# An Undo Framework for P2P Collaborative Editing

Stéphane Weiss, Pascal Urso, and Pascal Molli

Nancy-Université, LORIA,
F-54506, Vandoeuvre-lès-Nancy Cedex, France
{weiss,urso,molli}@loria.fr
http://www.loria.fr/∼{weiss,urso,molli}

**Abstract.** Existing Peer to Peer (P2P) collaborative editing systems do not allow any user to undo any modification. However, in such systems, users are not aware of others' modifications, hence, they could obtain duplicate work, conflicting modifications or malicious contents. In this paper, we propose a new undo framework called "UNO: Undo as a New Operation" in the Operational Transformation approach which does not limit the scalability of P2P algorithms. As a proof of concept, we apply our framework to build a P2P collaborative editor with undo capabilities in which any user can undo any modification.

**Keywords:** Group undo, P2P Collaborative Editing, Operational Transformation.

## 1 Introduction

Collaborative editing systems allow people distributed in time and space to work together on shared documents. The major benefits of collaborative writing include reducing task completion time, reducing errors, getting different viewpoints and skills, and obtaining an accurate document [1,2].

Nowadays, collaborative editing systems are becoming Peer to Peer (P2P): Version Control Systems (VCS) turn into Distributed Version Control Systems (DVCS), Wiki systems become P2P Wiki system [3,4,5] and even softphones change into P2P softphones such as skype. P2P architecture provides massive collaboration, resistance to failure and censorship. Such systems could also increase data availability and allow off-line work.

In a P2P environment, users are not immediately aware of others' concurrent modifications. Therefore, a site merging its document with another peer could obtain duplicate work, conflicting modifications or malicious contents. In such a context, an undo feature is mandatory. Every user must be able to revert any undesired change. Moreover, the modification to undo is not necessary the last received one, hence, users must be able to undo any operation in their history. Therefore, we need to provide the most general model of undo mechanism which allows any user to undo any edit operation at any time.

All existing undo approaches in the literature [6,7,8,9,10] are specific to a collaborative editing system. Unfortunately, none of these systems is designed to support P2P architectures. Therefore, none of the existing approaches could provide an undo for P2P systems. On the other hand, some existing collaborative softwares such as Wiki or DVCS propose an undo mechanism. However , wiki systems may fail to undo any operation. Moreover, such systems are centralized, hence, they cannot be used in P2P environment. DVCS allow any user to undo any operation. Nevertheless, DVCS do not bring any warranties about data consistency.

The Operational Transformation (OT) [9,11] framework, purposes a consistency model (CCI) to preserve Causality, Converge and Intention. OT is recognized as a suitable approach to maintain consistency of shared documents. This approach is applied to develop both real-time and asynchronous collaborative editors [12,13]. It consists of two main components: a generic integration algorithm which manages concurrent operations and transformation functions specific to a data model. For instance, MOT2 [13] is an integration algorithm designed for P2P architecture.

In this paper, we propose an undo in the OT framework that respects P2P algorithms' constraints. We call our approach UNO : *Undo as a New Operation*, because it aims to treat undo operation as any other operation newly generated by users. This approach, inspired from well established collaborative tools, allows building a generic schema which can be introduced in any OT approach without affecting its scalability.

As a proof of concept, we apply our framework on the TTF functions which are also the only published set of transformation functions which can be used with MOT2 (See appendix section A). Therefore, in this paper, we obtain a P2P collaborative text editor with undo features.

The structure of the paper is as follows. We first motivate the need of a novel undo approach for P2P systems in Section 2. We explain the main idea of the UNO in Section 3. Then, we briefly describe the OT approach in Section 4. We present our approach in Section 5 and apply it on the TTF functions in Section 6. Section 7 deals with the UNO framework correctness. Finally, we compare our approach with existing undo approaches in Section 9.

## 2 Motivation

P2P collaborative systems bring several exciting features:

- Massive editing: thousand of users can work on the same project and reduce drastically the task completion time,
- Off-line work: users edit their own copy, hence, they can work online or off-line,
- Privacy: users decide when they want to publish their modifications,
- Mobilibity: such systems support ad-hoc collaboration.

However, merging data over a P2P network implies some inconvenient behaviors:

- Obviously, each user cannot know what other users are doing: we can have duplicate work or conflicting modifications,
- Malicious peers could degrade existing work, merging with such peers will propagate this alteration to the whole network.

Such scenarios are not problematics as soon as users are capable of quickly reverting undesired changes. Therefore, in such systems, the undo feature is mandatory.

In OT, several approaches allow undoing any operation at any time such as GOTO-ANYUNDO[14], COT [10]. The GOTO-ANYUNDO approach uses state vectors (aka vector clocks [15]) while the COT approach uses extended state vectors [10] called “context vectors”. A state vector is a vector which contains a logical clock entry per host present in the network. As a consequence, state vector’s size grows linearly with the number of site in the system. In P2P networks, the number of site is potentially unbounded and not even known by network participants. Therefore, state vectors are not compatible with P2P constraints such as scalability or churn.

