A Flexible Undo Framework for Collaborative Editing

Stéphane Weiss, Pascal Urso, Pascal Molli

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Flexible Undo Framework for Collaborative Editing

Stéphane Weiss — Pascal Urso — Pascal Molli

N° 6516

Avril 2008

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--6516--FR+ENG

A Flexible Undo Framework for Collaborative Editing

Stéphane Weiss , Pascal Urso , Pascal Molli

Thème COG — Systèmes cognitifs
Projet ECOO

Rapport de recherche n° 6516 — Avril 2008 — 23 pages

Abstract: The Undo feature has been recognized as an important feature of collaborative systems. The Operational Transformation (OT) is a suitable approach to maintain consistency of shared documents. In all existing OT undo approaches, this feature is provided by the system without a specific design. In this paper, we argue that such feature could not always achieve the user's undo intention. Therefore, we propose a new generic and flexible undo framework called "Compensation". This framework allows defining an adequate modification to counterbalance any action performed by any user in the collaborative system.

Key-words: Compensation, Transformation Operationnal, Group undo, Collaborative Editing

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Un cadre flexible pour l'annulation dans l'édition collaborative

Résumé : Les systèmes d'édition collaborative fournissent un mécanisme d'annulation qui permet à un utilisateur de corriger ses erreurs ou de visionner des modifications précédentes. Il est difficile de fournir une telle fonctionnalité dans un environnement collaboratif où tous les utilisateurs sont autorisés à retirer les modifications effectuées par n'importe quel autre utilisateur. L'approche des Transformées opérationnelles permet d'assurer la cohérence des données partagées dans les systèmes d'édition collaborative. Toutes les approches existantes proposent une annulation indépendante du type d'application. Nous montrons, dans cet article, que ce type d'annulation peut donner des résultats inattendus pour un utilisateur. Par conséquent, nous proposons un nouveau framework générique et flexible pour l'annulation appelé "Compensation". Notre approche permet de définir une action adaptée pour contrebalancer les effets d'une modification.

Mots-clés : Compensation, Transformée operationnelle, Annulation de groupe, Édition collaborative

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].

The undo mechanism has been recognized as an important feature of collaborative editing systems [3, 4, 5, 6, 7]. In such systems, the most general model of undo mechanism allows any user to undo any edit operation at any time. Preserving consistency of shared data with the undo feature is a complex issue.

In collaborative editing, the Operational Transformation (OT) [7, 8] approach is recognized as a suitable approach to maintain consistency of shared documents. This approach has been applied to develop both real-time and asynchronous collaborative editors [**?**]. This framework considers two main components: the transformation functions, which are specific to an application, and the integration algorithm.

All undo algorithms proposed in the OT framework [9, 10, 11, 12] provide an undo at the integration algorithm level. The main goal is to return to a state on which the undone operation has never been performed. They are based on standard set of operations where the inverse of each operation is already included in this set of operations [11]. For instance, most of these algorithms are instantiated for operations insert and delete where the inverse of delete is insert and the inverse of insert is delete. This kinds of undo mechanisms has the advantage of not modifying the transformation functions specific to the application. However, through two motivating examples, we show that:

1. introducing undo feature in a collaborative editing systems impacts the definition of the operation set handled by the system,

2. a system undo may not achieve the user's undo intention.

Firstly, we consider a text collaborative editor. A user can produce three kind of modifications: insert a character, delete a character and update character's attributes (face, bold, italic, ...). Let's assume that a character "a" is inserted, put in bold "**a**" and finally deleted "". Now, a user wants to undo the deletion, the system generates the inverse operation, i.e. an insertion of the character "a". Obviously, the user expects to see the character "**a**" in bold. Unfortunately, the result is the character "a" which is no more in bold.

From this example, we note that the deletion removes the effects of two modifications: the insertion and the attribute update. Therefore, generating only an insertion to undo a deletion is not sufficient to achieve the expected effect. Undo mechanisms should also update the character attribute. Since existing undo approaches generate only one inverse operation, a solution consists in modifying existing operations[1]. Modifying the operation set implies to modify the transformation functions. Thus, to obtain a complete undo feature, even with

[1]Another solution may consist in generating many operations to undo one, but such a modification introduces consistency issues that have never been addressed.

an undo defined at the integration algorithm level, the modification of the operations and transformation functions cannot be avoided.

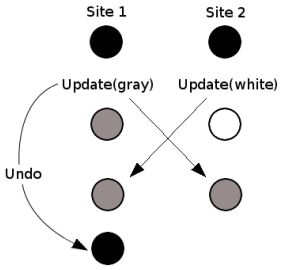

Figure 1: Undo in a graphical editor

Secondly, under certain circumstances, the system functionality undo may not achieve the expected undo effect from a user point of view. For instance, we consider a graphical editor (Figure 1) in which the fill color of a circle can be updated. Initially, the circle is black. Now, two users update its color at the same time: user 1 updates it to gray, while user 2 updates it to white. Since the circle could have only one color, the system have to arbitrary choose one color: let's say the color gray.

