Operational Transformation for Orthogonal Conflict Resolution in Real-time Collaborative 2D Editing Systems

Chengzheng Sun, Hongkai Wen, Hongfei Fan
School of Computer Engineering, Nanyang Technological University, Singapore

ABSTRACT
Operational Transformation (OT) is commonly used for conflict resolution in real-time collaborative applications, but none of existing OT techniques is able to solve a special type of conflict—orthogonal conflict, which may occur when concurrent operations are inserting/deleting an arbitrary number of objects in different dimensions of a two-dimensional (2D) workspace, such as spreadsheet documents. This paper is the first to identify and solve the orthogonal conflict problem by extending OT with a new capability of resolving 2D conflicts. Extending OT from one- to two-dimensional conflict resolution is fundamental to the theory and application of OT, and technically challenging as well because 2D orthogonal conflict is different from but intimately related to the one-dimensional positional shifting conflict and necessitates new and integral solutions for multi-dimensional conflicts. In this paper, we present formal definitions of orthogonal conflict, pseudo-code description, design rationale analysis, and correctness verification and complexity analysis of the 2DOT solution.

Author Keywords
Conflict Resolution, Operational Transformation, Real-time Collaborative Spreadsheet and Table Editing.

ACM Classification Keywords
H.5.3 [**Information Systems**]: Group and Organization Interfaces – Synchronous Interaction, Theory and Model.

General Terms
Algorithms; Theory.

INTRODUCTION
Operational Transformation (OT) is a technique for conflict resolution in collaborative computing systems, and a subject of continuous research in CSCW in past decades [1-4, 6-17]. In recent years, OT has been increasingly adopted as a core collaboration technique in industrial applications, e.g. Google Wave [17] and Docs[1], IBM OpenCoWeb[2], Novell Vibe[3], and Codoxware[4], which are creating a new wave of OT research and application.

Spreadsheets and word processors (with table functions) are among the most widely used computer software applications, and they are commonly used by data and knowledge workers in finance, business, and government organizations, as well as ordinary people in their daily lives. The need for real-time collaborative editing of spreadsheets and table-based documents has long been recognized and stimulated research and development efforts from both academia and industry [8,12,16]. Representative collaborative spreadsheet tools include Google Spreadsheet[5], eXpresso[6], etc. However, these collaborative tools lag in functionality far behind their desktop counterparts, such as Microsoft Excel[7] and Open Office Calc[8].

Modern single-user spreadsheet tools are capable of supporting comprehensive functionalities on two-dimensional (2D) tables with an arbitrary number of rows and columns. Inserting and deleting cells, rows, or columns are common operations in these applications, but supporting such operations in real-time collaboration is a challenge. One special characteristic of such operations is they have *global effect* ranges that cover not only the cells inserted or deleted directly by an operation (the *direct effect range*), but also those cells whose positions are shifted due to the execution of such an operation (the *indirect effect range*). Dynamic positional shifting of existing cells in a table may lead to inconsistencies if operations are generated concurrently and executed in different orders. Those technical obstacles are part of the reasons why existing collaborative spreadsheet tools (e.g. Google Spreadsheet) are unable to support many editing functionalities available in desktop spreadsheet tools (e.g. Microsoft Excel).

OT was originally invented to resolve positional shifting problems and maintain consistency while allowing concurrent operations to be executed in different orders [3,11,14]. Existing OT techniques are able to resolve positional shifting problems if concurrent operations are targeting objects in a single or hierarchical linear address space, like a sequence of cells in the same row/column, a sequence of rows/columns in the same table. However, none of existing OT techniques has ever addressed the *orthogonal operation conflict* problem, which occurs when concurrent operations are inserting or deleting objects (e.g. cells, rows, and columns) in orthogonal dimensions of a 2D workspace –

CSCW'12, February 11–15, 2012, Seattle, Washington, USA.

[1] http://en.wikipedia.org/wiki/Google_Docs
[2] https://github.com/opencoweb/coweb#readme
[3] http://en.wikipedia.org/wiki/Novell_Pulse
[4] https://www.codoxware.com
[5] http://spreadsheets.google.com/lv
[6] http://www.expressocorp.com/
[7] http://office.microsoft.com/enus/excel/default.aspx
[8] http://www.openoffice.org/product/calc.html.

a common phenomenon in collaborative spreadsheet applications. Two operations are orthogonal if their dimensions are different, e.g. one is to insert a cell in a row, and the other is to insert a cell in a column. Two orthogonal operations conflict if they are concurrent and have overlapping effect ranges.

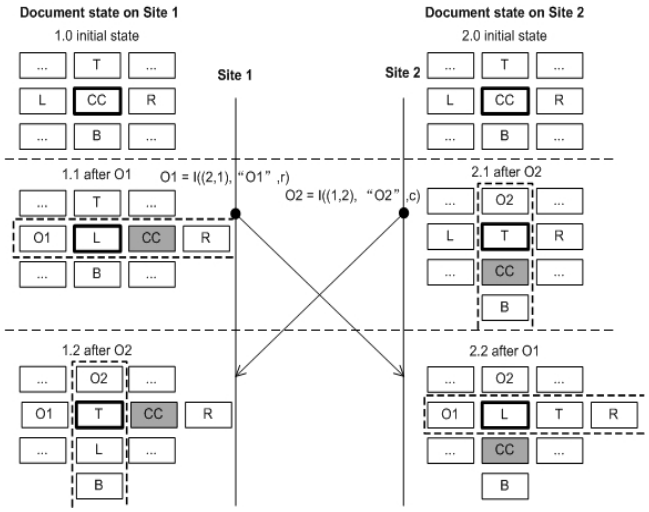

Figure 1 Conflicts between concurrent orthogonal operations in a 2D table. The effect range of each operation is highlighted in a dashed frame in the document after its execution. The critical cell position is highlighted by a box with thicker edges. The current position of the old critical cell is indicated by a shaded cell.

As shown in Figure 1, O1 is to insert a cell, denoted by [*O1*], at position (x_1, y_1) in *row*-x_1, O_2 is to insert a cell, denoted by [*O2*], into position (x_2, y_2) in *column*-y_2. These two operations are orthogonal as their operational dimensions are different, and there is a shared *Critical Cell* (denoted as [*CC*]) at (x_1, y_2) in their effect ranges. These two operations are generated concurrently and executed in different orders at the two sites. Due to the orthogonal nature of the two concurrent operations, the final results at the two sites are inconsistent: the old critical cell [*CC*] is shifted right to (x_1, y_2+1) by O_1 at site 1, but down to (x_1+1, y_2) by O_2 at site 2; the new cell at (x_1, y_2) is [*T*] at site 1, but [*L*] at site 2. Generally, concurrent orthogonal operations may result in inconsistent tables: old critical cells in their overlapping effect ranges are moved to different positions, and different new cell are moved into old critical cell positions in overlapping effect ranges.

Orthogonal conflict is a special 2D conflict problem: it involves critical cells at the intersection of the effect ranges of two orthogonal conflict operations, and those critical cells are shifted in different dimensions. No prior OT techniques can solve this problem because they are essentially one-dimensional (1D) in the sense that they work only if operations are targeting and shifting objects in the same 1D addressing space [3,6,7,9,11], or in a hierarchy of multiple 1D addressing spaces [2,12]. Another subtle aspect of orthogonal conflict is that it may occur in indirect effect ranges of concurrent operations. In Figure 1, the critical cell (x_1, y_2) is not directly inserted by O_1 or O_2, but indirectly affected by their shifting effects. Such indirect conflict relationships cannot be detected by conflict detection techniques that rely on the condition in which concurrent operations directly target common objects [13]. Orthogonal conflict may occur whenever two concurrent operations are inserting/deleting *any number of* cells, including a *full row or column,* in different dimensions.

