International Journal of Future Generation Communication and Networking
Vol. 9, No. 5 (2016), pp. 49-62
http://dx.doi.org/10.14257/ijfgcn.2016.9.5.06

Research on Conflict Resolution and Consistency Maintenance Supporting Intention Combination in Real-time Collaboration Environment

Qiongqiong Fu[1], Liping Gao[1*] and Naixue Xiong[2]

[1]School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China;[2]Shanghai Key Lab of Modern Optical System, and Engineering Research Center of Optical Instrument and System, Ministry of Education, University of Shanghai for Science and Technology, Shanghai 200093, China
lipinggao@fudan.edu.cn and xiongnaixue@gmail.com

Abstract

With the real-time group editing system, a group of users can view and edit the shared document by communication networks anytime and anywhere. Under the circumstance, it is surely inevitable that many operations from different users are going to conflict. Thus, two issues, the conflict resolution and the consistency maintenance, are the most important for designing and completing the system. In the past, the address space transformation algorithm, invented from the research about the real-time text editing system, could maintain consistency among more sites. The Multi Version conflict resolution approach could preserve users' intentions but not all when conflicts occur. This paper proposes a new solution of conflict resolution, named Intention Combination Conflict Resolution strategy with the document model of AST based on the idea of Multi Version approach. This solution can not only preserve all users' editing consistency by intention combination, but also keep all versions of operational objects including conflict operations' combination effects. In addition, the effectiveness of related algorithms is analyzed, and the availability of the strategy is described by a case and proved by our experiment.

Keywords*: Real-time Collaboration, Address Space Transformation, Multi Version, Intention Combination, Conflict Resolution, Consistency Maintenance*

1. Introduction

With the development of globalization, the phenomenon that one project is completed by cross-regional and cross-field cooperation, becomes more and more common. The demand of collaborative work in online social network becomes increasingly urgent. The groupware systems or the real-time collaborative systems [1-9] provide one possible platform with the increase of the demand about collaborative work in online social network [10]. People from different fields and different areas can edit the same shared document [2, 3, and 8] synergistically in the cooperation editing platform. It can largely improve the work efficiency. In order to hide the effect of network delay and reduce the response time of local operations, a replicated architecture [1-6, 8, 9, 11] has been widely adopted. In this architecture, local operations get responses immediately, then multicast to remote sites, and concurrent operations, whether local operations or remote, will be transformed before executed to keep the consistency of the document replicas. The whole process is the consistency maintenance [2-9, 11] of the document states.

At present, the consistency maintenance strategies are mainly divided into two categories: one is based on Operational Transformation (OT-based) [1-4, 8, 11] and the

ISSN: 2233-7857 IJFGCN

other is based on Address Space Transformation (AST-based) [3, 6, 9, 11]. The OT-based method is to compare remote operations with all operations in the local history buffer (HB)[2, 4,1 1], and then convert concurrent operations in the HB by comparing the timestamps of each operation, for that operations can be executed correctly under the current document state. OT has been applied to many collaborative text editors, including Grove [1], REDUCE [11], *etc*. AST is a technique different from OT for consistency maintenance and group undo [3-5, 8]. It can preserve operational intentions by transforming the address space, but not the operation itself and support the consistency maintenance of Insert, Delete and Update operations in a text editing field. Because it doesn't need to transform users' operations, the algorithm efficiency is higher than OT. Now AST has been an alternative of consistency maintenance and Undo/Redo [2, 4, and 9] technique for some real-time collaborative text editors.

In some applications, for one thing, the operational object may have rich types of attributes; for another users often not only insert or delete one object, but also update some attributes of one object. Moreover, this demand perhaps is inevitable for the collaborative editing. Thus, the collaborative system should gain the ability to update synergistically to serve the group work better. In [2], OT strategy is extended to support Update operations; however, this strategy requires traversing the whole historical sequence, which is going to lower the algorithm efficiency. Multi-Version Single-Display (MVSD) [2] is a typical technique to solve the conflict [2, 8] of concurrent Update operations. Update operation just modifies one object's attribution and conflicting Update operations will produce multiple versions for an attribute of one object. But Multi Version (MV) [2] technique just takes single operation effect into consideration and cannot preserve users' combined intentions. Therefore, in this paper, we propose a new conflict resolution strategy. It adopts the idea of AST's internal document management, and increase the algorithm efficiency due to reduce the comparison times of operations in HB. The strategy can not only make the document states consistent but also preserve combination effects of conflict operations.

The rest of this paper is organized as follows. In Section 2, the basic AST method, the conflict concept and the MVSD technique are introduced briefly. Then, a new conflict resolution approach is discussed and the algorithm efficiency is analyzed in Section 3. In Section 4, an example is described in detail. A prototype system called Co-NotePad is depicted in section 5. Finally, major contributions as well as shortages of this work are summarized in Section 6.