Now, user 1 wants to undo his operation. The system functionality undo generates a state on which only user 2's update was performed. Therefore, the circle turns into white. Depending on the context, this result could be desired. However, if we assume that a user is able to imagine the effect of its undo, user 1 will find this result surprising. Indeed, user 1 has seen the circle black and puts it in gray, so when he undoes his update, his intention is probably to obtain a black circle and not a white one as shown in Figure 1. On another hand, if it's the user 2 who undoes the "put in gray" operation after seeing the circle in white, his intention is probably to obtain a white circle.

This example illustrates that, depending on the context, the undo effect expected by a user could be different than a system undo effect. As a consequence, we claim that an undo framework must allow building any kind of undo effect: the regular one which we call **system undo** and other undo effects we call **user undo**. Based of these observations, we claim that an undo feature for an OT collaborative editing system must be designed specifically for an application. Thus, we introduce a novel undo approach, called **Compensation**, defined at the transformation function level. As suggested in [3], our framework allows designing a *user undo* as well as a *system undo*.

In the compensation approach, we generate a new operation to *counterbalance* the effect of another one. For instance, in the text editor previously defined, we could add an operation "insertFormatted" to compensate the deletion of a bold character. Similarly, in the graphical editor, we can compensate user 1's operation by an operation which updates the circle's color to black.

With the compensation approach, we can have the same result obtained in current undo approaches: we can define the compensation effect in a way that it rolls back the modification. Therefore, we can consider that *system undo* is a particular case of compensation.

The compensation approach brings the followings advantages. There is no specific mechanism required to integrate compensation operations when received on remote sites. The compensation relies only on a simple scheme which can be introduced in all existing OT algorithms. And finally, unlike existing undo approaches, the compensation does not limit the usage of log truncation algorithms [7] defined for approaches without undo feature.

As a proof of concept, we apply the compensation framework to the Tombstone Transformation Functions (TTF) [13]. Using the compensation, we design two different ways of undoing in the TTF approach: a *system undo* and a *user undo*. The correction of the obtained models is proved formally by the automated proof environment VOTE [14].

Based on this approach, we build Graveyard, a collaborative real-time text editor prototype. Graveyard combines a SOCT2 [15] algorithm that does not provide a native undo feature, with the TTF transformation functions extended with compensation operations. Finally, we obtain a reliable decentralized collaborative editor with an undo mechanism.

In this paper, we present our framework for compensation. We first describe the OT approach and its correctness criteria in the second section. Then, we present the compensation approach and show how it could be applied in section "Compensation approach". This section also deals with correctness issues and finally we show how our approach can be used with main integration algorithms. In the third section, we instantiate the compensation framework with the TTF transformation functions to obtain two different undo mechanisms for linear structures. We presents the Graveyard prototype which implements the approach in the forth section. Finally, we compare our approach with existing undo approaches in the fifth section.

2 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.

The *CCI* Model [7] considers an OT system correct if it preserves Causality, Convergence and Intention.

Causality This criterion ensures that all operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [16], 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.

To ensure these criteria, several integration algorithms were proposed [9, 15, 17, 18, 12, **?**]. Causality preservation is mainly achieved by the use of state vectors or a central time-stamper.

To ensure convergence, these algorithms use transformation functions to modify remote operations according to local ones. For instance, we consider two sites sharing the same text document (Figure 2). The initial state of the document is "Compnsation". On Site1, a user wants to insert a character 'e' to obtain "Compensation". Concurrently, on Site2, another user wants to insert a 's' at the end of the word. Each site sends its local operation to the other one. If Site1 and Site2 directly execute its remote operation, they do not obtain the same document. On Site1, the character 's' should have been inserted at position 12 instead of 11. In the OT model, a transformation functions T is devised in order to transform remote operations regarding concurrent operations. We call $T(op_1, op_2)$ the operation op_1 transformed against the operation op_2.

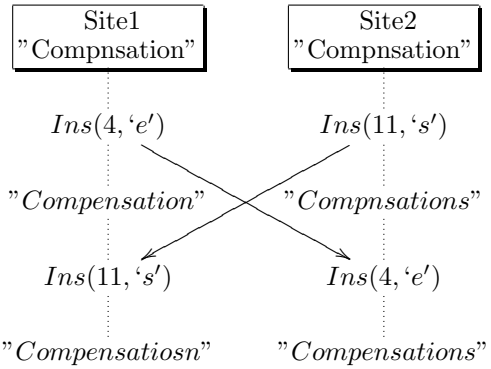

Figure 2: Divergence scenario

However, defining transformation functions is not sufficient to ensure convergence. In the OT approach, the correctness is based on two standard properties called $TP1$ and $TP2$. Some algorithms only require $TP1$ such as SOCT4 [18], MOT2 [**?**] or COT [12], while others require $TP1$ and $TP2$ such as Adopted [9], GOTO [17], SOCT2 [15].

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 $T(op_1, op_2)$:

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

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.

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

Finally, convergence and causality are not sufficient. For instance, in Figure 2, both sites could reach the state "Compensatiosn". Unfortunately, site2 's intention, which is to add the 's' after the 'n', is no more respected. The transformation functions have to be designed in order to ensure the intention preservation. Although a generic definition for intention has never been formalized, we assume that particular definitions can be expressed and verified for a given transformation functions set.

