Consistency Maintenance of Group/Ungroup Operations in Object-based Collaborative Graphical Editing Systems

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

Abstract

In real-time collaborative graphical editing systems, Object-based Group/Ungroup operations are frequently accessible and practically useful. However, the existing research on these operations of the graphical editing is rare and defective. In this paper, based on Multi-Version strategy and Address Space Transformation method, a new MVSDR algorithm, which is not only applied to simple operations (such as Create, Delete, ChangeATT, etc.), but also suitable for Group/Ungroup ones, is proposed to solve the consistency maintenance problem. The proposed algorithm abandons previous attempts to divide conflict operations into Real-Conflict operations and Resolvable-Conflict ones and to deal with them separately, thus making the algorithm more simple and effective. In addition, an example analysis is also given in this paper to prove the algorithm's correctness and effectiveness.

Keywords: *Real-time Collaboration, Graphical Editing System, Object-based, Group/Ungroup, Multi-Version Strategy, Address Space Transformation, Consistency Maintenance.*

1. Introduction

Real-time collaborative graphical editing system allows multiple users to view and edit the shared graphics at the same time from geographically different sites via network connections, which has higher editing efficiency and more conforms to the trend of the modern collaborative thoughts, compared with previous single-user editing system. Collaborative editing system can be classified into three types: Object-based, bitmap-based and hybrid [5]. Wherein, Object-based editing system is a special collaborative editing system, whose operation targets are objects such as points, lines, circles, triangles, etc., and each object has attributes such as color, coordinate, size, etc. Users can create, update and delete objects, and the attributes of objects can be also updated. Lots of research has been developed in this field, including Co-PowerPoint, Co-AutoCAD, Co-Visio, etc. However, existing studies are mostly aimed at simple operations such as Create, Delete, ChangeAtt, etc., by comparison, rarely at complex operations of Group/Ungroup, which are indispensable common operations in graphical editing system, thus making the research on this field of great significance.

For the sake of operations' consistence maintenance, existing collaborative graphical editing system usually adopts the following three strategies: Locking, Serialization and Operation Transformation (OT).

Locking [3] allows only one user to edit the shared documents at one time, and collaborative editing is allowed only if different users are locking and editing different objects. Locking is divided into Pessimistic Locking and Optimistic Locking. Wherein, Pessimistic Locking causes the operation's delay when obtaining the Lock, while Optimistic Locking avoids the delay, but the system is not clear what to do when the Lock is denied. In addition, the target objects by users must be stored in their initial states.

Serialization guarantees that the effects of all concurrent operations are just like their execution orders are the same at all sites. When there are conflicts between concurrent operations, the last operation is allowed to be executed only. Wherein, the applications in [1] [2] [3] belong to this method. The main problem with Serialization is the operational delay, thus making the response time too slow and reaching the requirement of real-time collaborative editing too difficult.

OT strategy transforms the to-be-executed operation against all executed concurrent operations before it being executed. OT ensures the operation to be executed correctly and the convergence and intention-preservation to be satisfied successfully. Some celebrated studies by this method include GOT/GOTO [6] and COT [7] [11] algorithm. The main limitation of OT is that the complex relationship between operations has to be considered, especially while dealing with Group/Ungroup, the transformation method will become completely complicated.

This paper adopts a novel approach named AST (Address Space Transformation) [8], which utilizes Mark-Retrace strategy. AST retraces the document state to that when the operation is generated, and retraces the document back to the current state after executing the operation. Without considering the complex relationship between operations, this method thus has higher algorithm efficiency and is also suitable for Group/Ungroup operations.

The following paper is organized as follows. In section 2, some previous works on graphical editing and the AST method are introduced briefly. An object-based document model is established and definitions of related operations are presented in section 3. In section 4, a new MVSDR algorithm is proposed to resolve Group/Ungroup operation's consistency maintenance problem, and an example analysis is also given to verify the correctness and effectiveness of the algorithm, which is the focus of the paper. Finally, the paper is concluded with a brief summary of major contributions and future work.