2. Related Work

2.1. AST

AST is originally proposed for synergistically editing the plain text document based on the linear structure. Given one site with the kind of document and the State Vector (SV)[1,3,4,8] based on Timestamp, an operation generated by the local site will be attached with the current SV and broadcasted to other sites. Now, we give the formats of three primitive operations.

1） Insert (p, obj)
2） Delete (p, obj)
3） Update (p, attribute, new-value, old-value)

In representations above, *obj* indicates the goal object of an operation. We need to note in this paper one operation is limited to cover only one object for the sake of simplicity although in the real cooperative system an operation can cover more than one object. *p* refers to the logical address of *obj* in the linear address space. *attribute* donates the goal attribute of Update.*obj*. *old-value* refers to the original value of this attribute, then *new-value* refers to new value of the attribute.

In a document based on linear structure, every operation has only one target object and the type of the target object just can be characters, while every character may be related with several operations. Besides, the address space transformation is achieved by Retracing procedure in which a few steps are needed, such as scanning all nodes and using *effective / ineffective* [3,9] to mark the effectiveness of the character of a node. The mark decides that the character is visible or not on the user interface. In [3] the AST document model is shown as in Figure 1.

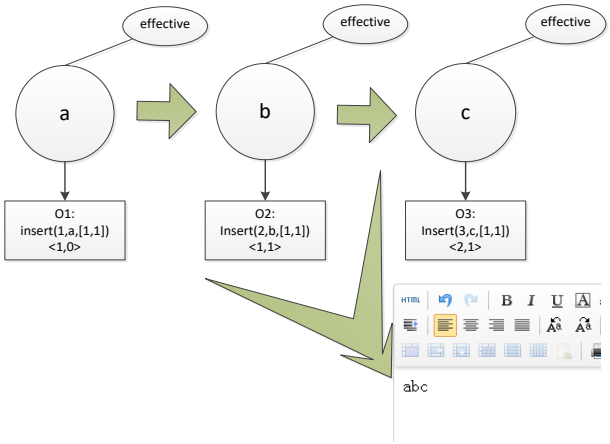

Figure 1. The Linear Document Structure and Model of AST

2.2. Conflict Relation

Every site in one group works on the same document in which every user in local site can see others' operation effects as soon as possible. However, some incompatible situations perhaps appear. For example, when two users' operations are concurrent and their target objects as well as the goal attributions are the same, and before that they don't know each other's operational intention, executing these two operations is bound to result in the conflict. The definition of conflict between Updates is given as follows.

Definition 1 Conflict Relation [2]: Given two Update U_a and U_b, they have a conflict relation, denoted as $U_a \otimes U_b$, iff: (1) U_a and U_b are concurrent; (2) their operational objects are the same, namely, $U_a.p = U_b.p$; (3) $U_a.attribute = U_b.attribute$. If a relation cannot meet these three conditions, it must not be the conflict relation, but the compatible relation for these Updates.

Here is an example to explain it. Suppose that there is a sentence "*I am a student.*". $site_1$ sends an Update operation U_1 aiming at '*a*' on the third position to change its color from *Black* to *Red*. Meanwhile, $site_2$ sends U_2 aiming at '*a*' on the third position to change its color to from *Black* to *Green*. Hence the U_1 and U_2 are conflict, which is denoted as $U_1 \otimes U_2$.

2.3. Multi-Version Single-Display (MVSD)

Each site generates an Update at the same time. If they conflict with each other, the possible effects of the operations can be divided into three categories:

1) Null-effect: none of the operations has any final effect on the target object.

2) Single-effect: only one operation has a final effect on the target object.

3) Combined-effect: more than one operations have the final effect on the target object.

The core idea of MVSD is as follows: when an object is updated by conflict operations, multiple versions of the object should be maintained internally, but only one version is displayed on the user interface.

Definition 2 Combination Version: Two or more operations update the same object's same attribute, which can lead to produce not only one possible effect of this attribute but also the combined effect formed by new values of conflict operations. Therefore, it can be called the combination version of the operations. For example, the original value of the color attribute of a character is *Black*, but if there are two new values, *Red* and *Green*, the combination version of the character object or to say mixed effect of the two operations is *Yellow*.

3. Conflict Resolution Approach

3.1. Primitive Operation

In order to apply the MV technique to collaborative editors with the linear document structure, the formats of these three primitive operations should be modified accordingly to preserve different intentions of conflict operations. The new definitions of them are given to support new document structure and maintain combination version conveniently. Also, each operation is attached with SV. However, another parameter of an operation should be mentioned - the source site number. The source site has own number, called siteId, where an operation is generated.