3 Compensation approach

The compensation is a generic framework which allows defining how an operation should be counterbalanced. To obtain any kind of effect when the user undoes an operation, we produce a new operation which has the wished effect. This new operation is treated as regular ones when integrated on local and remote sites.

In existing OT undo mechanisms, the undo effect is to return to a state on which the undone operation was never performed. We name such undo a **system undo**. If we want to obtain such a *system undo*, we can automatically prove that the operations and the transformation functions designed for such a compensation ensure this effect thanks to a formal property we call TPC.

In contrast to *system undo*, a **user undo** allows generating a new state which fits to user's expectation. The correctness of the transformation functions thereby relies on the intention preservation which, unfortunately, cannot be formally defined on a generic purpose. However, this intention preservation should be defined and verified for the considered transformation functions.

Given a set of operations, an instance of the compensation framework is built in three steps. Firstly, define the (possibly new) operations that counterbalance original ones. Secondly, define the transformation function for the new operations, if any. Thirdly, formally verify the properties required by the targeted integration algorithm.

3.1 Compensation mechanism

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

$C(op)$ is not necessary the inverse operation of op even in the case of *system undo*. For instance, in the text editor introducing example, the operation "insertFormatted" is not the inverse operation of the deletion, nevertheless, after its application, the document is in the state that we should have obtained if the deletion was never performed. In fact, the compensation in this example is a complete *system undo*.

However, since op may not be the last executed operation, we need to compute an operation $C(op)'$ which is defined on the current state. $C(op)'$ is the transformation of $C(op)$ according to all operations which have been executed on the local site since the execution of op. Therefore, $C(op)'$ is treated as any other newly generated operation (i.e.

sent to all remote sites in order to be integrated). To compute $C(op)'$, we use the naive undo algorithm [10], see Figure 3.

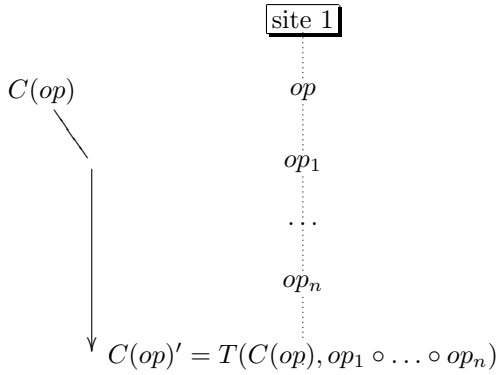

Figure 3: Algorithm of the compensation

3.2 Instantiation

The first step to instantiate the compensation framework is to define all operations $C(op)$. We need to define each operation $C(op)$ in such a way that $S \circ op \circ C(op)$ is a state where the effect of op has been counterbalanced.

When the definition of $C(op)$ introduces new operation(s), the following step is to write transformation functions for each new operation(s). These transformation functions are standard transformation functions, and thus must be proven correct according to CCI criteria.

In the OT approach, the correctness is ensured by a set of properties that must be satisfied by the transformation functions. Since compensation operations are integrated as regular operations, we have to demonstrate that the new set of operations and transformation functions respect the standard properties for convergence and intention.

The transformation functions defined for regular and compensation operations have to ensure:

- $TP1$ and $TP2$ (or only $TP1$, depending on the integration algorithm) are required for consistency,
- The intention preservation: Like regular operations, we have to verify that the effect of a compensation operation will be preserved.
- In case of *system undo*, we also need to satisfy a Transformation Property for Compensation[2] (TPC). This property ensures that the compensation effect will always

[2]We discuss of this property for a *user undo* is section "Comparing system undo and user undo"

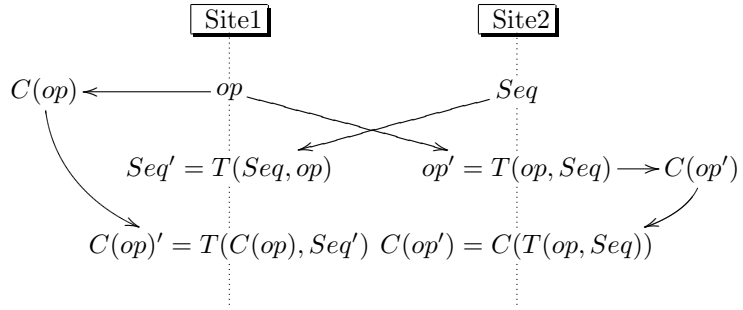

Figure 4: Respect of the system undo effect

be the same, even if the operation compensated is not the last executed one. This condition is similar to the condition $C4$ [10] and the condition $IP3$ [12, 19, 11]. The condition TPC is formally defined as:

$$TPC : T(C(op), T(seq, op)) = C(T(op, seq))$$

Figure 4 explains the TPC property. 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 compensate the same operation on the same state, they must obviously generate the same operation. Site1 generates $C(op)$ and transforms it against following operations $T(seq, op)$. Site2 compensates the last received operation which is $T(op, seq)$. These two compensation operations are defined on the same state, they compensate the same operation, so the resulting operation must be the same. The verification of this property ensures that whenever an operation is compensated, the compensation effect remains the same.

