Compensation in Collaborative Editing

Stéphane Weiss, Pascal Urso, Pascal Molli

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Compensation in Collaborative Editing

Stéphane Weiss — Pascal Urso — Pascal Molli

N° 6160 — version 2

version initiale Avril 2007 — version révisée Mai 2007

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--6160--FR+ENG

INRIA
LORRAINE

Compensation in Collaborative Editing

Stéphane Weiss , Pascal Urso , Pascal Molli

Thème COG — Systèmes cognitifs
Projet ECOO

Rapport de recherche n° 6160 — version 2 — version initiale Avril 2007 — version révisée Mai 2007 — pages

Abstract: In order to support users to recover from erroneous changes or to explore previously executed modifications, editing systems require an undo mechanism. In collaborative editors, users are allowed to revert any changes performed by any user, anytime. Operational transformation has been devised as a suitable mechanism for maintaining consistency in collaborative editing systems. Therefore, in this paper we present a novel undo approach in the context of operational transformation mechanism. Our approach is based on the notion of compensation used in databases where compensating operations semantically undo other operations. Moreover, our compensation mechanism is less restraining than any undo approaches and is generic in the sense that it can be used in association with any existing operational transformation algorithm.

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

Compensation 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 totalement décentralisé 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. Nous présentons, dans cet article, une nouvelle approche pour l'annulation dans le contexte des transformées opérationnelles. Notre approche est basée sur la notion de compensation utilisée dans les bases de données. Dans cette approche, les opérations de compensation annulent sémantiquement les effets des autres opérations. Notre mécanisme de compensation est moins contraignant que les autres approches pour l'annulation. Notre approche est également génerique du fait qu'elle peut être utilisée en association avec n'importe quel algorithme de transformation opérationnelle.

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

Categories and Subject Descriptors
C2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems
; H5.3 [**INFORMATION INTERFACES AND PRESENTATION**]: Group and Organization Interfaces
; I7.1 [**DOCUMENT AND TEXT PROCESSING**]: Document and Text Editing

General Terms
Algorithms,Verification,Reliability

Keywords
CSCW, Collaborative editing, undo, compensation, 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].

Undo/Redo has been recognized as an important feature of collaborative editing systems [3, 4, 5, 6, 7]. In collaborative 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) [4, 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.

All undo algorithms proposed in the OT framework [9, 10, 11, 12] provide an undo feature as a rollback of previous modifications. Such approaches assume there is an inverse operation, associated to each operation affecting the document, that makes the document return syntactically to a previous state. This kind of recovery, sometimes called backward error recovery, is not always possible. For instance, when a bookkeeper validates an operation, he cannot cancel this operation. If he makes a mistake, he has to compensate his operation by generating a new operation that semantically undoes the previous one. The resulting state is not equal to initial state.

This concept is well known in the database community as forward error recovery or compensation [13]. The main idea of our own approach is to execute an adequate compensating modification instead of rolling back the erroneous modification. This choice have two main advantages.

- In essence, the compensation approach includes the rollback approaches. As we can define the compensating effect in a way that it rolls back the modification, we can consider that backward recovery is a particular case of compensation. In addition, this approach can provide a recovery mechanism even in a system which does not allow rollback of modifications.
- Since compensation is a forward error recovery, the compensation operations are treated as any other newly produced operations. There is no specific mechanism required to handle such operations when received on remote sites. This simple scheme can be introduced in all existing OT algorithms including GOTO [11], COT [12], adOPTed [14]. It can provide undo capabilities even to algorithms that do not provide this feature natively as SOCT2 [15], and SOCT4 [16].

As a proof of concept, we applied the compensation framework to the Tombstone Transformation Functions (TTF) [17] which handles non inversible operations. The correction of the obtained model is proved formally by the automated proof environment VOTE [18].

Based on this approach, we build the Graveyard real-time editor prototype. Graveyard com-

bines a SOCT2 algorithm that does not provide a native undo feature, with TTF transformation functions extended with compensation operations. Finally, we obtain a reliable fully-decentralized collaborative editor with an undo mechanism.

qIn this paper, we detail our framework for compensation. We first describe the OT approach and its correctness criteria in Section 2. We then present the algorithm generating compensation operations and the properties required on the transformation functions in Section 3. We show that our approach can be integrated in well-known OT integration algorithms in Section 3.2. We also show how this framework is applied on the tombstone transformation functions approach including the proof of required properties and an implementation in Section 4. Finally, we compare our approach with existing backward recovery approaches in Section 5.

2. THE 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 that have to be integrated for execution.

In the Ressel Model [9], the system is correct if it preserve Causality and Convergence.

Description Considering two operations op_1 and op_2, operation op_1 is said to *precede* op_2 if and only if op_2 is generated on a copy after op_1 was executed on this copy. Subsequently, op_2 may depend on effects of execution of op_1. *Causality preservation* criterion ensures that all operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [19], will be executed in the same order on every copy. Two operations op_1 and op_2 that are not related by a precedence relation (neither op_1 precedes op_2, nor op_2 precedes op_1) are said to be concurrent.

Convergence The system converge if all copies are identical when the system is idle.

For example, we consider two sites sharing the same text document (Figure 1). 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. If Site1 and Site2 directly execute the 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.

