Operation Context and Context-based Operational Transformation

David Sun
Computer Science Division, EECS
University of California
Berkeley, CA
davidsun@cs.berkeley.edu

Chengzheng Sun
School of Computer Engineering
Nanyang Technological University
Singapore
CZSun@ntu.edu.sg

ABSTRACT

Operational Transformation (OT) is a technique for consistency maintenance and group undo, and is being applied to an increasing number of collaborative applications. The theoretical foundation for OT is crucial in determining its capability to solve existing and new problems, as well as the quality of those solutions. The theory of causality has been the foundation of all prior OT systems, but it is inadequate to capture essential correctness requirements. Past research had invented various patches to work around this problem, resulting in increasingly intricate and complicated OT algorithms. After having designed, implemented, and experimented with a series of OT algorithms, we reflected on what had been learned and set out to develop a new theoretical framework for better understanding and resolving OT problems, reducing its complexity, and supporting its continual evolution. In this paper, we report the main results of this effort: the theory of operation context and the COT (Context-based OT) algorithm. The COT algorithm is capable of supporting both do and undo of any operations at anytime, without requiring transformation functions to preserve Reversibility Property, Convergence Property 2, Inverse Properties 2 and 3. The COT algorithm is not only simpler and more efficient than prior OT control algorithms, but also simplifies the design of transformation functions. We have implemented the COT algorithm in a generic collaboration engine and used it for supporting a range of novel collaborative applications.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative computing; Synchronous interaction*

General Terms

Algorithms, Design, Theory

Keywords

Operation context, OT, context-based OT, consistency maintenance, undo, group editors, distributed applications.

CSCW'06, November 4–8, 2006, Banff, Alberta, Canada.

1. INTRODUCTION

Operational Transformation (OT) was originally invented for consistency maintenance in plain-text group editors [4]. In over 15 years, OT has evolved to support an increasing number of applications, including group undo [15, 19, 18, 21], group-awareness [28], operation notification and compression [20], spreadsheet and table-centric applications [14, 27], HTML/XML and tree-structured document editing [3, 7], word processing and slide creation [29, 25, 24], transparent and heterogenous application-sharing [1, 10, 24], and mobile replicated computing and database systems [6, 16].

To effectively and efficiently support existing and new applications, we must continue to improve the capability and quality of OT in solving both old and new problems. The soundness of the theoretical foundation for OT is crucial in this process. One theoretical underpinning of all existing OT algorithms is *causality/concurrency* [9, 17, 4, 22]: causally related operations must be executed in their causal order; concurrent operations must be transformed before their execution. However, the theory of causality is inadequate to capture essential OT conditions for correct transformation.

The limitation of the causality theory had caused correctness problems from the very beginning of OT. The dOPT algorithm was the first OT algorithm and was based solely on the concurrency relationships among operations [4]: a pair of operations are transformable as long as they are concurrent. However, later research discovered that the concurrency condition alone is not sufficient to ensure the correctness of transformation. Another condition is that the two concurrent operations must be defined on the same document state. In fact, the failure to meet the second condition was the root of the dOPT-puzzle [22]. This puzzle was solved in various ways, but the theory of causality as well as its limitation were inherited by all follow-up OT algorithms.

The causality theory limitation became even more prominent when OT was applied to solve the undo problem in group editors. The concept of causality is unsuitable to capture the relationships between an inverse operation (as an interpretation of a meta-level undo command) and other normal editing operations. In fact, the causality relation is not defined for inverse operations (see Section 2). Various patches were invented to work around this problem, resulting in more intricate complicated OT algorithms [18, 21].

After having designed, implemented, and experimented with a series of OT algorithms of increased complexity, we reflected on what had been learned and set out to develop a uniformed theoretical framework for better understanding and resolving OT problems, reducing its complexity, and

supporting its continual evolution. In this paper, we report the main results of this effort: the theory of operation context and the COT (Context-based OT) algorithm.

The rest of this paper is organized as follows. First, we define causal-dependency/-independency and briefly describe their limitations in Section 2. Then, we present the key elements of the operation context theory, including the definition of operation context, context-dependency/-independency relations, context-based conditions, and context vectors in Section 3. In Section 4, we present the basic COT algorithm for supporting consistency maintenance (do) and group undo under the assumption that underlying transformation functions are able to preserve some important transformation properties. Then, these transformation properties and their pre-conditions are discussed in Section 5. The COT solutions to these transformation properties are presented in Section 6. Comparison of the COT work to prior OT work, OT correctness issues, and future work are discussed in Section 7. Finally, major contributions of this work are summarized in Section 8.

2. LIMITATIONS OF CAUSALITY

The theory of *causality* is central to distributed computing and to the design of all existing OT algorithms. Following Lamport [9], causal-dependency/-independency relations among editing operations can be defined in terms of their generation and execution sequences [4, 23].

Definition 1. *Causal-dependency relation "$\rightarrow$"*

Given two operations O_a and O_b, generated at sites i and j, O_b is *causal-dependent* on O_a, denoted by $O_a \rightarrow O_b$, *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. *Causal-independency relation " $\parallel$ "*

Given two operations O_a and O_b, O_a and O_b are *causal-independent* or *concurrent*, denoted by $O_a \parallel O_b$, *iff* neither $O_a \rightarrow O_b$, nor $O_b \rightarrow O_a$. □

Just as Vector Logical Clocks are used for capturing casuality in distributed systems [17], State Vectors have been used for capturing causal relationships among operations and for representing document states in OT systems [4, 19, 23].

To illustrate causal relations among operations, consider a real-time group editing session with two sites in Figure 1. There are three editing operations in this scenario (the undo command $Undo(O_2)$ and its relation with other operations shall be explained later): O_1 generated at site 0, and O_2 and O_3 generated at site 1. According to Definitions 1 and 2, we have $O_2 \rightarrow O_3$ because the generation of O_2 happened before the generation of O_3; $O_1 \parallel O_2$ and $O_1 \parallel O_3$ because for each pair, neither operation's execution happened before the other operation's generation.

In the following discussion, we shall use the term *IT-transform* to mean the use of the *IT (Inclusion Transformation)* function: $IT(O_a, O_b)$, which transforms operation O_a against operation O_b in such a way that the impact of O_b is effectively included in O_a [23]. This term is introduced to differentiate this special transformation function from other steps involved in a transformation process.

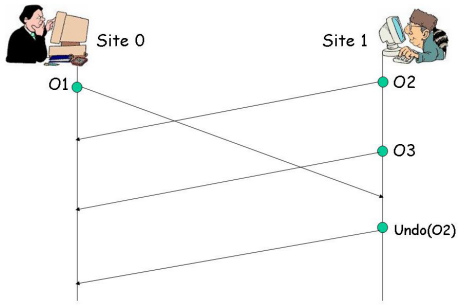

Figure 1: A real-time group editing scenario.

