Consistency Maintenance of Compound Operations in Real-time Collaborative Environments

Liping Gao*, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China. Email: lipinggao@usst.edu.cn

Fangyu Yu, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China. Email: fangyuyu@st.usst.edu.cn

Lily Gao, Pre-school and Special Education Normal School, Weifang University, Shandong, 261021, China. Email: snowgao@163.com

Naixue Xiong*, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China. Email: nxiong@coloradotech.edu.cn

Guisong Yang, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China. Email: gs_yang@yahoo.cn

Abstract: In real-time collaborative environments, address space transformation strategy can be used to achieve consistency maintenance of shared documents. However, as for the execution of compound operations, they are firstly decomposed into primitive operations, the relationships between the referencing objects and referenced objects are lost during the decomposition process. Besides, the Undo operations in this environment are targeted at compound operations, but not decomposed basic ones. However, the traditional algorithms take primitive operation as the manipulation unit, thus leading to semantic inconsistencies of compound Undo operations. This paper appends two history buffers to maintain the relationships between the original operations and the decomposed ones and introduces “Retrace-Undo-VT-Redo-Retrace” strategy to realize the consistency maintenance of compound operations. Also, this paper introduces the version-decomposition strategy, describes the main algorithms of the compound Undo operations and analyses the validity of the strategy. Case analysis is given to show the effectiveness of the strategy.

Keywords: Real-time Collaboration, Address Space Transformation, Consistency Maintenance, Compound Operation, Undo/Redo

1. INTRODUCTION

In parallel computing systems, a task is decomposed into multiple subtasks which are allowed to be carried out simultaneously, and parallel computing has been applied to different fields. Within the parallel collaborative design field, collaborative groupware systems support the concurrent editing of people who are geographically dispersed all over the world. In order to achieve high responsiveness and unlimited collaboration, replicated architecture is adopted widely [1-2]. In this architecture, local operations are executed immediately while remote operations are required to execute inclusion transformation (IT) or exclusion transformation (ET) to include or exclude the execution effects of other concurrent operations (that is the Operation Transformation strategy, abbr. as OT) [1-2, 23-25], or remote operations are executed under the retraced document status (that is the Address Space Transformation strategy, abbr. as AST) [3, 26]. Many issues in this research field have attracted the focus of the researchers and practitioners, such as consistency maintenance [1-4], Undo/Redo [5-6, 27], conflict resolution [7-8, 28-30], intention preservation [1-4] etc. There are some prototype systems in both text editor field (such as CoWord [9], CoPowerpoint [10] etc.), multimedia field (such as CoFlash [31] etc.), and computer-aided design field (such as CoAutoCAD [11], CoMaya [12] etc.). These

systems provide Insert, Delete, Update and Undo/Redo operations which are targeted at basic objects.

However, not all objects in the collaborative design environment are basic ones and not all operations in this field are Insert, Delete, Update or Undo/Redo operations. In fact, the operations in the collaborative design environment can be categorized according to two factors: the number of the target objects and the reference attribute of the operations (See Table 1). In table 1, the SO-SO Ops are also called primitive or atomic ones. And only these operations can be tackled directly by the traditional consistency maintenance algorithms (such as OT, AST, etc.). Compound operations refer to the operations, which need to be decomposed into one or several primitive operations before execution [13-14]. MO-SO, SO-RO, and MO-RO belong to compound operations. During the decomposition and broadcasting process of the compound operations, the relationship between the decomposed operations and the original operation is lost, which leads to the independent execution of these operations. And in execution process, the dependency relationship between the target object and the reference object is lost, thus resulting in the intention violation of the compound operations.

Table 1. The classification of the operations in the collaborative design environment

	Single Object	Multiple Object
Single Operation	SO-SO Op	MO-SO Op
Reference Operation	SO-RO Op	MO-RO Op

Take Fig.1 as an example. Two designers are responsible for the design of a building's outline. One designer completes the outline design of one window. Another designer releases *Mirror* operation to create a symmetric window. However, during the release process of the *Mirror* operation, if the first designer releases an *Update* operation to change the window's height, due to the lack of semantic consistency maintenance, the two windows with different height attribute will be obtained (See Fig. 1(b)), which violates the semantic intention of the *Mirror* operation for the two designers.

Figure 1. Semantic inconsistency of Concurrent Mirror and Update Operations

When users press the Undo or Redo button through the interface, only the last executed operation (locally or globally according to the different undo schema [6]) will be selected as the target operation (See Fig. 2), which destroys the user's semantic intention. Moreover, as for the reference operations, since the execution effect has been reflected in its constraint following operations by virtual transformation (VT) function [13], the Undo operation of that operation cannot be executed without removing its effect on its constraint following operations (See Fig. 3). Thus, strategies must be designed to solve these problems from the perspective of multiple objects and reference operations. Note that Obj_1 denotes the whole annulus, Obj_3 denotes the outer circle and Obj_4 is the inner circle in Fig 2 and Fig 3 below.

Figure 2. Undo Operation of the Operations with Multiple Target Objects

Figure 3. Undo Operation of the Reference Operations

Our research group has proposed to use CLAF (Check-Last-Avoid-Future) strategy to maintain the semantic consistency of the step-wise operations [13]. In fact, compound operations include not only step-wise operations (which need N steps to be completed, where $N \geq 2$), but also some single-step operations (which need to reference other objects to get the parameters of itself). The paper focuses on the semantic consistency maintenance of the compound operations as well as the compound Redo/Redo operations based on AST strategy. The paper introduces to add Original Op HB and Local-exe Op HB to the AST framework to maintain the relationship between the reference operations, modifies the operation generation process of the local site and the operation reception process of the remote sites to maintain the relationship during the broadcasting process, introduces “Retrace-Undo-VT-Redo-Retrace” to “Mark & Retrace” process of the AST strategy to realize the consistency maintenance of the compound operations. Besides, the paper analyses the deficiencies of the resolution process of the Undo/Redo operations with the traditional Undo/Redo schema, discusses the pre-condition of the Undo/Redo operations, gives new definition for the conflict operations between Undo/Redo operations and other operations, devises conflict resolution strategy based on multi-version decomposition strategy, describes the work flow of compound Undo/Redo operations both in the local sites and the remote sites, and details the main algorithms.

The rest of the paper is organized as follows. Section 2 describes the related works. In Section 3, the AST strategy and concepts of basic operations are introduced. Section 4 describes the consistency maintenance algorithm of the compound operations based on AST. In Section 5, we analyze the consistency maintenance of the compound Undo/Redo operations. Section 6 discusses the algorithm’s validity. Finally, the paper is concluded with summarizing our main contributions and discussing future work.