1） Insert (p, obj, [level,id])
2） Delete (p, [level,id])
3） Update (p, attribute, new-value, old-value, [level,id])

Where *[level, id]* indicates the position of an operation in the operation sequence of a character object. *Level* is the corresponding layer where the operation lies in, and *id* refers to a serial number of the operation in the layer. Apparently, the position indication *[level,id]* of an Insert operation for any object node shall be [1,1]. Specially, *id* in *[level, id]* of Update operation refers to the version ID for any attribute of one object. The order of version IDs is sorted by the siteIds. In addition, *level* depends on whether operations conflict or not by comparing SVs. *old-value* is the original value of an attribute effect of one object and *new-value* is the new value. Every new effect may be one of three types mentioned above. Suppose that the effect depends on one or more Update operations from $\{U_1, U_2, \ldots, U_n\}$. If it is determined by only one operation, new-value is the value of single effect. If it is determined by more than one operation, new-value is the value of combined effect of these operations. But if null, it will keep original effect unchangeable (Null-effect is not described in the following because Undo can support this kind of effect). Other parameters' definitions are still remained. We can understand these concepts further combining with the document model in Figure1.

For simplicity, each site is defined as an integer {1, 2, ..., N}. The siteId mentioned at the beginning of this section can be regarded as a sequence of Numbers in order, from smaller to bigger. Such assumption is reasonable because Single-effect is corresponding to only one site, whereas Mixed-effect is corresponding to several sites. The comparison rules of siteIds should be given as follows. If there are two siteIds, siteId_i and siteId_j, three comparison results are that:

1) $siteId\_i < siteId\_j$: when the length of siteId_i is less than that of site_j, *i.e.*, the sequence of siteId_i is part of siteId_j, or more than one parameter of siteId_j is larger than that of of siteId_i; when their lengths are equal or the former is longer than the latter, more than one parameter of siteId_j is larger than any one of siteId_i.

2) $siteId\_i = siteId\_j$: each version has only one siteId and the corresponding version ID is unique, so this case must not exist because *i* is not equal to *j*.

3) $siteId\_i > siteId\_j$: the conditions of this case are opposite of those in case 1).

3.2. Document Storage Structure for Combined-effect

In AST document model, the execution effect of an Update operation can be visible on the users' interface, because the operational object is marked as effective. And Update operation must be behind Insert operation in the operation sequence related to one node. Besides, Update just aims at attributes of an object, while Insert and Delete aim at the position of an object in the linear address space. In sum, Update can't conflict with Insert or Delete.

According to the idea of AST and MVSD techniques, both the operation sequence andmultiple versions of the target object should be maintained. Consequently the document model is shown as in Figure 2. A linear sequence called Version_list is added to store the version ID, relevant siteIds and the operation_list of each version. Among them, the operation_list consists of a few operations that can form a version. The document structure model can be illustrated through an example in Figure 2. obj1 has Insert(1,obj1,[1,1]) called O_1. obj2 has Insert(2,obj2,[1,1]) called O_2, Update(2,color,red,black,[2,1]) called O_4, Update(2,color,green,black,[2,2]) called O_5 and Update(2,color,yellow,black,[2,3]) called O_6. obj3 has Insert(3,obj3,[1,1]) called O_3.