Finally, we need to design an undo approach which does not limit the integration algorithm’s scalability.

## 3 UNO Idea

The main idea of our approach is to treat undo operations as any other operation generated by users. To illustrate this idea, assume that a text document is replicated (Figure 1). At the beginning, the document contains only “Rendezvous”. Then a first user inserts the sentence “at nine.” at line 2. Concurrently, a second user inserts “At 8 in the park:” at the beginning of the document. And finally, a user wants to undo the operation “ins(2, at nine. )”.

Existing OT undo approaches roughly consist in forming do-undo-pair by coupling the undo operation with the undone operation (Figure 1(a)). Unfortunately, integration algorithms are designed to generate and integrate operations only on the current state. Thus, specific mechanisms, such as “context vectors” [10], are designed to support such insertion. As a result, these undo mechanisms are tightly coupled to a specific integration algorithm that may not scale.

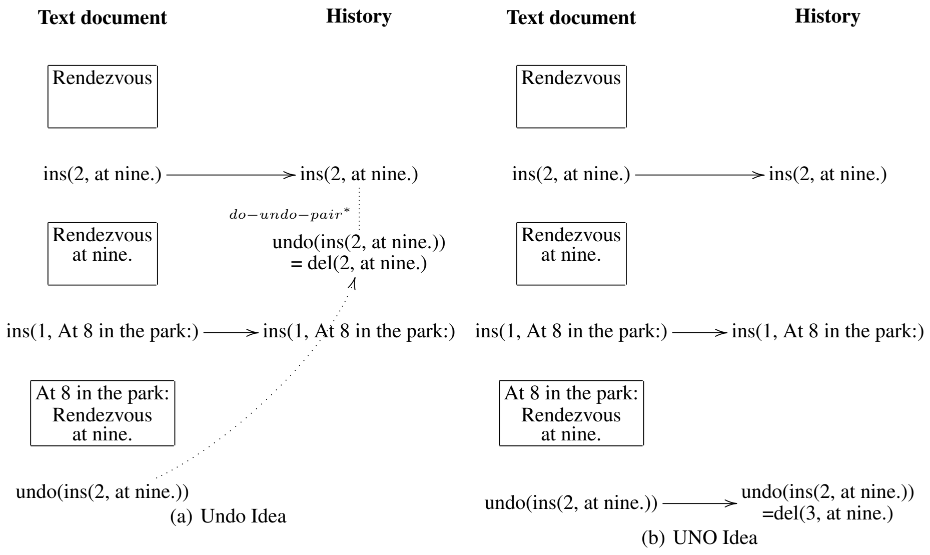

Text document
History
Rendezvous
ins(2, at nine.)
ins(2, at nine.)
do−undo−pair*
Rendezvous at nine.
undo(ins(2, at nine.)) = del(2, at nine.)
ins(1, At 8 in the park:)
ins(1, At 8 in the park:)
At 8 in the park: Rendezvous at nine.
undo(ins(2, at nine.))
(a) Undo Idea
Text document
History
Rendezvous
ins(2, at nine.)
ins(2, at nine.)
Rendezvous at nine.
ins(1, At 8 in the park:)
ins(1, At 8 in the park:)
At 8 in the park: Rendezvous at nine.
undo(ins(2, at nine.))
undo(ins(2, at nine.)) =del(3, at nine.)
(b) UNO Idea

**Fig. 1.** UNO idea

To obtain a generic undo mechanism, we propose to use the same schema as massively used in collaborative tools such as VCS or Wiki. For instance, in SVN ([16] page 91), undoing a previous revision consists in applying a modification and then committing it normally. In the UNO, we produce a new operation that has the wished effect on the current state, then we integrate it as any other user's operation. *Therefore, the undo operation will be placed in the history according the user's real schedule (Figure 1(b)).*

Thus, we obtain an undo mechanism that can be introduced in any OT collaborative editing system without modifying the way operations are integrated. Consequently, the scalability of the targeted OT system is not limited by the undo mechanism.

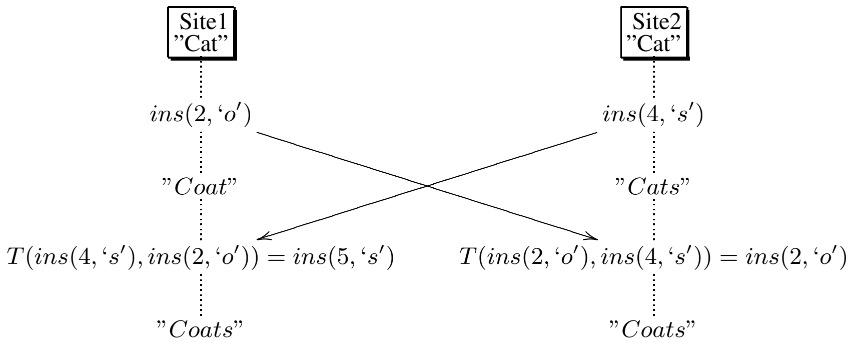
In the Figure 1, when we want to undo the operation "insert(2, at nine. )", we want to remove the third line "at nine.". Our idea is simply to generate a new operation which removes the third line instead of undoing the insertion (see Figure 1(b)). This new operation is generated on the current state, hence, all integration algorithms can handle it.