2. RELATED WORK

As for the intention preservation in collaborative design environments, different researchers have proposed different resolution strategies. Norrie [15] adds Group/Ungroup operations to extend the primitive operations based on OT, which supports the intention preservation of the group operations in the graphical editing environment. This is the first try to extend primitive operations in the text editor field to graphical field for supporting more operations. Li [16] presents an active rule based approach to

model user-defined semantic relationships in collaborative applications. However, he does not address more complex issues such as constraint violation. Also, CAB only focuses on maintaining constraints among different objects but not among different operations. Lin [17] proposes to extend OT to represent the rule constraints between different operations, however, the reference operations in the CAD field cannot be unified into mathematical function expressions. Hala [18] achieves semantic consistency by integrating semantic constraints into the operational transformation method. However, SAMS is based on XML resources and it is not clear whether SAMS can be used in other environments. Moreover, it does not guarantee constraint satisfaction. Wang [19] intends to achieve semantic preservation in real-time collaborative graphical systems and devises semantic expressions to express constraints. However, it cannot represent temporal constraints between objects. CoGSE can maintain both tree-structured constraints and system consistency in the face of concurrent operations. However, it supports only static constraints, but not dynamic ones. All the systems mentioned above only address constraints among different operations. The work of Lin [20] is somewhat similar with that of Wang. Tang [21] proposes to use feature re-arrangement technology to guarantee the intention preservation of different feature modeling operations in the 3D environment, which has different research context of our research work, since we mainly focus on two dimensional collaborative design environments. Sun [22] proposes to maintain the dependency relationship between different objects in the directed graph model. However, the graph model itself maintains the reference relationship between different objects, and the document model in our system is still linear structure, but not directed graph model. Fan [30] proposes a Dependency-based Automatic Locking (DAL) approach for supporting semantic conflict prevention in real-time collaborative programming environments. His work is somewhat similar with ours in that both the works intend to preserve the dependency relationship between different objects. However, the DAL strategy uses lock to realize that, which may reduce the concurrency of the whole system.

Undo/Redo operations are used frequently in single-user application, which are used to recover from an error condition or to motivate users to try different schemas to accomplish some objectives or tasks [5-6, 27]. Most design software (such as AutoCAD, Inventor, etc.) provides users with sequential Undo mechanisms to operate Undo operations from back to front in order. As it has been pointed out in [6], the undo mode is defined according to two facts: undo scope (global or local) and undo model (single-step undo, chronological undo and selective undo). The global undo supports to undo any operation released by all users (including both him/her and other users) while local undo restricts the undo scope of the operations to those generated at local site. Single-step undo indicates that user can undo and only undo the latest operation released by the local or another remote user. Chronological undo means that user can undo a sequence operation from the latest to the oldest sequence. And Selective undo supports any undo at any time.

However, the classification of the above undo modes is related not with the original operations but with the decomposed ones. For example, in Fig.3, the intention of the Undo operation is to undo the creation of the two objects, but not the last created object. This phenomenon occurs because the target operation of the Undo operation is the decomposed operation but not the original ones. So strategies must be adopted to maintain the relationship between the original ops and the decomposed ones. Bin [5] extends the Undo object from single character to string. That's to say, we can undo a set of characters, but not one character at one time. In this work, the operations of Insert or Delete are delayed to be combined as a string operation and then broadcasted to remote sites. This work style is fit for the mobile environment since the network bandwidth is limited. However, in order to achieve high

concurrency, operations in the ordinary environment should be broadcasted to remote sites as soon as possible. Cheng [32] employs a complete re-run strategy based on full operation for 3D collaborative modeling systems. It simply re-executes all the operations from the beginning except the Undo target and operations depending on it. The complete re-run strategy is simple and its correctness can be guaranteed. However, it has low efficiency and cannot be adaptive to large-scale design environments. He [27] proposes a group Undo/Redo mechanism for 3D collaborative modeling systems to support the “anytime, anywhere” Undo/Redo. It uses an Undo State Vector to ensure the Undo/Redo intention preservation and model consistency maintenance. However, this work is mainly focused on the feature modeling operations and the document structure in this work is modeled as Topological Entity Structure Tree (TEST) which is different from our model at all.

3. THE AST STRATEGY AND COMPOUND OPERATIONS

3.1 Address Space Transformation Strategy

Address space transformation strategy, proposed by Gu [3], is a consistency maintenance method based on document mark and retrace. Different with traditional OT strategy, AST takes the perspective of document status and thinks that the intention violation of concurrent operations results from different document statuses of the operation execution and the operation generation. Based on this perspective, AST introduces to retrace the document status to that when the operation is generated by comparing the timestamp of the operation with those of the operations in the history buffer.

AST is firstly used in the text editor field, aiming to solving consistency maintenance problem of document consisting of plain characters. It formats the document as a linear structure composed of character nodes, with every node appended to a history buffer containing the operations that have taken this node as the operation target. Operations in the list are arranged according to the execution order. The Effective/Ineffective status of the node is determined by comparing the timestamp of the node with that of the site. In order to adapt AST strategy to collaborative design field, some strategies must be adopted including address space mapping etc. Please consult to [11] for more information about this.

The main idea of AST is as follows: 1) Operations released from local site are executed immediately after their releases; 2) As for a remote operation, before its execution, causality-ready condition will be first examined. If it is not causally-ready, it will be queued, since some other operations which have been executed in their generation sites have not been executed in this site. If it is causally-ready, the document status of current site will first be retraced to that when the operation is generated and then the operation is executed in this retraced document status. After executing the operation, the document status will be retraced to current status. The readers are recommended to consult to [3] for more description about AST strategy.

3.2 Compound Operations

In this paper, compound operations refer to those original operations which have to be decomposed into one or several primitive operations before their execution. Based on Multi-Object operations and Reference operations, compound operations can be classified into MO-SO, SO-RO and MO-RO. Here, definitions of four compound operations are introduced, and some examples of compound operations including Copy-Paste, Mirror, Array and Extend are also detailed.

Compound Operations: operations that need to be decomposed into primitive operations before

execution by the basic consistency maintenance algorithms such as OT or AST. They either need some steps to be completed, or target multiple objects at one time.

Copy-Paste: create an object with the same attribute of the reference object, which can be decomposed into one Copy and several Insert primitive operations.

Mirror: create an object relative to a reference line or plane with the same attribute of the reference object, which can be decomposed into one Select and several Insert primitive operations.

Array: create several objects at a time, which can be decomposed into one or several Insert primitive operations.

Extend: change one vertex position of an object according to the attribute of the reference object, which can be decomposed into one or several Update primitive operations. For instance, extending a line to the center of a circle implies that one vertex of the line shall be changed to the position of the circle's center.

However, only primitive operations can be executed directly in AST framework, it is feasible that compound operations should be decomposed into primitive operations whose formats are defined in detail below:

Insert: Insert(Obj, "Obj_Type(Key₁(value₁), key₂(value₂), ..., key_n(value_n))", ReferenceObj, ReferenceType, OriginalOp);

Delete: Delete(Obj, ReferenceObj, OriginalOp);

Update: Update(Obj, "Key(old_value, new_value)", ReferenceObj, ReferenceType, OriginalOp);

Select: Select(Obj, OriginalOp);

Copy: Copy(Obj, OriginalOp).

Note that, Obj and ReferenceObj denote target object and reference object of the operation separately, Obj_Type denotes the type of the target object, Key denotes the attribute type of the object, such as Position, Radius, etc. ReferenceType denotes the constraint type, such as Parallel, Tangency, Endpoint-Intersect, etc.

4. SEMANTIC CONSISTENCY MAINTENANCE OF COMPOUND OPERATIONS

4.1 Add Original Op and Local-exe Op HB

An inner linear structure is used in AST strategy to maintain the Effective/Ineffective status of the object nodes, and every node is appended to a history buffer to contain operations that have been executed on this node. However, only the All-exe decomposed operations are stored in this list. The Local-exe decomposed operations and the original operations are not stored or maintained. Therefore, no information can be used to support the semantic consistency maintenance of compound operations.

Based on the AST framework, histories storing Original Ops and Local-exe Ops are introduced to store the relationships between the Original Ops and Decomposed ones at local sites (Fig. 4). The Local-exe Op HB and All-exe Op HB are linked with the object nodes which the Ops are manipulating while the Original Ops are stored in the linear structure with an additional state vector (or to say timestamp) to indicate the relationship between the Local-exe Ops and the Original Ops as well as that between the All-exe Ops and the Original Ops (See Fig. 4).