2. Related Work

2.1. Preparatory Work

In graphical editing system, Group/Ungroup are all-important operations. In short, Group intends to combine a series of objects (including group) into a group, while Ungroup attempts to divide a group into a series of objects (including group). However, existing objet-based graphical editing is mainly focused on simple operations [5], the research on Group/Ungroup is quite scanty.

In previous works, Ignat [1] [2] [3] classifies conflict operations into two types: Real-Conflict operations and Resolvable-Conflict operations. Real-Conflict operations refer to the situation that executing one operation will make it impossible to execute the other operation or will mask the execution effect of the other one. In this case, a priority-based policy is adopted in which only the operation with the highest priority will be executed.

Resolvable-Conflict operations are the situation that conflict operations can be executed correctly by changing their execution orders. On this occasion, Ignat adopts Serialization method, which ensures all conflict operations being executed in the same order at each site. Obviously, the way of categorizing conflict operations is not only cumbersome but also may omit some cases. Besides, the adopted priority-based policy makes it impossible to preserve the effects of any other operation whose priority is not the highest, which goes against the collaborative idea of maintaining all users' intentions. Note that the priority is an artificial rule and it will become meaningless if any one user does not comply with the rule. Moreover, in order to ensure operational order consistency, Serialization has to undo and redo certain operations repeatedly, thus leading to low efficiency of the algorithm.

Related to these studies, Xia [4] proposes Multi-Version Single-Display (MVSD) strategy, i.e., all operations' effects are preserved, but one version is displayed on the user interface only. Meanwhile, combined with an Operation Transformation (OT) technique, the remote operation is guaranteed to be executed aright by transforming against all executed concurrent operations before its execution. However, although the MVSD strategy preserves all operations' intentions, users have to pause and then to choose which one version to be displayed once multiple versions are generated, which cause higher overhead and lower efficiency for executing operations. Besides, due to the addition of Group/Ungroup, the process of operational transformation will be surprisingly complicated, accordingly increasing greatly the complexity of the algorithm.

In this paper, the attempt of classifying operations into Real-Conflict and Resolvable-Conflict operations is discarded. Combined with the AST method, the Multi-Version strategy is adopted, which maintains all users' intentions without considering complex relationships between operations, thereby improving the algorithm's efficiency. Details will be described in the following parts.

2.2. Overview of the AST Method

Different with OT which transforms the operation itself, AST [8] retraces the document state to that at the time of the operation's generation so as to conceal the effects of executed concurrent operations without considering complex relations among operations. AST is originally applied in text document environments [10] supporting users to insert, delete and update characters, by contrast, little research has been done on graphical editing systems. Here, we continue to use Timestamp scheme and status Mark technique. Each operation is attached with its generating site's current state vector and then broadcast to other sites, and the target object or group of every operation is added with an Effective/Ineffective Mark which indicates whether it is visible or not on the user interface. Each object or group may have several operations targeting itself and each operation may target several objects or groups equally. As shown in Fig.1, given three operations O_1, O_2 and O_3, generated at sites 1, 2 and 3 separately. Wherein, O_1=Group([G_1, Obj_3], G_2), O_2=Ungroup(G_1) and O_3=Group([G_1, Obj_4], G_3). Execution orders are different at different sites, assuming that the order is: O_2, O_1 and O_3 at site 2. O_1 cannot be executed directly after O_2's execution, because the current document state has been changed. To execute O_1 correctly, we should retrace the document state to that when O_1 is generated, and execute O_1 in this new document state, then retrace back to the current document state and finally execute subsequent operation O_3. At this moment, the state of G_1 is Ineffective, Obj_3 and Obj_4's are Effective. Fig.1 also shows the result of the execution of O_2 and O_1 on user's view.