The scenario in Figure 1 (without the undo command) has often been used to illustrate the dOPT-puzzle. Under the dOPT algorithm [4], when O_2 arrives at site 0, it will be IT-transformed against O_1 since $O_2 \parallel O_1$; this is correct because O_2 and O_1 are defined on the same (initial) document state. When O_3 arrives at site 0, it will also be IT-transformed against O_1 since $O_3 \parallel O_1$; but this is incorrect because O_3 is defined on the document state that contains the effect of O_2, whereas O_1 is defined on the initial document state. In this case, the parameters of O_3 and O_1 are not comparable and hence may not be IT-transformed correctly. The solution to this puzzle is first to IT-transform O_1 against O_2 to produce O_1', which is defined on the document state including the effect of O_2 (the same state on which O_3 is defined), and then to IT-transform O_3 against O_1' [22].

From Definitions 1 and 2, it is clear that the causal-dependency relation is only defined for original operations (e.g. O_1, O_2 and O_3) directly generated by users, but not for transformed operations (e.g. O_1'). Furthermore, the concurrency relation does not capture the essential condition for correct IT-transformation: the two input operations must be defined on the same document state [23].

Another major limitation of causality is its unsuitability for capturing OT conditions for inverse operations. The $Undo(O_2)$ command in Figure 1 is interpreted as an inverse operation $\overline{O_2}$. The correct undo effect for $\overline{O_2}$ is to eliminate the effect of O_2 but retain the effects of other operations (i.e. O_1 and O_3) [21]. To achieve this effect, $\overline{O_2}$ needs to be treated as an operation defined on the document state including the effect of O_2 but not O_1 and O_3, so that $\overline{O_2}$ can be transformed against O_1 and O_3 before its execution. However, according to Lamport's *happen-before* relation [9], $Undo(O_2)$ is causally dependent on O_1, O_2, and O_3. If $\overline{O_2}$ was to inherit the causal relation of $Undo(O_2)$, then it would be effectively treated as an operation defined on the document state with the effects of all three operations O_1, O_2, and O_3, which would prohibit $\overline{O_2}$ from being transformed against any operation, thus failing to achieve the correct undo effect. Moreover, after executing an inverse operation like $\overline{O_2}$, the document state can no longer be properly represented by the state vector, which is only capable of representing original normal editing operations.

3. OPERATION CONTEXT

3.1 Basic concept

Conceptually, each operation O is associated with a *context*, denoted by $C(O)$, which corresponds to the document

state on which the operation is defined. The significance of operation context is twofold: (1) an operation can be correctly executed only if its context and the current document state are the same; and (2) an operation can be correctly IT-transformed against another operation only if the contexts of these two operations are the same.

In Figure 1, both O_1 and O_2 are defined on the same initial document so they are associated with the same context; O_3 is defined on the document state which includes the effect of O_2, so $C(O_3)$ is different from $C(O_1)$ or $C(O_2)$. When O_2 arrives at site 0, it cannot be executed as-is since $C(O_2)$ does not match the current document state at site 0 which includes the effect of O_1. O_2 can be correctly IT-transformed against O_1 since their contexts corresponds to the same initial document state. When O_3 arrives at site 0, it cannot be executed as-is either since $C(O_3)$ does not match the current document state at site 0 which includes the effects of both O_1 and O_2. O_3 cannot be correctly IT-transformed against O_1 since their contexts are different, which is the root of the dOPT-puzzle. As discussed in Section 2, $Undo(O_2)$ should be interpreted as an inverse $\overline{O_2}$ defined on the document state with the effect of O_2 only.

3.2 Set representation of operation context

To facilitate comparison and manipulation of operation contexts for correct execution and transformation, it is necessary to *explicitly* represent operation context.

In OT systems, there are two different kinds of operation: *original* operations which are generated by users, and *transformed* operations which are the outcomes of some transformations. Original operations can be further divided into two classes: *normal* operations which are generated to *do* something, and *inverse* operations which are generated to *undo* some executed operations. For any operation O, its inverse is denoted by $\overline{O}$. Since every transformed operation must come from an original operation, we use the notation $org(O)$ to denote the original operation of O. If O is an original operation, then $org(O) = O$.

Since the context of an operation corresponds to the document state on which the operation is defined, the problem of context representation can be reduced into the problem of document state representation. In an OT-based group editor, each document state can be uniquely represented by the set of original operations executed so far on the document. These original operations may be executed in different orders or in different (original or transformed) forms at different sites, but the same document state must be achieved (according to the convergence requirement [23]). We use original (normal and inverse) operations, rather than their transformed versions, to represent a document state.

Definition 3. *Document state representation*

A document state can be represented by DS as follows:

1. The initial document state is represented by $DS = \{\}$.
2. After executing an operation O of any type on the document state represented by DS, the new document state is represented by $DS' = DS \cup \{org(O)\}$. □

This presentation does not specify what execution forms the original operations in DS should take to bring the document to the current state, but it captures essential and sufficient information for detecting whether two document states are the same and for deriving their differences in terms of original operations.

Based on the document state representation, the context of an original normal operation should be the same as the representation of the document state from which this operation was generated. To achieve the undo effect in [21], an original inverse operation $\overline{O}$ should be defined on the document state $DS = C(O) \cup \{O\}$, which is the state after executing the original operation O on the state $C(O)$. According to the definition of the IT function [23], a transformed operation O', where $O' = IT(O, O_x)$, should be defined on the document state $DS = C(O) \cup \{org(O_x)\}$, which is the state achievable by executing O_x on the state $C(O)$. More precisely, the context of an operation is defined blow.

Definition 4. *The context of an operation*

1. For an original normal operation O, $C(O) = DS$, where DS is the representation of the document state from which O was generated.
2. For an original inverse operation $\overline{O}$, $C(\overline{O}) = C(O) \cup \{O\}$, where O is the operation to be undone.
3. For a transformed operation O', $C(O') = C(O) \cup \{org(O_x)\}$, where $O' = IT(O, O_x)$. □

According to the above definition, the context of any type of operation can be represented as a set of original operations. For the scenario in Figure 1, we have $C(O_1) = \{\}$, $C(O_2) = \{\}$, and $C(O_3) = \{O_2\}$ according to Definition 4-Item 1. According to Definition 4-Item 2, we have $C(\overline{O_2}) = \{O_2\}$. From $O_2' = IT(O_2, O_1)$, we have $C(O_2') = \{O_1\}$ according to Definition 4-Item 3.

3.3 Context-dependency/-independency

We define the context-dependency/-independency relation among operations in terms of whether an original operation is included in the context of another operation of any type.