In the following sections, we give the details required to implement this idea.

## 4 The Operational Transformation (OT) Approach

In the OT approach, shared documents are replicated. Each site contains its own copy of the document, and a user is supposed to work at one site. OT approach allows any user to modify at any time his own copy of the document. Therefore, different copies of the same document can be modified in parallel. In the OT model, a modification is represented as an operation. Each site sends all the locally generated operations to the other sites. On these other sites, such operations are seen as remote operations which have to be integrated for execution.

To allow convergence, these algorithms use a transformation function $T$ to modify remote operations according to local ones. For instance, we consider two sites sharing the same text document (Figure 2). We call $T(op_1, op_2)$ the remote operation $op_1$ transformed against the local operation $op_2$. Of course the definition of the function $T$ is specific to the targeted data type. However, defining a transformation function is not sufficient to ensure correctness, the transformation function must also respect some formal properties (see Section 7).



Site1 "Cat"
Site2 "Cat"
$ins(2, `o')$
$ins(4, `s')$
$"Coat"$
$"Cats"$
$T(ins(4, `s'), ins(2, `o')) = ins(5, `s')$
$T(ins(2, `o'), ins(4, `s')) = ins(2, `o')$
$"Coats"$
$"Coats"$

**Fig. 2.** Transformations

## 5 Proposition: Undo as a New Operation

The main advantage of our undo scheme is that undo operations are treated as regular ones, i.e. do-operations, when integrated on local and remote sites. Thus, undo operations do not require any special treatment on remote sites.

Given a set of operations, an instance of the UNO framework is built in three steps. First, we define the (possibly new) operations that counterbalance original ones. Second, we define the transformation functions for these new operations, if any. Third, we formally verify the properties required by the targeted integration algorithm.

Then, a simple algorithm is used to provide the undo feature.

**Algorithm.** We call $undo(op)$ the undo operation of $op$, i.e. the operation which counterbalances the effect of $op$, if $op$ is the last executed operation. $undo(op)$ can be either a newly defined operation or an operation from the initial set.

However, since $op$ may not be the last executed operation, we need to compute an operation $undo(op)'$ which is defined on the current state. $undo(op)'$ is the transformation of $undo(op)$ according to all operations which have been executed on the local site since the execution of $op$. To compute $undo(op)'$, we use the following algorithm (also illustrated in Figure 3):

```
UNO(HB, i):
op:= undo(HB.get(i))
j:=i+1
while(j <= HB.size)
do
  op:= T(op, HB.get(j))
  j:=j+1
endwhile
return op
```

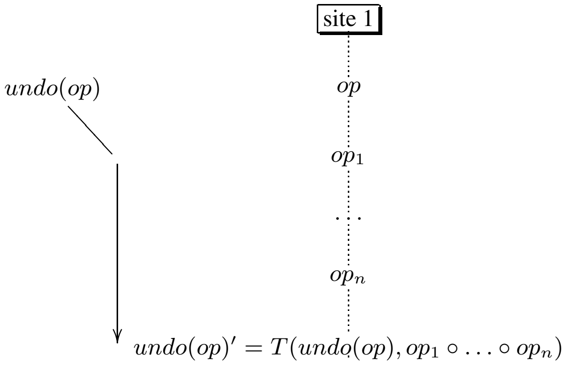


**Fig. 3.** Naive undo Algorithm

Therefore, $undo(op)'$ is defined on the current state of the document. Thus, $undo(op)'$ is considered as a normal operation directly done by a user when it will be integrated and transformed on remote sites.

Since undo operations are treated as normal operations, these transformation functions are standard transformation functions, and thus must be proven correct according to CCI criteria, see Section 7 about correctness of the approach.

**Complexity of the UNO Framework.** The UNO algorithm's time complexity is linear with the number of operation received. The space complexity is constant, indeed, the algorithm does not require any data structure except one operation. Since the UNO framework complexity does not depend on the number of site, it can be applied to P2P architecture.

## 6 Instantiation

Now, we want to apply our undo framework to provide the undo feature in a P2P collaborative text editor built with MOT2 and TTF.

### 6.1 The Tombstones Transformation Functions

The TTF approach is divided in two parts: the model and the transformation functions. A detailed explanation of the TTF approach and its correctness can be found in [17].

The main idea of the model is to keep deleted characters as tombstones. The document's view only shows visible characters: tombstones are hidden. Consequently, the model differs from the view. Figure 4 illustrates this. Assume that a document is in a state "abcd". Now, a user deletes the character 'b'. In the TTF model, the character is replaced by a tombstone (i.e. the character with a visibility flag set to false). The view differs from the model as the view only contains "acd" while the model contains "a$\cancel{b}$cd". Since tombstones are necessary to achieve consistency, they cannot be removed and thus, the operation "Ins" is not inversible.