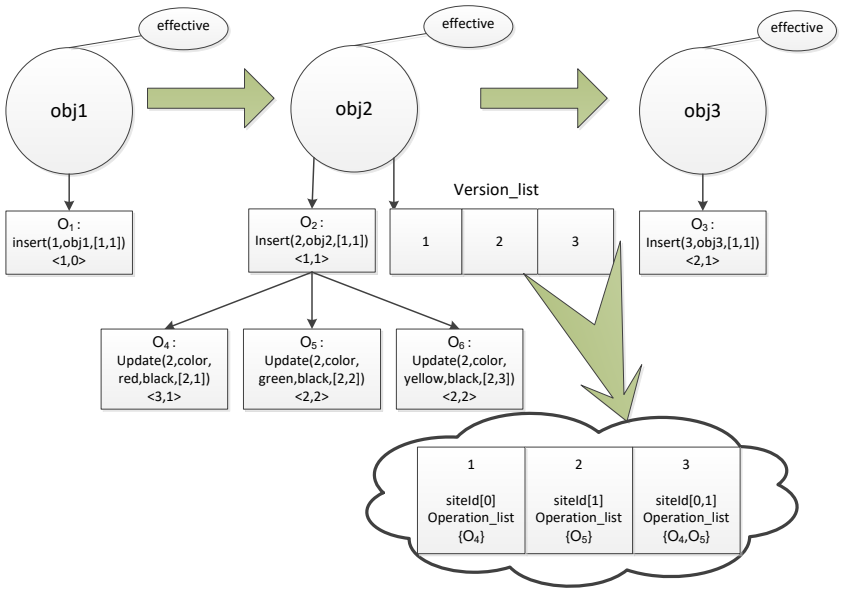

Figure 2. The Document Structure Model

If the relation between two operations, O_4 and O_5, is conflict relation, i.e. $O_4 \otimes O_5$, a combined effect should be formed by them. Correspondingly O_6 is generated for the combined effect of O_4 and O_5. Because the position indication of O_5 is behind O_4's, the SV and siteId of O_6 should be the same with O_5's. That is to say, the obj2 has a combination version attached with O_6.*new-value*. Then, these versions produced by three Updates operations O_4,O_5,O_6 should be added to the version_list in order, respectively called versions No.1,2,3. At the same time, siteId and operation_list of each version also should be recorded into the version_list. Specially, if the relation is compatible relation, every compatible operation should be recorded into the operation_list of all existing versions. Now we discuss the total number of versions among conflict operations. For example, two Update operations conflict, the total number is 3; three Update operations conflict, the total number is 7. That is to say, if N is the number of conflicting Update operations, the total number of operation versions should be 2^N-1.

3.3. Conflict Resolution Strategy Supporting Intention Combination

The procedure of the conflict resolution strategy supporting intention combination is shown as below. When an Update is generated, it can be immediately executed at local site. But at the remote sites it will be received to compare with all operations attached to the target node. Then, check if there exist other operations concurrent with the operation. The Update operation should be added to the layer of the operation sequence related to the node as long as operations match the definition of conflict relation. Meanwhile, the current document state should be retraced to the state that the operation wasn't executed by using the Retracing process in [3]. Then, compare site IDs to keep all versions consistent and ordered at all sites. Next, new combination versions are created by conflict operations in this layer. Finally, these versions need be added to version list, including record site Id and add corresponding operations to operation list accordingly.

Especially, there is no need to discuss how to solve the conflict between Insert and Delete because it can be solved by transforming address space, and the conflict is not in accordance with the conflict relation mentioned above. Hence, this paper just focuses on the conflict relation between Update operations. Just the relation between operations can satisfy the conditions of conflict relation in Section 2.2, the conflict can be solved reasonably by ICCR (Intention Combination Conflict Resolution) strategy we proposed in Procedure 1. Otherwise, the Retracing process and Range-Scan process [3] can be adopted to solve it.

Procedure 1: ICCR(U)
// U is a remote Update operation. SV_u means SV attached to U when it's generated. U_{org} is the first operation in the operation sequence of the same object and conflict with U. *i* is the *level* layer the conflicting Update operation lies in. *j* is the version ID (1<=j<=n, n is the current total number of versions of Updates that conflict with each other in the current layer)

```
Find all operations concurrent with U in the Node of U.obj which is effective by
scanning the list of operations according to SVs.
  If  there exists Uorg[level, id]  Upd_Conflict with U{
      If( the id version has not existed in Version_list){
            Add Uorg[level, id] to Version_list[Uorg.id];
      }Else{
            m=n;
            U.level=Uorg.level;
            U.id=++n;
            Add U to HB of the obj in the position of No. U.id, Level U.level;
            Add U[level, id] to Version_list[U.id];
            Retrace(Doc_s, SV_u);   // Retrace to the state of the operation not
executed.
            for(j=n-1; (U.siteId<U[i, j].siteId) && (j>0); j--){ // Maintain the layer
mark of each site
            consistency.
                Exchange(U.id, U[i, j].id);
                Exchange(Version_list[U.id], Version_list[U[i, j].id]);
            }
            for(i=level, j=id; j<=m; j++){ // Consider the case of Updates conflicted
            with each other in the same layer.
                U'=Create_newVersion(U[i, j], U);   // U' is an operation
                corresponding to the Mixed Version created by existing conflicting
                Update operations.
```



```
                    Add U' to HB of the obj in the position of No. U'.id, Level U'.level;
                    Add U'[level, id] to Version_list[U'.id];
                }
            }
    }
```

One of the most important steps in the conflict resolution strategy is to create the combination version. In other words, a new Update operation for the same attribute of the same object will be generated. So Function 1 is designed to solve this problem. To generate the new version, all parameters of the operation should be set properly according to the definition of the Update operation.

Function 1: Create_newVersion(U[i, j], U): U'
// Create combined Version U': U is a remote operation; U[i, j] is the j operation in the i layer of operation sequence conflicting with U; U' is a new operation to return.

```
If(j!=U.id  && U[i, j] and U are Upd_conflict) {
    U'.obj=U.obj;
    U'.p=U.p;
    U'.attribute=U.attribute;
    U'.new_value=Combined_effect(U.new_value, U[i, j].new_value);
    U'.old_value=U.old_value;
    U'.level=level;
    U'.id=++n;
    If (U.siteId > U [i, j].siteId) {   // U'.siteId is produced by the siteIds of the
    operations that will generate combined effect.
        U'.siteId ={U[i, j].siteId, U.siteId};
        SV=SVu;
    }Else{
        U'.siteId ={U.siteId, U[i,j].siteId};
        SV=SVui,j ;
    }
    for(k=n-1; (U.siteId<U[i, k].siteId) && (k>0); k--){
        Exchange(U'.id,U[i,k].id) ;
    }
}
```

According to the ICCR strategy, the conflict case between two sites, as shown in Figure 3, can be used to explain the procedure above. At first, $site_0$ sends an Insert operation O_1, and its purpose is to insert a character '*a*' in the first position of a plain text. O_1 is executed at once in local site, whose SV is <1, 0> and added to the operation sequence of the object, with the position mark [1, 1]. Then it will be broadcasted to $site_1$, and executed at once because there are not concurrent operations. Then, it is also added to the HB of the object at $site_1$, with the same mark as [1, 1].