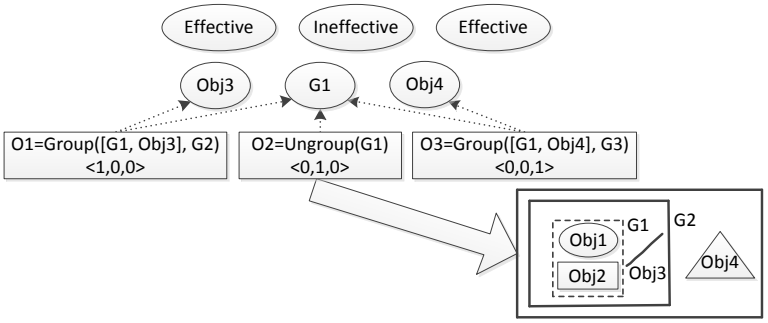


Fig.1 the AST Strategy and the User Interface View

3. Document Model and Basic Operations

3.1. Document Model

Here, an address tree is used to store the target objects. In the tree, Object is the basic unit as a leaf node, and Group can be represented both as a parent node and leaf node, which means that Group can contain Objects and Group at the same time. If one Group both has a parent and children, it is called an intermediate node, as G_3 shown in Fig.2, and if one Group has only children, it is the root node, as G_5 in Fig.2. The parent and children information, together with the state for each Object or Group shall be recorded, i.e., Obj/G :=(<Parent, Children>, State). Wherein, Obj denotes a simple object, and G is a group. Parent denotes one object or group's parent node of which has only a parent. Children, expressed with an unordered list ($Child_1$, $Child_2$,…, $Child_n$), denote the leaf nodes of one group which may have several children. State denotes one object or group's state, which has two states: Effective and Ineffective. As shown in Fig.2, G_5 :=(<Null, (G_4, Obj_7, Obj_8)>, Effective), G_3 :=(<G_4, (Obj_4, Obj_5, Obj_6)>, Effective) and Obj_3 :=(<G_2, Null>, Effective).

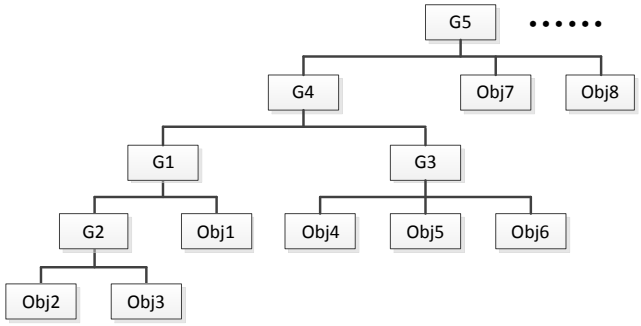


Fig.2 an Instance Diagram of Document Model

3.2. Basic Operations

In this part, five types of basic operations are introduced, including Create, Delete, ChangeAtt, Group and Ungroup. In addition, Conflict and Compatible Relations are also defined in detail.

Definition 1: Graphical Operations

Create(Obj): an object Obj is created.

Delete(ObjList): an ObjList is deleted.

ChangeAtt(ObjList, Attribute): the attribute of an ObjList is updated, wherein, the type of Attribute can be Position, Color, Size, Text, etc. For example, ChangePosition(ObjList, (dx_0, dy_0), (dx_1, dy_1)) moves ObjList from the initial coordinate position (dx_0, dy_0) to (dx_1, dy_1),

ChangeColor(ObjList, Color) changes the color of ObjList to color Color (such as Red, Green, Blue, etc.), ChangeSize(ObjList, (dx_0, dy_0), (Δdx, Δdy)) changes the size of ObjList by the ratio Δdx and Δdy, with (dx_0, dy_0) as the center.

Group(ObjList, G): an ObjList is grouped into a group G.

Ungroup(G): a group G is ungrouped, and the objects or groups contained in G still exist but no longer belong to the G.

Note that the ObjList mentioned above can be either an object or a group which includes objects or groups with an unordered list [Ojb_1, Obj_2, …, Obj_n].

Definition 2: Conflict Relations "⊗" [9]

Given two operations O_1 and O_2, they conflict with each other, denoted as $O_1 \otimes O_2$, iff:

(1) $O_1 \parallel O_2$;

(2) $Target(O_1) \cap Target(O_2) \neq \{\}$;

(3) $Att.Type(O_1) = Att.Type(O_2)$;

(4) $Att.Value(O_1) \neq Att.Value(O_2)$.

Note that Target(O) denotes the target object or group of operation O, Att.Type(O) denotes the attribute type of O, and Att.Value(O) denotes O's attribute value.

Definition 3: Compatible Relations "⊙" [9]

Given two operations O_1 and O_2, if they are not conflict with each other, they are compatible relations, denoted as $O_1 \odot O_2$.