This paper contributes a novel 2DOT solution to resolve orthogonal conflict problems in real-time collaborative 2D document editing applications. The rest of this paper is organized as follows. First, basic concepts and relations in orthogonal conflict resolution are defined. Then, combined effects for two orthogonal conflict operations are described. Next, the 2DOT solution for resolving conflicts among an arbitrary number of operations is discussed. Furthermore, a comprehensive example is used to illustrate how various 2DOT technical components work together. This is followed by theoretic verification of correctness and time complexity analysis of the 2DOT solution. Then, the 2DOT work is compared to related work, and future work is discussed. Finally, main contributions in this paper are summarized.

BASIC CONCEPTS AND DEFINITIONS

In this paper, a spreadsheet or table-based document is modeled as a 2D workspace consisting of multiple rows and columns; each row or column contains a sequence of cells; and each cell contains values, formulas, etc. A row (or column) in the 2D space is labeled by a sequence number from 0 to M − 1 (or N −1), where M (or N) is the maximum number of rows (or columns) in the 2D workspace.

Real-world spreadsheet applications (e.g. Microsoft Excel) may provide end-users with a rich set of operations. This work, however, focuses on two primitive ones: insert and delete of cells, as they are representative and specially related to the orthogonal conflict problem addressed in this paper. For the sake of clarity and simplicity, the direct effect range of an operation is restricted to a single cell, but the issues and solutions reported in this paper can be generalized to operations that insert/delete *any number* of cells (e.g. a full row or column). Two primitive operations are defined:

1. I(*p*,*c*,*dim*): insert a cell *c* at position *p* in dimension *dim*.
2. D(*p*,*dim*): delete the cell at position *p* in dimension *dim*.

The parameter *p* is a pair (*x*, *y*), where *x* and *y* represent the cell's row and column sequence numbers, respectively; *c* represents a cell object; and *dim* can take one of two values: *row* for a row-oriented operation, which has position shifting effects from *left-to-right* (for insert) or from *right-to-left* (for delete) in the target row; or *col* for a column-oriented operation, which has position shifting effects from *top-to-bottom* (for insert) or from *bottom-to-top* (for delete) in the target column. The dot notation is used to refer to the type and parameters of an operation *O*: *O.type* can be either *I* or *D*; O.*p* refers to the *p* parameter, and *O.p.x* and *O.p.y* refer to the coordinates in (*x*, *y*), respectively; *O.c* refers to the *c* parameter; and *O.dim* refers to the *dim* parameter.

The *Effect Range* of operation *O*, denoted as *ER*(*O*), is the set of cells directly or indirectly affected by the execution of *O*. If *O* is row-oriented, *ER*(*O*) includes all cells at or on the right of position *O.p* in the target row; if *O* is column-

oriented, $ER(O)$ includes all cells at or below position $O.p$ in the target column. Two operations are overlapping if their effect ranges have shared cells, as defined below.

Definition 1: *Overlapping Relation* "#". Given two operations O_a and O_b, they have an overlapping relation, denoted as $O_a \# O_b$, *iff*: $ER(O_a) \cap ER(O_b) \neq \{\}$.

Following Lamport [5], the causal (partial) ordering relation of operations is defined in terms of their generation and execution sequences as follows.

Definition 2: *Causal Ordering Relation* "→". Given two operations O_a and O_b, generated at sites i and j, O_a is causally ordered before O_b, denoted as $O_a \rightarrow O_b$, *iff:* (1) $i = j$ and the generation of O_a *happened before* the generation of O_b; (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 3: *Concurrent Relation* "||". Given two operations O_a and O_b, they are *concurrent*, denoted as $O_a || O_b$, *iff*: neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$.

In real-time collaborative editing systems, operations with causal relationships are executed in their causal order according to the causality preservation requirement in [10]; but concurrent operations may be executed in any orders at different collaborating sites.

Definition 4: *Orthogonal Relation* "⊥". Given two operations O_a and O_b, they are *orthogonal*, denoted as $O_a \perp O_b$, *iff*: $O_a.dim \neq O_b.dim$.

If orthogonal operations O_a and O_b are concurrent and their effect ranges are overlapping, their execution in different orders may result in inconsistent states (see Figure 1). Such a relationship is called *orthogonal conflict* (or *2D conflict*), as defined below.

Definition 5: *Orthogonal (or 2D) Conflict Relation* "⏀". Given two operations O_a and O_b, they have an *orthogonal conflict* relation, denoted as O_a ⏀ O_b, *iff*: (1) $O_a \perp O_b$; (2) $O_a || O_b$; and (3) $O_a \# O_b$.

Definition 6: *Parallel Operation Relation* "//". Given two operations O_a and O_b, they are parallel, denoted as O_a // O_b, *iff* they are not orthogonal.

Two parallel operations are overlapping if they are performed in the same row or column. If two parallel and overlapping operations are concurrent, their execution in different orders may result in inconsistent document states [1,2,9,14]. Such a relationship between the two operations is called *1D conflict*, which is defined below.

Definition 7: *1D conflcit Relation* "⍁". Given operations O_a and O_b, they have a *1D conflict* relation, denoted as O_a ⍁ O_b, *iff*: (1) $O_a // O_b$; (2) $O_a || O_b$; and (3) $O_a \# O_b$.

Definition 8: *Compatible Relation* "⊙". Given operations O_a and O_b, they are *compatible,* denoted as $O_a \odot O_b$, *iff*: neither O_a ⍁ O_b nor O_a ⏀ O_b.

Figure 2 is used to illustrate operation relations defined in this section. For four operations O_1, O_2, O_3 and O_4, their positional relationships in a 2D workspace are shown in Figure 2 (a), and their causal relationships are illustrated in Figure 2 (b). In Figure 2 (a), O_2 is illustrated as a column-oriented operation, but O_1, O_3, and O_4 are all row-oriented. Therefore, O_2 is orthogonal with O_1, O_3, and O_4, i.e. $O_1 \perp O_2$, $O_3 \perp O_2$, $O_4 \perp O_2$, according to Definition 4; and O_1, O_3, and O_4 are parallel operations, i.e. $O_1 // O_3 // O_4$, according to Definition 6. Moreover, O_1 is overlapping with O_4 and O_2, i.e. $O_1 \# O_4$ and $O_1 \# O_2$, but O_3 is not overlapping with any operation, according to Definition 1.

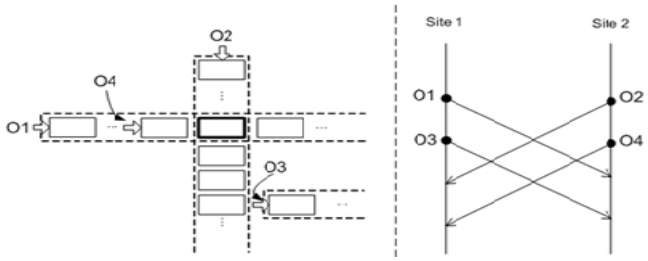

Figure 2 (a) Positional relations among four operations in a 2D table. (b) Operation causal relations in a time-space diagram.