So, there are properties to verify in order to ensure a correct OT system with compensation. 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, and many transformation functions supposed hand-proven finally revealed themselves false (all counter examples can be found in [13]).

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 [14] based on the theorem prover Spike [20, 21] which generates all the cases and ensures the verification of all properties.

We have now defined a complete and generic framework to provide compensation in the OT approach. This framework could be applied to many transformation functions and integration algorithm. The following section will discuss about compensation and existing integration algorithms.

3.3 Integrating the compensation in 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 compensation 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 a compensation algorithm. To compensate an operation *op*, we generate a compensation operation $C(op)$. The compensation algorithm transforms $C(op)$ against all operations which have been executed after *op*. For this stage, $C(op)$ 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 compensation operation. The resulting operation is treated as a normal operation. Thus, the compensation algorithm can be easily integrated in any integration algorithm.

In the following section, we instantiate the compensation approach on the Tombstone Transformation Functions (TTF). The TTF have two particularities: they have non-inversible operations and are the only transformation functions that ensure $TP1$ and $TP2$ [13].

4 Compensation in the TTF approach

In order to illustrate the compensation approach, we choose to apply it on the TTF functions. We present the TTF functions in the following and then, apply the compensation. In the first place, we apply the compensation to build a *system undo*. This kind of recovery is suitable for patch undo and , especially, for vandalism reverting. In the second place, we propose the compensation as a *user undo* which fits differently to user's intention.

4.1 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 [13].

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 5 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$\not$bcd". Since tombstones are necessary to achieve consistency, they cannot be removed and thus, the operation "Ins" is not inversible.

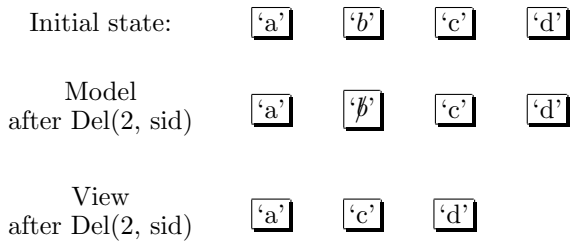

Figure 5: Model in the TTF approach.

The TTF transformation functions (Figure 6) can only be used with the TTF model. In other OT approaches, the deletion of a character decreases the position of all the following characters. The TTF model's particularity is that a character's position can only grow. Therefore, transforming an operation against any "Del" operation will never modify it.

```
T( Ins(p1, c1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) return Ins(p1, c1, sid1)
   else if (p1 = p2 and sid1 < sid2) return Ins(p1, c1, sid1)
   else return Ins(p1 + 1, c1, sid1)
end

T( Ins(p1, c1, sid1),Del(p2, sid2)):
   return Ins(p1, c1, sid1)
end

T( Del(p1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) return Del(p1, sid1)
   else return Del(p1 + 1, sid1)
end

T( Del(p1, sid1), Del(p2, sid2)):
   return Del(p1, sid1)
end
```

Figure 6: TTF transformation functions

4.2 Compensation as a "system undo"

The first step in applying the compensation framework is to define the compensation operations and their effect.

A *system undo* returns the system to a state on which the undone 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 compensated.

To achieve a such behavior, we propose to associate, to each character, a visibility level. This visibility level is an integer. Initially, an inserted character has a visibility level of 1. Each time we compensate an insertion operation, the visibility level of the corresponding character is decreased. Each time we compensate 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.

We choose to compensate the insertion with an operation "Del(p, sid)" which effect is to decrease the visibility level of the character at position "p". To compensate a deletion, we should increase the visibility level of the character. Since neither the operation "Ins", nor the operation "Del" could provide this effect, we add a new operation "Undel(p, sid)" which effect is to increase the visibility level of the character at position "p". Similarly, the compensation of an operation "Undel" have to decrease the visibility level. Fortunately, the operation "Del" produces this effect and can be used to compensate an operation "Undel".

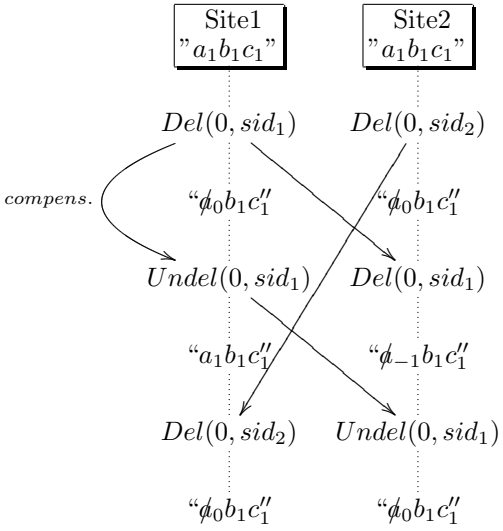

Figure 7: Visibility level

The use of visibility levels is illustrated in Figure 7.

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

$C(op)$:

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

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 presented in Figure 6.

