String Based New Operations –Find and Replace by New Operational Transformation Algorithms for Wide-Area Collaborative Applications

Santosh Kumawat
M.Tech Scholar
Poornima College of Engg.
Jaipur, Rajasthan, India

Ajay Khunteta
Asst Prof. Dept. of CS
Poornima College of Engg.
Jaipur, Rajasthan, India

ABSTRACT

Operational transformation (OT) is an established optimistic consistency control method in collaborative applications. This approach requires correct transformation functions. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive operations, insert and delete. In this paper we propose new algorithms that consider first time in history more new string operations that are Find and replace in addition to primitive operations like insert and delete. In history we are having first time algorithms for composite string operation - Find and replace. These algorithms for new Find and replace string operations also support earlier algorithms for primitive string operations-insert and delete. It also handles overlapping and splitting of operations when concurrent operations are transformed. These algorithms can be applied in a wide range of practical collaborative applications.

General Terms

Operational transformation (OT), optimistic consistency control method, Find-replace string operations.

Keywords

Operational transformation, transformation functions, string operations, Find and replace string operations, real-time cooperative editing systems.

1. INTRODUCTION

Operational Transformation (OT) [1] is an established optimistic consistency control method in collaborative applications network. Consistency control in this environment must not only guarantee convergence of replicated data, but also attempt to preserve intentions of operations. Fast local response and timely group awareness are accepted performance metrics in group editors. In general optimistic consistency control on linear data structures is done. In this context a family of optimistic concurrency control algorithms called OT has been well established. OT allows building real time groupware tools by correct transformation functions.

The objective of a collaborative environment [10] is to facilitate team working and, in particular, to enable a group of persons to manipulate shared objects, and modify them in a coherent manner. There are many collaborative activities, examples being simultaneous writing of a document by different authors, or cooperative design. In general, an object involved in a collaborative activity is submitted to concurrent accesses and real-time constraints. The real-time aspect necessitates every user seeing the effects of his own acts on the object immediately, and the effects of the acts of other users almost immediately. Consequently, the problem is to conciliate both real-time and consistency constraints, as the object may be modified concurrently by many users. To satisfy these requirements, it is necessary [16] [17] that concurrency control does not use a blocking protocol which could defer user actions.

Cooperative editing systems [12] are very useful groupware tools in the rapidly expanding areas of CSCW. They can be used to allow physically dispersed people to edit a shared textual document, to draw a shared graph structure, to record ideas during a brainstorming meeting, or to hold a design meeting. The goal of our research is to investigate, design, and implement cooperative editing systems with the following characteristics: (1) real-time: the response to local user actions is quick (ideally as quick as a single-user editor), and the latency for reflecting remote user actions is low (determined by external communication latency only); (2) distributed: cooperating users may reside on different machines connected by different communication networks with nondeterministic latency; (3) unconstrained: multiple users are allowed to concurrently and freely edit any part of the document at any time, in order to facilitate free and natural information flow among multiple users.

A plethora of OT algorithms have been proposed over the past two decades. Most of OT algorithms are developed under the framework of Sun et al [5], which includes an informal condition called "intention preservation". As a consequence, in general their correctness cannot be formally proved. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive operations, insert and delete. In real collaborative applications in which string based operations are common. The handling of string operations is very intricate, as confirmed in [5]. So there is an open challenge to handle more string operations.

To address the above challenges, this paper proposes a group of OT algorithm. It is based on the ABT framework [15, 17] which formalizes two correctness condition, causality and admissibility preservation. Causality preservation needed whenever an operation o is executed at a site, all operations that happen before o must have been executed at that site. Conceptually, admissibility requires that the execution of every operation not violate the relative position of effects produced by operations that have been executed so far. In general the ABT framework algorithms can be formally proved. The new proposed algorithms as per our knowledge first time in history are handling string operations like Find and replace in addition to primitive operations insert and delete and handles overlapping and splitting of operations when concurrent operations are transformed. These algorithms can be applied in a wide range of practical collaborative applications that require atomic string operations.

1.1 OT Functions- Inclusion and Exclusion Transformation

OT functions used in different OT systems may be named differently, but they can be classified into two categories.

