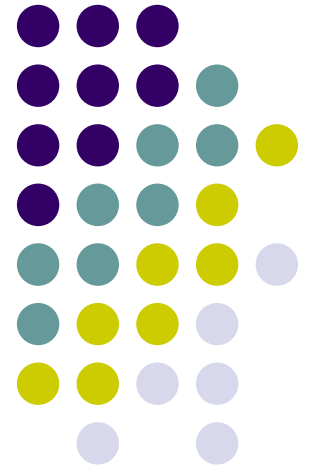


A Deadlock-free Multi-granular, Hierarchical Locking Scheme for Real-time Collaborative Editing

Jon A. Preston
Sushil K. Prasad





Agenda

- Motivation
- Related Work in Collaborative Editing Systems
- The Tree Data Structure
- Properties and Algorithms
 - InsertUser (with demotion)
 - RemoveUser (with promotion)
- Conclusions and Future Work



Motivation

- Collaborative Editing Systems (CES)
concurrency control falls into two approaches
 - Optimistic
 - Pessimistic
- Our scheme is a hybrid of these approaches
 - Avoid transformation/merge problems of optimistic
 - Provide high level of concurrency/access
- Locking is automatic (transparent to user)



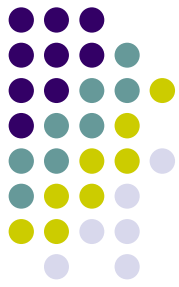
Related Work

- Increase parallelism in software engineering [10] and avoid bottleneck of traditional CMS [3]
- Avoids explicit turn-taking or system-specific approach [1], [11], and [12]
- Transparent to user [4], [5], [6], and [13]
- Avoids the merge problem [8]
- Provides multi-granular, hierarchical locking [7]
- Flexible, open-systems, Web-services based approach [2], [14] and [15]



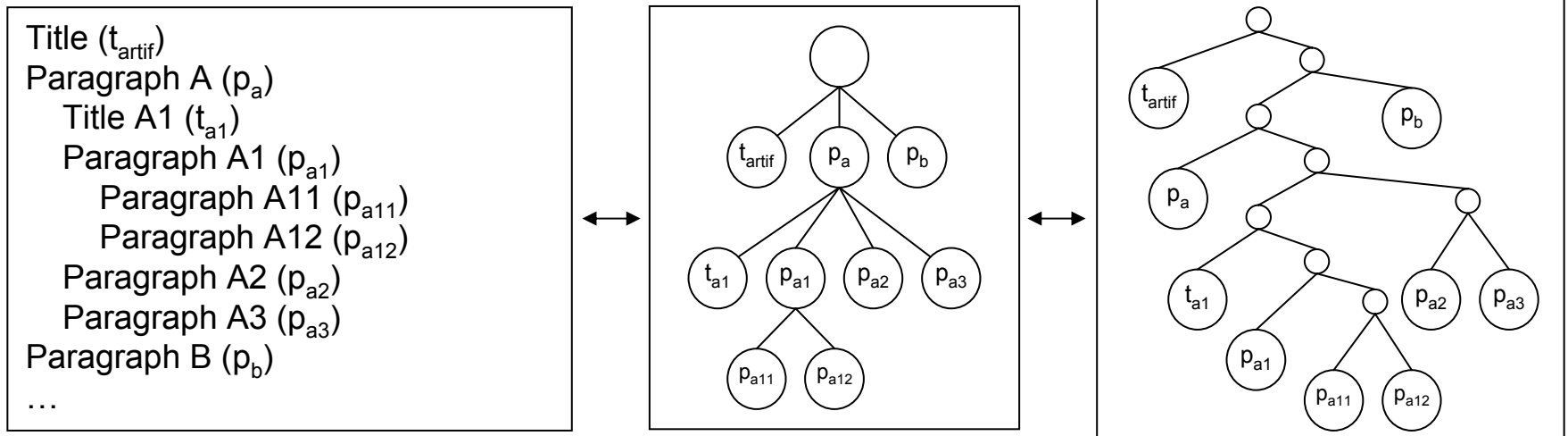
The Tree Structure

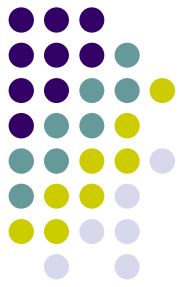
- Structure denotes sections and sub-sections
- Only leaf nodes contain satellite data
- All access requests from client will reference a specific leaf node (as the client knows only of document content, not tree)
- Nodes store ownership/lock information
 - If a section of the document is owned by a user u , then all subsections (i.e. child nodes in the tree) are also owned by the same user u .
 - Similarly, if a section of the document is not owned by any user, then all subsections (i.e. child nodes in the tree) are also not owned by any.