Figure 4. Adding Original Op HB and Local-exe Op HB in AST Frame

4.2 Introduce Bi-Vector to Transfer Relationship during Broadcasting

Section 4.1 introduces to use the Original Op and the Local_exe Op HBs to store the local relationship of the original operations and the decomposed operations. However, there is no strategy to transfer the relationship during the operation transmission process. In this paper Bi-Vector strategy is used to indicate it. That's to say, every operation is attached with two state vectors (See Fig. 5). One uses the traditional format $\langle S_{j1}, S_{j2}, \dots, S_{jN} \rangle$ with S_{ji} indicating how many operations released at site i have been executed at site j when the operation is released at site j , while another one uses the format $\langle \text{SiteID}, \text{SeqID} \rangle$ with SiteID indicating which site generates the operation and SeqID indicating the generation sequence number of the Original Op related with the Decomposed ones. For example, in Fig. 5, operations Select(4) $\langle 2, 1 \rangle \langle 1, 4 \rangle$, Insert(5,...,4) $\langle 2, 2 \rangle \langle 1, 4 \rangle$ and Insert(6,...,4) $\langle 2, 3 \rangle \langle 1, 4 \rangle$ are the decomposed Ops of the Original operation Mirror $\langle 1, 4 \rangle$ since they have the same state vector $\langle 1, 4 \rangle$.

Fig 5. Operation Sequence List in History Buffer

In general, the main idea of the Bi-Vector mechanism is as follows. Suppose a compound operation is released at certain local site, it is firstly decomposed into several primitive operations before execution, and the information of the original operation and its decomposed operations are stored in Original Op HB and Local_exe Op HB separately. It is the Bi-Vector strategy which appends two

timestamps to the original operations and the decomposed ones that their relationship can be preserved and thus broadcast to other sites. Once a remote site receives one decomposed operation, it is possible to reconstruct its original operation sequence by two additional state vectors, as shown in Fig. 5.

4.3 Modify the Local Operation Generation Process and the Remote Operation Receive Process

When one Op is released from the user interface, Generate process is used to transform the data format to primitive Ops that can be processed directly by AST strategy. If the Op is an Undo operation, it will be processed by Undo-Process which will be described in section 5. If the Op is a primitive operation, it will be executed immediately and appended to the All-exe Op HB of the target object node; If the Op is a Local-exe decomposed Op of a compound Op, it will be executed and appended to the Local-exe Op HB; If it is a All-exe decomposed Op of a compound Op, it will be executed and appended to the All-exe Op HB. Besides, if the Op is the first decomposed Op of a compound Op, the original Op will be appended to the Original Op HB. The *Generate* process is described in function 1¹.

Function 1: Generate O_u : O

1. If O_u is an Undo Op {
 2. Undo_Process(O_u); // call the Undo_Process that will be described in section 5.
 3. }
 4. Else If O_u is a primitive Op { // O_u is a primitive Op
 5. $O = \text{Pack}(O_u)$;
 6. Execute(O);
 7. Append($O.\text{Obj.All-exe Op HS}, O$); //Append to the All-exe Op HS
 8. Append($O.\text{Obj.Original Op HS}, O$); //Append to the Original Op HS
 9. }
 10. Else if O_u is a Local-exe Op { // O_u is Local-exe Op
 11. $O = \text{Pack}(O_u)$;
 12. Execute(O);
 13. Append($O.\text{Obj.Local-exe HS}, O$); //Append to the Local-exe HS
 14. }
 15. Else {
 16. $O = \text{Pack}(O_u)$;
 17. Execute(O);
 18. if O is the first All-exe operation of a complex operation
 19. Append($O.\text{Obj.Original Op HS}, O.\text{Original Op}$); //Append to the Original Op HS
 20. Append($O.\text{Obj.All-exe Op HS}, O$); //Append to the All-exe Op HS
 21. }
 22. Return O ;
-

After one operation is received by a remote site, *Receive* process is responsible for re-constructing original operation of the decomposed operation. In this process, *Receive* process first examines whether the operation is causally-ready. If it is not causally-ready, it will be queued. As for causally-ready operation, *Receive* process looks for the Original Op HB to examine whether the original Op is stored

¹Comment: All the functions and procedures in this paper are described with C pseudo code.

or not. If no such Op is found, the Original Op of this Op should be re-constructed and stored in the Original Op HB. The Receive process is described in Procedure 1.

Procedure 1. Receive O

1. If O is not causally ready
 2. Queue(O);
 3. else{ // Insert it into the Original Op HB according to the first timestamp of O
 4. Search for the Original Op HB to find whether there are operation O_o which satisfies that $O_o.Timestamp = O.Timestamp_2$
 5. If O_o is found
 6. Append(Original Op HB, O . Original Op);
 7. }
-

4.4 Modify the Execution Process of Remote Operations

After the reception of a remote operation O (Suppose that the target object of O is Obj , and the reference object of O is $referenceObj$), the “Retrace-Undo-VT-Redo-Retrace” procedure is described as follow.

- 1) Call *Receive* process to re-construct the Original Op of O ;
- 2) Call the *Retracing* process (procedure 2) to retrace to document status to the oldest status, in which O is generated (Retrace process).
- 3) Search the operation lists of all the object nodes to find whether there are operations O_c which satisfies $O \Rightarrow O_c$ (please consult to [13] to find the definition of “ $\Rightarrow$ ”). If no such operation exist, goto 4); otherwise, construct the Constraint Concurrent Tree (CCT_s) [13], and Undo all the operations in the tree from the bottom to up of the tree (Undo process).
- 4) If $O.referenceObj \neq \text{Null}$, check all the operations which take $referenceObj$ as the target object to find whether there are operations that are concurrent with O . If no such operations exist, goto 5); otherwise, suppose that operations list is $[O_{r1}, O_{r2}, \dots, O_{rm}]$ (the definition of $\xrightarrow{||^c}$, please consult to [13]) which satisfies that $O_r \xrightarrow{||^c} O$, O should execute Virtual Transformation (VT) [4] against $O_{r1}, O_{r2}, \dots, O_{rm}$ respectively to include the execution effect of these operations, i.e. $EO = VT(O, O_{r1})$, $EO = VT(EO, O_{r2})$, (VT process).
- 5) Execute EO (Do process).
- 6) All the operations in the CCT_s (except O) execute VT transformation against their parent node operation from top to down. And the transformed operations are re-executed (Redo process).
- 7) Retrace the document status to current timestamp (Retrace process).

Procedure 3 gives detailed description about the operation execution process of remote operations.