Definition 5. *Context-dependency relation "$\overset{c}{\rightarrow}$"*

Given an original operation O_a and an operation O_b of any type, O_b is *context-dependent* on O_a, denoted by $O_a \overset{c}{\rightarrow} O_b$, *iff*: (1) $O_a \in C(O_b)$; or (2) there exists an original operation O_x, such that $O_a \in C(O_x)$ and $O_x \in C(O_b)$. □

It should be noted that the context-dependency relation is defined only between an original (either normal or inverse) operation and another operation of any type (original or transformed). This is because any operation has a context, but only original operations can be included in a context.

Definition 6. *Context-independency relation "$\overset{c}{\parallel}$"*

Given two original operations O_a and O_b, O_a and O_b are *context-independent*, denoted by $O_a \overset{c}{\parallel} O_b$, *iff* neither $O_a \overset{c}{\rightarrow} O_b$, nor $O_b \overset{c}{\rightarrow} O_a$. □

It can be shown that if both O_a and O_b are original normal operations, then $O_a \overset{c}{\rightarrow} O_b$ is equivalent to $O_a \rightarrow O_b$; and $O_a \overset{c}{\parallel} O_b$ is equivalent to $O_a \parallel O_b$. In other words, the causal-dependency/-independency relation is a special case of the context-dependency/-independency relation.

3.4 Context-based conditions

The following Context-based Conditions (CC) capture essential requirements for operation execution and transformation in OT systems:

CC1: *$C(O) \subseteq DS$ is a necessary condition for an original operation O to be transformed to the document state DS for execution.*

CC1 ensures that O is always executed after the context-dependent operations included in $C(O)$. In other words, for any original operation O_x, if $O_x \overset{c}{\rightarrow} O$, then O_x must be executed before O. When O is an original normal operation, all operations which are causally before O must be included in $C(O)$ (according to Definition 1 and Definition 5), so CC1 preserves the causal ordering among original normal operations [4, 22]. When O is an original inverse operation, $C(O)$ must include the operation to be undone by O (see Definition 4-Item 2), so CC1 preserves the *do-undo* ordering among normal and inverse operations [21].

CC2: *$DS - C(O)$[1] is the set of operations that O must be transformed against before O is executed on the document state DS.*

CC2 ensures that O is transformed against all context-independent operations in DS before its execution. It can be shown that, for any O_x in $DS - C(O)$, it must be that $O_x \overset{c}{\parallel} O$. When O is an original normal operation, $DS - C(O)$ must include all executed operations which are concurrent with O, so CC2 covers the condition that O should be transformed against concurrent operations [4, 22]. When O is an inverse operation, CC2 covers the condition that O should be transformed against all operations which are executed after the operation to be undone by O [21].

CC3: *$C(O) = DS$ is a necessary condition for O to be executed on the document state DS.*

CC3 is required for correctly executing operations.

CC4: *$C(O_a) \subseteq C(O_b)$ is a necessary condition for O_a to be IT-transformable to the new context given by $C(O_b)$.*

CC4 is required because if $C(O_a) \not\subseteq C(O_b)$, then there must be an operation $O_x \in C(O_a)$ but $O_x \notin C(O_b)$, which means O_a cannot be IT-transformed to the new context $C(O_b)$ since IT-transformation cannot remove this O_x from $C(O_a)$ (see Definition 4-item3).

CC5: *$C(O_b) - C(O_a)$ is the set of operations that O_a must be transformed against before IT-transformed against O_b.*

CC5 ensures that O_a is transformed against context-independent operations in $C(O_b)$ before IT-transformed against O_b. It can be shown that, for any O_x in $C(O_b) - C(O_a)$, it must be that $O_x \overset{c}{\parallel} O_a$,

CC6: *$C(O_a) = C(O_b)$ is a necessary condition for O_a to be IT-transformed against O_b.*

CC6 is required for correctly applying IT functions.

In summary, CC1 and CC4 are required for ensuring correct ordering of operation execution/transformation; CC2 and CC5 are required for selecting correct transformation target operations; and CC3 and CC6 are required for ensuring correct operation execution/transformation. These context-based conditions form the foundation for the COT algorithm to be presented in Section 4 and Section 6.

[1] $DS - C(O)$ is the set difference between DS and C(O).

3.5 Context vector

An important element of the operation context theory is the *context vector*, which represents the set of operations of a context in an efficient way. For notational convenience, we assume that a collaborative editing session consists of N collaborating sites, identified by $0, 1, \ldots, N-1$.

3.5.1 Representing original normal operations

Original normal operations generated at each site are strictly sequential, so each of them can be uniquely identified by a pair of integers *(sid, ns)*, where *sid* is the site identifier and *ns* is the local sequence number of this operation.

Let O_{ij} be an original normal operation generated at site i with a sequence number j. If O_{ij} is included in a context $C(O)$, then $O_{i1}, O_{i2}, \ldots, O_{ij-1}$ must also be included in $C(O)$ according to Definition 3 and Definition 4. Therefore, all normal operations generated at the same site can be sufficiently characterized by the largest sequence number of these operations. All original normal operations in a context can be partitioned into N groups according to their generation sites, so N integers are needed for representing original normal operations in a context.

3.5.2 Representing original inverse operations

An original inverse operation can be generated to undo an original normal operation, or to redo an undone operation. Each original inverse operation directly or indirectly corresponds to exactly one original normal operation. For example, inverse operation $\overline{O}$ may be generated to undo O, and $\overline{\overline{O}}$ may be generated to undo $\overline{O}$. Both $\overline{O}$ and $\overline{\overline{O}}$ correspond to the same normal operation O. Based on this observation, all original inverse operations in an operation context can be grouped by their corresponding original normal operations: one inverse group for each undone original normal operation.

Inverse operations in the same inverse group can be further differentiated by a sequence number based on their execution order within this group. For example, $\overline{O}$ and $\overline{\overline{O}}$ are in the same inverse group corresponding to O, so $\overline{O}$ has the sequence number "1", and $\overline{\overline{O}}$ has the sequence number "2".

In general, an inverse can be identified by a triple (sid, ns, is), where *sid* and *ns* are the site identifier and sequence number of the corresponding normal operation, and *is* is the inverse sequence number within the group. Since inverses are sequentially executed, the largest sequence number in the group can be used to represent all inverses in the group.

Inverse groups can be further partitioned into N inverse clusters according to the site identifiers of their corresponding normal operations. The inverse cluster at site i – ic_i – can be expressed as follows:

$$ic_i = [(ns_0, is_0), (ns_1, is_1), \ldots, (ns_{k-1}, is_{k-1})],$$

where each pair (ns_j, is_j), $0 \leq j < k$, represents an inverse group with is_j inverse operations corresponding to the original normal operation with sequence number ns_j at site i. If no normal operation at site i has been undone, ic_i is empty.