As the operations "Del" and "Undel" just influence the fact that a character is visible in the view or not, they do not modify the position of the document's characters stored in the model. Consequently, an operation transformed against any "Undel" (or "Del") operation is not modified.

```
T( Ins(p1, c1, sid1),Undel(p2, sid2)):
   return Ins(p1, c1, sid1)
end

T( Del(p1, sid1), Undel(p2, sid2)):
   return Del(p1, sid1)
end

T( Undel(p1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) return Undel(p1, sid1)
   else return Undel(p1 + 1, sid1)
end

T( Undel(p1, sid1), Undel(p2, sid2)):
   return Undel(p1, sid1)
end

T( Undel(p1, sid1), Del(p2, sid2)):
   return Undel(p1, sid1)
end
```

Since these transformation functions are bijective, they can easily be reversed and consequently allow us to apply our approach with integration algorithms as SOCT2, GOTO which require reversible transformation functions.

The last step is to prove the correctness of the previous transformation functions.

Using the proof environment VOTE [14], we have proven that our transformation functions verify the properties $TP1$, $TP2$ and TPC. The system specification given to the theorem prover Spike[3] can be reviewed and tested at the following url: `http://graveyard.sf.net/`.

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

4.3 Compensation as a "user undo"

The first step in applying the compensation framework is to define the compensation operations and their effect.

In this case, we assume that, when a user compensates a deletion, the expected effect is to see the deleted character even if it has been deleted many times. In other words, user's intention is: "make this character visible".

As a result, an insertion is compensated by an operation "Del(position, id, sid)" which makes the character invisible. A deletion is compensated by making the character visible. Since no operation achieves this effect, we add a new operation "Undel(position, id, sid)".

Compensating an operation "Undel(position, id, sid)" consists in making the character invisible, then we use the operation "Del(position, id, sid)". The field "id" is an integer which indicates whether the operation has an effect or not. If "id" is superior to 0, the operation has no more effect since a concurrent operation has already achieve the expected effect.

Then we define the function C:

$C(op)$:
IF $op = Ins(p, c, sid)$ THEN $C(op) := Del(p, 0, sid)$
IF $op = Del(p, i, sid)$ THEN $C(op) := Undel(p, i, sid)$
IF $op = Undel(p, i, sid)$ THEN $C(op) := Del(p, i, sid)$

The second step is to write transformation functions for all operations (Figure 8[4]).

Figure 9 illustrates the behavior of this *user undo*. This scenario is similar to Figure 7 and emphasizes the difference between *system undo* and *user undo*. Site1 and Site2 generate concurrently an operation to delete the character 'a'. When Site1 cancels his deletion by generating an operation "$Undel(0, 0, sid_1)$", the user at this site expects to see the character 'a'. After integrating remote operations, Site1 and Site2 are in the same state and the user at Site1 sees the expected result.

The last step consists in proving the correctness of our transformation function.

Using the proof environment VOTE [14], we have proven that our transformation functions verify the properties $TP1$ and $TP2$. The system specification given to the theorem prover Spike can be reviewed and tested at the following url : `http://graveyard.sf.net/`.

In this approach, we define the following intentions for the operations :

"Ins" The order relationships between characters must be preserved. In [13], the authors prove that the TTF functions preserve these order relationships. The operations "Del" and "Undel" only modify the visibility of a character, not the position. Therefore, our approach preserves these relationships.

"Del" The intention of the "Del" operation is to obtain the character invisible. From the definition of the execution of "Del", $del(p, i, sid)$ has an effect only if $i = 0$. When generated by the user, we get $i = 0$, and the effect is realized. When transformed,

[4]Some transformation functions are similar to TTF functions and hence, are not reported on this figure.

```
T( Ins(p1, c1, sid1), Del(p2, i2, sid2)):
   return Ins(p1, c1, sid1)
end

T( Ins(p1, c1, sid1), Undel(p2, i2, sid2)):
   return Ins(p1, c1, sid1)
end

T( Undel(p1, i1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) return Undel(p1, i1, sid1)
   else return Undel(p1 + 1, i1, sid1)
end

T( Del(p1, i1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) return Del(p1, i1, sid1)
   else return Del(p1 + 1, i1, sid1)
end

T( Del(p1, i1, sid1), Del(p2, i2, sid2)):
   if (p1 = p2 and i2 = 0) return Del(p1, i1 + 1, sid1)
   else return Del(p1, i1, sid1)
end

T( Del(p1, i1, sid1), Undel(p2, i2, sid2)):
   return Del(p1, i1, sid1)
end

T( Undel(p1, i1, sid1), Undel(p2, i2, sid2)):
   if (p1 = p2 and i2 = 0) return Undel(p1, i1 + 1, sid1)
   else return Undel(p1, i1, sid1)
end

T( Undel(p1, i1, sid1), Del(p2, i2, sid2)):
   return Undel(p1, i1, sid1)
end
```

Figure 8: TTF transformation functions with a "user undo"

$i = 0$ remains true except if the $del(p, i, sid)$ operation is transformed against another $del(p, 0, sid')$, i.e. if the intention is already realized.

"Undel" The intention of the "Undel" operation is to obtain the character visible. Since the definitions of "Del" and "Undel" are the symmetric, the intention of "Undel" is also respected.