Procedure 2. Retracing(Doc_s, SV):

Retrace the document status Doc_s to that which is matched with timestamp SV

- 1: for each object node CN_i of linear structure of Doc_s {
 - 2: Set CN_i ineffective;
 - 3: Consider the Insert O_{ins} of CN_i
 - 4: If O_{ins} is timestamped by SVO_{ins} and $SVO_{ins} \leq SV$ {
 - 5: set CN_i effective
-

```

6.    }
7.    for each Delete  $O_{del}$  of  $CN_i$  {
8.        if  $O_{del}$  is timestamped by  $SVO_{del}$  and  $SVO_{del} \leq SV$  {
9.            set  $CN_i$  ineffective;
10.        }
11.    }

```

Procedure 3. Retrace-Undo-VT-Redo-Retrace (Doc_s, O):

Control the execution process of O on the document status of Doc_s

```

1:  Retracing( $Doc_s, SV$ );
2:   $CCT_s = \text{FindFollowing}(O, Doc_s)$ ;
3:  if  $CCT_s == \{O\}$  goto 7 ;
4:  Undo the operations in  $CCT_s$  from bottom to top of the tree except  $O$ ;
5:   $CCP_s = \text{FindPreceding}(O, Doc_s)$ ;
6:  for each  $O_p$  in  $CCP_s$  {
     $O = \text{VT}(O, O_p)$ ;
  }
7:  Execute the operation  $O$ ;
8:  Append  $O$  and its timestamp to corresponding target object node;
9:   $\text{VTF}(O, CCT_s)$ ;
10: Redo all the operations in  $CCT_s$  except  $O$ ;
11:  $SV[R] = SV[R] + 1$ ;
12: Retracing( $Doc_s, SV$ );

```

Function 2. FindFollowing(O, Doc_s): CCT_s

O : a concurrent operation

Doc_s : the document status which is represented with a linear structure containing the object nodes.

CCT_s : the constrain concurrent tree of O

```

1:   $flag = \text{false}$ ;
2:  For each object node  $o_n$  in  $Doc_s$  {
3:      For each operation  $O_p$  attached with node  $o_n$  {
4:          If  $O \xrightarrow{||^c} O_p$  {
5:               $CCT_s.\text{Append}(O_p)$ ;
6:               $flag = \text{True}$ ;
7:          }
8:      }
9:  }
10: If  $flag == \text{True}$  {
11:     For every child  $O_p$  of  $O$  {
12:          $CCT_s = CCT_s.\text{child}(O_p)$ ;
13:          $R = \text{FindFollowing}(O_p, Doc_s)$ ;
14:          $CCT_s.\text{append}(R)$ ;
15:     }

```

```

16:   }
17:   Else
18:       Return null;

```

Function 3. FindPreceding(O, Doc_s): CCP_s

O : a concurrent operation

Doc_s : the document status which is represented with a linear structure containing the object nodes.

CCP_s : operations list $[Op_1, Op_2, \dots, Op_n]$, with Op_i satisfies that $Op_i \xrightarrow{\parallel^c} O$

```

1:  $CCP_s = \text{null}$ ;
2: For each object node  $o_n$  in  $Doc_s$  {
3:   For each operation  $Op_p$  attached with node  $o_n$  {
4:     If  $O \xrightarrow{\parallel^c} Op_p$  {
5:        $CCP_s.Add(Op_p)$ ;
6:     }
7:   }

```

4.5 Case Analysis

In this section, an integrated example is detailed to show the work flow of the whole strategy. The initial document status and the execution sequence of different operations are shown in Fig. 6. Please consult to [13] for the definitions of Insert, Delete and Update.

Figure 6. Initial Document Status and Execution Order at Different Sites

The definition of O_1 to O_8 is as follows.

$O_1 = \text{Insert}(\text{Obj}_1, \text{"Circle(Position(180, 180, 0), 20)"}, \text{null}, \text{null}, \text{null})$; // O_1 wants to create a new circle object Obj_1

$O_2 = \text{Select}(\text{Obj}_2, 1)$; // O_2 is a Local-exe decomposed operation of Mirror operation

$O_3 = \text{Insert}(\text{Obj}_2, \text{"Circle(Position(300, 180, 0), 20)"}, \text{Obj}_1, \text{"Mirror()"}, 1)$; // O_3 is another decomposed operation of Mirror operation, which aims to create a circle with the same attribute of Obj_2

$O_4 = \text{Update}(\text{Obj}_1, \text{"Radius(20, 25)"}, \text{null}, \text{null}, \text{null})$; // O_4 intends to Update the radius of Obj_1 from 20 to 25

$O_5 = \text{Insert}(\text{Obj}_3, \text{"Line(Position(210, 120, 0), (270, 90, 0))"}, \text{null}, \text{null}, \text{null})$; // O_5 wants to create a new Line object Obj_5

$O_6 = \text{Copy}(\text{Obj}_4, 5)$; // O_6 is a Local-exe decomposed operation of Copy-Paste operation

$O_7 = \text{Insert}(\text{Obj}_4, \text{"Line(Position(270, 150, 0), (330, 120, 0))"}, \text{Obj}_3, \text{"Copy-Paste()"}, 5)$; // O_7 is another decomposed operation of Copy-Paste operation, which aims to create a line with the same attribute of Obj_3

$O_8 = \text{Update}(\text{Obj}_4, \text{"Position(p}_1, \text{p}_2)"$, $\text{Obj}_3, \text{"Extend(1,2)"}$, null); // O_8 wants to modify the first vertex of Obj_4 from p_1 to p_2 where p_2 is the second vertex of Obj_3

The document statuses of different sites are shown in table 2. Limited by the length of the paper, site 1 is taken as the example to describe the change process of the document status of AST by using "Retrace-Undo-VT-Redo-Retrace" process.

→ Site 1 : $\text{Docs}_0^1 = \{ \}$;

(1) Execute O_1 :

(2) Execute O_5 :

(3) Execute O_3 :

Since $O_1 \xrightarrow{\text{llc}} O_3$, $O_3' = \text{VT}(O_3, O_1) = \text{Insert}(\text{Obj}_2, \text{"Circle(Position(300, 180, 0), 20)"}$, $\text{Obj}_1, \text{"Mirror()"}, 1)$;

After the execution of O_3' , the document status is re-traced to current timestmap, and we get:

(4) Execute O_4 :

And under this condition, O_4 is executed. Then the document status is re-traced to current timestmap, and we get:

(5) Execute O_7 :

Since $O_5 \xrightarrow{\text{llc}} O_7$, $O_7' = \text{VT}(O_7, O_5) = (\text{Obj}_4, \text{"Line(Position(270, 150, 0), (330, 120, 0))"}$, $\text{Obj}_3,$

“Copy-Paste()”, 5);

After the execution of O_7 , the document status is re-retraced to current timestmap, and we get:

(6) Execute O_8 :

And under this condition, O_8 is executed. Then the document status is re-retraced to current timestamp, and we get:

Table 2. Different Document Status of Different Sites

Site1	Site2	Site3
Doc(1,0)= 	Doc(2,0)= 	Doc(3,0)=
Doc(1,1)= 	Doc(2,1)= 	Doc(3,5)=
Doc(1,5)= 	Doc(2,3)= 	Doc(3,1)=
Doc(1,3)= 	Doc(2,5)= 	Doc(3,7)=
Doc(1,4)= 	Doc(2,7)= 	Doc(3,3)=

From Table 2, we can see that after the execution of all the operations in all sites, the intentions of all the single operations and the compound operations are maintained, and the document statuses of all the sites are identical.

5. SEMANTIC CONSISTENCY MAINTENANCE OF COMPOUND UNDO/REDO OPERATIONS

The intention of the compound operations is maintained by using “Retrace-Undo-VT-Redo-Retrace” strategy. If later Undo/Redo operations of the compound operations occur, the following problem must be solved: The constraint following operations should exclude the execution effect of the undone operation. Besides, the intention of the Undo operation of the multiple object operation is to undo the execution effect of all the objects, but not only the last one. Therefore, some strategies must be taken to record the relationship between the decomposed operations and the original one, and the Undo operation should be defined on the original operation, but not the decomposed ones.

The following devises strategy to maintain the semantic consistency of compound Undo/Redo operations and describes the conflict resolution process of Undo/Redo conflicts.

