conditions [16]. We have learned that TP1 and TP2 have to be satisfied by transformation functions in order to ensure convergence, but we know little about how to verify whether an existing transformation function really satisfies TP1 and TP2. In fact, as illustrated in [17], some seemingly correct transformation functions do not really satisfy TP1 and TP2. More serious attention should be given to the design of transformation functions to better understand the intrinsic interactions (in the form of pre-/post-conditions) between transformation functions and transformation control algorithms.

Research should also be directed toward formal specification and verification of operational transformation concepts, properties, and algorithms. This formalization and verification is necessary for rigorously proving the correctness of the algorithms and for analyzing and improving the time and space complexities of existing algorithms. In [1], a Calculus for Concurrent Update (CCU) has been derived from the dOPT algorithm as a tool for the purpose of formal modeling and verification of consistency-preserving operational transformation. The *Team Automata* [6] is another mathematical model for describing the interaction of a groupware system components. More work needs to be done in developing and applying innovative theoretical tools to verify operational transformation algorithms and systems.

Future research should distinguish and explore two types of consistencies: one is *syntactic* consistency, which is concerned with whether all sites have the same view of the shared objects, regardless of whether the common view makes sense in the application context; and the other is *semantic* consistency, which is concerned with whether all sites have the same view of the shared objects, as well as whether the common view makes sense in the application context. There may exist many levels of syntactic consistency and semantic consistency in a particular application context. Previous work has mainly explored issues related to syntactic consistency. Particularly, the term *intention* as defined in [14, 17] and used in this paper has captured only a small piece of the much richer meaning of intention from the human user's perspective. This brings up interesting areas of research concerned with characterization and preservation of the human user's intentions in collaborative contexts, or *group intentions*. It may be infeasible for the system alone to automatically determine the human group intentions for different groups with divergent group goals. The system, however, could and should have mechanisms to help the group users decide their group intentions and resolve their conflicts. In general, we advocate a groupware system design paradigm, which builds a sufficient amount of generic supporting *mechanism* into the system, but leave the high level collaboration *policy* decisions up to the system users. A good

groupware system should be easily tunable by its users for supporting various collaboration needs [2, 10].

A lot of efforts have been putted on achieving the shortest response time (as short as single user editors), but not much research has been done on notification policy – when and how to make local updates public to achieve global consistency. Alternatives to notifying remote sites immediately after executing an operation at the local site include periodic notification, notification on demand, greying out the screen to tell user that the displayed information is out-of-date, etc. Future research should be conducted on mechanisms for supporting alternative notification policies and their applicability in different application environments.

Operation granularity is another unexplored issue. Current transformation algorithms are only capable of handling fine-grain primitive operations, such as *Insert* and *Delete*. Useful editors, however, must offer to the end user higher level compound operations, such as *Move*, and *Replace*. On one hand, the system needs additional mechanisms to support coarse-grain compound operations as an atomic sequence of primitive operations while still ensuring consistency properties. The richer semantics of the compound operations, on the other hand, could help the system to better understand and preserve the user's intentions.

A number of prototype group editors have been built in the past by various research groups for testing the feasibility of transformation-based consistency maintenance algorithms, and for investigating system design and implementation issues. GROVE has been used in several real groups for a variety of design activities to evaluate the system from users perspective and to gain usage experience [4, 5]. Since then, however, little has been reported on using this type of system in real-life collaborative environments to study the user's working modes in using the system, and to conduct statistics analysis of conflicts. Much more research efforts should be directed toward better understanding the potential effects of this type of system on people, their work and interactions.

Although all the transformation-based consistency maintenance algorithms and functions were designed in the context of text editing, many of them are actually quite general and potentially applicable in other domains of group editing. It would be interesting and useful to apply operational transformation in graphics/image/multimedia editors to further validate the generic algorithms and to gain more insights in the design and application of these types of systems. Even techniques used in transforming a sequence of characters could potentially be applicable in other real-time groupware systems, which allow concurrent insertion/deletion of any sequence of objects with a linearly ordered relationship. Moreover, operational transformation has been found very useful in supporting user-initiated collaborative undo operations [12, 13].

Consistency maintenance is a fundamental issue in many areas of computing systems, including operating systems, databases systems, distributed shared memory systems, and groupware systems. Research on real-time group editors, as a special class of distributed systems supporting human-computer-human interactions, has drawn inspirations from traditional distributed computing techniques (e.g., causal/total ordering of events, state-vector timestamping, serialization, etc.), and has also invented the non-traditional operational transformation technique to address its special issues, such as intention-preservation. The generalization and application of this unique operational transformation technique to other areas of distributed computing and CSCW is an exciting direction for future exploration.

References

[1] C. V. Cormack: "A calculus for concurrent update," Research Report CS-95-06, Dept. of Computer Science, University of Waterloo, Canada, 1995.

[2] P. Dourish: "Consistency guarantees: exploiting application semantics for consistency management in a collaborative tookit," In *Proc. of ACM Conference on Computer Supported Cooperative Work*, pp. 268-277, Nov. 1996.

[3] 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.

[4] C. A. Ellis, S. J. Gibbs, G.L. Rein: "Design and use of a group editor," In *Engineering for Human-Computer Interaction*. G. Cockton, Ed., North-Holland, Amsterdam, 1990, pp.13-25.

[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] C. A. Ellis: "Team Automata for Groupware Systems," In *Proc. of ACM Conference on Supporting Group Work*, pp.415-424, Nov. 1997.

[7] S. Greenberg and 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.

[8] A. Karsenty and M. Beaudouin-Lafon: "An algorithm for distributed groupware applications," In *Proc. of 13th International Conference on Distributed Computing Systems*, pp. 195-202, May 1993.

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

[10] J. Munson and P. Dewan: "A concurrency control framework for collaborative systems," In *Proc. of ACM Conference on Computer Supported Cooperative Work*, pp. 278-287, Nov. 1996.

[11] D. Nichols, P. Curtis, M. Dixon, and J. Lamping: "High-latency, low-bandwidth windowing in the Jupiter collaboration system," In *Proc. of ACM Symposium on User Interface Software and Technologies*, pp. 111-120, Nov. 1995.

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

[13] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenbauser:"An integrating, transformation-oriented approach to concurrency control and undo in group editors," In *Proc. of ACM Conference on Computer Supported Cooperative Work*, pp 288-297, Nov. 1996.

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

[15] C. Sun, X. Jia, Y. Zhang, and Y. Yang:"A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems," In *Proc. of ACM Conference on Supporting Group Work*, pp. 425-434, Nov. 1997.

[16] C. Sun, D. Chen, X. Jia: "Reversible inclusion and exclusion transformation for string-wise operations in cooperative editing systems," In *Proc. of The 21st Australasian Computer Science Conference*, pp.441-452, Springer-Verlag, Perth, Feb. 1998.

[17] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen: "Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems," *ACM Transactions on Computer-human Interaction*, 5(1), March 1998, pp.63-108.