Mappings

- It is possible to map a document to a tree and from this tree to a binary tree
- Without loss of generality, our algorithms work on this binary tree





Maintaining the Largest Sub-tree

- It is desirable to own the largest sub-tree possible
 - Can keep cached changes local
 - Reduces network messaging
- Only give up an owned portion of the tree when another user enters and injects contention (demotion)



Enter/Leave Document

- To clarify, we assume
 - Users can enter and leave the document at any time
 - To move from one section of the document to another, leave current section and enter new section (leave and re-enter document)
 - This is seamless/transparent to the user (i.e. the system removes and re-adds the user without informing the user)
- Issues of performance and cache optimization are still open for our research

Concurrent Reading/Sharing

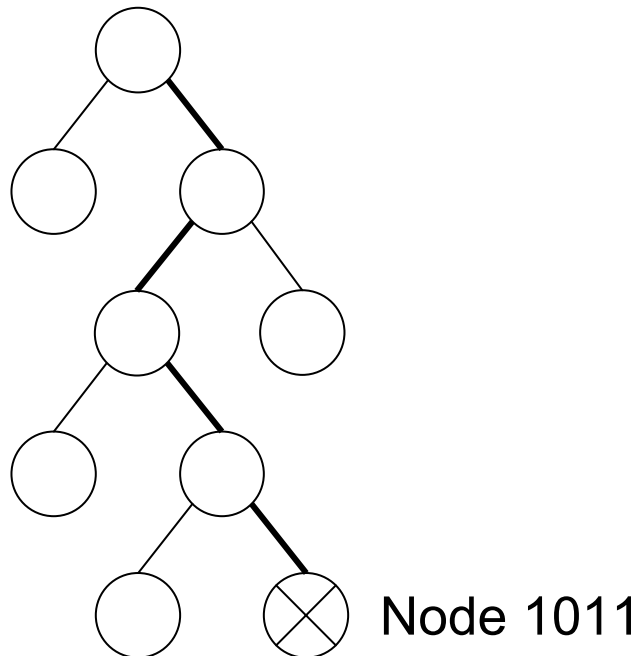


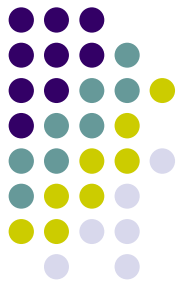
- As with other CES/CSCW systems, sharing is permissible
- This paper/presentation addresses exclusive write access
- Certainly, concurrent read access is permissible
 - Updating the view of read-only clients upon a change would occur as it does in other systems



Path Finding

- Since each node is uniquely identified, it is always possible to know the path to a node in the tree
 - Left = 0
 - Right = 1





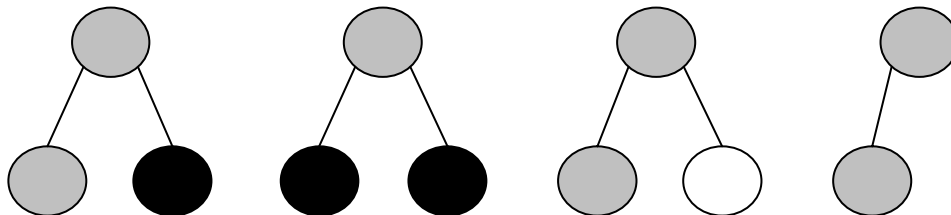
The Node Structure

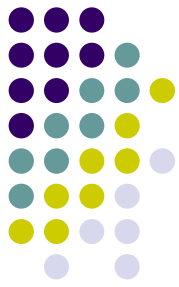
- The node structure contains
 - Left and right children
 - Sibling
 - Color (white, black, grey)
 - Owner (if black, what user owns this sub-tree)
 - Original Request (what sub-tree node was requested to cause ownership of this node)



Coloring

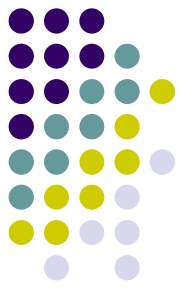
- White
 - Unowned
 - All sub-tree nodes are white logically
- Black
 - Owned
 - All sub-tree nodes are black logically
- Grey
 - Grey-color ≥ 2
 - There must exist ≥ 2 black nodes in the sub-trees
 - Possible grey configurations:





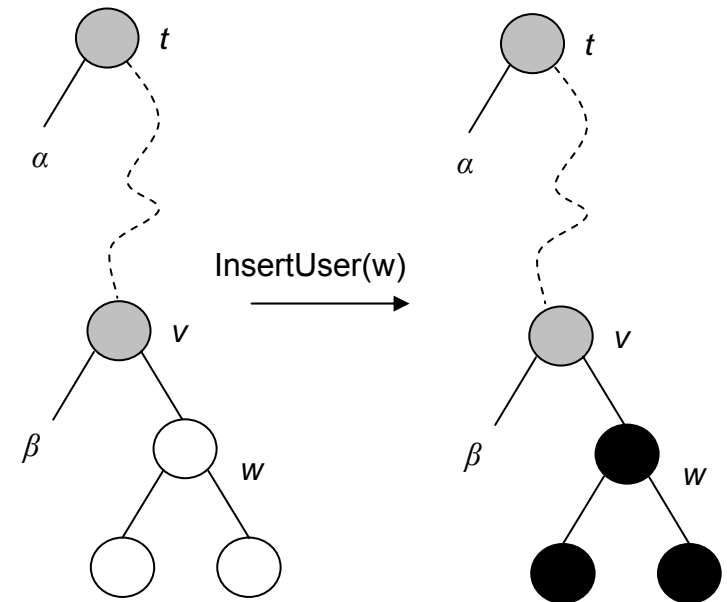
Algorithms

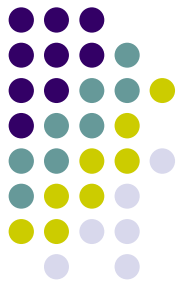
- Deadlock-free
- Always traverse top to bottom [9]
- Handshake lock
 - Acquire node
 - Acquire node's child
 - Release node
- Insert User
- Remove User



InsertUser

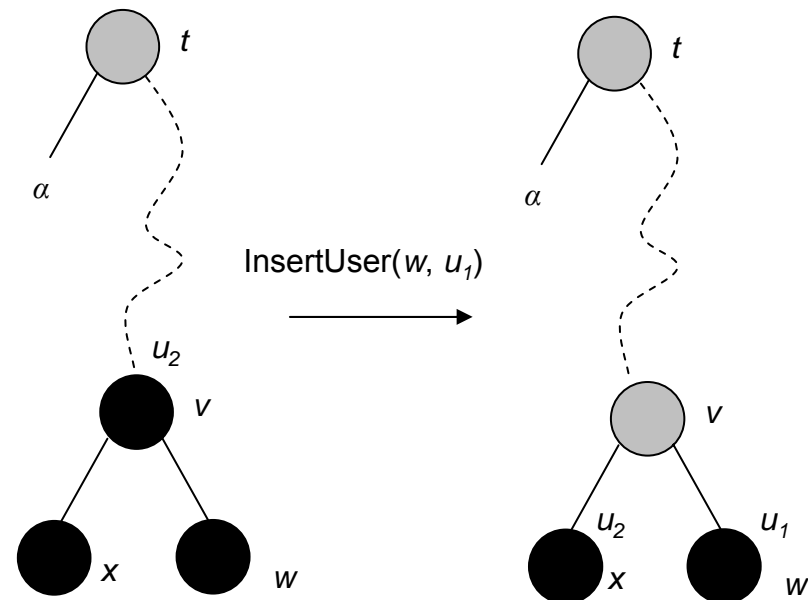
- User u_i requests lock on node w
- All nodes in the path from t to v must be grey
 - If white, lock higher
 - If black, lock fails
- Increase grey-color in path
- If contention on w
 - Demote w
 - If w is leaf “undo” inflated grey color





InsertUser - Demotion

- At times, demotion is necessary
 - When the top-to-bottom access encounters a black node
 - Push the ownership “down” (if a leaf, fail)
 - Requires that we maintain originalRequest (i.e., why is this node black)
 - Paint node grey
 - Recursion if we push along the same path to requested node



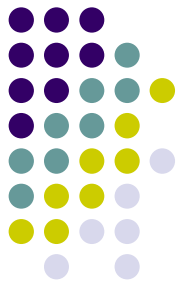
InsertUser



```
InsertUser(w, ui)  
  if w.owner ≠ ui  
    RecurseInsert(root, w, ui)
```