Next, $site_1$ sends operation O_2 to insert '*b*' to the second position in the document. The executing process is the same with that of the above. Its SV is <1, 1> and position mark is [1, 1]. Then it will be broadcasted to the remote site, with the mark of [1, 1]. The current document states at two sites are consistent ("*ab*") and their default original color is *Black*.

Afterwards, $site_0$ generates Update O_3 to update the color of '*b*' in the second position to *Red*, SV<2, 1>. At the same time, $site_1$, which does not receive O_3 from $site_0$, also generates an Update O_4 to update the color of '*b*' in the same position to *Green*, SV<1,2>. The local sites respond to the operations and put them into the operation sequence

respectively, marked as [2, 1] and broadcast them to others except themselves. $site_0$ finds the conflict relation between O_3 and O_4 from $site_1$, then invoke the ICCR strategy to solve this conflict. By comparing the siteIds, O_4 should be marked as [2, 2]. In order to solve the conflict, the document state need be retraced; then, the combination version of these conflict operations is created with SV<1, 2>, namely, O'; finally, the whole versions are recorded to Version_list. The process is also applied to $site_1$. The implement procedure, including add SV to an operation and change the document state, can be seen in Figure 3.

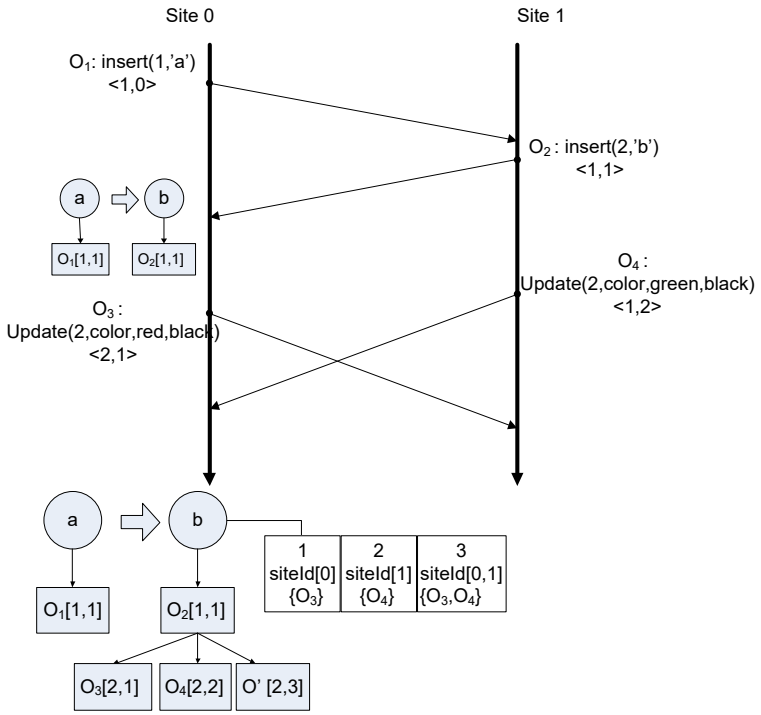

Figure 3. Conflict Resolution between Two Sites

3.4. Analysis of the Algorithm Effectiveness

The Procedure 1 can complete the conflict resolution among Update operations, which has been proved effective to a certain extent in the previous section. Then provided that there are M objects in the document and N Update operations conflict with each other. On one hand, the average time complexity of the searching process for a node in the linear sequence is O ((1+M)/2). And as for creating the combination version, suppose that there are N conflict operations, the total number of versions is 2^N-1 and the average times of invoking Create_newVersion function is (2^N-1-N)/N. On the other hand, Procedure1 and Function1 both have a loop to realize sorting the versions. And the best case is that the siteId of an operation is the biggest and ranked at the last, then the time complexity is O(1). The worst case is that the former N-1 operations have already been sorted, but the siteId of the last operation is smallest, then the loop comparison needs 2^{N-1}-1 times. So, the average time complexity of the sorting is O (2^{N-3}).

The overall time complexity of the algorithm can be computed as O ((1+M)/2+(2^N-1-N)/N+2^N-3). It should consider the whole execution procedure including find the target object, add new versions and sort version ID *etc*. If there are a large amount of operations, M can be considered as a constant. Also, the number of concurrent operations of one node can be considered as a constant.

