ISBN: 978-972-8924-97-3 © 2009 IADIS

COLLABORATIVE PEER TO PEER EDITION: AVOIDING CONFLICTS IS BETTER THAN SOLVING CONFLICTS

Martin Stéphane and Lugiez Denis
Laboratoire d'informatique fondamentale
39 rue Frédéric joliot-curie.
13013 Marseille France.

ABSTRACT

Collaborative edition is achieved by distinct sites that work independently on (a copy of) a shared document. Conflicts may arise during this process and must be solved by the collaborative editor. In pure Peer to Peer collaborative editing, no centralization nor locks nor time-stamps are used which make conflict resolution difficult. We propose an algorithm which relies on the notion or semantics dependence and avoids the need of any integration transformation to solve conflicts. Furthermore, it doesn't use any history file recording operations performed since the beginning of the edition process. We define editing operations for semi-structured documents i.e. XML-like trees, that are enriched with informations derived for free from the editing process. Then we define the semantics dependence relation required by the algorithm and we present preliminary results obtained by a prototype implementation.

KEYWORDS

Collaborative Editing, XML, Peer to Peer, Optimistic Reconciliation, Consistence Maintenance, Distributed Computing

1. INTRODUCTION

Collaborative edition becomes more and more popular (writing article with SVN, setting appointments with doodle, Wikipedia articles,...) and it is achieved by distinct sites that work independently on (a copy of) a shared document. Several systems have been designed to achieved this task but most of them use centralization and locks or weak centralization via time-stamps. A alternative approach is the Peer to Peer approach -P2P- where new sites can freely join the process and no central site is required to coordinate the work. This solution is more secure and scalable since the lack of central site protects from failures and allows for a huge number of participants. In this paper we focus on editing semi-structured documents, called XML trees from now on, using the basic editing operations *add*, *delete* for edges or *changing labels* in the document. Since the process is concurrent, conflicts can occur: for instance a site S_1changes the label of an edge by *summary* when another site S_2 want to relabel *Introduction* by *Abstract* and each change is propagated to the other site. A major challenge in this approach is the design of integration transformation that issues convergence of the editing process. Executing the corresponding operations leads to an incoherent state since the sites nor longer have identical copies of the shared document. In the optimistic P2P approach, each operation is accounting for and conflicts are solved by replacing the execution of an operation op_2 performed concurrently with op_1 by $IT(op_2,op_1)$ where *IT* is an integration transformation defined on the set of operations. This transformation computes the effect of the execution of op_1 on op_2, i.e. the *dependence of* op_2 *from* op_1.

We prensent an editing algorithm which relies on semantics dependances that eliminate conflicts, avoiding the divergences. Section 2 discusses the current approaches to collaborative editing, and we present our editing algorithm in section 3. The data structure used for XML trees is described in section 4 and our first results are given in section 5. Missing proofs can be found in the full research report. (http://www.lif.univ-mrs.fr/~martin/research)

2. RELATED WORKS

Many collaborative edition framework have been proposed, and we discuss only the most prominent ones. In the word case, the transformations proposed in [12, 3, 7, 10, 13] turned out to be non-convergent, see [6] for counter-examples. None of these transformations satisfy both properties *TP1* (a local confluence property) and *TP2* (integration stability) that are sufficient to ensure convergence [12] and, no convergent algorithm based on the integration transformation is known for words. For XML trees, algorithms and operations have been proposed (like in [1]), but they have the same problem as in the word case or use time-stamps (see [11]) i.e. are not true P2P.

IceCube (see [9]) is a operational-based generic approach for reconciliating divergent copies. Conflicts are solved on a selected site using optimization techniques relying on semantic static constraints (generated by document rules) and dynamic (generated by the current state of the document). This NP-hard problem and this approach is not a true P2P solution (each conflict is solved by one site). The *Harmony* project [8] is a state-based generic framework for merging two divergent copies of documents. These documents are tree-like data structure similar to the unordered trees that we discuss in section 4. The synchronization process exploits XML-schema information and is proved terminating and convergent for two sites.

So6 [11] is a generic framework based on the *Soct4* algorithm which requires the local confluence property (*TP1*). It relies on continuous global order information delivered by a times-tamper, which is not pure P2P since it relies on a central server for delivering these time-stamps.

Goto (Sun et al. [14]), *Adopted* (Ressel et al. [12]) and *SDT* (Du Li and Rui Li [2]) rely on the local confluence property (*TP1*) and on the integration stability property (*TP2*) to guarantee convergence. A main issue is to ensure that operation integration takes place in the same context and return the same result and each algorithm has its own solution. For instance, *Goto* uses a forward (*IT*) and a backward (*ET*) transformation to reorder the history (record of all operations performed). *Adopted* computes the sequence of integrations as a path in a multi-dimensional cube. The main drawback of these approach is that it is hard to design set of useful operations and integration transformations that satisfy both *TP1* and *TP2*. For instance, no such set exists in the word case nor for linearly ordered structures.