One is **Inclusion Transformation** (or Forward Transformation): IT(O_a, O_b) or T(op_1, op_2), which transforms operation O_a against another operation O_b in such a way that the impact of O_b is effectively included and the other is **Exclusion Transformation** (or Backward Transformation) : ET (O_a, O_b) or T^{-1}(op_1, op_2), which transforms operation O_a against another operation O_b in such a way that the impact of O_b is effectively excluded.

2. Background and Related Work

To explain string operations and the basic ideas of OT, consider a scenario in which two users, A and B, collaboratively edit a shared document which includes a list of students of a class. The document is replicated at the two sites when the users discuss about it online. Suppose that the list is initially "Ram" and the first position of a string is zero. User A extends the list to "Shyam,Ram" by operation OA = insert(0, "Shyam,"). At the same time, user B extends the list to "Ram,Raman" by operation OB = insert(3,",Raman"). The two sites diverge before their results are merged. When A receives OB, if the operation were executed as-is, the wrong result "Shy,Ramanam,Ram" would yield in the list of A. The intuition of OT [18] is to transform remote operations to incorporate the effects of concurrent local operations that have been executed earlier. In this scenario, for example, A transforms OB into a form O'B such that O'B can be correctly executed in current state "Shyam,Ram" of site A. Considering the fact that A has inserted a string of six characters on the left side of the intended position of OB, we must shift the position of OB by six to the right, yielding O'B = insert(9, "Raman"). Execution of O'B in state "Shyam,Ram" results in the right list of "Shyam,Ram,Raman". On the other hand, when user B receives OA, the operation can be executed as-is in current state of B because the target position of OA is not affected by the execution of OB. This results in list "Shyam,Ram,Raman". Now the two sites converge.

The philosophy of OT is to avoid operation overwriting so as not to lose user interaction results.

System Model and Notations

A number of collaborating sites are there in a system. The shared data is replicated at all sites when a session starts. Local operations are executed immediately and for local responsiveness, each site submits operations only to its local replica. In the background, local operations are propagated to remote sites. The shared data is like a linear string of atomic characters. Objects are referred to by their positions in the string, starting from zero. It consider two only primitive operations, namely, insert(p, s) and delete(p, s), which insert and delete a string s at position p in the shared data, respectively. Any operation o has attributes like o.id is the unique id of the site that originally submits o; o.type is the operation type which is either insert or delete; o.pos is the position in the shared data at which o is applied; o.str is the target string which the operation inserts or deletes. For a operation o, o.pos is always defined relative to some specific state of the shared data.

In the following table1[1] general notations of operation are summarized.

To support string wise transformation, we need to introduce a few more notations. Given any string s, notation |s| is the number of characters in s. If 0 <= i<j <= |s|, notation s[i:j] returns a substring of s starting from position i to position j -1. If j is not specified, s [i:] returns a substring from i to the end. For example, let s="abc", then |s|=3 and s[0:2]="ab" and s[1:]="bc".

Notation	Brief Description
o.id	the id of site that originally generates o
o.type	the operation type of o, either ins or del
o.pos	the position of o relative to the data model
o.str	the string inserted or deleted by o
$o_1 \rightarrow o_2$	o_1 happens before o_2
$o_1 \parallel o_2$	o_1 and o_2 are concurrent
$o_1 \sqcup o_2$	o_1 and o_2 are contextual equivalent
$o_1 \mapsto o_2$	o_1 and o_2 are contextually serialized
$[o_1, o_2]$	an ordered list of two operations
$< o_1, o_2 >$	a 2-operation sequence in which $o_1 \mapsto o_2$
$\|L\|$	the number of objects in list/sequence L
$L_1 \cdot L_2$	a concatenated list/seq of two lists/seqs

Table 1. A summary of the main notations.