In fact, for one thing, what happens not frequently in one collaborative editing work is that Update operation from each site with the same object and the same attribute conflict with each other. For another, setting the linear sequence to store versions can be very easy

for searching and selecting, and exactly save time to some extent. In addition, one version should contain just some necessary information. Based on the above reasons, the algorithm is feasible in theory and its time cost is within acceptable range.

4. The Example Analysis

The conflict resolution strategy ICCR is not only suitable for the case of two sites, but also for the case of more sites. The following example is a case of the Update_Conflict resolution applied to three sites. As shown in Figure 4, O_1 is Insert(1,a,[1,1]), O_2 is Insert(2,b,[1,1]), and O_3 is Insert(3,c,[1,1]). These three operations are causally before one by one. Thus the document state is "*abc*" after the Insert operations are executed. Then the three object nodes are all marked *effective*, namely, they can be visible to users. Next, $site_0$, $site_1$, and $site_2$ generate an Update operation O_4(Update(2,color,red,black)), O_5(Update(2,color,blue,black,)), and O_6(Update(2,color,green,black)) concurrently which aim at changing the color attribute of the object '*b*' to *Red*, *Blue*, *Green* respectively. According to the definition of the conflict relation, the relations of three operations are easily known, namely that $O_4 \otimes O_5$, $O_5 \otimes O_6$ and $O_4 \otimes O_6$. That is to say, these three operations conflict with each other. Therefore, the conflict should be solved by ICCR strategy proposed previously.

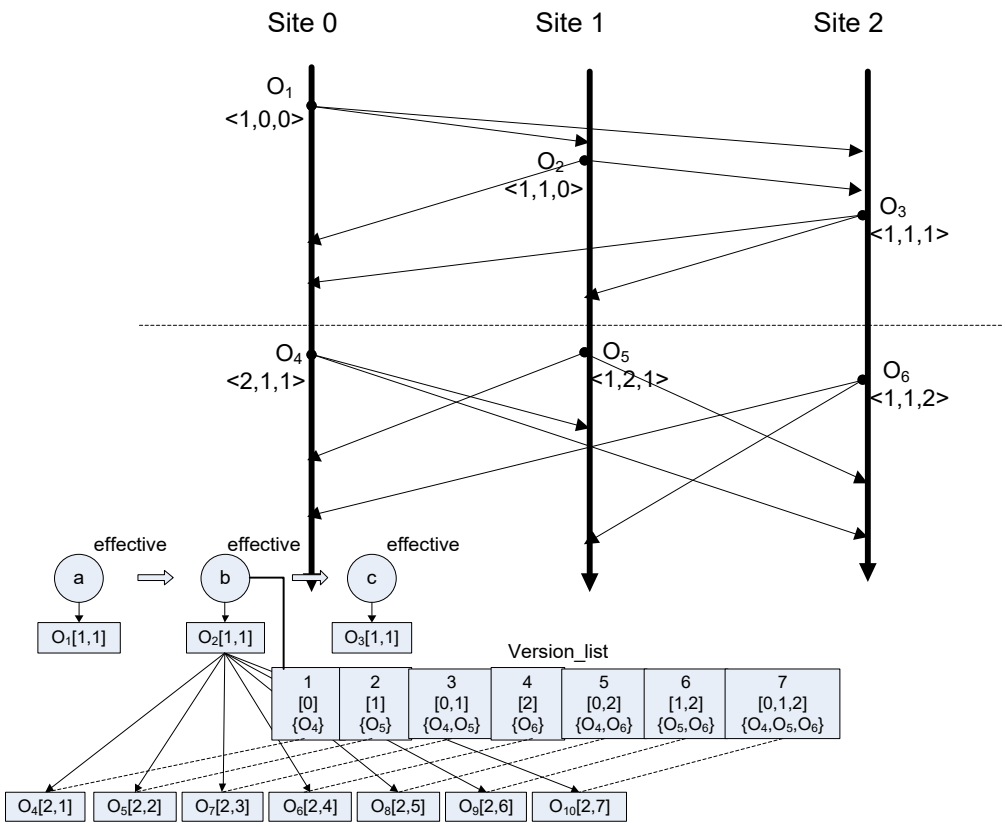

Figure 4. Conflict Resolution among Three Sites

In Figure 4, the conflict resolution procedure of Update operations at each site is very similar, so it can be depicted in detail by explaining the procedure at one of the sites, such as $site_0$.

1) At $site_0$, the local operation O_4 is executed immediately. Hence, the color of '*b*' is *Red*. O_4 is in the first position, the second layer of operation sequence of the object '*b*'

and is marked as [2, 1]. Then, the version information is added to Version_list.