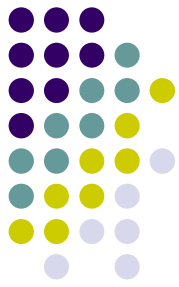
```
RecurseInsert(n, w, ui)  
  if n.color = white // ensures we always acquire largest lock  
    then SetOwner(n, ui, w)  
  else if n is a leaf node // a leaf node but not white, so failed insert (i.e. we can't demote a leaf)  
    then RecurseRemove(root, w, ui) // undo the false insert's effect on greyColors  
         return failure  
  else if n.color = grey // keep looking down  
    then n.greyColor = n.greyColor + 1  
         RecurseInsert(NextInPath(n, w), w, ui)  
  else // color of node in path to w is black, so we must demote it  
    b = NextInPath(n, w)  
    a = NextInPath(n, n.originalRequest)  
    SetOwner (a, n.owner, n.originalRequest) // demote n to a  
    n.color = grey  
    n.greyColor = 2  
    if a ≠ b  
      then SetOwner (b, ui, w) // the nodes are in separate paths  
      else RecurseInsert(b, w, ui) // the nodes are in the same path
```

```
SetOwner(w, ui, r)  
  w.color = black  
  w.owner = ui  
  w.originalRequest = r
```



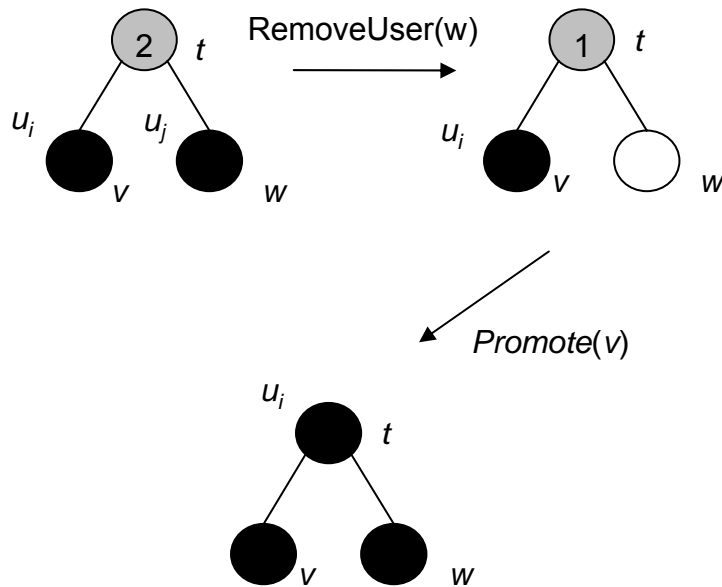
RemoveUser

- User u_i releases lock on node w
- All nodes in the path from t to v must be grey
 - If white, lock higher
 - If black, release higher
- Decrease grey-color in path
- Promote if possible when grey-color goes from 2 to 1