The set of operations given by Davis and Sun provides operations on trees for the Grove editor [1], but this set doesn't satisfy the local confluence property *TP1*. Therefore, there is little hope to get a convergent editing process. *OpTree* [5, 4] present a framework for editing trees and graphical documents using *Opt* or the *Soct2*, and relies extensively on history files containing all operations performed on the date. The complexity is at least quadratic in the size of the log file and no formal proof of correctness is given.

A main problem of all these solutions -even when convergence is guaranteed- is that they rely on computations of history files that record all operations performed which can become quite expensive.

3. CONFLICT-FREE SOLUTION

We propose a generic schema for collaborative editing which avoid the pitfalls of previous works by avoiding the need to solve conflicts. First we give an abstract presentation of this editing process and of the properties required to ensure its correctness, then we show how it works for XML trees. Each site participating to the editing process executes the same algorithm (given in Figure 1) and performs operations on his copy of the shared documents. Operations belong to a set of operations *Op*, and we assume that there is a partial ordering f_s (i.e. an irreflexive, antisymmetric, transitive relation) on operations. This ordering expresses causal dependencies of the editing process: $op_1 \mathsf{f}_s op_2$ iff op_1 depends from op_2 (for instance op_1 creates an edge and op_2 relabels this edge). We write $op_1 \|_s op_2$ iff $op_1 \not\mathsf{f}_s op_2$ and $op_2 \not\mathsf{f}_s op_1$ when op_1 and op_2 are independent.

We show how to compute this relation for XML trees in section 4. A sequence of operations is denoted by $[op_1,...,op_n]$ and the result of applying op_1, followed by $op_2,\ldots,op_n$ to the document t is denoted by $[op_1,...,op_n](t)$. The set of operations (Op,f_s) is independent iff $\forall op,op' \in Op$ $\forall t, op \|_s op' \Rightarrow [op,op'](t) = [op',op](t)$.

Proposition 1 Let (Op,f_s) an independent set of operations. Let $[op_1,...,op_n]$ be a valid sequence of operations in Op and let σ be a substitution compliant with f_s. Then $[op_1,...,op_n](t) = [op_{\sigma(1)},...,op_{\sigma(n)}](t)$

3.1 The (Fast Collaborative Editing) FCedit Algorithm

Using a *semantic dependence* of operations allows to reduce the integration transformation to a trivial one: $IT(op_2,op_1)=op_2$ which is possible since we enrich the data structure by adding informations coming for free from the editing process.This yield an important property: each edge is uniquely labelled. Furthermore labels also record the level of dependence of the sites that created or modified them. These properties allow to get a simple convergent editing algorithm which doesn't require any history file recording all operations done since the beginning of the edition process. Since a word can be encoded as a tree, this algorithm also solves the word case, at the price of a more complex representation. These ideas have been implemented in a prototype that proved that the editing is done efficiently and that the process is scalable

```
1 INITIALIZE():
2     ∀i, SReceived[i] = 0          // State Vector of
      received operations
3     (SiteId, Obj, OpCount, WaitingList) = (n, o, 1, {})

1 GENERATEREQUEST(op):      // User emit operation
2     Let r = (op, SiteId : OpCount)
3     if isExecutable(r) then
4         OpCount = OpCount + 1
5         t = op(t)                 // Apply operation
6         broadCast r to other participant.

1 EXECUTE(r):                 // Execute a request r
2     r = (op, #Site : #Op)
3     StateReceived[#Site] = #Op    // Update state
      vector
4     WaitingList = WaitingList/r  // remove r from
      waiting list
5     t = op(t)                 // Applies a operation

1 ISEXECUTABLE(r): // Check request r is executable
2     Let r = (op, #Site : #Op)
       // Check that the previous operation on same
      site has been executed
3     if SReceived[#Site] ≠ #Op − 1 then
4         return false
      // Check all dependencies was executed
5     for nSite : cSite ∈ dependancesOf(r) do
6         if SReceived[nSite] < cSite then
7             return false
8     return true

1 RECEIVE(r):                 // treats request r
2     WaitingList = WaitingList ∪ r
3     forall r ∈ WaitingList|isExecutable(r) do
4         execute(r).       // execute all executable
          request
```

Figure 1. FCEdit algorithm

The procedures (except *Main()*) of the generic distributed algorithm *FCedit* are given in Figure 1. Each site has an unique identification stored in *SiteId*, a operation numbering stored in *Opcount*, a copy of the document and a list *WaitingList* of requests awaiting to be treated. The function *dependenceOf(r)* returns the pairs with a site identifier, some operation count, such that depends from an operation issued from site with operation count. This function is defined simultaneously with the data structure, set of operations and dependence relation, see section 4.3 for the definition used for XML-trees. The *Main()* procedure calls *Initialize()* and enters a loop which terminates when the editing process stops. In the loop, the algorithm choose non-deterministically to set the variable *op* to some user's input and to execute.