3.5.3 Representing normal and inverse operations

To represent an operation context with both original normal and inverse operations, an N-dimensional context vector is defined below.

Definition 7. *Context Vector*

Given an operation O, its context $C(O)$ can be represented by the following context vector $CV(O)$:

$$CV(O) = [\ (ns_0, ic_0), (ns_1, ic_1), \ldots, (ns_{N-1}, ic_{N-1})\ ],$$

where, for $0 \leq i \leq N-1$,

1. ns_i represents all original normal operations generated at site i, and
2. $ic_i = [(ns_0, is_0), (ns_1, is_1), \ldots, (ns_{k-1}, is_{k-1})]$ represents all inverse operations for undoing normal operations generated at site i, where (ns_j, is_j), $0 \leq j < k$, represents an inverse group with is_j inverses related to the normal operation with sequence number ns_j. □

In the absence of inverse operations in the operation context, all ic_i, $0 \leq i \leq N-1$, would be empty and a Context Vector would be reduced to a State Vector [4].

The vector representation of operation context can also be used as the vector representation of the document state. As an example, consider the document state after interpreting the undo command $Undo(O_2)$ in Figure 1. Since $Undo(O_2)$ is interpreted as an inverse $\overline{O_2}$ (see Section 4.2), the document state after executing (the transformed) $\overline{O_2}$ shall be $DS = \{O_1, O_2, O_3, \overline{O_2}\}$. This document state cannot be represented by a state vector but can be represented as a context vector as follows: $CV(DS) = [(1, [\,]), (2, [(1, 1)]]$.

Based on Definition 7, it is straightforward to derive the scheme for maintaining the vector representation for the document state after executing each operation (according to Definition 3). Moreover, the vector representation of operation context can also be used to efficiently detect context-dependency/-independency relations. Due to space limitation, these technical details are omitted in this paper.

4. THE BASIC COT ALGORITHM

In the basic COT algorithm, we assume each site maintains a document state DS, which contains the set of *original* operations executed so-far. This is different from the *log* or the *History Buffer (HB)* schemes in prior OT algorithms [4, 22, 23], which record a list of *transformed* operations. We deliberately leave the internal data structure of DS unspecified to keep the COT algorithm independent of the operation buffering strategy.

In algorithm description, we shall use the context set representation $C(O)$, rather than the context vector representation $CV(O)$. When an operation O is propagated from the local site to remote sites, however, it is the context vector, not the context set, that is actually piggy-backed on O for propagation. The set of operations in $C(O)$ can be easily determined from DS based on the information in $CV(O)$.

The COT algorithm has two parts: the COT-DO part for supporting consistency maintenance (do), and the COT-UNDO part for supporting undo. Both parts share the same core context-based transformation procedure. Operation context and context-based conditions are central to the whole COT algorithm.

4.1 COT-DO

COT-DO takes two parameters: O – an original operation to be executed, and DS – the current document state representation. COT-DO is invoked only if $C(O) \subseteq DS$ (CC1), which ensures that all operations included in the context of O have already been executed on DS.

Algorithm 1. *COT-DO(O, DS)*

1. *transform*$(O, DS - C(O))$;
2. Execute O; $DS := DS \cup \{org(O)\}$.

Procedure 1. *transform(O, CD)*

Repeat until $CD = \{\ \}$:

1. Remove O_x from CD, where $C(O_x) \subseteq C(O)$;
2. *transform*$(O_x, C(O) - C(O_x))$;
3. $O := IT(O, O_x)$; $C(O) := C(O) \cup \{org(O_x)\}$.

COT-DO first invokes procedure *transform*() to transform O against operations in $DS - C(O)$ (CC2). This is to upgrade the context of O to DS. In Step 2, it must be that $C(O) = DS$ (CC3), so O is executed as-is, and the original of O is added to DS (according to Definition 3-Item 2).

The heart of COT-DO is *transform*(O, CD), whose task is to transform O against operations in CD, which represents the context difference between $C(O)$ and a new context on which O is to be defined. This procedure repeats the following three steps until CD becomes empty:

1. Remove an operation O_x from CD, where $C(O_x) \subseteq C(O)$ (CC4). An operation O_x meeting this condition can be determined if all operations in CD are sorted in the order of their execution and sequentially retrieved.
2. The procedure *transform*() is recursively invoked to transform O_x against operations in $C(O) - C(O_x)$ (CC5). This is to upgrade O_x to the context of O, so that they can be used for IT transformation in the next step.
3. After the recursive call to *transform*(), it must be that $C(O) = C(O_x)$ (CC6), so O is IT-transformed against O_x, and the context of O is updated by adding the original of O_x (according to Definition 4-Item 3).

To show how COT-DO works, we examine how it resolves the dOPT-puzzle in Figure 1. Consider the operation executions at site 0, with the initial document state $DS_0 = \{\ \}$.

1. After the generation of O_1, since $C(O_1) = DS_0$, O_1 is executed as-is and DS_0 is updated to $DS_1 = \{O_1\}$.
2. When O_2 arrives with $C(O_2) = \{\}$, *transform*$(O_2, DS_1 - C(O_2))$ is called, where $DS_1 - C(O_2) = \{O_1\}$.

   Inside *transform*$(O_2, \{O_1\})$, since $C(O_1) = C(O_2)$, we have $O_2' := IT(O_2, O_1)$, and $C(O_2') = \{O_1\}$.

   Returning from *transform*$(O_2, \{O_1\})$, we have $C(O_2') = DS_1$, so O_2' is executed, and DS_1 is updated to $DS_2 = \{O_1, O_2\}$, where $O_2 = org(O_2')$.
3. When O_3 arrives with $C(O_3) = \{O_2\}$, *transform*$(O_3, DS_2 - C(O_3))$ is called, where $DS_2 - C(O_3) = \{O_1\}$.

   Inside *transform*$(O_3, \{O_1\})$, *transform*$(O_1, C(O_3) - C(O_1))$ is recursively called, with $C(O_3) - C(O_1) = \{O_2\}$, which is the key step in detecting the dOPT-puzzle.

   In the recursive *transform*$(O_1, \{O_2\})$, since $C(O_2) = C(O_1)$, we have $O_1' := IT(O_1, O_2)$, and $C(O_1') = \{O_2\}$.

   Returning from the recursion, we have $C(O_1') = C(O_3)$, so $C(O_3') := IT(O_3, O_1')$ (the dOPT-puzzle resolved here), and $C(O_3') = \{O_1, O_2\}$, where $O_1 = org(O_1')$.