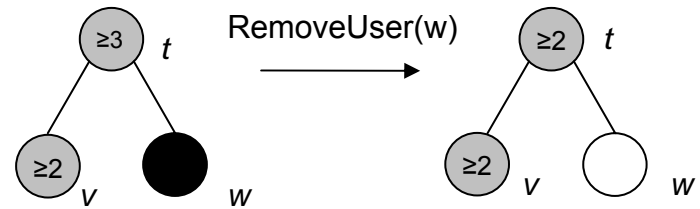


RemoveUser Cases

- With promotion



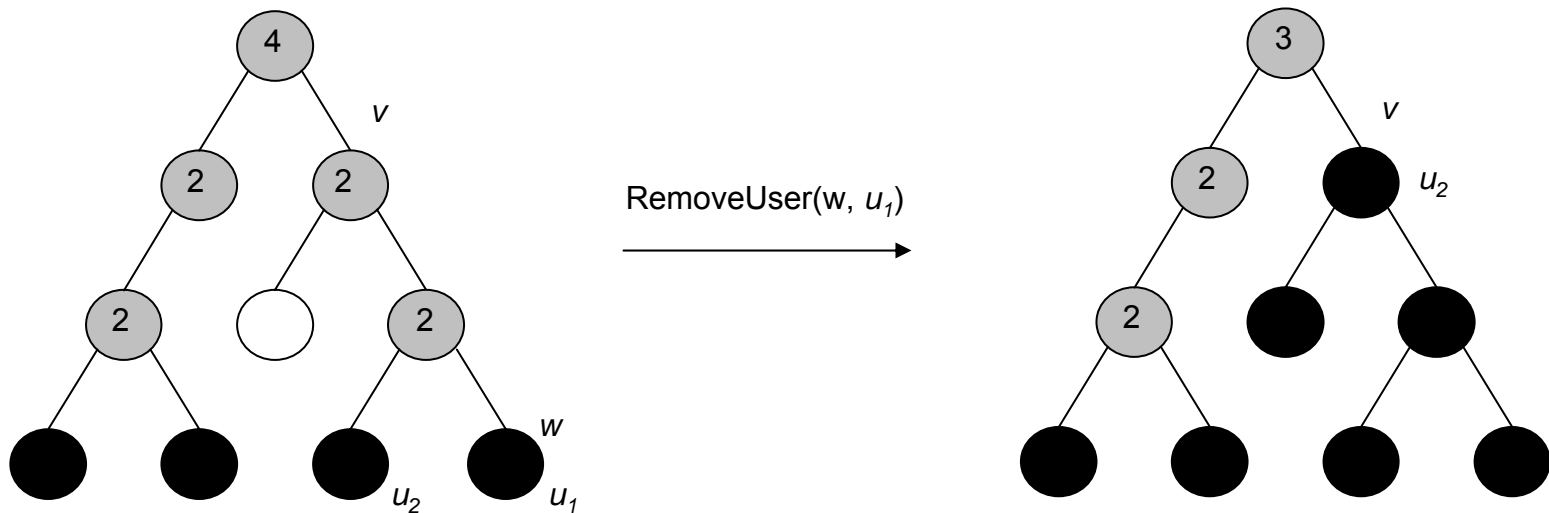
- Without promotion

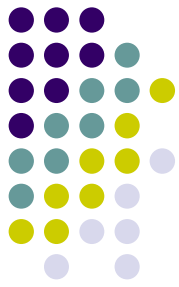




RemoveUser Promotion

When grey-color reduces from 2 to 1, promote





RemoveUser

RemoveUser(w, u_i)

if $w.owner = u_i$
 then RecurseRemove($root, w, u_i$)

RecurseRemove(n, w, u_i)

if $n.color = black$ and $n.owner = u_i$
 then ReleaseOwner(n)

else if $n.color = grey$ // keep looking down

then $n.greyColor = n.greyColor - 1$

if $n.greyColor = 1$ and $w.sibling.color = black$ // sibling promotion priority

then SetOwner($n, w.sibling.owner, w.sibling$)

else if $n.greyColor = 1$ and $w.color = black$ // w must be all that remains

then SetOwner($n, w.owner, w$)

else if $n.greyColor = 0$ and // paint n white as w and w.sibling are white

then ReleaseOwner(n)

else RecurseRemove(NextInPath(n, w), w, u_i)

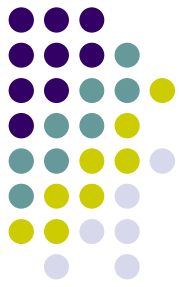
ReleaseOwner(w)

$w.color = white$

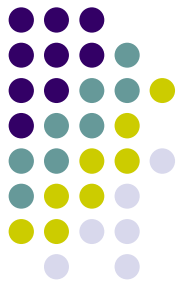
$w.owner = NIL$

$w.originalRequest = NIL$

RemoveUser Special Cases

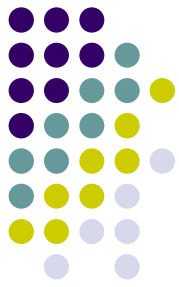


- If an insertion failed, then we invoke a RemoveUser to reduce the artificially-inflated grey-count along the path from root to desired node
- It is possible that legitimate removals occurred between the time of the failed insertion and the concomitant removal (to compensate for the failed insertion)
- Consequently, the grey-count of a node can fall to 1
 - Promote sibling or the node associated with the failed insert
- Additionally, the grey-count of a node could fall to 0
 - Promote nothing (everything is now removed below this node)
 - Paint this node white