5.1 Undo Conflict

During the execution process, intention conflicts may occur frequently when Undo operations are concurrent with other operations coming from remote sites. The definition of conflict including Undo operation is described in Definition 1.

Definition1Conflict ‘ $\otimes$ ’

Two operations O_a and O_b are conflict with each other, expressed as $O_a \otimes O_b$, if and only if: (1) they are concurrent; (2) they are targeting at the same object, i.e. $O_a.Obj = O_b.Obj$; (3) O_a has completely the same semantic meaning with O_b or O_a has completely controversy semantic meaning with O_b ;

That’s to say: 1) If two operations tend to indicate whether the target object exists or not, they conflict with each other if they are targeting the same objects with the following intentions: (1) Both

operations intend to make the same object invisible or invisible; (2) One operation intends to make the object visible while another one wants to make it invisible; 2) If two operations tend to change the attribute value of the target object, they conflict with each other if they change the same key with different values.

5.2 Control flow of the Compound Undo/Redo Operations

Figure 7. Undo Support of Compound Undo Operations

As has been shown in Fig.7, after the compound Undo operation is released from the user interface, it is treated as a special Undo operation. *Undo Capture procedure* is responsible for capturing the Undo operation and translating it into the IDs of the original Ops from the Original Op HB with the reference to the chosen undo mode. *Undo Decompose* process decomposes the IDs of the Original Ops ($Id(OriginalOp)$) into those of the Decomposed Ops ($Id(Op)^*$) by consulting both the Original Op HB and the All-exe Op HB. *Undo Pre-condition Check* examines whether the prerequisites of all the decomposed Undo operations are satisfied or not. If not, some visual clues are feed back to the user interface and the invalid operation will be blocked to be evoked. The decomposed Undo operations, whose Original Op has passed the pre-condition check process, are broadcasted to remote sites and executed in local sites simultaneously. The local process of Undo operations is as follows:

1) *Undo Cascade* process checks the constraint concurrent tree (CCT_s) of the Undo's manipulating operation so as to find out all the operations which need to be undone and to execute virtual transformation (VT) against the Undo operation so that the undo effect can be reflected in these operations. As for more detailed description about VT function, please consult to [6];

2) *Undo Generate* process transforms Undo operation into its corresponding Insert, Delete or Update operations. That is, Undo (Insert) operation is transformed into Delete operation; Undo (Delete)

operation is transformed into Insert operation; Undo (Update) operation is transformed into Update operation;

3) *Undo Execute* process is responsible for the execution of the undo operations. Undo Execute process controls the execution process recursively from the lowest operation nodes of CCT_s . After the execution of the CCT_s , in order to exclude the effect of the target operation of the undo operation from the constraint following operations, VT function is called to transform the CCT_s to get the new execution forms. Then, the transformed operations in the CCT_s are redone.

When remote sites receive the decomposed Undo operation, it will first be re-constructed to the Original Undo operation and attached to the Original Op HB. Then *Conflict Detection* process checks whether semantic conflicts occur. If conflicts exist, modified multi-version strategy is adopted to solve them. *Conflict Resolution* process will be discussed in detail in the following section. After *Conflict Resolution*, *Undo Cascade*, *Undo Generate* and *Undo Execute* process are evoked one by one to complete the execution process. The *Undo_Decompose* and *Undo_Cascade* algorithms are described in Functions 4 and 5.

Function 4. $Undo_DecomposeId(OriginalOp), Id_1(O), Id_2(O), \dots$

Decompose Original Undo with $Id(OriginalOp)$ to decomposed ones with $Id_1(O), Id_2(O), \dots$

1. for each object node CN_i of linear structure of Doc_s {
 2. for each operation O_j of CN_i {
 3. if $O_j.Timestamp_2 = Id(OriginalOp).Timestamp$
 4. $Id(O).Append(O_j)$;
 5. }
 6. }
 7. Return $Id(O)$;
-

Function 5. $Undo_Cascade(Id(O), Doc_s): CCT_s$

Check all the nodes in $Docs$ to construct the Constraint Concurrent Tree CCT_s

1. $flag=false$;
 2. for each object node CN_i of linear structure of Doc_s {
 3. for each operation O_j of CN_i {
 4. if $O \xrightarrow{||^c} O_j$ {
 5. $CCT_s.Append(O_j)$;
 6. $CCT_s = CCT_s.O_j$;
 7. $R = Undo_Cascade(O_j, Doc_s)$;
 8. $CCT_s.Append(R)$;
 9. $flag=true$;
 10. }
 11. }
 12. If $flag==false$
 13. Return null;
-

5.3 Undo Conflict Resolution

In the collaborative environment with Undo operations, the conflict may occur not only between

Update operations, but also between Undo/Redo operations and Insert/Delete operations. Section 5.1 describes the definition of the Undo conflict and the following will give the version decomposition function based on multi-version strategy.

Function 6. Version_Decomposition(Ver, O): new_Ver

Generate new version new_Ver based on Ver

Ver : a group of operations attached on one object node;

O : an operation targeting to be executed on the same object node;

new_Ver : the newly generated versions;

1. If O is not pre-condition ready { *//If the Pre-condition is not satisfied, O cannot be executed at this version*
 2. $new_Ver = Ver$;
 3. return;
 4. }
 5. $R = \emptyset$;
 6. for every operation O_i in Ver { *//Search all the operation which are conflict with O*
 7. if $O_i \otimes O$
 8. $R = R \cup \{O_i\}$;
 9. }
 10. If $R = \emptyset$ { *//If conflict operations don't exist*
 11. $new_Ver = Ver \cup \{O\}$;
 12. return new_Ver ;
 13. } else { *//If conflict operations exist, Ver needs to be split*
 14. $new_Ver[0] = (Ver - R) \cup \{O\}$;
 15. $new_Ver[1] = Ver$;
 16. return new_Ver ;
 17. }
-

As for the semantic conflict, a new Multi-Version technology [7] is given here. In the MV strategy, when a conflict occurs between two operation groups, the document version is split into two ones with each comprising one of the conflict operation groups. As for the newly-arrived operation, Pre-condition check is performed to check which version it can be executed on. Function 6 describes the version split process. Since version split process may produce duplicate versions, *Version_Check* is called to delete the redundant versions. As for the Delete and Update operations attached on the same object node, we oblige that Update operation should be executed before Delete operation. So that *Version_Exchange* process re-arranges the sequence of the two operation types.

Procedure 4. Version_Check

Check the new_Vers to decide whether there are duplicate versions and delete the duplicate ones;

1. for every V_i in new_Ver {
 2. for every V_j in new_Ver {
 3. if $i \neq j$ and $V_i == V_j$ {
 4. delete V_j ;
 5. }
 6. }
-

Procedure 3. Version_Exchange

Check every version in new_Vers to re-arrange the order of the concurrent Delete and Update operations;

1. for every V_i in new_Vers { *//as for every version*
 2. for every O_k in V_i { *//as for every operation of every version*
 3. If O_k is a Delete or Undo (Insert) or Redo (Delete) operation { *// O_k is an operation which has equal semantic meaning with the Delete operation*
 4. for every O_l in V_i {
 5. if $O_l // O_k$ and O_l is an Update or Undo (Update) or Redo (Update) operation {
 6. Re-arrange O_l before O_k ;
 7. }
 8. }
 9. }
 10. }
 11. }
-

5.4 Case Analysis

Figure 8. Example 1, Undo Support of Multiple Object Operations

In this section, two examples are given to demonstrate the work flow of the Undo consistency maintenance strategy. Example 1 analyses the Undo support of multiple object operation and example 2 analyses the Undo support of reference operation. The work flow of example 1 is shown in Fig. 8.