Proposition 2: The algorithm FCedit is convergent if the set of operations is independent.

4. CONFLICT FREE OPERATIONS FOR XML TREES

The basics editing operations on trees are *insertion, deletion or relabeling* of a node. Actually, since we consider edge labelled trees instead of node labelled trees, insertion and deletion are performed on edges instead of nodes. Trees are unordered trees enriched by a labelling tag allow to reorder son of the edge.i.e. : that's correspond to XML document. The information stored in nodes (or edges in our case) are described as a word on some finite alphabet. To get a independent set of operations containing relabeling, we use a more complex labeling that we describe now.

The set of identifiers *ID***.** Each site is uniquely designated by its identifier which is a natural number (IP numbers could be used as well). The set of identifier is the set *ID* of pairs *(SiteNumber:NbOpns)* where $NbOpns \in Nat$ is denotes some numbering of operations on site identified by *NbOpns*. That give a uniquenesse identifier.

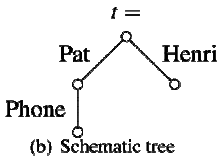
The set of labels $\mathcal{L}$. A label is a pair *(l,id)* where $id \in ID$ and *l* is a triple *(lab,id',v)* with $lab \in \Sigma_L^*$ (the "real" label) with Σ_L a finite alphabet, $id \in ID, v \in Nat$ (*v* is version number of Label).

Trees are unordered i.e. $\{n_1(t_1),...,n_p(t_p)\}$ is identified with $\{n_{\sigma(1)}(t_{\sigma(1)}),...,n_{\sigma(p)}(t_{\sigma(p)})\}$ for σ any permutation of $\{1,...,n\}$.

Example: We give an XML document and a its tree structure and its representation according to editing process.

```
1  <?xml version ...?>
2  <Pat>
3    <phone/>
4  </Pat>
5  <Henri>
6  </Henri>
```

(a) XML Document



(b) Schematic tree

$$t = \left\{ \begin{array}{l} ((Pat,(1:3),2)(\{((Phone,(3:4),5),(2:1))(\{\})) \\ ((Henri,(2:3),1),(2:2))(\{\}) \end{array} \right\}$$

(c) Data structure

Figure 2. Unordered tree representation

4.1 Editing Operations

Adding an edge. The operation $Add(id,id_p)$ if $id_p \neq id$ adds an edge labelled by *(l,id)* with *l=(NoValue,id,0)* under edge labelled by *(…,id)*. When id_p doesn't occur, the tree is not modified.

Deleting a subtree. The operation $Del(id)$ deletes the whole subtree corresponding to the unique edge labelled by *(…,id)*(including this edge). When id doesn't occur, the tree is not modified.

Changing a label. The operation $ChLab(id,id_{op},v_{new},lbl_{new})$ with $id,id_{op} \in ID$, $dep \in Nat$, $lbl_{new} \in \Sigma^*$ replaces the label (l_e,id) of the edge identified by *(…,id)* by (l_e',id) where $l_e=(lab_{old},id_{old},v_{old})$ and $l_e'=(lbl_{new},id_{op},v_{new})$. When $v_{old}>v_{new}$ or ($v_{old}=v_{new}$.and $id_{op}>id_{old}$) or id doesn't occur, the tree is not modified. The version number is incremented when *ChLab* is generated.

We write *Op* this operation set {*Add*, *Del* and *ChLab*}.

4.2 Semantic Dependence

- $Add(id,id_p)\ f_s\ Del(id)$: an edge can be deleted only if it has been created.
- $Add(id_p,id_p')\ f_s\ Add(id,id_p)$: adding edge under edge requires that edge has been created.
- $Add(id,id_p)\ f_s\ ChLab(id,id_{op},v,lbl)$: changing the labeling of edge requires that edge has been created.

4.3 Ordering

We associate a unique word (using priority tag and uniqueness identifier) on an alphabet Σ, and we use a lexicographic ordering $<<$ to order edges. The labels are augmented with this ordering information to store it in data structure. See the research report for detail.

Example: Let *e* an edge created on site 2 with operation number 10 ($id_e=2:10$), let *f* an edge created on site 1 with operation number 11 ($id_f=1:11$) and let *g* an edge on site 5 created with operation number 11 ($id_g=5:11$). The priority tags (assigned during the edition process) are: 6 for *e*, 7 for *f* and 6 for g (same time of *e* tag). Let '#' is smallest element of Σ* by $<<$. We obtain $6\#2.10 << 6\#5.11 << 7\#1.11$, we have $e\ p\ g\ p\ f$.