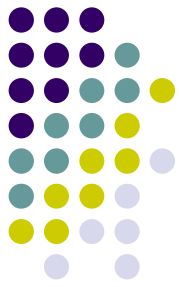
Analysis

- RemoveUser makes a single pass of the tree from top to bottom
 - $O(h)$ where h is equal to the height of the tree.
- InsertUser can fail and require an “undo” of the artificially-inflated grey-colors (via a call to RemoveUser)
 - Thus at most two passes of the tree from top to bottom
 - Also $O(h)$ where h is equal to the height of the tree
- Promotion occurs in $O(1)$ because we know the sibling is to be promoted (or in the special cases, the failed insert node is promoted or nothing is promoted)



Conclusion

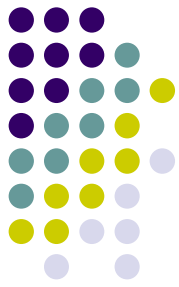
- Contributes hybrid concurrency policy to the field of CSCW/CES
 - Deadlock-free
 - Multi-granular, hierarchical locking
 - Maximizes sub-tree owned
 - Minimizes messaging/communication
 - Insert and remove algorithms efficient – $O(h)$
 - Promotion is efficient – $O(1)$



Future Work

- Address tree modification
 - Insert – obtain lock on parent, then insert below
 - Delete – obtain lock on node and remove
 - Split – obtain lock on node and create children
 - Join – obtain lock on both nodes and combine
 - Move – this is a delete + insert
- Simulation – in progress
- Further define cache policies for concurrent readers
- Place into larger framework of open-system, Web services based architecture and simulate editing

References



1. Netbeans Collaboration Project. <http://collab.netbeans.org>. Sun Microsystems. Viewed August, 2005.
2. V. Bharadwaj and Y.V. R. Reddy. A Framework to Support Collaboration in Heterogeneous Environments. SIGGROUP Bulletin. Vol 24, No 3. December 2003. pp. 103-116.
3. M. Chu-Carroll, J. Wright, and D. Shields. Supporting Aggregation in Fine Grained Software Configuration Management. SIGSOFT 2002/FSE-10. Charleston, SC. November 2002.
4. M. Chu-Carroll and S. Sprenkle. Coven: Brewing Better Collaboration through Software Configuration Management. SIGSOFT 2000. San Diego, CA. November 2000.
5. C. Ellis and S. Gibbs. Concurrency Control in Groupware Systems. In Proceedings of ACM SIGMOD Conference on Management of Data, pages 399-407. ACM Press, May 1989.
6. S. Greenberg and D. Marwood. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In Proceedings ACM Conferences on Computer Supported Cooperative Work, pages 207-217. ACM Press, Nov. 1994.
7. B. Magnusson, U. Asklund, S. Minör. Fine-Grain Revision Control for Collaborative Development. Proceedings of ACM SIGSOFT'93. Los Angeles, CA. December 1993.
8. J. Munson and P. Dewan. A Concurrency Control Framework for Collaborative Systems. Proceedings Computer Supported Cooperative Work, pages 278-287. ACM Press, Nov. 1996.
9. V. N. Rao and V. Kumar. Concurrent Access of Priority Queues. IEEE Transactions on Computers. Vol 37, No 12. pp. 1657-1665. 1988.
10. C. O'Reilly. A Weakly Constrained Approach to Software Change Coordination. In Proceedings of the 26th International Conference on Software Engineering. 2004.
11. H. Shen, C. T. Cheong, and C. Sun. CoStarOffice: Toward a Flexible Platform-independent Collaborative Office System. IWCES'04.
12. D. Sun, S. Xia, C. Sun, and D. Chen. Operational Transformation for Collaborative Word Processing. CSCW'04. Chicago, IL. November 2004. CHI Letters. Vol 6, Issue 3. pp. 437-446.
13. A. van der Hoek, D. Redmiles, P. Dourish, A. Sarma, R. S. Filho, and C. de Souza. Continuous Coordination: A New Paradigm for Collaborative Software Engineering Tools. Workshop on Directions in Software Engineering Environments. ICSE'04. 2004.
14. Y. Yang and D. Li. Separating Data and Control: Support for Adaptable Consistency Protocols in Collaborative Systems. Proceedings of CSCW'04. Chicago, IL. November 2004. CHI Letters. Vol 6, Issue 3. pp. 11-20.
15. M. Younas and R. Iqbal. Developing Collaborative Editing Applications using Web Services. IWCES'03. 2003.
16. B. P. Zeigler and H.S. Sarjoughian. Introduction to DEVS Modeling & Simulation with JAVA: Developing Component-based Simulation Models. Technical Document, University of Arizona. 2003.