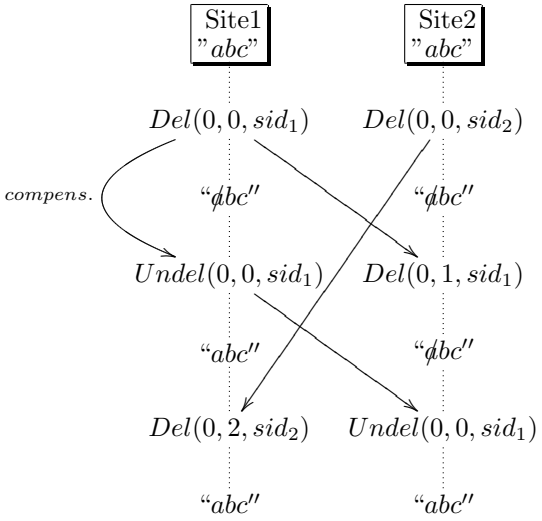

Figure 9: User undo

4.4 TPC and user undo

In the *system undo* strategy, we assume that the expected undo effect is to remove an operation from the system. On the contrary, in the *user undo* strategy, we remove an effect from the document. Thus TPC may contradict user intention in the case of a user undo.

An example of this contradiction is show in figure 10[5]. In this example, two users delete concurrently intersecting parts of the document "abc". User1 deletes "ab" while User2 deletes "bc".

- For User2, the observed effect of User1's operations is only to remove "a"; and if he undoes these operation, he wants only "a" to appear.
- For User1, the effect of his actions is obviously to remove "ab"; and if he undoes them, he wants "ab" to appear.

So, two user's undo intention of the same operation could lead to two different operations. Thus, TPC contradicts users' intentions in this case.

5 Implementation

In order to validate our approach, we have built the Graveyard prototype. Graveyard is a real-time collaborative text editor (cf. figure 11). It relies on the SOCT2 algorithm for

[5]This scenario does not lead to inconsistency, since sites will eventually converge when both set of undelete operation will remotely be integrated.

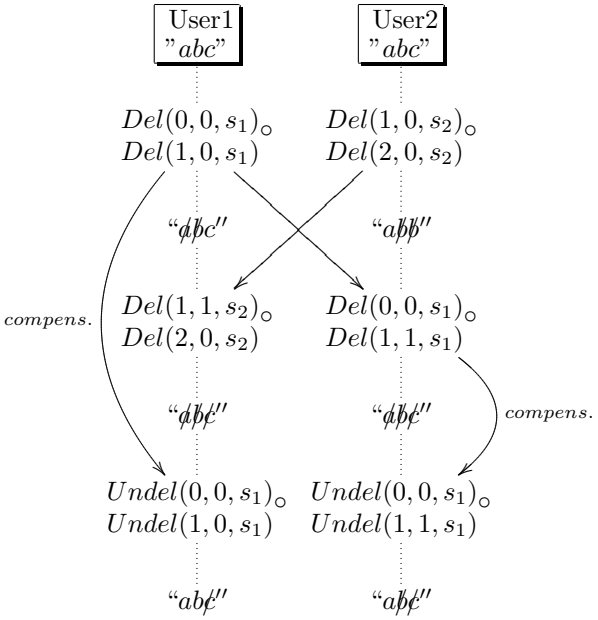

Figure 10: TPC and user undo

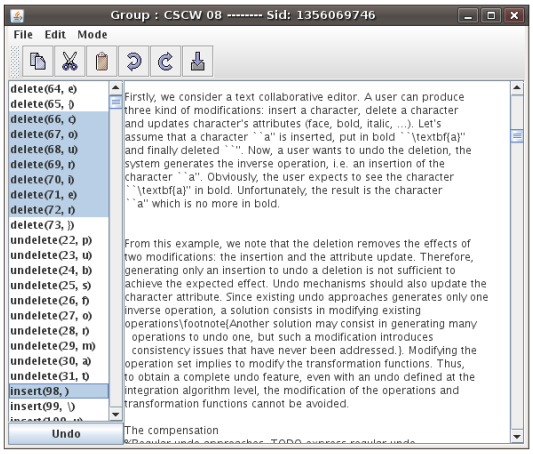

Figure 11: Graveyard real-time editor

integrating concurrent operations. SOCT2 does not provide natively undo capabilities. We used the TTF transformation functions with related compensation operations to obtain a real-time collaborative with undo feature. The general architecture of graveyard is described

in figure 12. For this implementation, we used SOCT2 however we can replace SOCT2 by SOCT4, MOT2, adOPTed or GOTO and obtain the same result.

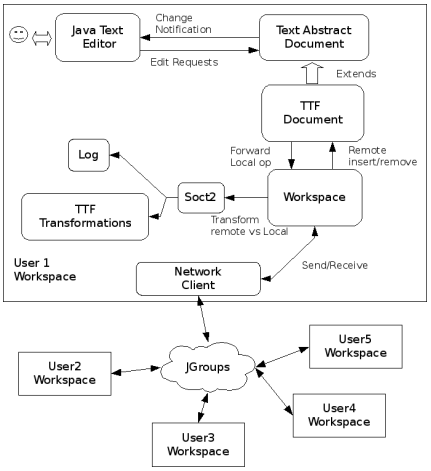

Figure 12: Graveyard architecture

The TTF transformation functions require to change the data model of the editor. Fortunately, the Model-View-Controller architecture of the Java Swing Framework [6] offers this functionality. Programmers can plug-in their own data model into the abstract document model of the text editor. The Eclipse RCP framework [7] offers the same functionality. By this way, the programmer model is notified of all changes requested by the controller i.e. typing a character, but also copy, paste and undo request.