2) When $site_0$ receives the remote operation O_5, O_5 is detected as conflict with O_4 by comparing SV of O_5 with that of each operation in the second node. Retrace the document state to timestamp <1, 2, 1>, cancel the operation effect of O_4 and put O_5 to the position [2, 2] since the siteId of O_5 is bigger than that of O_4. Then, the version information is added to Version_list.

3) At the same time, since they are conflict operations, a combination version O_7 consisted of O_4 and O_5 is added, and its position mark should be [2,3] as well as SV<1,2,1>. Then, add the version to Version_list and its siteId is [0, 1] which combines the siteId of O_4 and O_5.

4) Next, when O_6 arrives at $site_0$, another two combination versions should be created besides the version of O_6 itself because O_6 conflicts with O_4, O_5 and O_7. First, the document state is retraced and O_6 is put to the position of [2, 4] in the second node. Then, a combination version of O_8 marked with [2, 5] and attached with SV<1, 1, 2> will be created by O_6 and O_4 in the layer and be added to Version_list. In the same way, the combination version of O_9 marked with [2, 6] as well as SV<1, 1, 2> will be created by O_6 and O_5, and be added to Version_list. Another combination version of O_{10} marked with [2, 7] as well as SV<1, 1, 2> will be created by O_6, O_4 and O_5, and be added to Version_list.

On one side, there are seven versions totally, which is in accordance with the previous conclusion. The conclusion indicates that the number of N operations' versions is 2^N-1. On the other side, the final version ID and the order of versions at $site_1$ and $site_2$ is the same as those at $site_0$. To sum up, the case can prove that ICCR strategy can solve the kind of conflict and maintain the document consistency.

5. Co-NotePad Text Editing System

We have developed the Co-NotePad system supporting collaborative editing under Android development platform. Three operations (Insert, Delete, and Update) of the collaborative system, are realized. The Update operation in the system can only support the color modification of characters and ICCR strategy can be realized to preserve multiple intentions. More functions in the system will be added and improved in the future. To develop the Co-NotePad system, we has utilized the development tools of Eclipse, Java SDK, Android SDK, ADT *etc*. In addition, the editor can support two communication modes: GSM and WIFI.

First of all, the Co-NotePad system should be installed on multiple mobile devices. Then, each site starts to login in the server and select to download the same file or start an empty file. Here we select the same text file. Figure 5 shows the process that two clients, site 1 and site 2, want to update the color attribute of the first character *H* in the title simultaneously. From the figure we can see that site 1 has chosen the attribute of *Red* while site 2 has chosen the attribute of *Green*. Apparently, these two Update operations conflict with each other. Figure 6 shows the ICCR process result that the character *H* has three versions including the version of intention combination. The first version is from site 1, the second one is from site 2 and the last one is the combined effect.