### 6.2 Undo Operation

The first step in applying the UNO framework is to define the undo operations and their effects. Undoing an operation must return the system to a state on which the undone

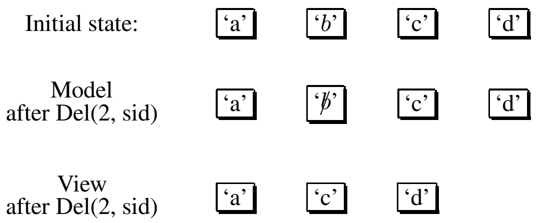


**Fig. 4.** Model in the TTF approach

operation was never performed. In our context, this definition implies that a character deleted concurrently by N sites should not be visible unless each of these N delete operations are undone.

To achieve such a behavior, we simply propose to replace the visibility flag of each character by a visibility level. This visibility level is an integer. Initially, an inserted character has a visibility level of 1. Each time we undo an insertion operation, the visibility level of the corresponding character is decreased. Each time we undo a deletion, we increase the visibility level of this character.

A character is said "visible" and appears in the document's view if its visibility level is at least 1. Similarly, a character is said "invisible" and does not appear in the document's view if its visibility level is less than 1. Since characters are just marked as invisible, we introduce a new operation "Undel(p,sid)" which effect is to increase the visibility level of the character at position "p". The use of visibility levels is illustrated in Figure 5.

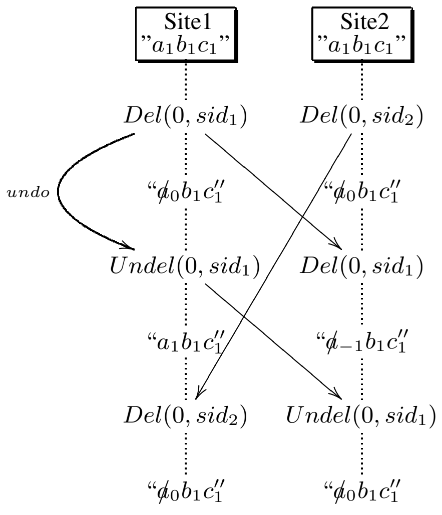


**Fig. 5.** Visibility level

The function $undo(op)$ links normal operations to undo operations. As we have defined undo operations, we can now write the function $undo(op)$.

```
undo(op):
IF $op = Ins(p, c, sid)$ THEN $undo(op) := Del(p, sid)$
IF $op = Del(p, sid)$ THEN $undo(op) := Undel(p, sid)$
IF $op = Undel(p, sid)$ THEN $undo(op) := Del(p, sid)$
```

The second step is to write transformation functions for all operations. The definition of the transformation functions for the operations "Ins" and "Del" are the same as in original TTF approach.

```
T( Ins$(p_1, c_1, sid_1)$,Undel$(p_2, sid_2)$):
    **return** Ins$(p_1, c_1, sid_1)$
**end**

T( Del$(p_1, sid_1)$, Undel$(p_2, sid_2)$):
    **return** Del$(p_1, sid_1)$
**end**

T( Undel$(p_1, sid_1)$, Ins$(p_2, c_2, sid_2)$):
     **if** $(p_1 < p_2)$ **return** Undel$(p_1, sid_1)$
     **else return** Undel$(p_1 + 1, sid_1)$
**end**

T( Undel$(p_1, sid_1)$, Undel$(p_2, sid_2)$):
    **return** Undel$(p_1, sid_1)$
**end**

T( Undel$(p_1, sid_1)$, Del$(p_2, sid_2)$):
    **return** Undel$(p_1, sid_1)$
**end**
```

Moreover, since these transformation functions are bijective, they can easily be reversed. Consequently, we can apply this approach with integration algorithms as SOCT2, GOTO which require reversible transformation functions.

## 7 Correctness

An OT system is considered as correct if it respects the CCI [9] criteria:

**Causality:** This criterion ensures that all operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [18], will be executed in the same order on every copy.

**Convergence:** The system converges if all copies are identical when the system is idle.

**Intention:** The expected effect of an operation should be observed on all copies. It must be ensured by the transformation functions and by the integration algorithm.

In [13], the authors show how MOT2 ensures Causality. Convergence is achieved if the transformation functions satisfy two properties [19]:

* $TP1$: The transformation property $TP1$ defines a state equality. The state obtained by the execution of an operation $op_1$ on a state $S$ followed by the execution of the operation $T(op_2, op_1)$ should be equal to the state obtained by the execution of $op_2$ on a state $S$ followed by the execution of $T(op_1, op_2)$ :

$$TP1 : S \circ op_1 \circ T(op_2, op_1) = S \circ op_2 \circ T(op_1, op_2)$$

* $TP2$: The property $TP2$ ensures that the transformation of an operation against a sequence of operations does not depend on the transformation order of operations in this sequence.

$$TP2:\ T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$$

Based on some problematic scenarios called “undo puzzles”, prior works expressed the need of additional properties to obtain a correct undo. Two properties $IP2$ and $IP3$ are proposed [7,20] to solve these scenarios. However, none shows that these properties are necessary and sufficient to ensure a correct undo. For instance, the property $IP2$ is not respected by our transformation functions while the corresponding problematic scenario does not appear (Figure 6).