2.2 Literature Survey

Research on real-time group editors in the past decade has invented an innovative technique for consistency maintenance, called operational transformation In it presents an integrative review of the evolution of operational transformation techniques, with the goals of identifying the major issues, algorithms, achievements, and remaining challenges. First, it use a linear time interval based logical clock[6] for the same purpose of causality preservation as the more complex vector clock approach in existing operational transformation algorithms. This increases system scalability in terms of accommodating late comers in a dynamic collaboration environment. Second, it solve the dOPT puzzle with a one-dimensional history buffer (as compared to the N-dimensional storage in adOPTed [6]) and the time complexity the TIBOT control algorithm is O(n) (as compared to O(n^2) in GOTO [4][5]). Third, it solve the TP2 puzzle in a fully replicated architecture and without using ET (as compared to GOT [4]) or extra mechanisms (as compared to notification server in [7] and sequencer in [3]). The assumptions it made in it that lead to these results are reasonable in the target application domain of distributed interactive groupware systems.

The notification mechanism for a group editor consists of two algorithms AnyONE and AnyINE, and a propagation protocol SCOP.

In addition, it contribute a new operational transformation control algorithm SLOT for concurrency control, which is significantly simpler and more efficient than existing algorithms. Furthermore, it is free of state vectors, free of ET transformation functions, and free of the TP2 transformation condition.

It have contributed the theory of operation context and the COT (Context-based OT) algorithm. The theory of operation context is capable of capturing essential relationships and conditions for all types of operation in an OT system; it provides a new foundation for better understanding and resolving OT problems.

To ensure the convergence of the copies while respecting the user intention, it have proposed two new algorithms, called SOCT3 and SOCT4.

A novel state difference based transformation (SDT) approach which ensures convergence in the presence of arbitrary transformation paths.

It proposes an alternative framework, called admissibility-based transformation (ABT), that is theoretically based on formalized, provable correctness criteria and practically no longer requires transformation functions to work under all conditions. Compared to previous approaches, ABT simplifies the design and proofs of OT algorithms.

Next it is having ABTS for string handling. First, it is based on a recent theoretical framework with formal conditions such that its correctness can be proved. Secondly, it supports two string-based primitive operations and handles overlapping and splitting of operations. As a result, this algorithm can be applied in a wide range of practical collaborative applications.

3. Algorithms

In this section we are discussing our new proposed algorithms for replace-Find operations of strings.

Syntax for function replace is replace(st1,st2,start,end,occr), where st1 is existing string in document and st2 is new string by what replacement is to be done, start is starting position from where function start and end is the ending position where function stops. Also occr is the number that is equal to the number of occurances of st1 for what replacement will be done by function replace. Note st1 and st2 can be of different length. And Find(st1,start,end), it return the position of first occurance of string st1 from start. Here also start is equal to starting position of function and end is ending position of function and st1 is string what get find out by function in given document text . Note start always be equal to or less than end. If start not specified then by default it is starting of text and if end not specified then by default it is end of given text.

Example

To make clear basic idea of OT, consider a scenario in which two users, A and B, collaboratively edit a shared document which includes the text about facts in world. The document is replicated at the two sites when the user discuss about it online. Suppose the document is initially “SrashtiNirmataRam-Hae”, and the first position of string is zero. At site-1 User A modify the document by command replace(“Nirmata”, “Rachiyata”, 0, ,1) , the resulting document is “SrashtiRachiyataRam-Hae”. At the same time user B at site-2 extend the document to “SrashtiNirmataRam-HanumanHae” by insert(18,”Hanuman”). The two sites diverse before their results are merged.

When A receives O_B if the operation are executed as-is, the wrong result SrashtiRachiyataRaHanumanm- Hae” would yield in the list of A. The intutions of OT[20] is to transform remote operations to incorporate the effects of concurrent local operations that have been executed earlier. In this scenario, for example, A transforms O_B into a form O'_B such that O'_B can be correctly executed in current state “SrashtiRachiyataRam-Hae” of site A. Considering the fact that A has replaced a string of 7 characters by a string of 9 characters on the left side of the intended position of O_B, we must shift the position of O_B by 2 to the right, yielding O'_B = insert(20,”Hanuman”). Execution of O'_B in state “SrashtiRachiyataRam-Hae” results in the right list - “SrashtiRachiyataRam-HanumanHae”. On the other hand, when user B receives O_A, the operation can be executed as-is in current state of B because the target position of O_A is not affected by the execution of O_B. This results in list “SrashtiRachiyataRam-HanumanHae”. Now the two sites converge.