In example 1, O_1 and O_2 are the decomposed operation of Mirror<1, 1> and O_3 and O_4 are the decomposed operation of Undo<1, 2>. The execution process of site 1 and 2 is as follow.

Site 1:

1) When Mirror<1, 1> operation is released from the user interface, it is decomposed into two operations: O_1 and O_2 . After the local execution of these two operations, they are broadcasted to site 2;

2) When Undo(Mirror)<1, 2> operation is released from the user interface, it's decomposed into two operations: O_3 and O_4 . Since the pre-condition of the two operations are satisfied, and there are no constraint following operations of them, they are executed immediately;

3) Site 1 receives the remote operation O_5 =Delete(Object3)<1, 1> <2, 1> from site 2. Since $O_3 \otimes O_5$, the document status is split into two versions: new_Ver₁ = { O_1, O_2, O_3, O_4 }, new_Ver₂ = { O_1, O_2, O_4, O_5 };

4) Site 1 receives the remote operation $O_6 = \text{Undo}(O_5)$ from site 2. Since the pre-condition of O_6 is only satisfied on version 2, it is executed only on version 2 and the new_Ver₂ = { O_1, O_2, O_4, O_5, O_6 } is achieved.

the process at site 2 is omitted here due to this paper's length limitation, with the same result as at site 1.

In example 1, the document statuses of site 1 and site 2 are identical and the intention of the Undo operation is maintained successfully.

The operation execution process of example 2 is shown in Fig. 9. In example 2, O_1 and O_2 are decomposed operations of Array<1, 1>. Suppose that Obj₁ is the reference object of Array<1,1>, the execution process of site 1 and 2 are as follows.

Figure 9. Example 2, Undo Support for Reference Operation

Site 1:

1) When Array<1, 1> operation is released from the user interface, it is decomposed into two operations: O_1 and O_2 . After the local execution of these two operations, they are broadcasted to site 2;

2) Site 1 receives the remote operation $O_5 = \text{Update}(\text{Obj1}) <0, 1> <2, 1>$ from site 2. Since $O_5 \xrightarrow{\parallel c} O_1$ and $O_5 \xrightarrow{\parallel c} O_2$, O_1 and O_2 are undone and executed VT transformation against O_5 , which results in: $O_1' = \text{VT}(O_1, O_5)$, $O_2' = \text{VT}(O_2, O_5)$. Execute O_5 , O_1' and O_2' respectively;

3) When Undo(Array)<1, 2> operation is released from the user interface, it is decomposed into two operations: O_3 and O_4 . Since the pre-condition of these two operations are satisfied, O_3 and O_4 are executed respectively.

4) When site 1 receives the $O_6 = \text{Undo}(O_5) <0, 2> <2, 2>$ operation, it first searches the Original Op HB to find whether there is an operation with the timestamp of <2, 2>. Thus, it appends an original operation of Undo(Array)<1, 2> to the Original Op HB. Because $O_1 \xrightarrow{\parallel c} O_5$ and $O_2 \xrightarrow{\parallel c} O_5$, Undo Cascade process is called to construct the constraint concurrent tree $\text{CCT}_s = \{O, O_1, O_2\}$. Execute the reverse operation $\overline{O_5}$ of O_5 . Undo O_1 and O_2 operations. Execute VT transformation of O_1 and O_2 against $\overline{O_5}$, and we get: $O_1' = \text{VT}(O_1, \overline{O_5})$, $O_2' = \text{VT}(O_2, \overline{O_5})$. Redo O_1' and O_2' ;

Equally, the process at site 2 is omitted in this paper, with the same result as at site 1.

In example 2, the document statuses of site 1 and site 2 are identical and the intention of the Undo operation and Update operation are maintained successfully.

6. DISCUSSIONS OF THE STRATEGY'S VALIDITY

The consistency maintenance algorithm adopted in this paper is correct or not is dependent on whether it satisfies the CCI model. Hereinto, the CCI model ensures the validity of the AST, which implies that the final document statuses of all sites are consistent and all users' intentions are maintained, if all operations at all sites are released and executed on the basis of the AST algorithm. The "Retrace-Undo-VT-Redo-Retrace" strategy's correctness is based on the correctness of the AST method, and AST guarantees the correct execution for causal-ready operations only and the consistent results for concurrent operations (not constraint concurrent operations) with different execution orders.

There is a special relationship called constraint concurrent ($\parallel^c$) except causal ($\rightarrow$) and concurrent ($\parallel$) relationships between operations in "Retrace-Undo-VT-Redo-Retrace" strategy, and the causal and concurrent operations can be solved by AST, so it is doable that the strategy's validity can be proved by assuring constraint concurrent operations executed in different orders at different sites can achieve the consistent document statuses.

Suppose n constraint consistent operations $O_1, O_2, \dots, O_n$, meeting the condition of $O_1 \parallel^c O_2 \parallel^c O_3 \parallel^c \dots \parallel^c O_n$, with $n!$ different execution orders. In this case, the prove shall be verified that whichever orders those n operations are executed, the final execution result is $[O_1, O_2', O_3', \dots, O_n']$, wherein $O_2' = VT(O_2, O_1)$, $O_3' = VT(O_2', O_2')$, $\dots$, $O_n' = VT(O_n', O_{n-1}')$.

- 1) If $n=1$: all execution order is the same, i.e., $[O_1]$;
- 2) If $n=2$: two cases are considered into account.
 - (1) If O_1 is reached before O_2 at certain site, the execution procedure is as follows:
 - i. execute O_1 , and $HB=[O_1]$;
 - ii. $O_2' = VT(O_2, O_1)$;
 - iii. execute O_2' , and $HB=[O_1, O_2']$;
 - (2) If O_2 is reached before O_1 at certain site, the execution procedure is as follows:
 - i. execute O_2 , and $HB=[O_2]$;
 - ii. $Undo(O_2)$, and $HB=[]$;
 - iii. execute O_1 , and $HB=[O_1]$;
 - iv. $O_2' = VT(O_2, O_1)$;
 - v. execute O_2' , and $HB=[O_1, O_2']$;

According to case (1) and (2), despite that O_1 and O_2 are executed in different orders, the final result is both $[O_1, O_2']$.