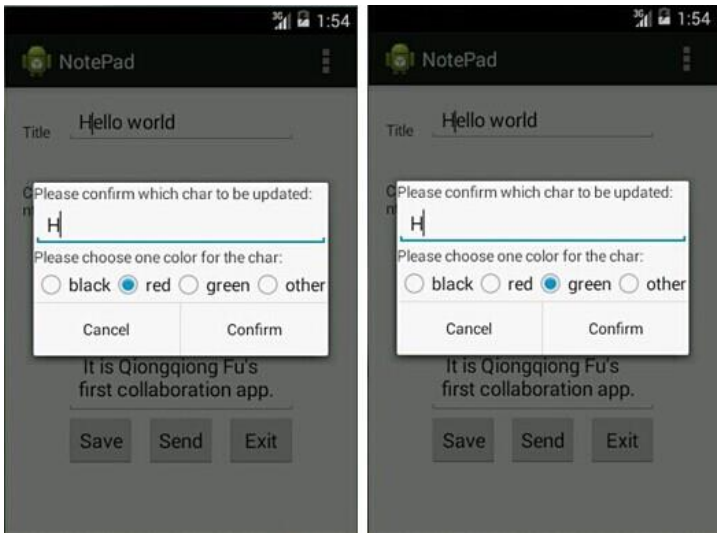

Figure 5. Concurrent Update Operations of Two Clients

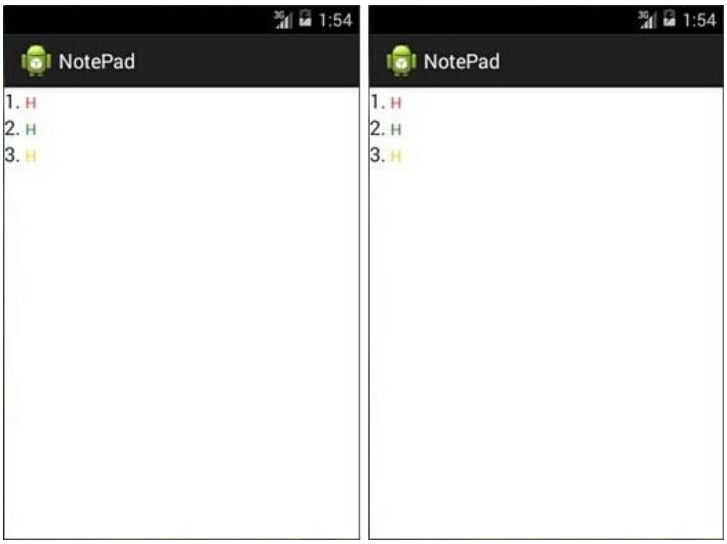

Figure 6. Different Versions Produced by Conflict Update Operations

6. Conclusions and Future Work

This paper proposes a conflict resolution strategy called ICCR strategy, mainly for consistency maintenance in the real-time collaborative editing environment. And in order to support all users' intentions of update operations, we define the combined effect and extend MVSD technique to adapt to the internal document model of AST. In this way the idea of Multi Version can be applied to the conflict solution of Update operations under the new document structure model.

In the approach the primitive operation formats should be redefined and a new document model presented need integrate MVSD to AST document structure in which a linear sequence should be added to save all versions for searching conveniently. Meanwhile, we study further on the document consistency maintenance technique. The advantages of the approach lie in reducing times of operation comparison, in other words, saving time of conflict resolution, and taking account of combined versions produced by the conflict relation to maintain all operation intentions.

The conflict solution approach has a few disadvantages, such as it is not suitable for all situations and may be not well used in different collaborative editors. Moreover, we will

make further research on the conflict resolution issues based on AST in other real-time collaborative systems. The current prototype system is functionally tested and just verified the availability of the strategy, but still in the debugging. One of our work is to continue improving the system, including more attributes selection of Update operation and better consistency maintenance of combination version, which is also the direction of our future work.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61202376 and No.61572325, 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] C. A. Ellis and S. J. Gibbs, “Concurrency control in groupware systems”, Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data, ACM, **(1998)**, pp. 59-68.
[2] D. Sun, S. Xia, C. Z. Sun and D. Chen, “Operational transformation for collaborative word processing”, Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work, **(2004)**, pp. 437-446.
[3] N. Gu, J. M. Yang and Q. W. Zhang, “Consistency maintenance based on the Mark & Retrace technique in groupware systems”, Proceedings of the 2005 international ACM SIGGROUP conference on Supporting Group Work, **(2005)**, pp. 264-273.
[4] D. Sun and C. Z. Sun, “Operation context and context-based operational transformation”, Proceedings of the 2006 20th anniversary conference on Computer Supported Cooperative Work, **(2006)**, pp. 279-288.
[5] L. P. Gao and T. Lu, “Maintaining semantic consistency of compound Undo operations replicated collaborative graphical editing environments”, Application Research of Computers, vol. 27, no. 9, **(2010)**, pp. 3434-3438.
[6] L. P. Gao, “Consistency maintenance of reference operations based on Add Space Transformation Strategy in replicated collaborative design environments”, Journal of Chinese Computer Systems, vol. 2, no. 4, **(2011)**, pp. 599-605.
[7] H. F. Shen and C. Z. Sun, “Achieving data consistency by contextualization in web-based collaborative applications”, ACM Transactions on Internet Technology, vol. 10, no. 4, **(2011)**, pp. 1-37.
[8] Agustina, C. Z. Sun and D. Xu, “Operational Transformation for dependency conflict resolution in real-time collaborative 3D design systems”, Proceedings of the 2012 ACM Conference on Computer Supported Cooperative Work, **(2012)**, pp. 715-728.
[9] L. P. Gao, S. S. Wang, S. X. Guo, Q. Q. Chen, Y. B. Zhang and T. Lu, “Solving two special dependency conflicts in real-time collaborative design systems”, Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design, **(2013)**, pp. 11-16.
[10] H. S. Gu, H. J. Hang, Q. Lv, and D. 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.
[11] H. H. Xia, T. Lu, B. Shao, G. Li, X. H. Ding and N. Gu, “A partial replication approach for anywhere anytime mobile commenting”, Proceedings of the 2014 ACM Conference on Computer Supported Cooperative Work, **(2014)**, pp. 530-541.

Authors

Qiongqiong Fu, she is a postgraduate student in University of Shanghai for Science and Technology. And she obtained her BSc degree in 2013 at Zhengzhou Institute of Aeronautical Industry Management. Her current research interests include CSCW, collaborative design and collaborative computing.

Liping Gao, she graduated from Fudan University, China with a PhD in Computer Science in 2009. 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.

Naixue Xiong, he 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 (about software engineering), and Japan Advanced Institute of Science and Technology (about dependable networks), respectively. Before attending Colorado Technical University, he worked in Wentworth Technology Institution, Georgia State University for many years. His research interests include Cloud Computing, Security and Dependability, Parallel and Distributed Computing, Networks, and Optimization Theory.