4. Consistency Maintenance Strategy of Group/Ungroup Operations

4.1. Description of the Algorithm

The main idea of the algorithm is that the local operation can be executed immediately, and then attached with its generating site's state vector, broadcast to other sites. As for the remote operation, it cannot be executed at once. First of all, it is checked to find whether it is a causally ready operation or not. If not, it has to be queued since the sending site has executed operations which have not been executed at this site. If so, steps are as follows: if the operation desires to Delete/Ungroup one object or group that has been deleted/ungrouped, it is cancelled. If not, firstly, retrace the document state to that when the operation is generated, and then find all executed conflict operations, if there is no such operation, execute the operation directly in this new document state, if there does exist such operations, retain the effects of the operation and all other conflict ones to create multiple versions with Multi-Version Strategy. Finally, retrace back to the current document state and add the operation into the history buffer (HB).

The MVSDR (Multi-Version Strategy based Double Retracing) algorithm reveals the execution process of remote operation O_i at one site, assuming that O_i is a causally-ready operation. Wherein, Docs denotes the current document state, SVo_i is the state vector at the time of O_i's generation, and SV_c is the state vector of current document state. Besides, all executed operations are stored in HB.

Algorithm: MVSDR(Docs, O_i, HB):

Note: Given executed operations O_1, O_2,…, O_{i-1}, and O_i is the operation to be executed.

Begin:

1. HB ={ O_1, O_2,…, O_{i-1}};
2. If O_i is a Delete/Ungroup Operation whose target object/group has been Deleted/Ungrouped
3. O_i is refused to be executed;

4. else
5. Retracing(Docs, SVo_i); //call Retracing function
6. If there is any executed operation O_j ($1 \le j \le i-1$), such that $O_j \otimes O_i$ then
7. FindConflict (O_{set}, O_i); //call FindConflict function
8. Multi-Version(O_{set}, O_i, VS_i); //call Multi-Version function
9. else
10. execute O_i directly;
11. end if
12. end if
13. $SV_c = SV_c + 1$;
14. Retracing(Docs, SV_c); //call Retracing function again
15. HB =HB+{O_i};

End

The Retracing function specifies the procedure of retracing the document state to that at a given timestamp SVo_i, wherein, SVo_i is the state vector when O_i is generated, and ON is the object or group node in the tree structure of the document Docs.

Function 1: Retracing(Docs, SVo_i): Docs

Note: Before executing O_i, retrace the document's state to the time when O_i is generated. Wherein, ON is the Object (including group) Node, in the tree structure of the document Docs.

Begin:

1. For any Ungroup/Delete Operation O_{ugr}/O_{del} of ON
2. If the O_{ugr} is timestamped by SV_{ugr}, and $SV_{ugr} < SVo_i$ then
3. Set ON Ineffective;
4. ON.children ←ON.parent;
5. else if the O_{del} is timestamped by SV_{del}, and $SV_{del} < SVo_i$ then
6. Set ON Ineffective;
7. else
8. Set ON Effective;
9. end if
10. end for
11. For any other Operation O_{any} of ON
12. Set ON Effective;
13. end for

End

The FindConflict function specifies that all executed operations which are conflict with O_i are stored in O_{set}.

Function 2: FindConflict (Oset, O_i): Oset

Note: Given executed operations O_1, O_2,…, O_{i-1}, and O_i is the operation to be executed.

Begin:

1. O_{set} ={};
2. Remove O_j from HB and repeat until HB ={};
3. If $O_j \otimes O_i$ then
4. $O_{set} = O_{set} + \{O_j\}$;
5. else
6. do nothing;

7. end if
8. Return O_{set};

End

The Multi-Version function specifies the process of preserving the effects of O_i and all executed conflict operations that are stored in O_{set} to generate multiple versions. Mainly four cases are listed in the function, and other cases can be accomplished based on the Multi-Version idea.

Function 3: Multi-Version(O_{set}, O_i, VS_i): VS_i
Note: O_i is the operation to be executed, O_{set} is the set of all executed operations that conflict with O_i, and VS_i is the new generating versions set.