- 3) If $n=k$:

Suppose the conclusion is tenable when $n=k-1$, and the proof can be made that for a set of

operation sequence $[O_1, O_2, \dots, O_{i-1}, O_i, O_{i+1}, \dots, O_k]$ with the corresponding relation: $O_1 \parallel^c$

$O_2 \parallel^c \dots \parallel^c O_k$ (See Fig. 10), out-of-order executed operations between the subsequences $[O_1, \dots, O_{i-1}]$ and $[O_{i+1}, \dots, O_k]$ emerge the same result: $[O_1, O_2', \dots, O_{i-1}']$ and $[O_{i+1}, O_{i+2}', \dots, O_k']$, wherein $O_2' = VT(O_2, O_1)$, $\dots$, $O_{i-1}' = VT(O_{i-1}, O_{i-2}')$, $O_{i+2}' = VT(O_{i+2}, O_{i+1}')$, $\dots$, $O_k' = VT(O_k, O_{k-1}')$. However, it is difficult to predicate the operations' constraint concurrent relations so as to judge their execution order due to the lack of O_i . Three situations shall be discussed as follows:

Figure 10. the Constraint Concurrent Dependency Relation Graph of $[O_1, \dots, O_k]$

Figure 11. O_i is reached before $[O_1, O_2, \dots, O_{i-1}, O_{i+1}, \dots, O_k]$

- (1) If O_i is reached before $[O_1, O_2, \dots, O_{i-1}, O_{i+1}, \dots, O_k]$ (See Fig. 11) at certain site, the execution procedure is as follows:
 - i. Execute O_i , and $HB = [O_i]$;
 - ii. Whichever order the subsequences $[O_1, \dots, O_{i-2}]$ and $[O_{i+1}, \dots, O_k]$ are reached, the execution result is the same: $[O_1, \dots, O_{i-2}']$ and $[O_{i+1}', \dots, O_k']$, wherein $O_2' = VT(O_2, O_1')$, ..., $O_{i-2}' = VT(O_{i-2}, O_{i-3}')$; $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$, and $HB = [O_i] + [O_1, \dots, O_{i-2}'] + [O_{i+1}', \dots, O_k']$;
 - iii. When O_{i-1} is reached, $Undo(O_k', \dots, O_{i+1}', O_i)$ firstly, and $HB = [O_1, \dots, O_{i-2}']$;
 - iv. $O_{i-1}' = VT(O_{i-1}, O_{i-2}')$;
 - v. Execute O_{i-1}' , and $HB = [O_1, \dots, O_{i-2}', O_{i-1}']$;
 - vi. $O_i' = VT(O_i, O_{i-1}')$;
 - vii. Execute O_i' , and $HB = [O_1, \dots, O_{i-2}', O_{i-1}', O_i']$;
 - viii. $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$;
 - ix. Execute $O_{i+1}', \dots, O_k'$, and $HB = [O_1, \dots, O_i', O_{i+1}', \dots, O_k']$.
- (2) If O_i is reached between $[O_1, O_2, \dots, O_{i-1}, O_{i+1}, \dots, O_k]$ (See Fig. 12) at certain site, the execution procedure is as follows:
 - i. Whether operations in the subsequence $[O_1, \dots, O_{i-1}]$ are executed in order or not after reaching O_i , the same result can be achieved: $[O_1, O_2', \dots, O_{i-1}', O_i']$ and $O_2' = VT(O_2, O_1')$, ..., $O_i' = VT(O_i, O_{i-1}')$ by a series of procedure of Undo, VT and Redo.
 - ii. For operations in the subsequence $[O_{i+1}, \dots, O_k]$ after O_i is reached, the result is the same: $[O_i, O_{i+1}', \dots, O_k']$ and $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$ when they are executed in different orders by a series of procedure of Undo, VT and Redo. If O_{i-1} has not been reached yet before executing O_{i+1} , execute O_i and then goto iii; else $O_i' = VT(O_i, O_{i-1}')$, the document state is: $[O_1, \dots, O_i', O_{i+1}', \dots, O_k']$ and exit the procedure;
 - iii. If O_i is executed after reaching O_{i-1} , a series of Undo shall be executed to undo all the follow operations that are constraint concurrent with O_i and O_i itself, and then $O_i' = VT(O_i, O_{i-1}')$, $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$. Finally, execute $O_i', O_{i+1}', \dots, O_k'$ again, and $HB = [O_1, \dots, O_i', O_{i+1}', \dots, O_k']$.

Figure 12. O_i is reached between $[O_1, O_2, \dots, O_k]$

Figure 13. O_i is reached after $[O_1, O_2, \dots, O_k]$

(3) If O_i is reached after $[O_1, O_2, \dots, O_k]$ (See Fig. 13) at certain site, the execution procedure is as follows:

- i. Whether operations in the subsequence $[O_1, \dots, O_{i-1}]$ are executed in order or not, the same result can be achieved: $[O_1, O_2', \dots, O_{i-1}', O_i']$ and $O_2' = VT(O_2, O_1')$, ..., $O_i' = VT(O_i, O_{i-1}')$ by a series of procedure of Undo, VT and Redo;
- ii. For operations in the subsequence $[O_{i+1}, \dots, O_k]$ after O_i is reached, the result is the same: $[O_i, O_{i+1}', \dots, O_k']$ and $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$ when they are executed in different orders by a series of procedure of Undo, VT and Redo; $HB = [O_1, O_2', \dots, O_{i-1}'] + [O_{i+1}', \dots, O_k']$;
- iii. After O_i is reached, undo $O_k', \dots, O_{i+1}'$, and $HB = [O_1, O_2', \dots, O_{i-1}']$;
- iv. $O_i' = VT(O_i, O_{i-1}')$, $O_{i+1}' = VT(O_{i+1}, O_i')$, ..., $O_k' = VT(O_k, O_{k-1}')$;
- v. Execute $O_i, O_{i+1}', \dots, O_k'$, and $HB = [O_1, O_2', \dots, O_i', O_{i+1}', \dots, O_k']$

According to analysis of three situations above, the conclusion is still holds water when $n=k$.

As for the correctness of Undo/Redo operations, it is based on the correctness of the Do operations, since it treats the Undo/Redo operations as the reverse or the same operations of the original operations. The correctness of the Undo/Redo solutions is referred to [5, 32]. Due to the correctness of the Do process, the Undo/Redo is also correct.

7. CONCLUSION AND FUTURE WORK

In this paper, based on the AST method and Multi-version strategy, novel approaches are proposed to resolve the semantic consistency maintenance problem of compound operations in collaborative systems. This paper's main contribution includes that we append two additional history buffers (Original Op HB and Local-exe Op HB) for maintaining the relationships between original operations and decomposed ones, propose the "Retrace-Undo-VT-Redo-Retrace" strategy to realize the consistency maintenance of compound operations, and devise the Undo/Redo conflict strategy as the resolution method for compound Undo/Redo operations. In addition, the adopted algorithms' correctness and effectiveness in this paper are verified with case analyses and proof discussion.

However, there is an issue that has to be taken into consideration, i.e., the rendering pattern of multiple versions on the user interface. As depicted in this paper, multiple versions are generated by version splitting strategy and can be reserved at all sites. Users can select the highest-score version with a score mechanism which marks each version a different score according to their specified rule. Once a version is elected, the necessary information of the version shall be broadcast to all sites and will be displayed at all users' interface. Due to the field is not the focus research in this paper, explicit details will not be given here.

The future work includes the implementation of the "Retrace-Undo-VT-Redo-Retrace" strategy on the CoAutoCAD prototype as well as the solution of semantic Redo/Undo in document structure whose objects have the constraint relationships.

ACKNOWLEDGMENTS

The work is supported by the National Science Foundation of China (NSFC) under Grant No. 61202376, Shanghai Natural Science Foundation under Grant No. 15ZR1429100, Innovation Program of Shanghai Municipal Education Commission under Grant No. 13YZ075, Shanghai Key Science and Technology Project in Information Technology Field under Grant No. 14511107902, Shanghai Leading Academic Discipline Project under Grant No. XTKX2012, and Shanghai Engineering Research Center Project under Grant No. GCZX14014 and C14001.