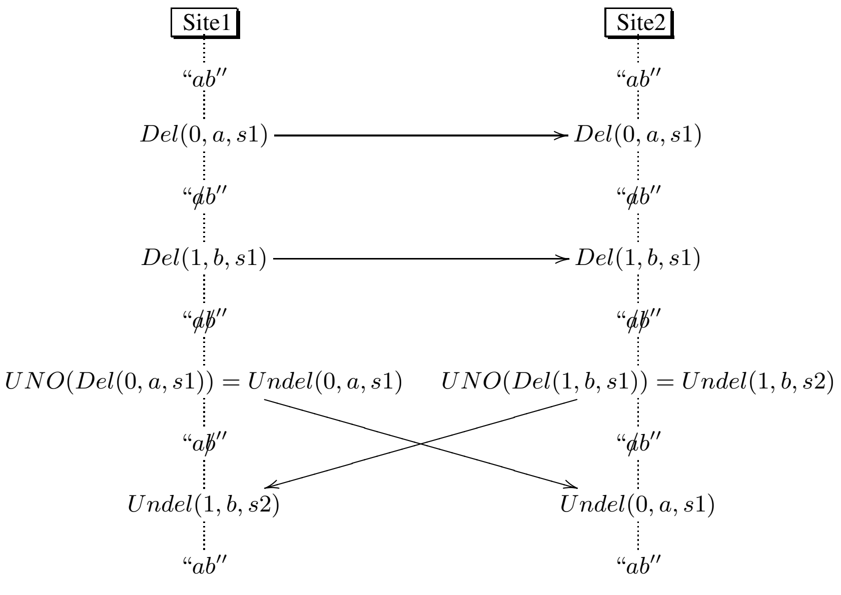


**Fig. 6.** IP2 undo puzzle

The condition $IP3$ is formally defined as:

$$IP3:\ T(undo(op), T(seq, op)) = undo(T(op, seq))$$

To illustrate this condition, Figure 7, two sites make concurrent operations. Site1 generates $op$ while site 2 generates a sequence of operations $seq$. Both sites receive remote operations, transform and integrate them. Now, they are on the same state. Consequently, if they want to undo the same operation on the same state, they must obviously generate the same operation. Site1 generates $undo(op)$ and transforms it against following operations $T(seq, op)$. Site2 undoes the last received operation which is $T(op, seq)$. These two undo operations are defined on the same state, they undo the same operation, so the resulting operation must be the same. The verification of this property ensures that whenever an operation is undone, the undo effect remains the same.

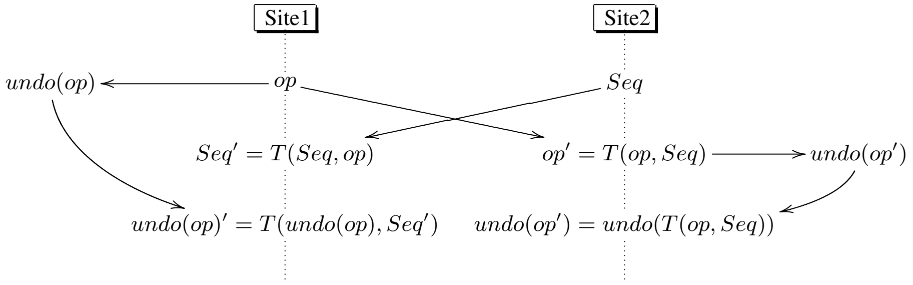


**Fig. 7.** Respect of the undo effect

So, there are properties to verify in order to ensure a correct OT system with undo. Due to their conciseness, these properties are theoretically easy to prove. However, one of the particularity of the OT approach is the huge numbers of cases to check.

In such conditions, a hand proof is error-prone. Indeed, many hand-proven transformation functions were finally revealed false (all counter examples can be found in [17]).

On another hand, each of the cases to check can be easily handled by an automated formal theorem prover. Consequently, we choose to use the proof environment VOTE [21] based on the theorem prover Spike [22,23] which generates all the cases and ensures the verification of all properties.

Using the proof environment VOTE [21], we have proven that our transformation functions verify the properties $TP1$, $TP2$ and $IP3$. The system specification given to the theorem prover Spike[1] can be reviewed and tested at the following url: `http://graveyard.sf.net/`.

## 8 Integrating the UNO with Existing Integration Algorithms

In the OT framework, integration algorithms (SOCT2, SOCT4, GOTO, COT, MOT2) are defined for operations (with no assumption about the number or the kind of operations) and need transformation functions to deal with these operations.

The UNO framework extends a set of operations and transformation functions to support a recovery mechanism. We obtain a new set of operations and transformation functions. Consequently, the resulting set can be handled by any existing integration algorithm.