Begin:
Case 1: O_i and O_{set} are Group Operations // Case 1: Group Operations
1. Pre: O_i =Group($ObjList_1$, G_1) and O_{set} =Group($ObjList_2$, G_2) and $ObjList_1 \cap ObjList_2 \neq \{\ \}$
2. $VS_i \leftarrow \{V_{ii}$ ($ObjList_1$, G_1); V_{ij}($ObjList_2$, G_2)$\}$; //j is the number of the operation in O_{set}

Case 2: O_i and O_{set} are ChangePosition Operations // Case 2: ChangePosition Operations
3. Pre: O_i =ChangePosition(ObjList, (dx_0, dy_0), (dx_1, dy_1)) and O_{set} =ChangePosition (ObjList, (dx_0, dy_0), (dx_2, dy_2))
4. $VS_i \leftarrow \{V_{ii}$ ($ObjList_1$, (dx_1, dy_1)); V_{ij}($ObjList_2$, (dx_2, dy_2))$\}$;

Case 3: O_i and O_{set} are ChangeColor Operations // Case 3: ChangeColor Operations
5. Pre: O_i =ChangeColor($ObjList_1$, $Color_1$) and O_{set} =ChangeColor($ObjList_2$, $Color_2$) and $ObjList_1 \cap ObjList_2 \neq \{\ \}$
6. $VS_i \leftarrow \{V_{ii}$ ($ObjList_1 \cap ObjList_2$, $Color_1$; V_{ij}($ObjList_1 \cap ObjList_2$, $Color_2$; $\}$;
7. Put {$ObjList_1$ - $ObjList_2$, $Color_1$} on VS_i;
8. Put {($ObjList_2$ - $ObjList_1$, $Color_2$} on VS_i;

Case 4: O_i and O_{set} are ChangeSize Operations // Case 4: ChangeSize Operations
9. Pre: O_i =ChangeSize(ObjList, (dx_0, dy_0), (Δdx_1, Δdy_1)) and O_{set} =ChangeSize(ObjList, (dx_0, dy_0), (Δdx_2, Δdy_2))
10. $VS_i \leftarrow \{V_{ii}$ ($ObjList_1$, (Δdx_1, Δdy_1)); V_{ij}($ObjList_2$, (Δdx_2, Δdy_2))$\}$;

Case 5: O_i and O_{set} are other types of Operations //Other Situations
11. Do something accordingly based on the Multi-Version idea;
12. end Case
13. CheckVersion(VS_i); //Check and Delete the redundant versions
14. Return VS_i;

End

The CheckVersion function checks whether there are duplicate versions or not, and if so, those redundant versions will be deleted.

Function 4: CheckVersion(VS_i): VS_i
Note: VS_i is the new generating versions set.

Begin:
1. For each V_i in VS_i {
2. For each V_j in VS_i{
3. If $i \neq j$ and $V_i = V_j$
4. Delete V_j;
5. end if
6. }
7. }

4.2. Example Analysis

Assume that the shared editing area for users is a square, and the graphical initial state on the user interface view is shown in Fig.3. Wherein, group G_1 includes two objects Obj_1 and Obj_2, Obj_3 and Obj_4 are the other objects.

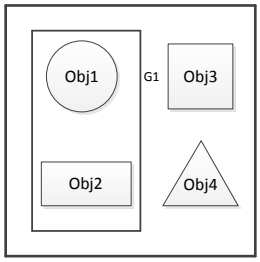


Fig.3 the Initial State

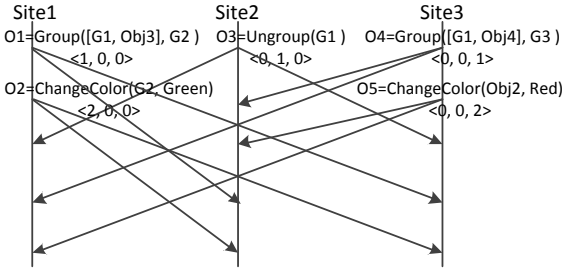


Fig.4 Example Analysis

As shown in Fig.4, O_1 and O_2 are generated at site 1, O_2 at site 2, O_3 and O_4 at site 3, and their relations are: $(O_1 \rightarrow O_2) \parallel O_3 \parallel (O_4 \rightarrow O_5)$.