In Figure 2 (b), the four operations have the following causal and concurrent relationships: $(O_1 \rightarrow O_3) \parallel (O_2 \rightarrow O_4)$, according to Definitions 3 and 4. Conflict/compatible relations among these four operations can be derived from considering both the positional and causal relation scenarios in Figure 2 (a) and (b) together. Among the three pairs of orthogonal operations, only one pair is orthogonally conflicting, i.e. O_2 ⏀ O_1, according to Definition 5. This is because O_2 is concurrent and overlapping with only O_1, but is causally before O_4 and not overlapping with O_3. Among the three parallel operations, only one pair is *1D conflicting*, i.e. O_1 ⍁ O_4, according to Definition 7. This is because O_1 is concurrent and overlapping with only O_4, but O_1 is causally before O_3, and O_4 is not overlapping with O_3.

In summary, causally-related operations are compatible and their execution order is constrained by their causal relationships [3,5]. Concurrent operations may be compatible (if non-overlapping) or conflicting (if overlapping). Compatible concurrent operations can be executed in any orders and achieve consistency without special treatment. Conflict (1D/2D) concurrent operations may cause workspace inconsistency if they are executed in different orders. The focus of this paper is to devise new 2DOT techniques to resolve 2D orthogonal conflicts and to adapt existing 1DOT techniques in 2D workspace.

RESOLVING CONFLICT BETWEEN TWO OPERATIONS

Combined Effect for Orthogonal Conflict Operations

From the example scenario in Figure 1, it can be observed that, after executing O_1 and O_2 on the same 2D document at two sites separately, two document states are produced, each preserving the effect of one operation only. Based on these two states, one can envisage a merged document state that combines the effects of the two orthogonal conflict

operations, as shown in the final document states at both sites in Figure 3.

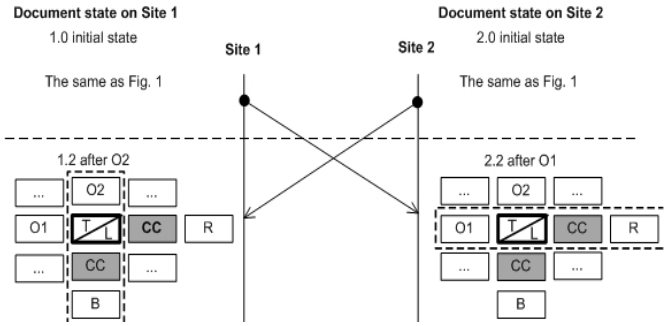

Figure 3 Combined effect of two orthogonal conflict operations.

In the merged document state, effects of both operations are preserved as if each operation was executed alone, except that the critical cell position (x_1, y_2) becomes a special *Multi-Version* (MV) position capable of hosting two cells, denoted by [*T*/*L*], where *L* is shifted into this position by O_1 and *T* by O_2. This combined effect is termed as the *Union Effect*, as defined below.

Definition 9: *Union Effect (UE)*: Given two orthogonal conflict operations O_a and O_b defined on the same 2D document, their union effect consists of the following:

1. an MV position is created at the interaction of the effect ranges of O_a and O_b;
2. effects of O_a (or O_b) are the same as if O_a (or O_b) was executed alone on the 2D document state on which O_a (or O_b) was defined; and
3. cells outside effect ranges of O_a and O_b are untouched.

This union effect is intuitive for users to understand as it retains the original effect of an operation as if the operation was executed alone, and it is capable of preserving all users' work even in the face of conflict. The 2D union effect is also compatible with the 1D union effect achieved by the basic OT technique for insert and delete operations in a linear (1D) workspace [10,14], and hence well suited for OT support, which will become clear late in this paper.

Required Properties of the Multi-Version Position

One key element of the union effect is the creation of an MV position to accommodate the two cells shifted into by two orthogonal conflict operations. The implementation of the MV position is required to preserve the following properties:

1. **Uniqueness and Fixed Location:** An MV position is a *unique* location in the 2D workspace. This location is fixed in the sense that it cannot be changed by operations, though cells in this MV position can be shifted by operations according to OCOD (see below).
2. **One Cell One Dimension (OCOD):** Each cell place in the MV position belongs to one specific dimension. When a new MV position is created, it has two initial cell places: one is occupied by the current cell located at this position and the other is vacant for new contents. A vacant cell place can be filled by directly inserting a cell into it or indirectly shifting a cell into it. When an existing MV position falls into the effect range of an operation *O*, only one cell place belonging to *O*'s dimension will be affected. If the cell place in *O*'s dimension is occupied, then that cell will be shifted out of the MV position. If the cell place in *O*'s dimension is vacant, then the position shifting effect of *O* will not propagate beyond the MV position, which means those cells beyond the MV position in the indirect effect range of *O* will not be affected.
3. **Multi-Version Single-Display (MVSD):** A MV position maintains two cells internally, but displays only one cell on the user interface. The determination of which cell to display must be consistent across all sites. There exist various ways to implement this MV property. For example, the MVSD technique in [13] can be adapted to support the MVSD property. Suitable user interfaces can be provided for users to select (i.e. commit to) one cell among the two cells in an MV position. After the selection, the MV position is removed and the location becomes a normal one.

The feasibility of preserving the above properties has been experimentally tested in our designing and implementing real-time collaborative spreadsheet tools and word processors [12,13,16]. The rest of the paper will assume these properties for the MV cell are supported by underlying MV implementation mechanisms, without further elaborating their technical details.

Achieving the Union Effect

In OT-based collaborative systems, it is the operations, not the documents that are propagated among collaborating sites. To incorporate the orthogonal conflict solution in the OT framework, it is necessary to devise techniques to achieve the union effect *incrementally* by (1) executing one (local) operation as-is to produce one intermediate 2D document, and (2) executing the other (remote) operation in a *transformed* form that merges its effect into this intermediate 2D document to achieve the full union effect.

In Figure 3, for example, when O_2 arrives at site 1, O_1 has been executed with effects of inserting a new cell [*O1*] at (x_1, y_1), shifting cell [*L*] into the critical cell position (x_1, y_2), and shifting the old critical cell [*CC*] to position (x_1, y_2+1). To achieve the union effect of O_1 and O_2 under this 2D document state, the following steps can take place to process O_2 (a similar process occurs in processing O_1 at site 2):

1. *create* an MV position at location (x_1, y_2), which hosts the existing cell [*L*] in its row-dimension cell place and leaves its column-dimension cell place vacant;
2. *copy* cell [*CC*] at position (x_1, y_2+1) and *insert* it to position (x_1+1, y_2) in the column dimension, which will shift the following cells one position downwards; and
3. *execute* the original O_2 to insert a new cell [*O2*] at position (x_2, y_2), which will shift cell [*T*] into the MV position (x_1, y_2), but will not shift the cells below the MV position because the MV position has a column-dimension vacant place for cell [*T*], which effectively blocks the shifting effect downwards, thanks to the OCOD property of the MV position.