The philosophy of OT is to avoid operation overwriting so as not to lose user interaction results.

Basic IT Functions

In the most basic form, function IT(o_1,o_2) transforms a primitive operation o_1 with another primitive operation o_2 and outputs result o_1'. The output result can be a composite operation or atomic operation. According to [20], the precondition of IT(o_1, o_2) is o_1 U o_2 and the postcondition is $o_2 \rightarrow o_1'$.

Now we are discussing basic IT functions ITIR, ITRI, ITDR, ITRD for our replacement operation replace(st1,st2,start,end,occr). Here |st2|-|st1|=pc. Also Find(st1,start,end) returns position of first occurance of st1 from start, let it be ‘p’. In ITIR we are passing these pc as parameter . If operation o is replace then o.pos is equal to start parameter of replace function. If IT function return o' then o' is new start parameter for replace and Find functions where operation o is replace.

Algorithm 1:
ITIR(o_1, o_2, pc, start, end, st1): o_1'

$o_1' \leftarrow o_1$
for (i=0; i<occr and start<=end ; i++)
p $\leftarrow$ Find(st1,start,end)
if(p!= null) then
if pc>0 then
if p< o_1.pos then
o_1'.pos $\leftarrow$ o_1'.pos + | pc|
else if p=o_1.pos and o_2.id<o_1.id then

```
o1'.pos ← o1'.pos + | pc|
endif
else if pc=0 then
o1' ← o1'
else if pc<0 then
if p< o1.pos  then
if o1.pos  >= p + |pc| then
o1'.pos ← o1'.pos - | pc|
else
o1'.pos ← p
endif
endif
endif
endif
start = p
endfor
return o1'
```

Algorithm 1 transforms operation insertion o_1 with another operation that is replacement o_2 to incorporate the effects of o_2 in o_1. Let s be their common definition state. In it o_2 replace a substring st1 that is already in s with other new substring st2 and o_1 is to do insertion on s. In it both insertion and replacement are getting operated on same string s.

Here all IT algorithms takes as its parameter o_1, o_2, pc, start, end, st1. In it o_1, o_2 are two operations. Also pc is difference in |st1| and |st2|, that is (|st2|-|st1|) , where st1 is string existing in present document and st2 is new string by what we need to replace st1. Again start is starting position of function and end is ending position of function.

Algorithm 2:

ITRI(o_1, o_2, pc, start, end, st1): o_1'

```
o1' ← o1
p ← Find(st1,start,end)
if o2.pos <p  then
p' ← p+| o2.str |
else if p=o2.pos and o2.id<o1.id then
p' ← p+| o2.str |
endif
return p'
```

Algorithm 2 transforms operation insertion o_2 with another operation that is replacement o_1 to incorporate the effects of o_2 in o_1. Let s be their common definition state. In it o_1 replace a substring st1 that is already in s with other new substring st2 and o_2 is to do insertion on s. In it both insertion and replacement are getting operated on same string s.

Here p' what is getting returned by function ITRI is equal to new start position of replace and Find functions.

Algorithm 3:

ITDR(o_1, o_2, pc, start, end, st1): o_1'

```
o1' ← o1
for (i=0;i<occr and start<=end ;i++)
p ← Find(st1,start,end)
if(p!= null) then
if pc>0 then
if p<= o1.pos then
o1'.pos ← o1'.pos + | pc|
else if o1.pos<p<o1.pos+ |o1.str| then
oL ← oR ← o1
oL.str ← o1.str[0: p- o1.pos]
oR.pos ← p+ | pc|
oR.str ← o1.str[p- o1.pos: ]
o1'.sol ← [ oL, oR]
endif
else if pc<0 then
o2.pos ← p and |o2.str| ← |pc|
o1' ← MSITDD(o1, o2)
elseif pc=0 then
 o1' ← o1'
endif
endif
start = p
endfor
return o1'
```

Algorithm 3 transforms operation deletion o_1 with another operation that is replacement o_2 to incorporate the effects of o_2 in o_1. Let s be their common definition state. In it o_2 replace a substring st1 that is already in s with other new substring st2 and o_1 is to do deletion on s. In it both deletion and replacement are getting operated on same string s.