   After returning from *transform*$(O_3, \{O_1\})$, $C(O_3') = DS_2$; so O_3' is executed, and DS_2 is updated to $DS_3 = \{O_1, O_2, O_3\}$, where $O_3 = org(O_3')$.

4.2 COT-UNDO

To undo an operation O, a meta-level undo command $Undo(O)$ must be issued by a user. How to generate the undo command for selecting any operation to undo is part of the *undo policy* [21]. This paper is confined to the discussion of the *undo mechanism*, which determines how to undo the selected operation in a given context.

In COT-UNDO, $Undo(O)$ is interpreted as an inverse $\overline{O}$, that is *context-dependent* on operations in $C(O)$ and O itself. COT-UNDO takes two input parameters: O is the operation selected to be undone, which can be any operation done so-far, and DS is the current document state representation.

Algorithm 2. *COT-UNDO(O, DS)*

1. $\overline{O} := makeInverse(O)$; $C(\overline{O}) := C(O) \cup \{O\}$;
2. $COT\text{-}DO(\overline{O}, DS)$.

COT-UNDO works by first creating an inverse $\overline{O}$ by invoking $makeInverse(O)$[2], with its context $C(\overline{O}) := C(O) \cup \{O\}$ (according to Definition 4-Item 2), and then invoking COT-DO to handle $\overline{O}$.

For example, to interpret $Undo(O_2)$ in Figure 1, COT-UNDO is invoked with parameters O_2 and $DS = \{O_1, O_2, O_3\}$. First, $\overline{O_2}$ and $C(\overline{O_2}) = \{O_2\}$ are created. Then, COT-DO is invoked with parameters $\overline{O_2}$ and DS. Inside COT-DO, $transform(\overline{O_2}, DS - C(\overline{O_2}))$ shall be invoked, and $\overline{O_2}$ shall be correctly transformed against O_1 and O_3 since $CD = DS - C(\overline{O_2}) = \{O_1, O_3\}$. This example shows that an inverse operation can be handled by COT-DO in the same way as other normal operations. This is because context-based conditions CC1 - CC6 are uniformly applicable to both normal and inverse operations.

The basic COT algorithm is simple yet powerful – capable of doing and undoing any operations at anytime. Among all prior OT systems, only the combination of GOTO and ANYUNDO (referred as GOTO-ANYUNDO) has similar capabilities [22, 21].

5. TRANSFORMATION PROPERTIES

COT is a high-level control algorithm responsible for determining which operation should be transformed against other operations and in which order according to context-based conditions. Another important component of an OT system is the low-level transformation functions responsible for transforming operations according to their types and parameters. Past research has identified a range of transformation properties/conditions that must be maintained for ensuring the correctness of an OT system. Different OT systems may have different control algorithms, different transformation functions, and different divisions of responsibilities among these components.

Unlike GOTO-ANYUNDO, the basic COT algorithm does not use ET (Exclusion Transformation) functions [21], thus avoiding the requirement of the *Reversibility Property* (RP) between IT and ET functions [21].

Similar to GOTO-ANYUNDO, the basic COT algorithm assumes that underlying transformation functions are capable of preserving the following properties [4, 15, 19, 23, 21]:

1. **Convergence Property 1** (CP1)[3]. Given a document state DS, and operations O_a, O_b, if $O'_a = IT(O_a, O_b)$, and $O'_b = IT(O_b, O_a)$, then it must be:

   $$DS \circ [O_a, O'_b] = DS \circ [O_b, O'_a],$$

   which means that $[O_a, O'_b]$ and $[O_b, O'_a]$ are equivalent with respect to the effect on the document state DS.

2. **Convergence Property 2** (CP2). Given three operations O, O_a and O_b, if $O'_a = IT(O_a, O_b)$ and $O'_b = IT(O_b, O_a)$, then it must be:

   $$IT(IT(O, O_a), O'_b) = IT(IT(O, O_b), O'_a),$$

   which means that $[O_a, O'_b]$ and $[O_b, O'_a]$ are equivalent with respect to the effect in transformation.

3. **Inverse Property 2** (IP2)[4]. Given any operation O_x and a pair of operations $[O, \overline{O}]$, it must be:

   $$IT(IT(O_x, O), \overline{O}) = IT(O_x, I) = O_x,$$

   which means that $[O, \overline{O}]$ and I are equivalent with respect to the effect in transformation.

4. **Inverse Property 3** (IP3). Given two operations O_a and O_b, if $O'_a := IT(O_a, O_b)$, $O'_b := IT(O_b, O_a)$, and $\overline{O_a}' := IT(\overline{O_a}, O'_b)$, then it must be:

   $$\overline{O_a}' = \overline{O'_a},$$

   which means the transformed inverse operation $\overline{O_a}'$ is equal to the inverse of the transformed operation $\overline{O'_a}$.

The above transformation properties are important discoveries of past research, but they are not unconditionally required. The pre-conditions for requiring them, however, were never explicitly stated in their specifications, which has unfortunately caused quite some misconceptions in OT literature. To explore alternative solutions to these properties, we explicitly state the Pre-Conditions (PC) for CP1, CP2, IP2, and IP3 as follows:

1. **PC-CP1:** CP1 is required only if the OT system allows the same group of context-independent operations to be executed in different orders.

2. **PC-CP2:** CP2 is required only if the OT system allows an operation to be transformed against the same group of context-independent operations in different orders.

3. **PC-IP2:** IP2 is required only if the OT system allows an operation O_x to be transformed against a pair of do and undo operations (O and $\overline{O}$) one-by-one.

4. **PC-IP3:** IP3 is required only if the OT system allows an inverse operation $\overline{O_a}$ to be transformed against another operation O_b that is context-independent of O_a.

[2]The reader is referred to [25] for precise definitions of three primitive operations *Insert, Delete* and *Update* and their corresponding inverses. The $makeInverse(O)$ procedure directly follows these definitions.

[3]**Convergence Property 1 & 2** in this paper (and in [21]) are the same as **Transformation Property 1 & 2** in [19].

[4]There is another **Inverse Property 1** (IP1) that is required in an OT system for achieving the correct undo effect [21], but IP1 is not related to IT functions.

There are generally two ways to achieve OT correctness with respect to these transformation properties: one is to design transformation functions capable of preserving these properties; the other is to design control algorithms capable of breaking the pre-conditions for requiring these properties.

Past research has shown that it is relatively easy to design transformation functions capable of preserving CP1, but non-trivial to design and formally prove transformation functions capable of preserving CP2, IP2 and IP3. Counter-examples illustrating the violation of these properties in some early published transformation functions can be found in [23, 21, 8, 11]. IT functions capable of preserving IP2 and IP3 had been devised in the context of ANYUNDO [21], but our experience in implementing these functions revealed that those solutions are quick intricate and inefficient (more analysis can be found in Section 7).