As the effect achieved in the above process is only part of the union effect for two orthogonal conflict operations, the term *Partial Union Effect* is used to refer to this part of the union effect. Also, all actions involved in the above process, i.e. creating an MV position, copying and inserting the critical cell, and executing the original operation, can be determined by the parameters of the two conflict operations.

```
PU(Oa, Ob) {
  if (Oa.dim = row && Oa.type = I && Ob.type = I) {
     1. create an MV position at (Oa.p.x, Ob.p.y);
     2. copy the cell at position (Oa.p.x, Ob.p.y+1) and
        insert it to position (Oa.p.x +1, Ob.p.y) in the
        dimension of Oa; and
  } else { // other cases are omitted for conciseness }
  execute the original Oa;
}
```

The *Partial Union* procedure, $PU(O_a, O_b)$, is defined to achieve the partial union effect of O_a on top of O_b's effect:
The full union effect of two conflict operations O_a and O_b can be achieved by $O_b \circ PU(O_a, O_b)$, or $O_a \circ PU(O_b, O_a)$, where "○" denote the execution of an operation or procedure, as stated in the following proposition.

Proposition 1: Given two orthogonal conflict operations O_a and O_b, defined on the same 2D document state S, we have:

$$S \circ O_b \circ PU(O_a, O_b) = S \circ O_a \circ PU(O_b, O_a),$$

Proof: It follows directly from UE and PU definitions.

One corollary of this proposition is that using the PU procedure to execute two orthogonal conflict operations in different orders will achieve the same union effect, which is the basis to ensure convergence in 2D OT transformation functions (to be discussed in late sections).

RESOLVING CONFLICTS IN ARBITRARY OPERATIONS

In real-time collaborative editing systems, multiple operations may be generated with arbitrary concurrency and positional relationships, resulting in complex conflict scenarios, as illustrated in Figure 2. In this section, the orthogonal conflict solution is extended from two to multiple operations, and both 2D and 1D conflict resolution solutions are integrated in the same OT framework.

Adopting Existing OT Control Algorithms

An OT system consists of generic control algorithms and domain-specific transformation functions [3,7,9-12]. OT control algorithms ensures conditions governing operation execution and transformation [14,15]. One important condition is that operations with causal relationships must be executed in their causal orders; concurrent operations may be executed in any order [14,15]. There exist various distributed techniques to achieve causally ordered message delivery or operation execution [3,7,10,14,15,17].

Another important condition is that a pair of operations must be *context-equivalent* (i.e. defined on the same document state) before being transformed with each other [10,11,14,15]. Violation of this condition was the root of a major algorithmic flaw in the first OT system [3,11]. Various OT control algorithms, such as adOPTed [9], Jupiter[7], GOT/GOTO [10,11], and COT [15], have been proposed to ensure the context-equivalency condition.

As causal and context relationships among operations are generic and independent of document types or dimensions (1D/2D). Existing solutions to them can be directly adopted for supporting orthogonal conflict resolution in collaborative 2D document editing systems. In this work, we focus on the design of OT transformation functions that are special to orthogonal conflict resolution. In particular, we design special *IT (Inclusion Transformation)* functions that re-define an operation on a 2D document by including the impact of another operation [10,11]. Such transformation functions for orthogonal conflict resolution can be combined with any OT control algorithm that requires only *IT* functions (e.g. COT [15], adOPTed [9], Jupiter[7], and Google Wave OT [17], etc).

Representing Partial Union Effects

The *PU* procedure would have been adequate to solve orthogonal conflicts if two orthogonal conflict operations O_a and O_b are always executed one after the other, without interleaving with other operations in-between. In unconstrained collaborative computing environments, however, execution of two orthogonal conflict operations may be interleaved with executions of arbitrary operations. Consider the scenario in Figure 2. When O_1 arrives at site 2, it must be transformed against concurrent operations O_2 and O_4 in sequence [17]. In transforming O_1 against O_2, their orthogonal conflict relation can be detected and resolved by invoking $PU(O_1, O_2)$ to achieve the partial union effect for O_1. However, as O_2 is not the only concurrent operation that has been executed before O_1 at site 2, O_1 has to be further transformed against O_4 (to resolve potential conflict with O_4) before its execution. Therefore, the partial union effect of O_1 has to be recorded inside O_1, which may be further transformed and performed later. Mechanisms are needed to represent partial union effects within operations and to execute an operation with partial union effects properly.

Parameters of an operation are the places to record transformation effects on an operation so that execution of the transformed operation can achieve the desirable effect. Normal positional parameters of an operation, however, can only record positional shift effects caused by 1D conflict operations targeting in the same dimension, but cannot record partial union effects. Therefore, it is necessary to add a new operation parameter, called *Partial Union Effect Record* (*pur*), to record the partial union effect of an operation O with respect to another conflict operation.

Definition 10: Partial Union Effect Record (*pur).* A *pur* is a tuple of three elements, i.e. *pur* = (*mv*, *sc*, *dc*), where:

1. *mv* records the MV position for this partial union effect;
2. *sc* records the position of the source cell that is to be copied for realizing this partial union effect;
3. *dc* records the position of the destination cell to insert the copied *sc* cell.

As an operation may be transformed with multiple orthogonal conflict operations, a *Partial Union Effect Record List* (*purl*) is needed to keep track of all records

accumulated in transformation. We use the term *Internal Parameter* to refer to the new parameter *purl* in an operation, and the term *External Parameter* to refer to the normal positional parameter of an operation.

Executing Operations with Partial Union Effects

```
PUX(O) {
  1. for each pur in the purl of O, create an MV position
     at each pur.mv position;
  2. for each pur in the purl of O, copy the cell at
     position pur.sc and insert the copied cell at position
     pur.dc in the same dimension of O; and
  3. execute O according to its external parameters.
}
```

To execute an operation O with the internal parameter *purl*, a *Partial Union effect eXecution* (PUX) procedure is devised.

The *PUX* procedure is an extension of the *PU* procedure for handling multiple partial union effects: creating *multiple* MV positions (Step 1), and copying *multiple* source cells and inserting them into destination cells (Step 2). Inside each of the two steps, a loop is used to handle multiple records in the *purl*, but the order of handling these records is insignificant. The reasons for this order-independency are: (1) multiple MV positions can be created in any order since each MV position is unique (a property of an MV position) and the creation of an MV position has no positional shifting effect on any cell in the document; (2) source and destination cells can be copied and inserted in any order because the positional shifting effect caused by destination cell insertions can always be absorbed by existing MV cells, thanks to the OCOD property of the MV cell. By definition, the effect achieved by $PUX(O_a)$ is the same as that by $PU(O_{a,,}O_b)$, as stated in the following proposition.

Proposition 2: Given two orthogonal conflict operations O_a and O_b, defined on the same 2D document state S, we have:

$$S \circ O_b \circ PU(O_{a,}, O_b) = S \circ O_b \circ PUX(O_a),$$

Proof: It follows directly from PUX and PU definitions.

Furthermore, it is important to point out that among the three steps in the *PUX*, only the last two steps may cause positional shifting effects along the dimension of O in the 2D document, and their overall effects are fully captured and represented by O's external parameters. Therefore, when transforming an operation O_x against O, we only need to examine the impact of O's external parameters on O_x.