In Algorithm 3 and Algorithm 4 MSITDD is from our paper[21].

Algorithm 4:

ITRD (o_1, o_2, pc, start, end, st1): o_1'

```
o1' ← o1
p ← Find(st1,start,end)
if (pc>=0)
if o2.pos <p  then
if p >= o2.pos+| o2.str | then
p' ← p'- | o2.str |
else
p' ← o2.pos
endif
endif
else if (pc <0) then
o1.pos ← p and |o1.str| ← |pc|
p' ← MSITDD(o1, o2)
endif
return p'
```

Algorithm 4 transforms operation deletion o_2 with another operation that is replacement o_1 to incorporate the effects of o_2 in o_1. Let s be their common definition state. In it o_1 replace a substring st1 that is already in s with other new substring st2 and o_2 is to do deletion on s. In it both deletion and replacement are getting operated on same string s.

Here p' what is getting returned by ITRD is equal to new start position of replace and Find functions.

3.3 Basic Swap Functions

The basic swapping function for swapping two operations. Given two operations o_1 and o_2, where $o_1 \rightarrow o_2$, function swap(o_1, o_2) transposes them into o_1' and o_2' such that $o_2' \rightarrow o_1'$.The precondition of swap(o_1, o_2) is $o_1 \rightarrow o_2$.

Algorithm swapRI and swapRD is to swap replace operation on string with other primitive operations like insertion and deletion on strings.

Algorithm5:
swapRI(o_1,o_2,pc,start,end,st1):(o_2',o_1')

```
o1' ← o1 ; o2' ← o2
p ← Find(st1,start,end)
if(p > o2.pos) then
p' ← p-| o2.str |
o1'.pos ← p'
else if (p < o2.pos) then
for (i=0;i<occr and start<=end ;i++)
p ← Find(st1,start,end)
if(p!= null) then
if pc>=0 then
o2'.pos ← o2'.pos - | pc|
else if pc<0 then
o2'.pos ← o2'.pos + | pc|
endif
endif
start = p
endfor
endif
return(o2', o1')
end
```

Algorithm 5 swaps replace o_1 with an insertion o_2. Here it takes as its parameter o_1, o_2, pc, start, end, st1 . Also pc is difference in |st1| and |st2|, where st1 is string existing in present document and st2 is new string by what we need to replace st1,so pc=(|st2|-|st1|). Again start is starting position of function and end is ending position of function. It returns o_2', o_1' that are modified operations where o_2'.pos is new position for insertion and o_1'.pos is new starting position for modified replace and Find functions.

In Algorithm 6 swaps replace o_2 with a deletion o_1. Here parameters are like algorithm5. It returns o_2', o_1' that are modified operations where o_1'.pos is new position for deletion and o_2'.pos is new starting position for modified replace and Find functions. In these swapping algorithms overlapping of replace with insertion or deletion is not considered for reducing extra overhead.

Algorithm 6:
swapDR(o_1,o_2,pc,start,end,st1):(o_2', o_1')

```
o1' ← o1 ; o2' ← o2
p ← Find(st1,start,end)
if(p > o1.pos) then
p' ← p+| o1.str |
o2'.pos ← p'
else if (p < o1.pos) then
for (i=0;i<occr and start<=end ;i++)
p ← Find(st1,start,end)
if(p!= null) then
if pc>=0 then
o1'.pos ← o1'.pos + | pc|
else if pc<0 then
o1'.pos ← o1'.pos - | pc|
endif
endif
start = p
endfor
endif
return(o2', o1')
end
```

4. CONCLUSION

This paper contributes a group of new optimized generic operational transformation algorithms that first time in history consider new composite string operations, Find and replace. It also support existing primitive operations like insert and delete.

First time in history birth of composite string operations like Find and replace in multi user shared environment take place in this paper.

Most of OT algorithms are developed under the framework of Sun et al [5], which includes an informal condition called "intention preservation". As a consequence, in general their correctness cannot be formally proved. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive operations ,insert and delete.