Clearly, solving CP2, IP2 and IP3 at the control algorithm level has the benefit of simplifying the design of transformation functions and the OT system as a whole. In the following section, we extend the basic COT algorithm to provide simple and efficient solutions to CP2, IP2 and IP3 at the control algorithm level.

6. COT SOLUTIONS TO CP2, IP2, AND IP3

A distinctive feature of COT is that in every transformation process (i.e. an invocation of *transform*(O, CD)), the whole set of transformation target operations are determined in advance, and available in the context-difference parameter CD (calculated by using context-based conditions CC2 and CC5). With the knowledge of all operations involved in the transformation process, we are able to properly arrange these operations to break the pre-conditions for CP2, IP2, and IP3.

6.1 Extended *transform()* procedure

We extend the core procedure *transform*(O, CD) to take advantage of the global knowledge of operations in the context-difference parameter CD for breaking PC-CP2, PC-IP2 and PC-IP3. The extended *transform*(), as shown in Procedure 2, retains the structure and main elements of Procedure 1, but adds solutions to CP2, IP2, and IP3 in Step 1 (*ensure_TPsafety*()) and in Step 2-(c) (the *if-then* part).

Procedure 2. *transform(O, CD)*

1. If $CD \neq \{\ \}$, *ensure_TPsafety*(O, CD);
2. Repeat until $CD = \{\ \}$:
   (a) Remove the first operation O_x from CD;
   (b) *transform*$(O_x, C(O) - C(O_x))$;
   (c) If O_x is a *do-undo-pair*,
   then $C(O) := C(O) \cup \{org(O_x), org(\overline{O_x})\}$;
   else $O := IT(O, O_x)$; $C(O) := C(O) \cup \{org(O_x)\}$.

Procedure 3. *ensure_TPsafety(O, CD)*

1. Ensure *CP2-safety*: sort operations in CD in a total order that respects their context-dependency order.
2. Ensure *IP2-safety*: for any $O_x \in CD$, if $\overline{O_x} \in CD$, then mark O_x as a *do-undo-pair*, remove $\overline{O_x}$ from CD.
3. Ensure *IP3-safety*: if O is inverse, the invoke *make_IP3safe_Inverse*(O, CD).

Procedure 4. *make_IP3safe_Inverse($\overline{O}$, CD)*

1. $O :=$ *makeInverse*$(\overline{O})$; $C(O) := C(\overline{O}) - \{O\}$;
2. $NCD := \{O_x \mid O_x \in CD$ **and** $O_x \overset{c}{\parallel} O\}$;
3. *transform*(O, NCD);
4. $\overline{O} :=$ *makeInverse*(O); $C(\overline{O}) := C(O) \cup \{O\}$;
5. $CD := CD - NCD$.

6.2 Breaking the pre-condition for CP2

The COT solution to CP2 is to sort all operations in CD in a total order which respects their context-dependency order (in Step 1 of *ensure_TPsafety*()). If an operation O is transformed against the same group of context-independent operations in multiple invocations to *transform*(O, CD), this group of operations must be included in CD and sorted in the same total order. Therefore, O can never be transformed against the same group of operations in different orders, thus breaking PC-CP2.

It should be noted that CD becomes an ordered set after the sorting. The first O_x in CD must meet the condition $C(O_x) \subseteq C(O)$ in Step 2(a) of *transform*(O, CD) (Procedure 1), so this condition is no longer explicitly specified in Procedure 2. A correct total order for breaking PC-CP2 can be conveniently determined by using the context-dependency relations among all operations plus the site identifiers of context-independent operations.

There have been several prior OT systems capable of breaking PC-CP2, including the GOT system (by an undo/redo scheme based on total ordering) [23], the SOCT4 system (by a control strategy based on global sequencing) [26], the NICE system (by a central transformation-based notifier) [20], and the TIBOT system (by a distributed synchronization protocol based on time-internal) [12]. The COT solution to CP2 is unique and avoids the use of any undo/redo or global sequencing/synchronization.

6.3 Breaking the pre-condition for IP2

The basic idea of the COT solution to IP2 is to make sure that an operation is never transformed against a pair of do and undo operations one by one, thus breaking PC-IP2. This solution consists of two parts: (1) Step 2 of *ensure_TPsafety(CD)* couples operations with their corresponding inverses if they are all included in the context difference CD, and remove these inverses from CD; (2) In Step 2-(c) of *transform*(), if O_x is found to be a *do-undo-pair*, the IT-transformation of O against O_x is skipped (effectively treating this pair as an identity operation) and the context of O is updated by adding two operations: $\{org(O_x), org(\overline{O_x})\}$.

6.4 Breaking the pre-condition for IP3

The COT solution to IP3 is encapsulated in the procedure *make_IP3safe_Inverse*$(\overline{O}, CD)$, which makes $\overline{O}$ an *IP3-safe* inverse with respect to the context difference CD. An inverse $\overline{O}$ is *IP3-safe* with respect to CD if it is made from a transformed version of O, which has included all operations in CD that are context-independent of O. Under the control of COT, the *IP3-safe* inverse $\overline{O}$ shall never be transformed against operations that are context-independent of O, thus breaking PC-IP3.

The *make_IP3safe_Inverse* procedure works as follows: (1) create operation O (the inverse of $\overline{O}$) and $C(O) = C(\overline{O}) -$

$\{O\}$; (2) select all operations from CD which are context-independent of O and create a new context difference NCD; (3) transform O against operations in NCD (by recursively invoking *transform*()); (4) create a new inverse from the transformed O; and (5) create a new CD by subtracting NCD from the old CD (the new CD must maintain the total order as required for solving CP2). This new inverse $\overline{O}$ must be *IP3-safe* because it is created from a transformed operation whose context has included all operations in NCD. The *IP3-safe* inverse $\overline{O}$ shall never be transformed against the operations in NCD since these operations have been removed from the new CD in Step (5).

7. DISCUSSIONS

7.1 The theory of operation context

The notion of operation context was first proposed in the GOT algorithm [23] and used in conjunction with the theory of causality in follow-up GOTO and ANYUNDO algorithms [22, 21]. In prior work, the context of an operation O was defined as a *sequence* of *transformed* operations which can be executed to bring the document from its initial state to the state on which O is defined. This definition is directly coupled to the sequential history buffering strategy, which saves executed operations in their execution forms and orders. There was no explicit representation of an operation context. Context relationships among operations are derived from the causality relationships *plus* the history buffer position relationships among operations [23, 21].

In this paper, the concept of operation context is defined as a *set* of *original* operations corresponding to the document state on which this operation is defined. This new concept of operation context is independent of the underlying operation buffering strategy and is explicitly represented as an operation set. Based on the set representation of operation context, essential OT conditions (CC1 – CC6) have been precisely and concisely captured. Moreover, the context vector has been devised to efficiently represent both normal and inverse operations in a context. The context vector is more general than the state vector and potentially applicable to other distributed computing systems as well.