Our framework also requires the UNO algorithm. To undo an operation $op$, we generate an undo operation $undo(op)$. The UNO algorithm transforms $undo(op)$ against all operations which have been executed after $op$. For this stage, $undo(op)$ is considered as concurrent to all operations after $op$. Fortunately, the main goal of every integration algorithm is to transform an operation against a set of concurrent operations.

Consequently, any OT integration algorithm can determine the undo operation. The resulting operation is treated as a normal operation. Thus, the UNO algorithm can be easily integrated in any integration algorithm. Of course, this requires to remove the

[1] `http://lita.sciences.univ-metz.fr/~stratula`

undo dedicated treatment from these algorithm (such as the "mirror" and "fold" functions of adOPTed or "ensure-IPXSafety" of COT).

As a proof of concept, we build the Graveyard prototype[2]. Graveyard is an open-source collaborative text editor that allow any user to undo any operation. It relies either on the MOT2 algorithm or on the SOCT2 algorithm for integrating concurrent operations and it uses the TTF transformation functions with related undo operations. However, we can replace MOT2 by SOCT4, adOPTed or GOTO and obtain the same result.

## 9 Related Work

The first selective undo was proposed in [7]. To undo an operation, the authors propose to swap it with following operations in the history. Then, the resulting operation's inverse is applied. However, swapping two operations in the history is not always possible, hence, the authors also add the notion of conflict. If a conflict occurs, the undo is aborted. Therefore, this framework does not allow undoing any operation.

In [24], the authors introduce an undo specific to the adOPTed algorithm by adding two functions called "mirror" and "fold". This solution allows undoing operations in the inverse chronological order, i.e. from the last operation to the first one without skipping one. This approach does not allow undoing any operation. Since the adOPTed algorithm requires transformation functions satisfying $TP1$ and $TP2$, we can use the TTF functions in association with our undo approach to provide an undo for any operation.

The GOTO-ANYUNDO approach [20] is associated with the GOTO integration algorithm. This approach introduces a new undo algorithm called ANYUNDO-X. This undo approach is the first to solve known undo problematic scenarios while allowing any user to undo any operation at any time. GOTO-ANYUNDO treats specifically undo operations: undone and undo operations are grouped to create do-undo pairs. While integrating an undo operation, the history buffer is modified in order to remove the undone operation effect from the history. The GOTO-ANYUNDO approach is designed for real-time editing. In such a context, state vectors are adequate, however they are not compatible with P2P environment.

In [25], the authors define two properties $C3$ and $C4$ which are similar to $IP2$ and $IP3$. To ensure the verification of these two properties, the authors introduce a specific operation "undo(op)". This approach defines generic transformation functions for this operation "undo(op)" using the proposed transformation functions. The main idea to enforce $C4$ is to swap the operations and undo the resulting operation. Unfortunately, the authors do not discuss the case of causally dependent operations. This leads to incorrect results.

The COT approach [10] is an OT system designed for real-time editing which introduces the notion of "context vector". A context vector is associated to each operation and represents the operations executed before its generation. Unlike state vectors, context vectors captures undo operations causality and concurrency. As state vectors, the cost of context vector is compatible with real-time editing, however, they are not suitable in a P2P environment.

[2] `http://graveyard.sf.net`

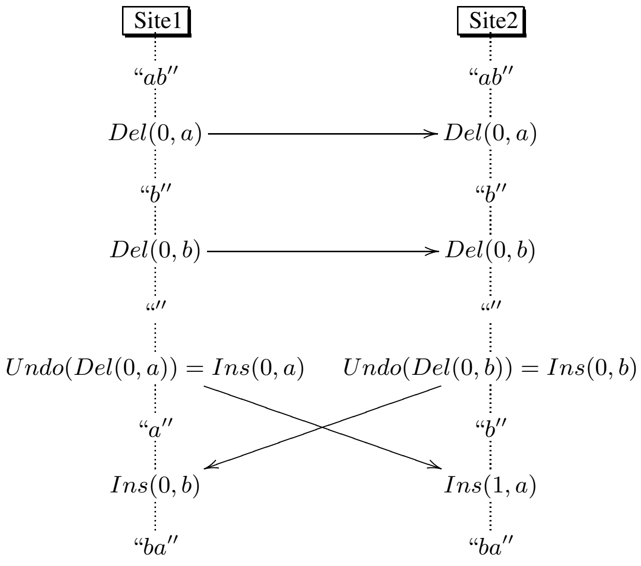


**Fig. 8.** IP2 undo puzzle

Distributed version control systems (DVCS) as Git [3] are P2P collaborative systems mainly used for source code editing. They propose an undo comparable to the UNO approach: they compute a new patch to remove the effect of a previous one and treat it as a new patch. However, DVCS lack of a formal framework, indeed, there is no property to validate DVCS' correctness. On the contrary, the UNO approach is based on the CCI framework.

Repliwiki, Distriwiki and Wooki are P2P wiki systems. Unfortunately, they do not provide an undo mechanism.

**Discussion.** Algorithms similar to the UNO algorithm already appear in the literature. For instance, [25] calls it the naive undo algorithm. However, it was never supported as a correct undo algorithm. The reason is its apparent inability to solve known undo-puzzles.