Now, the process of executing operations will be analyzed detailedly at all sites, and the execution result is shown in Fig.5.

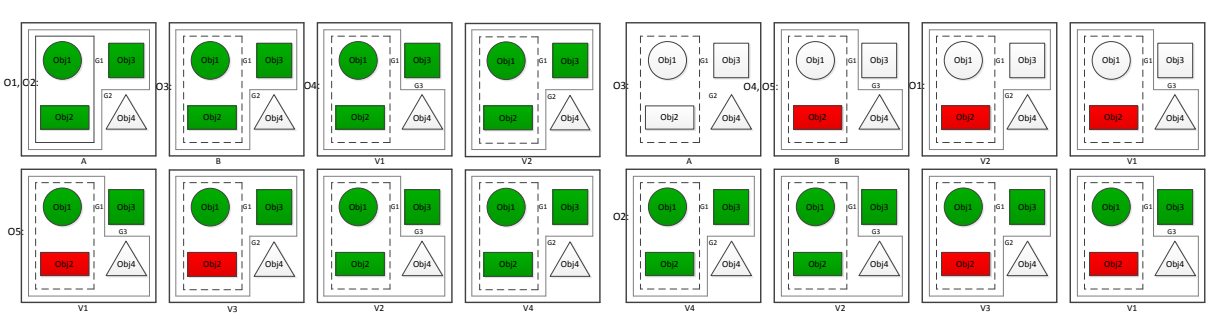


Fig.5 Execution Result of Operation Process

At site 1: (the execution order: O_1, O_2, O_3, O_4, O_5)

1. O_1 and O_2 are executed immediately at the local site 1, generating the version A.

2. With the arrival of O_3, HB ={O_1, O_2} at the moment. Firstly, retrace the document state to that when O_3 is generated, whose state vector is SVo_3<0, 1, 0>. Then for executed operations O_1 and O_2, the state of their target object or groups G_1, Obj_3 and G_2 shall be set Effective. Due to both O_1 and O_2 do not conflict with O_3, so O_3 is executed directly in this document state. Secondly, retrace back to the current document state whose state vector is SV_c<2, 1, 0>. Since O_3 is an Ungroup operation and $SVo_3 < SV_c$, set the state of O_3's targeting group G_1 Ineffective, and G_1's parent node G_2 is given to Obj_1 and Obj_2. Finally, add O_3 into the HB, and the version B is generated finally.

3. When O_4 arrives at site 1, HB ={O_1, O_2, O_3}. Firstly, retrace the document state to that at the time of O_4's generation, whose state vector is SVo_4<0, 0, 1>. Then, put the state of target group G_1 Effective since $SVo_3 \geq SVo_4$, Obj_3 and G_2 are set Effective. And then, there is found executed operation O_1 are conflict with O_4, meeting the case 1 in Multi-Version function, thus generating two versions V_1 and V_2. Secondly, retrace back to the current document state whose state vector is SV_c<2, 1, 1>. Since O_3 is

Ungroup and $SVo_3 < SV_c$, set the state of O_3's targeting group G_1 Ineffective. And G_1's parent node is given to Obj_1 and Obj_2, thus making their parent node become G_3 in V_1 and the parent node is G_2 in V_2. Finally, O_4 is added into the HB.

4. At the time of O_5's arrival, HB ={O_1, O_2, O_3, O_4}. Firstly, retrace the document state to that at the time of O_5's generation, whose state vector is SVo_5<0, 0, 2>. Then, put the state of target group G_1 Ineffective since $SVo_3 < SVo_5$, Obj_3, G_2 and Obj_4 are set Effective. And then, it is found executed operation O_2 conflicts with O_5, meeting the case 3 in Multi-Version function, thus generating four versions V_1, V_2, V_3 and V_4. Secondly, retrace back to the current document state whose state vector is SV_c<2, 1, 2>. Since O_3 is Ungroup and $SVo_3 < SV_c$, still set the state of its targeting group G_1 Ineffective. And G_1's parent node is given to Obj_1 and Obj_2, thus making their parent node become G_3 in V_1 and V_2 and the parent node is G_2 in V_3 and V_4. Finally, add O_5 into the HB so that HB ={ O_1, O_2, O_3, O_4, O_5}.

