JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXXX-XXXX (2024)
DOI: XX.XXX.XXXX.XX

Appendix: All-Operation-Effects Intention Preservation based on Operational Transformation for Multimedia Collaboration

WEIWEI CAI, XIAOHU YAN[†+], CHUNMEI ZHANG, ZHEN HU[‡]
School of Mathematics and Computer Science
Wuhan Polytechnic University
Wuhan, 430048 P.R. China
† Undergraduate School of Artificial Intelligence
Shenzhen Polytechnic University
Shenzhen, 518055 P.R. China
E-mail: yanxiaohu@szpu.edu.cn[+]
‡ Wuhan Hazhengyu Environmental Technology Co., Ltd

Recent decades have seen a growing interest and demand for multimedia collaboration systems that aim to provide a multi-user environment in which geographically distributed users can collaborate on multimedia over the Internet. Operational transformation (OT) is a widely adopted collaboration technique. However, OT-based systems may suffer a false-tie puzzle in consistency maintenance, causing the problem of operation intention preservation. While several research efforts have been made to address the puzzle, most of these efforts achieve the goal at the price of sacrificing the efficiency and flexibility of OT-based systems. In this paper, we propose a novel false-tie solution that preserves all operation effects, minimizing the impact on flexibility and efficiency as much as possible. The concepts and implementation invented in the proposed solution would be beneficial to the development and progress of multimedia collaboration.

Keywords: Multimedia Collaboration, Intention Preservation, Operation Transformation, Collaborative Systems

1. Object Sequence Manipulation

An object sequence consists of a list of objects. Each object (denoted as Ω) has the following attributes:

1. Lop(Ω) records a set of timestamped operations that take effect on Ω;
2. Len(Ω) represents the number of characters that are introduced by the operations in Lop(Ω) or a part of the characters in the replicated and shared document;
3. Next(Ω) represents the successor object.

An empty object sequence *OS* has three objects: the head object Ω_{beg}, the middle object Ω_{mid}, and the tail object Ω_{end}:

1. Lop(Ω_{beg})=Lop(Ω_{end})=Lop(Ω_{mid})=[ ],
2. Len(Ω_{beg})=Len(Ω_{end})=0, Len(Ω_{mid})=∞,

Received March 11, 2014; revised June 20, 2014; accepted September 28, 2014.
[+]Corresponding author

3. Next(Ω_{beg})=Ω_{mid}, Next(Ω_{mid})=Ω_{end}.

Three subroutines *Visible*, *Appeared*, and *Split* are used. *Appeared*(Ω, O) determines whether Ω appeared before O is applied. *Visible* and *Appeared* depend on the context representation. Assume a context is represented by a vector clock. If O_a is included in CT_{O_b}, then $VC(O_a)[Sid(O_a)] \leq VC(O_b)[Sid(O_a)]$. Split($\Omega$,*num*) splits an object Ω into two neighboring objects Ω_1 and Ω_2: Len(Ω_1)=*num*, Len(Ω_2)=Len(Ω)-*num*, Next(Ω_1)=Ω_2. If $num \geq Len(\Omega)$, nothing is done.

Algorithm 1 FindPrev(OS, O) : Ω

```
1:  num = 0, Ω = Ω_beg;
2:  if Pos(O) > 0 then
3:      while Next(Ω) ≠ Ω_end and Pos(O) ≠ num do
4:          Ω = Next(Ω);
5:          if Visible(Ω,O) is true then
6:              Split(Ω, Pos(O) − num);
7:              num = num + Len(Ω);
8:          end if
9:      end while
10: end if
11: return Ω;
```

According to the given operation O, *FindPrev* in Algorithm 1 searches the object sequence for a target object Ω: (1) Ω is visible to O; Pos(O) equals the total length of visited objects from Ω_{beg} to Ω (inclusion) that are visible to O. In the case of Pos(O)$= 0$, the head object Ω_{beg} is returned. In the case of Pos(O)> 0 (lines 2-10), the object sequence is traversed until the target object is found.

Algorithm 2 Mark(OS, Ω_p, O)

```
num = 0, Ω = Ω_p;
while Next(Ω) ≠ Ω_end and Len(O) ≠ num do
    Ω = Next(Ω);
    if Visible(Ω, O) is true then
        Split(Ω, Len(O) − num);
        num = num + Len(Ω);
        Lop(Ω)=Lop(Ω)+O;
    end if
end while
```

According to the given operation O, *Mark* in Algorithm 2 marks a list of neighboring objects following after Ω_p. The marked objects hold the conditions: (1) they are visible to O; and (2) their total length equals Len(O).

According to the given operation O, *Add* in Algorithm 3 finds a proper position after Ω_p in the object sequence OS for a new object Ω_{new} that is introduced by O. In lines 2-11, the algorithm traverses the object sequence from the successor of Ω_p to the tail object. For each visited object Ω, if Ω appears, then Ω_{new} is placed before Ω; otherwise, Ω is

Algorithm 3 Add(OS, Ω_p, O)

1: create a new object Ω_{new} with O;
2: $\Omega_{pre} = \Omega_p$;
3: **while** true **do**
4:     Ω = Next(Ω_{pre});
5:     **if** $Appeared(\Omega, O)$ is true **then**
6:         **break**;
7:     **else if** Pri(Lop(Ω)[0], O)>0 and $Appeared(\Omega$, Lop(Ω_{pre})[0]) is true **then**
8:         **break**;
9:     **end if**
10:     Ω_{pre} = Next(Ω_{pre});
11: **end while**
12: add Ω_{new} after Ω_{pre};

introduced by an operation context-independent to O, and the priorities are compared. It should be noted that in line 7, Lop(Ω)[0] must be an insert context-independent to O, because Appeared(Ω,O) is false; $Appeared(\Omega$, Lop(Ω_{pre})[0]) means Ω appears before Ω_{pre}. More details can be found in [1, 2].