Based on the theory of causality, prior OT algorithms have used state vectors to capture causal-dependency relationships among original normal operations and to represent document states in terms of original normal operations. However, causal-dependency relationships are not defined for inverse or transformed operations, and state vectors cannot represent document states with original inverse operations. The theory of causality is unable to capture essential OT conditions (CC1 – CC6) for all types of operation – original and transformed, normal and inverse operations.

7.2 COT versus GOTO-ANYUNDO

Both COT and GOTO-ANYUNDO are capable of doing and undoing any operations at anytime. The main difference is that COT achieves this capability without using ET functions (thus eliminating the RP requirement for IT functions), and without requiring IT functions to preserve CP2, IP2 and IP3. The avoidance of RP, CP2, IP2, and IP3 has significantly simplified the design of transformation functions and the OT system as a whole.

COT is simpler than GOTO-ANYUNDO (and prior OT algorithms based on the causality theory) because of the use of a single theory of operation context for capturing all OT-related conditions (CC1–CC6), the uniformity of context-based conditions for treating all types of operation, and the conciseness of these context-based conditions.

The COT-based system is more efficient than the GOTO-ANYUNDO-based system in solving IP2 and IP3. In GOTO-ANYUNDO, the do-part (a normal operation) and the undo-part (an inverse operation) need to be coupled for the purpose of preserving IP2 [21]. An *eager* coupling strategy was adopted: an inverse operation is coupled with its corresponding normal operation immediately after its execution. Under this scheme, inverse operations are not explicitly represented in the history buffer. When a normal operation is to be executed, however, it may need to be transformed against only the undo-part of a *do-undo-pair*. To cope with this problem, an extra *DeCouple-GOTO-ReCouple* scheme has to be used to decouple a *do-undo-pair* before invoking GOTO and then recouple them afterwards [21]. However, the implementation of this decouple-recouple scheme revealed it was rather intricate and causing many repeated transformations.

In the COT algorithm, COT-DO and COT-UNDO are seamlessly integrated. Inverse operations are explicitly represented in the operation context, and a *lazy* coupling strategy is adopted: the coupling of a *do-undo-pair* occurs not immediately after executing each inverse, but only when both the do-part and the undo-part appear in the same transformation process at some late stage. These strategies help to avoid overhead transformations caused by the eager coupling scheme and the decouple-recouple scheme.

In the GOTO-ANYUNDO-based system, the solution to IP3 is encapsulated in an IP3-preserving IT function, called *IP3P-IT* [21]. Inside this function, an extended ET function has to be used, which may invoke the expensive GOTO algorithm to ensure RP with the corresponding IT function. In contrast, the COT solution to IP3 is encapsulated in the high-level procedure *make_IP3safe_Inverse*($\overline{O}, CD$), which is more efficient since (1) it avoids converting $\overline{O}$ to O back and forth multiple times for each $O_x \in NCD$ (if *IP3P-IT*($\overline{O}, O_x$) were used instead); and (2) the *transform()* procedure is much cheaper than GOTO.

7.3 OT buffering strategies

Another distinctive feature of the COT algorithm is the separation of the algorithm from the underlying operation buffering strategy. This has not only resulted in a cleaner and simpler logical structure to the algorithm itself, but also allowed a range of performance optimizations at the operation buffering level.

We have devised and implemented a buffering structure in which not only original operations but also transformed versions can be saved; and all transformed operations from the same original operation are organized in the same *version group*. When an original operation is required at the COT algorithm level, the corresponding version group is searched for a version that matches the context requirement. If such a version already exists, it is used to represent the original operation in the transformation process, thus saving the overhead to transform the original operation into this version. Under this buffering structure, various heuristics can be used to selectively save transformed versions to maximize their reuse and minimize their space usage. By experimentation, we have identified some useful heuristics that are

effective in saving transformations for a number of common patterns of operation sequence.

COT is not the first OT algorithm that buffers and uses original operations for transformation. Several prior OT algorithms, including CCU [2], adOPTed [19], and GOTO-ANYUNDO [21], have also buffered original operations. COT is unique in its way of buffering and using original, as well as transformed, operations.

7.4 OT correctness

OT correctness is a central topic of discussion in OT research. In this section, we provide our observations and opinions on some important OT correctness issues.

OT is a complex system with multiple interrelated components. A system-oriented approach is needed for addressing OT issues. An experimental method, called *puzzle-detection-resolution*, has commonly been used in exploring and refining OT solutions. Puzzles are subtle but representative scenarios in which certain OT properties/conditions may be violated and the system may produce incorrect results. The ability to solve all known puzzles is a necessary condition and an important indicator of the soundness of an OT system. In research literature, simple puzzle scenarios are often used to illustrate the key reasons why an OT system works or fails. In real OT system design, however, a real implementation and comprehensive testing cases based on complex puzzle scenarios are crucial in validating a design.

Theoretical methods have also been used to formally verify OT correctness with respect to some identified transformation properties/conditions. Formal verification can be effective if the correctness issues have been well-understood and the verification criteria and boundary conditions have been well-defined. In this regard, experimental methods like *puzzle-detection-resolution* can play an important role in gaining the necessary insights into the real correctness issues, and establishing suitable criteria and conditions for formal verification.

A systematic approach is needed in conducting both experimental and theoretic OT research. Many OT components and issues are intimately related, and a solution to one issue, if examined in isolation, is unlikely to be correct or complete. For example, a solution that works well for consistency maintenance (do), may fail when both do and undo problems are considered; and an undo solution (e.g. preserving IP2) may violate the solution to consistency maintenance [21]. A complete OT solution to both do and undo problems is significantly more difficult to design than a partial solution to only one of them.

On the other hand, a difficult issue in one OT component may be resolved easily, or avoided altogether, if this issue is addressed from a different OT component. For example, it is known that devising and proving transformation functions capable of preserving properties CP2, IP2, and IP3 are difficult. However, these difficulties can be avoided by devising control algorithms (like COT) capable of breaking the pre-conditions for requiring these properties; it is also easier to prove a control algorithm is capable of breaking the pre-conditions for these properties, than to prove transformation functions are capable of preserving them.

Different OT systems may have different divisions of responsibility among their components and hence different correctness requirements for these components. Caution must be taken in interpreting correctness results. For example, CP1 and CP2 were proven to be necessary and sufficient for adOPTed-based systems to converge [19, 13], but this result cannot be generalized to all OT systems. In fact, CP1 and CP2 are neither sufficient nor necessary for many OT systems. They are insufficient because an OT system may need to preserve additional properties/conditions, such as IP2, IP3, and those summarized in [21]. They are unnecessary if the pre-conditions for requiring them have been broken. For example, neither CP1 nor CP2 is required in the REDUCE system based on the GOT algorithm for ensuring convergence [23]. CP2 is also not required by OT systems based on COT or some prior OT algorithms [26, 20, 12].