In this paper, we have shown that it provides a correct undo

- even if the $IP2$ condition is relaxed
- and if the operation set and the transformation functions are able to respect the CCI criteria including the undo intentions.

For instance, the operation "Ins" does not realize exactly the undo intention of the operation "Del", in Figure 8, this leads to swap the characters 'a' and 'b'. The intention of "Ins" is "*insert a new element*". On the other hand the operation "Undel" realizes the intention "*makes a deleted element reappear*". Thus, if a framework wants to use the operation "Ins" to undo the operation "Del", it is forced to introduce complex integration

[3] http://git.or.cz/

mechanisms to avoid undo-puzzle to occurs. On the contrary, using an operation expressing the undo intention solves this puzzle as show in Figure 6.

We thus strongly claim, that operations and transformation functions which ensure undo intention preservation allow building more generic, more efficient and correct undo mechanism.

## 10 Conclusions

In all existing OT approaches, undo is designed for real-time editing. In this paper, we introduced our undo framework which provides undo feature for all integration algorithms even P2P ones.

In this paper, we propose:

- a generic undo framework: This framework can be applied to all set of transformation functions to provide an undo mechanism on several date type. An important feature of our approach is that the resulting transformation functions remain generic towards integration algorithms. Consequently, we can apply these functions with COT, SOCT2, SOCT4, MOT2, GOTO and adOPTed.
- a CCI compliant framework: The CCI criteria is a formal framework for collaborative editing correctness. The UNO framework uses the CCI to ensure its correctness.
- a scalable framework: The UNO approach time and space complexity is constant with the number of site. Therefore, it is particularly adequate for P2P environment. The UNO algorithm is efficient since it is only linear in time with the number of operation received. We also show that a simple and efficient algorithm considered incorrect can be instantiated to provide a correct undo.
- an implementation: We have a complete solution to build correct P2P text editors with a flexible undo capability. The TTF transformation functions with the undo proposed in this paper are implemented in the Graveyard[4] collaborative text editor. Graveyard is a prototype which can be used with MOT2 for P2P or SOCT2.

The CCI framework includes the Intention definition which is not formally defined in the general case. In the UNO, the undo behavior depends on the Intention criterion. Therefore, as future work, we will try to formally defined the Intention. We plan also to implement a DVCS and a wiki system using the UNO approach.

## References

1. Tammaro, S.G., Mosier, J.N., Goodwin, N.C., Spitz, G.: Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing. Computer-Supported Cooperative Work - JCSCW 6(1), 19–51 (1997)
2. Noël, S., Robert, J.-M.: Empirical study on collaborative writing: What do co-authors do, use, and like? Computer Supported Cooperative Work - JCSCW 13(1), 63–89 (2004)

[4] http://graveyard.sf.net

3. Morris, J.C.: Distriwiki: a distributed peer-to-peer wiki network. In: Int. Sym. Wikis, pp. 69–74 (2007)
4. Kang, B.B., Black, C.R., Aangi-Reddy, S., Masri, A.E.: Repliwiki: A next generation architecture for wikipedia (unpublished), http://isr.uncc.edu/repliwiki/repliwiki-conference.pdf
5. Weiss, S., Urso, P., Molli, P.: Wooki: a p2p wiki-based collaborative writing tool. In: Web Information Systems Engineering, December 2007, Springer, Nancy (2007)
6. Berlage, T., Genau, A.: A framework for shared applications with a replicated architecture. In: ACM Symposium on User Interface Software and Technology, pp. 249–257 (1993)
7. Prakash, A., Knister, M.J.: A framework for undoing actions in collaborative systems. ACM Trans. Comput.-Hum. Interact. 1(4), 295–330 (1994)
8. Choudhary, R., Dewan, P.: A general multi-user undo/redo model. In: ECSCW, pp. 229–246 (1995)
9. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) 5(1), 63–108 (1998)
10. Sun, D., Sun, C.: Operation Context and Context-based Operational Transformation. In: Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006, November 2006, pp. 279–288. ACM Press, Banff (2006)
11. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: Clifford, J., Lindsay, B.G., Maier, D. (eds.) SIGMOD Conference, pp. 399–407. ACM Press, New York (1989)
12. Molli, P., Oster, G., Skaf-Molli, H., Imine, A.: Using the transformational approach to build a safe and generic data synchronizer. In: Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003, November 2003, pp. 212–220. ACM Press, Sanibel Island (2003)
13. Cart, M., Ferriè, J.: Asynchronous reconciliation based on operational transformation for p2p collaborative environments. In: CollaborateCom (2007)
14. Sun, C., Chen, D.: Consistency maintenance in real-time collaborative graphics editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) 9(1), 1–41 (2002)
15. Mattern, F.: Virtual time and global states of distributed systems. In: Cosnard, M. (ed.) Proceedings of the International Workshop on Parallel and Distributed Algorithms, October 1989, pp. 215–226. Elsevier Science Publishers, Château de Bonas (1989)
16. Collins-Sussman, B., Fitzpatrick, B.W., Pilato, C.M.: Version Control with Subversion. O'Reilly Media, Sebastopol (2007), http://svnbook.red-bean.com/
17. Oster, G., Urso, P., Molli, P., Imine, A.: Tombstone transformation functions for ensuring consistency in collaborative editing systems. In: The Second International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006), November 2006. IEEE Press, Atlanta (2006)
18. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
19. Ressel, M., Nitsche-Ruhland, D., Gunzenhäuser, R.: An integrating, transformation-oriented approach to concurrency control and undo in group editors. In: CSCW, pp. 288–297 (1996)
20. Sun, C.: Undo as concurrent inverse in group editors. ACM Transactions on Computer-Human Interaction (TOCHI) 9(4), 309–361 (2002)
21. Imine, A., Molli, P., Oster, G., Urso, P.: Vote: Group editors analyzing tool: System description. Electr. Notes Theor. Comput. Sci. 86(1) (2003)
22. Nieuwenhuis, R. (ed.): RTA 2003. LNCS, vol. 2706. Springer, Heidelberg (2003)
23. Stratulat, S.: A general framework to build contextual cover set induction provers. J. Symb. Comput. 32(4), 403–445 (2001)