Proposition 3: *The ordering* p *on edges is a total ordering on edges and this ordering is compatible with group editing.*

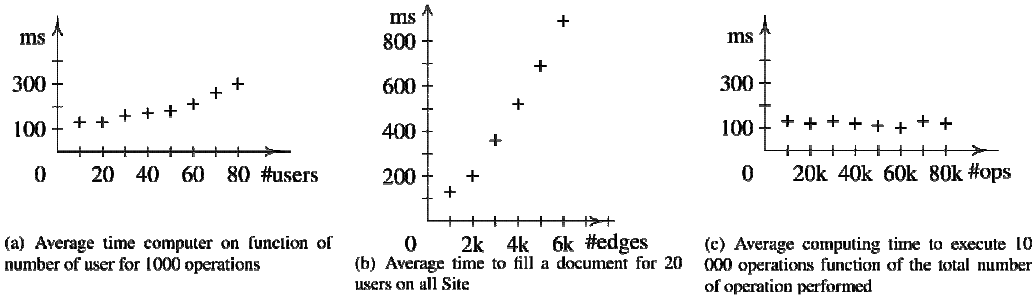
Note that *ChLab* can be used to change the ordering of edges. (actually, we introduce a specialized version of *ChLab* named *ChOrder* for this task).

Proposition 4 : *The set* (Op,f_s) *is an independent set of operations.*

Therefore our collaborative editing algorithms works for ordered trees, i.e. XML trees.

5. CONCLUSION

We have implemented the algorithm and the data structure for XML trees (ordered) in Java on a Mac with a 2.53GHz processor. The P2P framework is simulated by random shuffling of the messages that are broadcast.



(a) Average time computer on function of number of user for 1000 operations

(b) Average time to fill a document for 20 users on all Site

(c) Average computing time to execute 10 000 operations function of the total number of operation performed

Figure 3. Prototype statistic

Future works: We plan to extend this word by adding type information like DTD or XML schemas which are used to ensure that XML documents comply with for general structure. The second main extension that we investigate is the ability to *undo* some operations, which may require a limited use of an history file to recover missing information (needed for instance to recover a deleted tree).

REFERENCES

[1] A.H. Davis and C. Sun and J. Lu. Generalizing operational transformation to the standard general markup language. *CSCW '02: Proceedings of the 2002 ACM*, pages 58--67, New York, NY, USA, 2002. ACM.

[2] L. Du and L. Rui. Preserving operation effects relation in group editors. *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 457--466, New York, NY, USA, 2004. ACM.

[3] L. Du C. A. Ellis AND S. J. Gibbs , Concurrency control in groupware systems. *In SIGMOD Conference (1989), vol. 18, pp. 399–407.*

[4] C. Ignat and M.C. Norrie. Customisable Collaborative Editing Supporting the Work Processes of Organisations. *Computers in Industry*, 57(8-9):758--767, 2006.

[5] C. Ignat and M.C. Norrie. Tree-based Model Algorithm for Maintaining Consistency in Real-time Collaborative Editing Systems. *CSCW 2002, IEEE Distributed Systems online*, 2002.

[6] A. Imine. *Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à -Pair*. PhD thesis, Université Henri Poincaré, Nancy, 2006.

[7] A. Imine and P. Molli and G. Oster and M. Rusinowitch. Proving Correctness of Transformation Functions in Real-Time Groupware. *8th European Conference of Computer-supported Cooperative Work*, 2003.

[8] J. Foster and M. Greenwald and C. Kirkegaard and B. C. Pierce and A. Schmitt. Exploiting Schemas in Data Synchronization. *J. of Computer and System Sciences*, 73(4), 2007.

[9] A. Kermarrec and A. Rowstron and M. Shapiro and P. Druschel. The IceCube approach to the reconciliation of divergent replicas. *PODC '01: Proceedings of the twentieth annual ACM*, pages 210—218,NY, USA, 2001. ACM.

[10] Du Li and Rui Li. Ensuring Content Intention Consistency in Real-Time Group Editors. *24th International Conference on Distributed Com- puting Systems*, 2004. IEEE Computer Society.

[11] G. Oster and H. Skaf-Molli and P. Molli and H. Naja-Jazzar. Supporting Collaborative Writing of XML Documents. *Proceedings of ICEIS 2007: Software Agents and Internet Computing*, pages 335-342, Funchal, Madeira, Portugal.

[12] M. Ressel and D. Nitsche-Ruhland and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *CSCW '96: Proceedings of the 1996 ACM* pages 288--297, NY, USA.

[13] M. Suleiman and M. Cart and J. Ferrié. Serialization of concurrent operations in a distributed collaborative environment. *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 435-445, NY, USA, 1997. ACM.

[14] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. *CSCW '98: Proceedings of the 1998 ACM*, pages 59--68, New York, NY, USA, 1998. ACM.