To address the above challenges, this paper proposes a novel OT algorithm. It is based on the ABT framework [15, 17] which formalizes two correctness condition, causality and admissibility preservation. In general the ABT framework algorithms can be formally proved. The new proposed algorithms first time in history are handling string operations like Find and replace in addition to primitive operations insert and delete and handles overlapping and splitting of operations when concurrent operations are transformed in particular situations. These algorithms can be applied in a wide range of practical collaborative applications that require string operations.

This paper proposed new algorithm like swapRI, swapDR, ITRI ,ITIR, ITDR and ITRD for replace operation of strings , where swapRI and swapDR are basic swap functions and ITRI, ITIR, ITRD and ITDR are basic IT functions for string replace operations.

4.1 Future Work

There is a lot of efforts needed to preserve intention preservation and also to preserve semantic consistency and syntactic consistency. There is still scope to extend the support to other composite operations of string handling and char handling. Also it can support other better data structures also. A lot of work is done to reduce space complexity and time complexity. Still there is a scope to reduce space complexity and time complexity.

5. REFERENCES

[1] ABTS: A Transformation-Based Consistency Control Algorithm for Wide-Area Collaborative Applications Bin Shao , Du Li , Ning Gu . IEEE Paper published in 2009

[2] C. A. Ellis and S. J. Gibbs.Concurrency control in groupware systems. In Proceedings of the ACM SIGMOD'89 Conference on Management ofData, pages 399-407, Portland Oregon, 1989.

[3] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Transactions on Computer Systems, 20(3):239–282, Aug. 2002.

[4] N.Vidot, M.Cart, J.Ferrie,and M.Suleiman. Copies convergence in a distributed realtime collaborative environment. In ACM CSCW'00, pages 171–180, Dec. 2000.

[5] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality- preservation, and intention-preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63–108, Mar. 1998.

[6] H. Shen and C. Sun. Flexible notification for collaborative systems. In ACM CSCW'02, pages 77–86, Nov. 2002.

[7] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In ACM CSCW'98, pages 59–68, Dec. 1998.

[8] R. Li and D. Li. A new operational transformation framework for real-time group editors. IEEE Transactions on Parallel and Distributed Systems, 18(3):307-319, Mar. 2007.

[9] G. Oster, P. Urso, P. Molli, and A. Imine. Proving correctness of transformation functions in collaborative editing systems. Technical Report 5795, INRIA, Dec. 2005.

[10] M. Suleiman, M. Cart, and J. Ferrie. Concurrent operations in a distributed and mobile collaborative environment. In IEEE ICDE '98 International Conference on Data Engineering, pages 36-45, Feb. 1998.

[11] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proceedings of the ACM Conference on Computer- Supported Cooperative Work, pages 59-68, Dec. 1998.

[12] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality- preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63108, Mar. 1998.

[13] D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. IEEE Transactions on Parallel and Distributed Systems, 20(10):1454-1470,2009.

[14] Decouchant D., Quint V., Vatton I.; "L'édition Coopérative de documents avec riffon," Colloque IHM'92, Paris, Décembre 1992.

[15] R. Li and D. Li. Commutativity-based concurrency control in groupware. In Proceedings of the First IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '05), San Jose, CA, Dec. 2005.

[16] Ellis C.A., Gibbs S.J., Rein G.L.; "Groupware: Some issues and experiences," Commun. ACM,vol.34, n°. 1, pp. 39-59, January 1991.

[17] Greenberg S., Marwood D.; "Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface," in Proc. ACM Int. Conf. on Computer Supported Cooperative Work, Canada, October 1994, pp. 207-217.

[18] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In Proceedings of the ACM SIGMOD'89 Conference on Management ofData, pages 399-407, Portland Oregon, 1989.

[19] D. Li and R. Li. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work: The Journal of Collaborative Computing, Aug. 2009. Accepted.

[20] Prakash A., Shim H. S.; "DistView: Support for Building Efficient Collaborative Applications using Replicated Object," in Proc. ACM Int. Conf. on Computer Supported Cooperative Work, October 1994, pp. 153-164.

[21] Santosh Kumawat and Ajay Khunteta
A Transformation based New Algorithm for Transforming Deletions in String Wise Operations for Wide-Area Collaborative Applications ,International Journal of Computer Applications (0975 – 8887) Volume 4– No.12, Aug