24. Ressel, M., Gunzenhäuser, R.: Reducing the problems of group undo. In: GROUP, pp. 131–139 (1999)
25. Ferrié, J., Vidot, N., Cart, M.: Concurrent undo operations in collaborative environments using operational transformation. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 155–173. Springer, Heidelberg (2004)

## Appendix

## A MOT2 and TTF

In [13], the authors claim that MOT2 only require the $TP1$ property. Therefore, we could use the following transformation functions since they satisfy $TP1$.

```
T( Ins(p1, c1) , Ins(p2, c2) ):
if p1 < p2 or ( p1 = p2 and c1 < c2 )
   Ins(p1, c1)
else Ins(p1 + 1, c1)

T( Ins(p1, c1), Del(p2, c2) ):
if p1 <= p2
   Ins(p1, c1)
else Ins(p1 − 1, c1)

T( Del(p1, c1), Ins(p2, c2) ):
if p1 < p2
  Del(p1, c1)
else Del(p1 + 1, c1)

T( Del(p1, c1), Del(p2, c2) ):
if p1 < p2
  Del(p1, c1)
else Del(p1 − 1, c1)
```

Unfortunately, Figure 9 illustrates a divergence scenario which can occur. This problem is an instance of the $TP2$ puzzle [17].

Finally, MOT2 requires transformation functions which satisfy the properties $TP1$ and $TP2$. Therefore, the TTF transformation functions are particularly adequate for MOT2 since they are the only published set of transformation functions satisfying $TP1$ ant $TP2$.

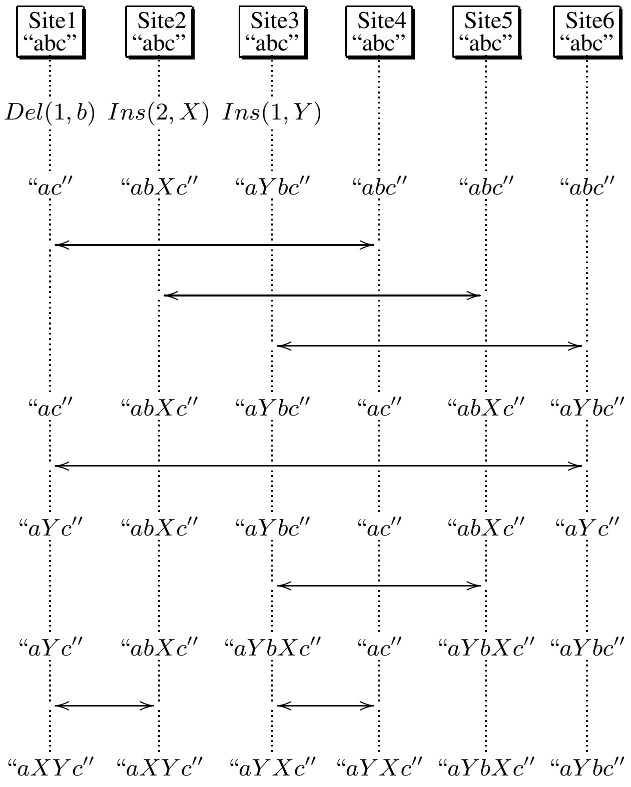

Site1 "abc"
Site2 "abc"
Site3 "abc"
Site4 "abc"
Site5 "abc"
Site6 "abc"
$Del(1,b)$
$Ins(2,X)$
$Ins(1,Y)$
"ac" "abXc" "aYbc" "abc" "abc" "abc"
"ac" "abXc" "aYbc" "ac" "abXc" "aYbc"
"aYc" "abXc" "aYbc" "ac" "abXc" "aYc"
"aYc" "abXc" "aYbXc" "ac" "aYbXc" "aYbc"
"aXYc" "aXYc" "aYXc" "aYXc" "aYbXc" "aYbc"

**Fig. 9.** Divergence scenario