Transforming External Parameters

To transform operation O_a against O_b, we must consider O_b's impact on O_a's external parameters. Three cases can be differentiated. First, O_a and O_b are compatible, i.e. $O_a \odot O_b$, so no change is made to O_a's external parameters. Second, O_a and O_b are 1D conflicting, i.e. $O_a \textcircled{/} O_b$, so the positional parameter of O_a must be adjusted according to the impact of O_b. The *1D Conflict Resolution (1DCR)* function is defined to achieve this. The *1DCR* function invokes the *NIT* (*New IT*) function to do the real work. The *NIT* function encapsulates the address conversion between 2D to 1D according to operation dimensions, and the invocation of suitable existing 1D *IT* functions according to operation types. We leave the definition of 1D *IT* functions unspecified because there are a variety of existing 1D *IT* functions (e.g. [3,9,10, 13]) to choose from and any suitably matching 1D *IT* function can be used.

```
1DCR (Oa, Ob) {
   Oa.p := NIT(Oa.p, Ob.p, Oa.dim, Oa.type, Ob.type);
   return Oa;
}
NIT (p1, p2, d, t1, t2) {
   if (d = row) { // row-oriented
      if (t1 = I && t2 = I) p1.x := IT_II(p1.x, p2.x);
      else {// other cases are omitted for conciseness }
   } else {// column-oriented
      if (t1 = I && t2 = D) p1.y := IT_ID(p1.y, p2.y);
      else {// other cases are omitted for conciseness }
   }
   return p1;
}
```

Third, O_a and O_b are 2D conflicting, i.e. $O_a \textcircled{\perp} O_b$. In this case, a *pur* is created to record the partial union effect, which can be achieved by the 2*D Conflict Resolution (2DCR)* function defined below. For conciseness, the *2DCR* definition assumes O_a is a row-oriented *Insert* while O_b is column-oriented *Insert* (definitions for other cases can be derived by following the same ideas illustrated here).

```
2DCR (Oa, Ob) {
  pur := new PUR; // create an empty pmr record
  if (Oa.dim = row && Oa.type = I && Ob.type = I) {
       pur.mv := (Oa.p.x, Ob.p.y);
       pur.sc := (Oa.p.x, Ob.p.y+1);
       pur.dc := (Oa.p.x+1, Ob.p.y);
   } else { //other cases are omitted for conciseness }
   if UniqueMVCheck(pur.mv,Oa,Ob) then Record(pur, Oa)
   return Oa;
}
```

In *2DCR*, a subroutine *UniqueMVCheck*(*pur.mv, O_a, O_b*) is invoked to check whether the MV position *pur.mv* is already in O_a or O_b. Only if the *pur.mv* does not exist in O_a and O_b, the new *pur* is recoded in O_a by calling *Record* (*pur, O_a*). This check is to ensure the uniqueness of an MV position in the 2D space. Actually, the OT control algorithm (e.g. COT), which ensures context-equivalence of transformed operations, also plays a role in enforcing MV uniqueness: if an operation is orthogonally conflicting with multiple operations at the same critical cell position, only one MV position will be created at this position regardless of the order of operation execution.

Transforming Internal Parameters

To transform operation O_a against O_b, it is also necessary to examine O_b's impact on internal parameters of O_a as well. The purpose of this transformation is to adjust O_a's internal parameters according to the positional shifting effect of O_b. Among three cell positions, *mv, sc*, and *dc*, in an internal parameter *pur*, only the *sc* and *dc* cells are subject to positional shifting effects of other concurrent operations. To illustrate the positional shifting effect on source and destination cells and to motivate key technical elements in

transforming internal parameters, consider the scenario in Figure 4. There are five operations with following causal and positional relations: $O_1 \parallel (O_2 \rightarrow O_3 \rightarrow O_4 \rightarrow O_5)$; O_1 is orthogonal with O_2, O_3 and O_5; O_1 and O_4 are performed on the same column; O_2 and O_3 are performed on the same row; O_5 is performed on a parallel row; O_1 is orthogonally conflicting with O_2, O_3, and O_5, and 1D conflicting with O_4. When O_1 arrives at site 2, it needs to be transformed against O_2, O_3, O_4 and O_5 in sequence, which is determined by the control algorithm (e.g. COT).

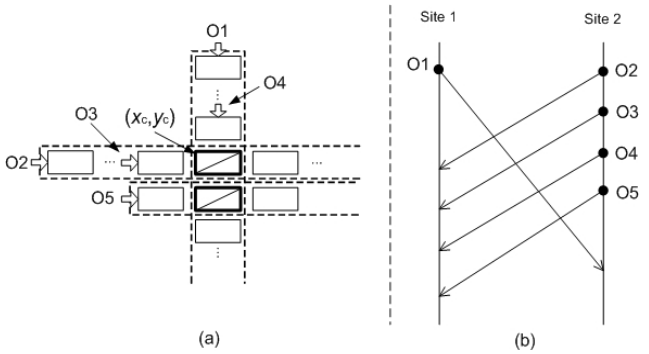

Figure 4 A scenario for illustrating transformation of source and destination cells in a *pur*.

The following steps occur at site 2. First, O_1 is transformed against O_2, and a *pur*-1 is created to resolve their orthogonal conflict and recorded in O_1. Then, O_1 is further transformed against O_3, which records no new *pur* in O_1 due to the MV uniqueness check. However, O_3 has positional shifting effects on the source cell *pur*-1.*sc* in O_1, so the position *pur*-1.*sc* should be adjusted according to the shifting effect of O_3. This adjustment is essentially resolving a 1D conflict between an internal parameter (*pur*-1.*sc)* of O_1 and O_3, and can be achieved by invoking the *NIT* on *pur*-1.*sc* against $O_3.p$.

Next, O_1 is transformed against O_4, which has positional shifting effects on not only O_1's external parameter (which can be resolved by the *1DCR* function), but also on the internal destination cell *pur*-1.*dc* which falls in the effect range of O_4. The shifting effect on *pur-1.dc* is also a 1D conflict relation between an internal parameter (*pur*-1.*dc)* of O_1 and O_4, and can be resolved by invoking the *NIT* on *pur*-1.*dc* against $O_4.p$ as well.

Finally, O_1 is transformed against orthogonal conflict operation O_5, and a new *pur*-2 is created and recorded in O_1 since it is unique. There is one thing special: O_5 has no shifting effect on the destination cell *pur*-1.*dc*, even though the effect range of O_5 covers *pur*-1.*dc*. This is because *pur*-1.*dc* has become an MV position in the new *pur*-2, due to orthogonal conflict resolution between O_1 and O_5. The *pur*-1.*dc* cell is always inserted in the same dimension as its hosting operation O_1 (see the definition of *PUX*), and hence must be orthogonal to the dimension of O_5. So, O_5 cannot shift *pur*-1.*dc* due to the ODOC property of an MV cell.

By analyzing all cases in this example, we can derive the following general conditions for a source/destination cell of an operation O_a to be affected by the positional shifting effect of another concurrent operation O_b:

1. the source/destination cell of O_a falls in the effect range of O_b (e transforming O_1 against O_3, and against O_4, in the above example), and
2. the source/destination cell is not an MV position (e.g. transforming O_1 against O_5 in the above example).

The positional shifting effect to source and destination cells is essentially a 1D conflict, stated as a proposition below.

Proposition 3. Given two operations Oa and Ob defined on the same 2D state, the positional shifting effect of Ob on Oa's an internal source/destination cell is a 1D conflict.
Proof: It follows directly from Definition 7.