Algorithm 4 Apply(OS, O)

Ω_p = FindPrev(OS, O);
if O is insert **then**
    Add(OS, Ω_p, O);
else ▷ O is delete
    Mark(OS, Ω_p, O);
end if

According to the given operation O, *Apply* in Algorithm 4 illustrates how to apply insert/delete on object sequence, with respect to Definition 2 and 3.

Algorithm 5 TPOS(O, OS)

$num = 0$;
for $\Omega = Next(\Omega_{beg})$; $\Omega \neq \Omega_{end}$; Ω = Next(Ω) **do**
    if Lop(Ω) is not empty and Lop(Ω)[0] is O **then**
        break;
    end if
    $num = num$ + Len(Ω);
end for
return num;

According to the given operation O, *TPos* in Algorithm 5 counts the total length of objects positioned before O's target objects by traversing the object sequence.

2. Correctness Verification

Lemma 1 *After applying the same set of operations on an empty object sequence in different orders that respect causal orders, the resulting object sequences are identical.*

Lemma 2 *Given two context-independent operations O_a and O_b and an object sequence OS, Apply(O_a,OS) and Apply(O_b,OS) are commutable, i.e. applying O_a and O_b in different orders on OS produces the same object sequence.*

The object sequence manipulation of FTF is the same as that of AST and ASTO. Lemma 1 and Lemma 2 has been proved in AST [1] and ASTO [2].

Lemma 3 *Given two context-independent insert operations O_a and O_b, TPos(O_a,OS)-TPos(O_b,OS) is consistent under the common context, where CT(OS) = CT(O_a,O_b).*

According to Lemma 2, context-independent operations cannot reverse the orders of concurrently inserted objects.

Lemma 4 *Assume a set of operations have applied on an empty object sequence OS and is included in the context of any forthcoming operations, the resulting object sequence OS' is equivalent to the empty object sequence, i.e., $OS' \equiv OS$.*

Proof. Assume U is the operation set. For any object Ω in OS', there are 3 cases:

(1) Lop(Ω) is empty;

(2) Lop(Ω) has only one operation O, O must be included in U; obviously Ω is visible to any forthcoming operations and it can be merged into adjacent objects that are also visible to any forthcoming operations;

(3) Lop(Ω) has more than one operation, Ω must be invisible to any forthcoming operations and it can be removed.

After the target objects of operations in U are merged or removed, OS' is identical to OS. Readers can refer to [2] for more in-depth discussions about the mergeable and removable conditions.

Lemma 5 *Given two context-independent insert operations O_a and O_b, and two object sequences OS and OS', if $OS \equiv OS'$, then TPos(O_a,OS)-TPos(O_b,OS) is consistent with TPos(O_a,OS')-TPos(O_b,OS').*

Theorem 1 *Given two context-independent insert operations O_a and O_b, TPos(O_a,OS)-TPos(O_b,OS) is consistent with TPos(O_a,OS')-TPos(O_b,OS') , where CT(OS)=CT(O_a,O_b) and CT(OS')=MICT(O_a,O_b).*

Proof. Context relations between O_a and O_b are enumerated. There are two cases.

(1) the first case, O_a and O_b are context-equivalent, $CT_{O_a} \cap CT_{O_b} = CT_{O_a} \cup CT_{O_b}$, $CT_{O_a} \triangle CT_{O_b}$ and CD(O_a,O_b) are empty; thus $CT(O_a, O_b) = MICT(O_a, O_b)$, leading to $OS = OS'$.

(2) the second case, O_a and O_b are not context-equivalent, $CT_{O_a} \triangle CT_{O_b}$ and CD(O_a, O_b) are not empty. According to Lemma 1, after applying CD(O_a,O_b) on an empty object sequence, the object sequences converge. For each operation O in CD(O_a, O_b), O is included in the context of an operation O_t in MICT(O_a,O_b), i.e., O has been executed at the time of O_t's generation. According to Lemma 4, the object sequence built on an empty object sequence with MICT(O_a,O_b) is equivalent to the object sequence built from an empty object sequence with CT(O_a,O_b).

(3) ether $OS = OS'$ or $OS \equiv OS'$, according to Lemma 5, the theorem holds.

REFERENCES

1. N. Gu, J. Yang, and Q. Zhang, "Consistency maintenance based on the mark & retrace technique in groupware systems," in *Proc. of ACM GROUP*, 2005, pp. 264–273.
2. W. Cai, F. He, S. Yang, and X. Lv, "Self-compressing object sequence for consistency maintenance in co-editors," *Softw. Pract. Exp.*, Vol. 52, no. 8, 2022, pp. 1802–1825.