One OT correctness issue, which is often discussed in relation to the CP2-violation problem, is the "false-tie" problem: when two (or more) insert operations with the same position are IT-transformed with each other, the position tie may be *false* if it was not original but caused by previous transformations. An OT system may fail to produce correct results if the normal tie-breaking rule (e.g. based on site identifiers) is used to break "false-ties". This problem was long discovered in early OT work and a concrete scenario related to this problem was illustrated in Fig. 6 of [23]. It is beyond the scope of this paper to discuss solutions to this problem, but it is worth pointing out that the "false-tie" problem is different from the CP2-violation problem: a "false-tie" may occur without violating CP2. In our view, the "false-tie" problem is an issue at the transformation function level and its solution could and should be localized at this level as well. For alternative views and approaches to this problem, the reader is referred to [8, 11, 5].

The COT algorithm has been implemented and validated by a comprehensive testing suite covering all known OT puzzle scenarios. In this paper, informal analysis and simple puzzle scenarios have been used to show the correctness of COT with respect to various transformation properties/conditions. Formal verification of COT correctness with respect to these properties/conditions, and quantitative analysis of the time and space complexity of COT, shall be reported in a journal version of this paper.

8. CONCLUSIONS

We have contributed the theory of operation context and the COT (Context-based OT) algorithm. The theory of operation context is capable of capturing essential relationships and conditions for all types of operation in an OT system; it provides a new foundation for better understanding and resolving OT problems. The COT algorithm provides uniformed solutions to both consistency maintenance and undo problems; it is simpler and more efficient than prior OT control algorithms with similar capabilities; and it significantly simplifies the design of transformation functions. The COT algorithm has been implemented in a generic collaboration engine and used for supporting a range of novel collaborative applications [24].

Real-world applications provide exciting opportunities and challenges to future OT research. The theory of operation context and the COT algorithm shall serve as new foundations for addressing the technical challenges in existing and emerging OT applications.

Acknowledgments

The authors are grateful to Bo Begole and anonymous reviewers for their valuable comments and suggestions which have helped improve the presentation of the paper.

9. REFERENCES

[1] J. Begole, M. Rosson, and C. Shaffer. Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. *ACM Trans. on Computer-Human Interaction*, 6(2):95–132, 1999.

[2] G. Cormack. A calculus for concurrent update. In *Research Report CS-95-06, Dept. of Computer Science, University of Waterloo, Canada*, 1995.

[3] A. Davis, C. Sun, and J. Lu. Generalizing operational transformation to the standard general markup language. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 58 – 67, Nov. 2002.

[4] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proc. of the ACM Conf. on Management of Data*, pages 399–407, May 1989.

[5] N. Gu, J. Yang, and Q.Zhang. Consistency maintenance based on the mark & retrace technique in groupware systems. In *Proc. of ACM Conf. on Supporting Group Work*, pages 264–273, Nov. 2005.

[6] R. Guerraoui and Corine Hari. On the consistency problem in mobile distributed computing. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, pages 51–57, New York, Octo 2002. ACM.

[7] C. Ignat and M.C. Norrie. Customizable collaborative editor relying on treeOPT algorithm. In *Proc. of the European Conf. of Computer-supported Cooperative Work*, pages 315–324, Sept. 2003.

[8] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving correctness of transformation functions in real-time groupware. In *Proc. of the European Conf. on Computer-Supported Cooperative Work*, Sept. 2003.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communication of ACM*, 21(7):558–565, 1978.

[10] D. Li and R. Li. Transparent sharing and interoperation of heterogeneous single-user applications. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 246–255, Nov. 2002.

[11] D. Li and R. Li. Preserving operation effects relation in group editors. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 457–466, Nov. 2004.

[12] R. Li, D. Li, and C. Sun. A time interval based consistency control algorithm for interactive groupware applications. In *Proc. of International Conference on Parallel and Distributed Systems*, pages 429–436, July. 2004.

[13] B. Lushman and G. Cormack. Proof of correctness of Ressels adOPTed algorithm. *Information Processing Letters*, (86):303–310, 2003.

[14] C. Palmer and G. Cormack. Operation transforms for a distributed shared spreadsheet. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 69–78, Nov. 1998.

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

[16] N. Preguica, M. Shapiro, and J. Legatheaux Martins. Automating semantics-based reconciliation for mobile databases. In *Proceedings of the 3th Conference Francaise sur les Systems d'Exploitation*, Octo 2003.

[17] M. Raynal and M. Singhal. Logical time: capturing causality in distributed systems. *IEEE Computer Magazine*, 29(2):49–56, Feb. 1996.

[18] M. Ressel and R. Gunzenhäuser. Reducing the problems of group undo. In *Proc. of the ACM Conf. on Supporting Group Work*, pages 131–139, Nov. 1999.

[19] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 288–297, Nov. 1996.

[20] H.F. Shen and C. Sun. A flexible notification framework for collaborative systems. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 77–86, Nov. 2002.

[21] C. Sun. Undo as concurrent inverse in group editors. *ACM Trans. on Computer-Human Interaction*, 9(4):309–361, December 2002.

[22] C. Sun and C. A. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 59–68, Nov. 1998.

[23] 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 Trans. on Computer-Human Interaction*, 5(1):63–108, March 1998.

[24] C. Sun, Q. Xia, D. Sun, D. Chen, H.F. Shen, and W. Cai. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Trans. on Computer-Human Interaction*, 2006.

[25] D. Sun, S. Xia, C. Sun, and D. Chen. Operational transformation for collaborative word processing. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 437–446, Nov. 2004.

[26] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 171–180, Dec. 2000.

[27] S. Xia, D. Sun, C. Sun, and D. Chen. A collaborative table editing technique based on transparent adaptation. In *Proc. of the International Conf. on Cooperative Information Systems, LNCS Vol. 3760, Springer Verlag*, pages 576–592, Nov. 2005.

[28] S. Xia, D. Sun, C. Sun, and D. Chen. Object-associated telepointer for real-time collaborative document editing systems. In *Proc. of the IEEE Conf. on Collaborative Computing: Networking, Applications and Worksharing*, Dec. 2005.

[29] S. Xia, D. Sun, C. Sun, D. Chen, and H.F. Shen. Leveraging single-user applications for multi-user collaboration: the CoWord approach. In *Proc. of the ACM Conf. on Computer-Supported Cooperative Work*, pages 162–171, Nov. 2004.