Based on the general conditions for detecting positional shifting effect on internal parameters and the 1D conflict nature of such effects, the *Internal Parameter Transformation* function *IPT*(O_a, O_b) is devised, which uses the same *NIT* function based on an existing 1D *IT* to resolve this 1D conflict. One detail in invoking the *NIT* function is worth noting: the operation type for the source/destination cell position is *P* (for Pointer), which is neither *I* (for Insert) nor *D* (for Delete) because 1D *IT* functions for *P* against *I*/*D, i.e. IT_PI* and *IT_PD*, are different from *IT_II*, or other *I*/*D* combinations. The extra condition NOT($O_a \textcircled{\scriptsize ⊥} O_b$) is to ensure a source cell is not adjusted by O_b if it is orthogonally conflicting with O_a and hence is responsible for creating the current *pur* under exam.

```
IPT(Oa, Ob) {
   pl := Oa.purl;
   for (i = 0; i < |pl|; i++) {
      if pl[i].sc is in ER(Ob) but not an MV in Ob and
                                        NOT (Oa ⊥ Ob )
   then pl[i].sc:=NIT(pl[i].sc, Ob.p, Ob.dim, P, Ob.type)
      else if pl[i].dc is in ER(Ob) but not an MV in Oa
   then pl[i].dc:=NIT(pl[i].dc, Ob.p, Ob.dim, P, Ob.type);
   }
   return Oa;
```

2DIT Function for Resolving 1D and 2D Conflicts

Based on the component solutions, a *2D Inclusive Transformation* (*2DIT*) function is defined to provide an integrated solution for resolving both 1D and 2D conflicts.

```
2DIT (Oa, Ob) {
        if Oa ⫽ Ob then Oa := 1DCR (Oa, Ob);
        else if Oa ⊥ Ob then Oa := 2DCR (Oa, Ob);
        Oa := IPT (Oa, Ob);
        return Oa;
}
```

The *2DIT* function requires the two input operations O_a and O_b to be context-equivalent, but may have arbitrary positional relationships in the 2D document. The *2DIT* function first transforms O_a's external parameters by invoking *IDCR* to resolve 1D conflict, or *2DCR* to resolve 2D conflict, and then internal parameters by invoking *IPT* to resolve 1D conflict with internal position parameters.

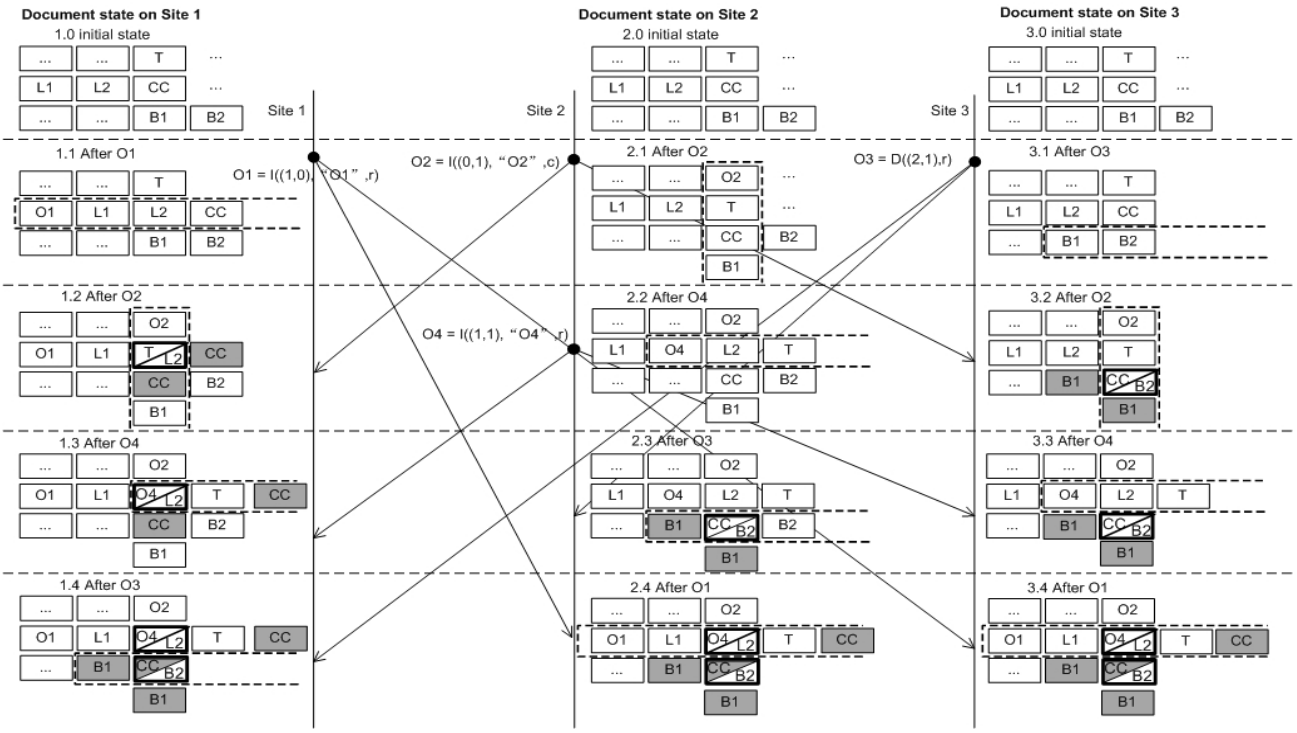

Figure. 5 A comprehensive example for illustruting the working of the 2DOT solution.

A COMPREHENSIVE EXAMPLE

In this section, one comprehensive example is used to illustrate how all components in the 2*DOT* solution work together to resolve 2D and 1D conflicts. The illustration assumes that, in each invocation to *2DIT*(O_a, O_b), O_a and O_b are *context-equivalent* ensured by the companion OT control algorithm (e.g. COT [15]). A 3-site collaborative 2D editing scenario is shown in Figure 5. Based on the positional and causal relationships of the four operations, their conflict relations can be derived: $O_1 \textcircled{\scriptsize 1} O_2$, $O_2 \textcircled{\scriptsize 1} O_3$, $O_1 \textcircled{\scriptsize 2} O_4$, and all other pairs are compatible. Main steps in processing each operation at all sites are explained below.

Site 1: in order of O1, O2, O4, and O3

1. O_1 is executed as-is.
2. To process O_2, *2DIT*(O_2, O_1) is invoked. Due to $O_2 \textcircled{\scriptsize 1} O_1$, *2DCR*($O_2$, O_1) is invoked, which creates *pur-1* =(*mv*(1, 2), *sc*(1, 3), *dc*(2, 2)) in O_2. Then, *IPT*(O_2, O_1) is invoked but does not change O_2. *PUX*(O_2) is invoked to create an MV at *pur*-1.*mv* with content [*/L2*], copy cell [*CC*] from *pur-1.sc* to *pur-1.dc*, and insert [O_2] at position $O_2.p$ = (0, 1), which shifts cell [*T*] into the MV position, resulting in [*T/L2*].
3. To process O_4, *2DIT*(O_4, O_1') is first invoked, where O_1' = *2DIT*(O_1, O_2). Due to $O_4 \textcircled{\scriptsize 2} O_1$', *1DCR*($O_4$, O_1') is invoked to shift $O_4.p$ from (1, 1) to (1, 2). Then, *IPT*(O_4, O_1') is invoked but has no effect as O_4 has no internal parameter. Finally, *PUX*(O_4) is invoked to insert [O_4] at the MV position (1, 2), which changes the MV content from [*T*/*L2*] to [O_4/*L2*].
4. To process O_3, *2DIT*(O_3, O_1) is first invoked but makes no change to O_3 since $O_3 \odot O_1$. Second, *2DIT*(O_3, O_2') is invoked, where O_2' = *2DIT*(O_2, O_1). As $O_3 \textcircled{\scriptsize 1} O_2$, *2DCR*($O_3$, O_2') is invoked to create *pur*-2 = (*mv*(2, 2), *sc*(3, 2), *dc*(2, 1)) in O_3, Third, *2DIT*(O_3, O_4') is invoked, where O_4' = *2DIT*(O_4, O_1'). Since $O_3 \odot O_4$, neither *1DCR* nor *2DCR* is invoked, and *IPT*(O_3, O_4') makes no change to O_3. Finally, *PUX*(O_3) is invoked to create an MV position at *pur*-2.*mv* with content [*CC*/], copy cell [*B*1] from *pur*-2.*sc* to *pur*-2.*dc*, and delete [O_3] at the cell at $O_3.p$ = (2, 1), which shifts [*B*2] into the MV position, resulting in [*CC*/*B*2].