So, we wrote the TTF document that extends the Swing text editor abstract document. This document manages invisible characters when inserting and removing characters. This model also forward local operations to the other connected editors through the workspace manager.

The workspace manager sends local operations to other sites and handles remote operations reception. When a remote operation is received and is causally ready, this operation is transformed against concurrent operations using the SOCT2 algorithm and TTF transformation functions. The resulting operation is executed on the TTF document. This update is notified to the swing text editor. For managing membership and network broadcast, we use the JGroups [8] toolkit. This toolkit allows us to test various broadcast protocols.

The undo feature is accessible in graveyard with traditional keys. By typing 'control-z', the user can undo his operations that are not always the last executed operations. The undo

[6] `http://en.wikipedia.org/wiki/Swing_(Java)`
[7] `http://www.eclipse.org/`
[8] `http://www.jgroups.org`

panel at the left (see figure 11) contains all operations integrated on the local site. The user can select any operation and compensate them by clicking on the button "Undo".

The graveyard prototype is available under the GPL license on sourceforge[9].

6 Related Work

In [22], the authors present an undo specific to the adOPTed algorithm by adding two functions called "mirror" and "fold". Unfortunately, this solution cannot allow undoing any operation at anytime. Since the adOPTed algorithm requires transformation functions satisfying $TP1$ and $TP2$, we can use the TTF functions in association with the compensation approach. Therefore, we can instantiate adOPTed and provide an undo for any operation.

The ANYUNDO algorithm [11] is associated with the GOTO integration algorithm. This approach introduces three undo properties called $IP1$, $IP2$ and $IP3$. The property $IP3$ is similar to the property TPC. The property $IP1$ illustrates the neutrality of do-undo pairs toward the document state while the property $IP2$ illustrates the neutrality of do-undo pairs toward transformation functions. The properties $IP2$ and $IP3$ are enforced by the ANYUNDO algorithm. Therefore, the GOTO-ANYUNDO approach needs transformation functions which satisfy three properties $TP1$, $TP2$ defined for operation satisfying $IP1$. Unfortunately, transformation functions satisfying $IP2$ and $IP3$ defined for operations satisfying $IP1$ have never been published. The TTF approach satisfies $TP1$ and $TP2$ but is defined for operations which do not satisfy $IP1$. However, we can still provide compensation in GOTO. Operations and transformation functions described in this paper can be used in the GOTO integration algorithm to provide either a *system undo* or a *user undo* without using the ANYUNDO algorithm.

In [10], 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.

In the COT approach [12], an undo operation is defined on the generation state of the operation that we want to undo. The state is represented by a "context vector" associated to each sent operation. Unlike state vectors which only capture normal operations , context vectors are designed to capture also undo operations. Unfortunately, context vector's size grows linearly with the number of undone operations. Therefore, the use of context vector limits the number of undo which can be performed during an editing session. One can use the compensation approach with the COT's integration algorithm. This allows obtaining an undo mechanism in the COT approach without using "context vectors".

From a generic point of view, existing undo mechanisms place into the operation log undo operations directly next to original one in order to form do-undo-pairs [22, 11, 10, 12]. This

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

behavior limits the supports of log truncation features. Indeed, if the original operation is dropped, undo operation cannot be integrated. Since compensation operations are treated as do operation, every operation in the log of the local site can be compensated and remotely integrated even if it has been dropped by the other sites.

From a correctness point of view, the existing undo mechanisms rely on some properties that the transformation functions must ensure or that the undo mechanism enforces. These properties are designed to resolve some known undo puzzles. Mechanisms which enforce such properties could break causality, convergence or intentions. However, since compensation operations are integrated as other operations, their correctness rely directly on the CCI criteria:

- Causality is ensured by the integration algorithm
- Convergence is ensured by the respect of TP1 and, if required, TP2.
- Intention must be ensured by the execution of the operations and their transformation as well for compensation operation as for standard operation[10]. Unfortunately, OT framework does not provide a generic scheme to show that an operation's intention is respected. This issue must be addressed for each set of operations as we done for the instances of compensation for TTF.

However, compared to undo built at the integration level, the compensation approach has a drawback. The compensation implies an additional development cost on account of the addition of new operations. But, this additional cost cannot be avoided if original operations are not inversible – such as TTF operations – or if the original set of operations is not sufficient to produce a complete undo – such as the "insertFormatted" example. This cost is mostly a property verification cost and can be reduced thanks to automated verification techniques [**?**].

Finally, compared to the above approaches, the use of the compensation approach brings several improvements: it improves the performances, extends the undo functionality add/or even replaces a non-instantiable undo mechanism. One main characteristic of compensation approach is to realize the undo feature at the transformation functions level. It makes the undo feature independent of the integration algorithm. So the compensation approach can be used with any integration algorithm.