The process at site 3 is not described in this paper because of its similar to that at site 1, and site 2's process is omitted here due to this paper's length limitation, with their same results as at site 1.

As we can see, the final result is always the same even though the execution orders are diverse at diverse sites, thus proving the algorithm's correctness and effectiveness.

5. Conclusions and Future Work

In this paper, based on the AST method and the Multi-Version Strategy, we propose a novel approach to resolve the Group/Ungroup consistency maintenance problem in graphical editing systems. Since abandoning the attempt to classify conflict operations into Real-Conflict and Resolvable-Conflict operations, the algorithm adopted becomes simpler and easier to achieve. This paper's main contribution includes that it is the first time to adopt AST method to solve Group/Ungroup questions in graphical environments, which does not have to consider complex relations among operations and reduces delay greatly, with comparison to previous approaches such as Serialization and OT. In addition, the Multi-Version Strategy is proposed to resolve conflict operations' problem, which reflects the effects of all users and satisfies the users' intentions better than the priority-based policy.

However, there are other operations such as Undo/Redo of Group/Ungroup and the version identification in practical graphical editing systems, which is the next focus on these issues in the future work.

Acknowledgements

The work is supported by the National Natural 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.

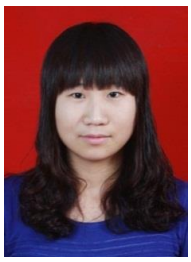
References

[1] Claudia-Lavinia Ignat, Moira C. Norrie. Grouping/ungrouping in graphical collaborative editing systems[C]. In: Proceeding of the 5th Int. Workshop on Collaborative Editing Systems, **(2003)**, Helsinki, Finland.

[2] Claudia-Lavinia Ignat, Moira C. Norrie. Grouping in collaborative graphical editors[C]. In: Proceedings of the 2004 ACM conference on Computer supported cooperative work. ACM, **(2004)**, pp: 447-456.
[3] Claudia-Lavinia Ignat, Moira C. Norrie. Draw-Together: Graphical editor for collaborative drawing[C]. In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. ACM, **(2006)**, pp: 269-278.
[4] Xia S, Sun D, C Sun, Chen D. Collaborative object grouping in graphics editing systems[C]. Collaborative Computing: Networking, Applications and Worksharing, **(2005)**, 2005 International Conference on. IEEE.
[5] Xueyi Wang, Jiajun Bu, and Chun Chen. Achieving Undo in Bitmap-based Collaborative Graphics Editing Systems. In: Proceeding of the Conference on Computer Supported Cooperative Work, **(2002)**, pp: 68-76.
[6] Sun C, Ellis C. Operational transformation in real-time group editors: issues, algorithms, and achievements[C]. In: Proceedings of the 1998 ACM conference on Computer supported cooperative work. ACM, **(1998)**, pp: 59-68.
[7] David Sun and Chengzheng Sun. Operation Context and Context-based Operational Transformation [A]. In: Proceeding of the Conference on Computer Supported Cooperative Work, **(2006)**, pp: 279-288.
[8] Ning Gu, Jiangming Yang and Qiwei Zhang. Consistency Maintenance Based on Address Space Transformation Technique in Group Editor [J]. Chinese Journal of Computers, 30(5): 763-774 **(2007)**.
[9] Y Cheng, F He, B Xu, S Han, X Cai and Y Chen. A multi-user selective undo/redo approach for collaborative CAD systems. Journal of Computational Design and Engineering, 103-115 **(2014)**.
[10] Hansu Gu, Haojie Hang, Qin Lv and Dirk Grunwald. Fusing Text and Frienships for Location Inference in Online Social Networks. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on, vol.1, **(2012)**, pp. 158-165. IEEE.
[11] 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, **(2013)**, pp. 629-638. ACM.

Authors

Liping Gao(1980-) graduated from Fudan University, China with a PhD in 2009 in Computer Science. She received her BSc and master degree in Computer Science from Shandong Normal University, China in 2002 and 2005 respectively. 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 computer.