References

- [1] Xia H, Lu T, Shao B, et al. A partial replication approach for anywhere anytime mobile commenting[C]. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM, 2014: 530-541.
- [2] Xu Y, Sun C, Li M. Achieving convergence in operational transformation: conditions, mechanisms and systems[C]. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM, 2014: 505-518.
- [3] N. Gu, J. Yang and Q. Zhang. Consistency Maintenance Based on Address Space Transformation Technique in Group Editor [J]. Chinese Journal of Computers, 30(5), 2007: 763-774.
- [4] Haifeng Shen, Chengzheng Sun. Achieving Data Consistency by Contextualization in Web-Based Collaborative Applications. ACM Transactions on Internet Technology, Vol. 10(4), 2011: Article No. 13.
- [5] Bin Shao, Du Li and Ning Gu. An algorithm for selective undo of any operation in collaborative applications. In: Proceedings of ACM SIGGROUP Conference on Supporting Group Work, 2010: 131-140.
- [6] SUN C. Undo as concurrent inverse in group editors. ACM Transactions on Computer-Human Interaction, 9(4), 2002: 309-361.
- [7] Khazalah F, Malik Z, Rezgui A. Automated conflict resolution in collaborative data sharing systems using community feedbacks[J]. Information Sciences, 2015, 298: 407-424.
- [8] Agustina, Chengzheng Sun. Dependency-conflict detection in real-time collaborative 3D design systems. In: Proceedings of the 2013 conference on Computer supported cooperative work, 2013: 715-728.
- [9] CoWord. <http://reduce.qpsf.edu.au/coword>.
- [10] CoPowerPoint. <http://reduce.qpsf.edu.au/copowerpoint>.
- [11] Liping Gao, Bin Shao, Lin Zhu, Tun Lu and Ning Gu. Maintaining Time and Space Consistencies in Hybrid Engineering Environments: Framework and Algorithms. Computers in Industry, Vol. 59(9), 2008: 894-904.
- [12] Agustina, Fei Liu, Steven Xia, Haifeng Shen, Chengzheng Sun. CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools. In: Proceedings of the 2008 ACM conference on Computer supported cooperative work, 2008: 5-8.
- [13] Liping Gao, Tun Lu and Ning Gu. CLAF: Solving intention violation of step-wise operations in CAD groupware. Advanced Engineering Informatics, 24(2), 2010: 121-137.
- [14] Hansu Gu, Mike Gartrell, Liang Zhang, Qin Lv and Dirk Grunwald. AnchorMF: towards effective event context identification. In: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. ACM, 2013: 629-638.
- [15] Claudia-Lavinia, Ignat and Noira C. Norrie. Grouping in Collaborative Graphical Editors. In: Proceedings of the 2004 ACM conference on Computer supported cooperative work, 2004: 447-456.
- [16] D. Li, J. Patrao. Demonstrational customization of a shared whiteboard to support user-defined semantic relationships among objects. In: Proceedings of the International ACM SIGGROUP

- Conference on Supporting Group Work, 2001: 97-106.
- [17] K Lin, D Chen, G Dromey, C Sun. Maintaining constraints expressed as formulas in collaborative systems. In: Proceedings of the International Conference on. IEEE, 2007: 318-327.
- [18] S. Hala, M. Pascal, O. Gérald. Semantic consistency for collaborative systems. In: Proceedings of the 5th International Workshop on Collaborative Editing Systems, 2003.
- [19] X.Wang, J. Bu, C. Chen. Semantic preservation in real-time collaborative graphics designing systems. In: The 4th International Workshop on Collaborative Editing Systems, 2002.
- [20] K. Lin, D. Chen, C. Sun, G. Dromey. Maintaining constraints in collaborative graphic systems: the CoGSE approach. In: Proceedings of the 9th European Conference on Computer-Supported Cooperative Work, 2005: 185-120.
- [21] M. Tang, S. Chou, J. Dong. Conflicts classification and solving for collaborative feature modeling. *Advanced Engineering Informatics*, Vol. 21(2), 2007: 211-219.
- [22] Agustina, Chengzheng Sun, Dong Xu. Operational transformation for dependency conflict resolution in real-time collaborative 3D design systems. In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, 2012: 1401-1410.
- [23] Xia X, Wang X, Li J and Zhou X. Multi-objective mobile app recommendation: A system-level collaboration approach[J]. *Computers and Electrical Engineering*, 2014, 40(1): 203-215.
- [24] S.J.K. jagadeesh Kumar and R.Saraswathi. Consistency Management Strategies in Data Replication Using Ant Colony Optimization Method. *Journal of Global Research in Computer Science*. Vol. 4(4), 2013: 175-181.
- [25] S.Thulasi, P.SaiPrathap, N. Mohan Krishna, R.Sandeep Kumar. Consistency Maintenance in Fractious P2P System. *International Journal of P2P Network Trends and Technology*, Vol. 3(2), 2013: 80-86.
- [26] Bin Shao, Du Li, Tun Lu and Ning Gu. An operational transformation based synchronization protocol for web 2.0 applications. In: Proceedings of the ACM conference on Computer supported cooperative work, 2011: 563-572.
- [27] Yuan Cheng, Fazhi He, Xiantao Cai, Dejun Zhang. A group Undo/Redo method in 3D collaborative modeling systems with performance evaluation. *Journal of Network and Computer Applications*, 36(6), 2013: 1512-1522.
- [28] Cai H, Ji X, Bu F. Research of consistency maintenance mechanism in real-time collaborative multi-view business modeling[J]. *Journal of Shanghai Jiaotong University (Science)*, 2015, 20: 86-92.
- [29] Agustina, Chengzheng Sun. Xpointer: an x-ray telepointer for relaxed-space-time wysiwiw and unconstrained collaborative 3d design systems. In: Proceedings of the conference on Computer supported cooperative work, 2013: 729-740.
- [30] Hongfei Fan, Chengzheng Sun. Supporting semantic conflict prevention in real-time collaborative programming environments. *Applied Computing Review*. Vol. 12(2), 2012: 39-52.
- [31] <http://cooffice.ntu.edu.sg/coflash/action.html>
- [32] Cheng Y, He F, Xu B, Han S, Cai X and Chen Y. A multi-user selective undo/redo approach for collaborative CAD systems[J]. *Journal of Computational Design and Engineering*, 2014, 1(2): 103-115.

Authors

Liping Gao(1980-) graduated from Fudan University, China with a PhD in Computer Science. She received her BSc and master degree in Computer Science from Shandong Normal University, China. She is doing her research work in University of Shanghai for Science and Technology as an assistant professor. Her current research interests include CSCW, heterogeneous collaboration, consistency maintenance and collaborative engineering.

Fangyu Yu(1989-) is a postgraduate student in University of Shanghai for Science and Technology.

She obtained her BSc degree in Electronic Information Engineering from Henan University of Science and Technology, China. Her current research interests include CSCW, collaborative design and collaborative computing.

Lily Gao(1982-) graduated from Shandong Normal University, China with a master degree in Computer Science in 2008. She received her BSc in Computer Science from Shandong Normal University in 2005. She is doing her research work in Weifang University as a lecturer. Her current research interests include Cultural Algorithm and CSCW.

Naixue Xiong(1978-) is a Professor at School of Computer Science, Colorado Technical University, Colorado Spring, CO, USA and University of Shanghai for Science and Technology, China. He received his both PhD degrees in Wuhan University and Japan Advanced Institute of Science and Technology, respectively. His research interests include Cloud Computing, Security and Dependability, Parallel and Distributed Computing and Optimization Theory.

Guisong Yang(1982-) received his BSc degree from Zhengzhou University, China, and M.S. and Ph.D. degrees from Tongji University, Shanghai, China, respectively. He is an Assistant Professor with the Computer Science and Engineering Division, University of Shanghai for Science and Technology, China. His research interests include wireless sensor networks and pervasive computing, distributed and embedded systems, wireless and mobile systems.