7 Conclusions

In all existing OT approaches, undo is designed at the integration algorithm level and proposes a *system undo*. In this paper, we introduced our compensation framework which provides the *system undo* and, furthermore, a new way to undo called *user undo*. The *user undo* allows expressing the user's intention undo [3, 23].

[10] Even with traditional undo properties (IP2/IP3 or C3/C4) or our property TPC, intention preservation is never formally proved to be ensured in any case.

With a little additional design cost – that may not be avoided to build a correct undo – our framework allows defining any adequate action to counterbalance the effect of an operation. Since *system undo* is just one of these possible actions, we claim that the compensation is more generic than existing undo approaches. 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. We have a complete solution to build correct decentralized text editors with a flexible undo capability and a log truncation feature. The TTF transformation functions with the *system undo* proposed in this paper has been implemented in the Graveyard collaborative text editor.

In future work, we will integrate the TTF transformation functions with the *user undo*. Thus, a user study will be settled to determine which undo strategy is the most adequate depending on the context. The major drawback of the compensation as a *user undo*, which is shared with the OT approach in general, is the lack of a formal definition and generic properties to ensure user's intention. As a result, we will try to obtain a formalization of user's intention.

References

[1] S. G. Tammaro, J. N. Mosier, N. C. Goodwin, and G. Spitz, "Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing," *Computer-Supported Cooperative Work - JCSCW*, vol. 6, no. 1, pp. 19–51, March 1997.

[2] S. Noël and J.-M. Robert, "Empirical study on collaborative writing: What do co-authors do, use, and like?" *Computer Supported Cooperative Work - JCSCW*, vol. 13, no. 1, pp. 63–89, March 2004.

[3] G. D. Abowd and A. J. Dix, "Giving undo attention." *Interacting with Computers*, vol. 4, no. 3, pp. 317–342, 1992.

[4] T. Berlage and A. Genau, "A framework for shared applications with a replicated architecture." in *ACM Symposium on User Interface Software and Technology*, 1993, pp. 249–257.

[5] A. Prakash and M. J. Knister, "A framework for undoing actions in collaborative systems." *ACM Trans. Comput.-Hum. Interact.*, vol. 1, no. 4, pp. 295–330, 1994.

[6] R. Choudhary and P. Dewan, "A general multi-user undo/redo model." in *ECSCW*, 1995, pp. 229–246.

[7] 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 (TOCHI)*, vol. 5, no. 1, pp. 63–108, Mars 1998.

[8] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems." in *SIGMOD Conference*, J. Clifford, B. G. Lindsay, and D. Maier, Eds. ACM Press, 1989, pp. 399–407.

[9] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors." in *CSCW*, 1996, pp. 288–297.

[10] J. Ferrié, N. Vidot, and M. Cart, "Concurrent undo operations in collaborative environments using operational transformation." in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, ODBASE 2004*, ser. Lecture Notes in Computer Science, vol. 3290. Springer, Novembre 2004, pp. 155–173.

[11] C. Sun, "Undo as concurrent inverse in group editors," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 9, no. 4, pp. 309–361, Dcembre 2002.

[12] D. Sun and C. Sun, "Operation Context and Context-based Operational Transformation," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Banff, Alberta, Canada: ACM Press, Novembre 2006, pp. 279–288.

[13] G. Oster, P. Urso, P. Molli, and A. Imine, "Tombstone transformation functions for ensuring consistency in collaborative editing systems," in *The Second International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*. Atlanta, Georgia, USA: IEEE Press, November 2006.

[14] A. Imine, P. Molli, G. Oster, and P. Urso, "Vote: Group editors analyzing tool: System description." *Electr. Notes Theor. Comput. Sci.*, vol. 86, no. 1, 2003.

[15] M. Suleiman, M. Cart, and J. Ferrié, "Concurrent operations in a distributed and mobile collaborative environment," in *Proceedings of the fourteenth International Conference on Data Engineering - ICDE'98*. Orlando, Floride, tats-Unis: IEEE Computer Society, Fvrier 1998, pp. 36–45.

[16] L. Lamport, "Time, clocks, and the ordering of events in a distributed system." *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[17] C. Sun and C. A. Ellis, "Operational transformation in real-time group editors: Issues, algorithms, and achievements." in *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*. New York, New York, tats-Unis: ACM Press, Novembre 1998, pp. 59–68.

[18] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman, "Copies convergence in a distributed real-time collaborative environment." in *CSCW*, 2000, pp. 171–180.

[19] C. Sun, "Undo any operation at any time in group editors." in *CSCW*, 2000, pp. 191–200.

[20] R. Nieuwenhuis, Ed., *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2706. Springer, 2003.

[21] S. Stratulat, "A general framework to build contextual cover set induction provers." *J. Symb. Comput.*, vol. 32, no. 4, pp. 403–445, September 2001.

[22] M. Ressel and R. Gunzenhäuser, "Reducing the problems of group undo." in *GROUP*, 1999, pp. 131–139.

[23] H. Garcia-Molina and K. Salem, "Sagas." in *SIGMOD Conference*, U. Dayal and I. L. Traiger, Eds. ACM Press, 1987, pp. 249–259.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399