Site 2: in order of O2, O4, O3, and O1

1. O_2 and O_4 are executed as-is in sequence.
2. To process O_3, *2DIT*(O_3, O_2) is first invoked, which in turn invokes *2DCR*(O_3, O_2) due to $O_2 \textcircled{\scriptsize 1} O_3$, and creates *pur*-3 = (*mv*(3, 2), *sc*(3, 2), *dc*(2, 1)) in O3. Second, *2DIT*(O_3, O_4) is invoked but makes no change to O_3 since $O_3 \odot O_4$ and O_4 has no effect on *pur*-3 in O_3. Finally, *PUX*(O_3) is invoked to execute O_3.
3. To process O_1, *2DIT*(O_1, O_2) is first invoked, which in turn invokes *2DCR*(O_1, O_2) due to $O_1 \textcircled{\scriptsize 1} O_2$, and creates *pur*-4 = (*mv*(1, 2), *sc*(2, 2), *dc*(1, 3)). Second, *2DIT*(O_1, O_4) is invoked, which in turn invokes *1DCR*(O_1, O_4) due to $O_1 \textcircled{\scriptsize 2} O_4$, and does not change $O_1.p$; *IPT*(O_1, O_4) shifts *pur*-4.*dc* = (1, 3) to (1, 4) as the *pur*-4.*dc* falls in the effect range of O_4, and *pur*-4.*dc* is not an MV in O_4. Third, *2DIT*(O_1, O_3') is invoked, where O_3' = *2DIT*(*2DIT*(O_3, O_2), O_4) . Since $O_1 \odot O_3$, neither *1DCT* nor *2DCR* is invoked, and *IPT*(O_1, O_3') does not change O_1, though *pur*-4.*sc* = (2, 2) falls in the effect range of O_3', because O_3' has an MV at *pur*-4.*sc* which was created in *2DIT*(O_3, O_2). Finally, O_1 is executed by invoking *PUX*(O_1).

Site3: in order of O3, O2, O4, O1

1. O_3 is executed as-is.

2. To process O_2, $2DIT(O_2, O_3)$ is first invoked, which in turns invokes $2DCR(O_2, O_3)$ due to $O_2 \textcircled{\scriptsize ⊥} O_3$, and creates *pur*-5 = (*mv*(2, 2), *sc*(2, 1), *dc*(3, 2)). Finally, O_2 is executed by $PUX(O_2)$.
3. To process O_4, $2DIT(O_4, O_3')$ is first invoked, where $O_3' = 2DIT(O_3, O_2)$. Since $O_4 \odot O_3$, neither *1DCT* nor *2DCR* is invoked, and $IPT(O_4, O3)$ does not change O_4 either since O_4 has no internal parameter. Finally, O_4 is executed as-is by $PUX(O_4)$.
4. To process O_1, $2DIT(O_1, O_3)$ is invoked, which makes no change to O_1 since $O_1 \odot O_3$ and O_1 has no internal parameter. Then, $2DIT(O_1, O_2')$ is invoked, where $O_2' = 2DIT(O_2, O_3)$. As $O_1 \textcircled{\scriptsize ⊥} O_2$, $2DCR(O_1, O_2')$ is invoked to create *pur*-6 = *mv*((1, 2), *sc*(2, 2), *dc*(1, 3)) in O_1. Next, $2DIT(O_1, O_4')$ is invoked, where $O_4' = 2DIT(O_4, O_3')$. Since $O_1 \textcircled{\scriptsize //} O_4$, $IDCR(O_1, O_4')$ is invoked, but makes no change to O_1 as $O_1.p$ is at the left side of $O_4'.p$. $IPT(O_1, O_4')$ changes *pur*-6.*dc* from (1, 3) to (1, 4) since *prm*-6.*dc* falls in the effect range of O_4' and is not an MV in O_4'. O_1 is executed by $PUX(O_1)$.

From the final document states at three sites, we see consistent results have been achieved by using *2DIT* functions under the control of a COT-like algorithm.

CORRECTNESS AND COMPLEXITY ANALYSIS

One assumption: *2DIT* functions are used under the control of the COT algorithm for consistency maintenance only. We only need to verify the correctness of *2DIT* functions with respect to *CP1* (Convergence Property 1) because other criteria, such as *CP2* (*Convergence Property* 2), and *IP1-3 (Inverse Properties* for supporting group undo), are irrelevant to *2DIT* under this assumption [15]. Another assumption is that the *1DIT* functions adopted by *2DIT* functions are able to preserve *CP1,* which is easy to achieve [3,9,10,11,14,15]. Thus, the following theorem establishes the correctness of *2DIT*.

Theorem 1: Given two operations O_a and O_b defined on the same 2D document state, if $O_a' = 2DIT(O_a, O_b)$, and $O_b' = 2DIT(O_b, O_a)$, then we have:

$$S \circ O_a \circ PUX(O_b') = S \circ O_b \circ PUX(O_a')$$

Proof: For the given pair of operations O_a and O_b, they may have three possible conflict/compatible relationships: compatible, 1D conflict, or 2D conflict. We show that the theorem holds in all cases. In the compatible case, the theorem holds trivially. The 1D conflict case covers both external and internal parameters because the positional shifting effects on internal parameters are 1D conflicts according to Proposition 3. In this case, the theorem holds because of the *CP1*-preserving property of 1D *IT* functions (by assumption) adopted by *2DIT* to resolve 1D conflict for both external parameters (in *2DCR*) and internal parameters (in *IPT*). In the 2D conflict case, $2DIT(O_a, O_b)$ invokes $2DCR(O_a, O_b)$ to record the partial union effect in the internal parameters of O_a, and the transformed O_a' will be executed by invoking $PUX(O_a')$. According to Proposition 2, the effect of $PUX(O_a')$ is exactly the same partial union effect achieved by $PU(O_a, O_b)$, i.e.

$$S \circ O_b \circ PUX(O_a') = S \circ O_b \circ PU(O_a, O_b).$$

Following the same reasoning, we can derive:

$$S \circ O_a \circ PUX(O_b') = S \circ O_a \circ PU(O_b, O_a).$$

By Proposition 1, the theorem holds in the 2D conflict case as well. Thus the theorem holds for all cases.

For OT time complexity, a common measurement is the number of transformation function invocations for each operation to be executed [14]. From the assumption that *2DIT* functions are used under the control of COT, which has a linear time complexity (with respect to concurrent operations) for consistency maintenance [15], we derive the same linear time complexity for the 2DOT solution in terms of 2*DIT* function invocations. A *2DIT* invocation may include one *1DIT* call for transforming external parameters, plus additional *1DIT* calls for transforming internal parameters (source/destination cells) due to orthogonal conflict, which is bounded by the number of concurrent operations as well. So, the time complexity of the *2DOT* solution in terms of *1DIT* function invocations is $O(n^2)$, where n is the number of operations that are concurrent or context independent with the operation to be executed. More work is needed for real performance measurement and evaluation of 2DOT in the future.

Some caveats for theoretic verification of OT correctness and complexity analysis. Theoretic verification plays an important but limited role in validating an OT solution since verification criteria (e.g. *CP1-2, IP1-3*) cover only a restricted part of a complete OT solution, and not all criteria are relevant to every OT solution. Many aspects of an OT solution do not lend themselves well to rigorous description and verification, and can be more effectively addressed by real implementation and benchmarking [14]. Worst case time complexity rarely matters in real-time collaborative sessions since the degree of concurrency is often small and a transformation function call is cheap [14]. In reality, time efficiency of a real OT system has much more to do with its practical implementation than theoretic complexity. More comprehensive discussions on the facts and myths related to OT correctness and complexity can be found in [14].

COMPARISON TO RELATED WORK

Conflict resolution has been explored in a wide range of collaborative systems. Floor control, locking, and serialization were commonly used conflict prevention and resolution strategies for supporting collaborative work [4], but they were found too restrictive for collaborative work that require concurrent and free interactions [2,3,10,13]. Due to its lock-free and unconstrained interaction properties, OT is particularly suitable for supporting real-time collaboration over the Internet [1,7,9,11,12,14-17].

To the best of our knowledge, there exist only two prior work directly related to applying OT to collaborative spreadsheet/table editing systems [8,16]. In [8], transformation functions were defined for operations with 2D addresses, rather than 1D addresses as in most other OT systems. A set of spreadsheet operations were derived from a public domain spreadsheet system *sc*, which include: *insert* (full rows/columns), *delete* (full rows/columns), and

set, *format*, and *copy* the cell value in a spreadsheet. Concurrent semantics for the set of operations were defined and transformation functions were proposed to resolve linear positional shifting conflicts (or 1D conflicts) among concurrent operations. 2D orthogonal conflicts, however, may occur among concurrent operations inserting or deleting *any number of* cells, including a full row/column of cells, but this 2D conflict issue was not addressed in [8].

In [16], a 2D table is represented as a sequence of rows (row-based) or columns (column-based); and 1D OT techniques (control algorithms and transformation functions) are directly used to support operations on tables. In a row-based table representation, cells belonging to the same row can be addressed by their indexes in the row, but cells belonging to the same column are scattered among multiple rows. Consequently, row-oriented operations can be processed efficiently, but column-oriented operations are inefficient as they have to be decomposed into multiple operations targeting individual cells. The reverse is true for column-based representation. This 1D hierarchical approach works reasonably well when the table size (the number of cells in a row or column) is relatively small (e.g. tables in word processor documents), but it does not scale well to tables with a very large number of rows/columns, like Microsoft Excel spreadsheets (with 64K rows or columns). Moreover, the technique in [16] was unable to resolve the orthogonal conflict issue either.

2DOT provides a solution to orthogonal conflict, but it alone is not a complete solution to supporting collaborative 2D applications. 2DOT needs to be integrated with suitable OT control algorithms like COT [15], and with other transformation functions like those from [8, 16] to provide a comprehensive OT solution to collaborative 2D systems. In our future work, 2DOT will be extended to handle data schemas (or constraints) and support group undo in real-time collaborative spreadsheet editing sessions.

CONCLUSIONS

In this work, we have contributed a novel OT technique for resolving the orthogonal conflict problem in collaborative 2D document editing systems. This work is the first to identify, formalize and solve the orthogonal conflict problem in real-time collaborative systems. The extension of OT from 1D to 2D conflict resolution is fundamental to the theory and practical application of OT, and represents an important advancement to the state-of-the-art of OT.

When OT was originally invented to solve the 1D positional shifting problem in concurrent inserting/deleting characters in a string, few people predicted such primitive capability would evolve into a tech for supporting a range of real-world collaborative applications. 2DOT was invented to resolve orthogonal conflict (2D positional shifting problem) in spreadsheets, but we anticipate potential application of 2DOT to any collaborative computing problem that can be modeled as concurrent operations inserting/deleting objects in multi-dimensional workspaces. Future exploration will discover more applications as well as limitations of OT.

ACKNOWLEDGEMENT

The authors wish to thank anonymous expert reviewers for their insightful and encouraging comments and suggestions.

REFERENCES

1. Agustina, C. Sun, and D. Xu,"Operational transformation for dependency conflict resolution in real-time collaborative 3D design systems," *ACM Conf. on CSCW*, 2012, Feb. 2012.
2. J. Begole, M. B. Rosson, and C. A. Shaffer, "Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems," *ACM Transactions on Computer-Human Interaction,* 6(2), pp. 95-132, 1999.
3. C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," *ACM SIGMOD Conf.*,1989, pp. 399-407.
4. S. Greenberg and D. Marwood, "Real time groupware as distributed system: concurrency control and its effect on the interface." *ACM Conf. on CSCW*, pp.207–217, Oct. 1994.
5. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *CACM*, 21(7), pp. 558-565, 1978.
6. D. Li and R. Li, "Transparent sharing and interoperation of heterogeneous single-user applications," *ACM Conf. on CSCW*, 2002, pp. 246-255.
7. D. A. Nichols, P. Curtis, M. Dixon and J. Lamping, "High-latency, low-bandwidth windowing in the Jupiter collaboration system," *ACM Symp. on UIST*, 1995, pp. 111-120.
8. C. Palmer and G. Cormack, "Operation transforms for a distributed shared spreadsheet," *ACM Conf. on CSCW*, 1998, pp. 69-78.
9. M. Ressel, et al, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," *ACM Conf. on CSCW*, 1996, pp. 288-297.
10. 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), pp. 63-108, 1998.
11. C. Sun and C. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements," *ACM Conf. on CSCW*, 1998, pp.59-68.
12. C. Sun, S. Xia, D. Sun, D. Chen. H.F. Shen and W. Cai: "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM Transactions on Computer-Human Interaction,*13(4), pp. 531 – 582, Dec. 2006.
13. D. Sun, S. Xia, C. Sun and D. Chen, "Operational transformation for collaborative word processing," *ACM Conf. on CSCW, 2004,* pp. 437-446.
14. C. Sun,"OTFAQ: Operational Transformation Frequent Asked Questions and Answers," http://cooffice.ntu.edu.sg/otfaq,2011.
15. D. Sun, and C. Sun, "Context-Based Operational Transformation in Distributed Collaborative Editing Systems," *IEEE Transactions Parallel Distributed Systems*, 20(10), pp. 1454-1470, 2009.
16. S. Xia, D. Sun, C. Sun, and D. Chen, "A collaborative table editing technique based on transparent adaptation," *Intl. Conf. on CoopIS*, 2005, pp. 576 - 593.
17. D. Wang and A. Mah, "Google Wave Operational Transformation",h*ttp://www.waveprotocol.org/whitepapers/operational-transform*.