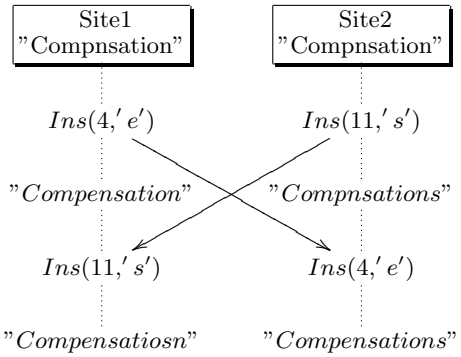

Figure 1: Divergence scenario

However, defining a transformation function is not sufficient to ensure convergence. In the OT approach, the correctness is based on two standard properties TP_1, TP_2. Some algorithms only requires TP_1 such as SOCT4 [**?**] or COT [12],

others require TP_1 and TP_2 such as Adopted [9], GOTO [20], SOCT2 [15].

The transformation property TP_1 defines a *state equality*. The state obtained by the execution of an operation op_1 on the initial 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 the initial state S followed by the execution of $T(op_1, op_2)$:

$$TP_1 : S \circ op_1 \circ T(op_2, op_1) = S \circ op_2 \circ T(op_1, op_2)$$

The property TP_2 ensures that the transformation of an operation against a sequence of operation does not depend on the transformation order of the operation in the sequence.

$$TP_2 : T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$$

3. COMPENSATION APPROACH

Existing undo methods for OT enforce the document to return to a previous state after the execution of an undo operation. The compensation approach do not require the system returns to a previous state but only to a state semantically equal to a previous state. However, in some cases, this semantically equal state can be syntactically equal to the previous state.

We call $C(op)$ the operation which compensates op. The execution of the operation $C(op)$ undoes, from a semantic point of view, the effect of the operation op. $C(op)$ is not necessary the inverse operation of op but it is defined in order to compensate op just after the execution of 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 semantically undone.

On Figure 2, a site generates two operations op_1 and op_2. Now, it wants to compensate the last executed operation which is here op_2. Using a function C, we determine the operation $op_3 = C(op_2)$ which compensate op_2. For this example, we chose $C(Ins(p, c)) = Del(p)$.

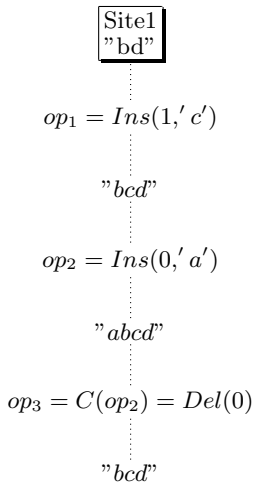

Figure 2: Compensation of a last executed operation

We define the effect of compensating operations on the state obtained after the execution of the operation to be compensated. To compensate an operation op just after its execution, we generate an operation $C(op)$. The operation $C(op)$ will compensate the operation op if op is the last executed operation.

In order to compensate any operation, and not only the last executed operation, we have to transform the compensation operation. On Figure 3, the initial state of the document is "bd". First we add a 'c' between 'b' and 'd'. Then, we add a 'a' at the beginning of the document. Now we want to compensate the operation op_1 which is not the last executed operation. The compensation of the insertion of a 'c' is obviously the deletion of this character. Unfortunately, the direct execution of the operation $C(op_1)$ deletes the characters 'b'.

Indeed, the operation $C(op_1)$ does not take account of the effect of the operation op_2 executed after op_1. We need to compute an oper-

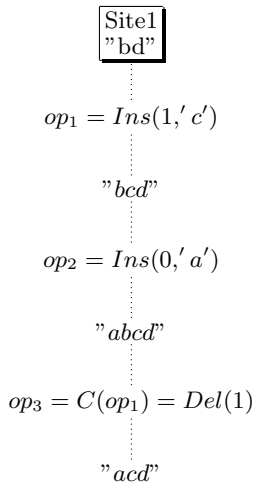

Figure 3: Wrong compensation of a non-last executed operation

ation $C(op)'$ which realize the effect of $C(op)$ on the current state.

To compute the operation $C(op)'$, we need the following algorithm (also illustrated by the Figure 4).

1. First, we determine the operation $C(op)$ which compensates op. $C(op)$ should have compensated op if op was the last operation executed on the current site.
2. We use the forward transformation functions in order to transform $C(op)$ with all operations which have already been executed on this site. The resulting operation is called $C(op)'$. $C(op)'$ is defined on the current state and will be send to other sites where it will be integrated as any other operation.

This algorithm is known as the naive algorithm for undo [10].

We need to ensure that the effect of $C(op)'$ is the same as $C(op)$ if op was the last executed

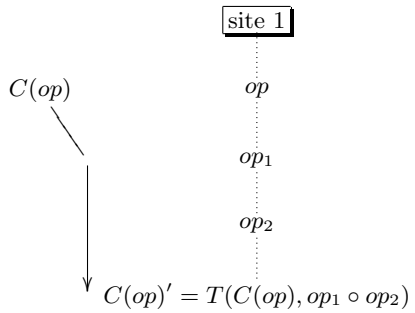

Figure 4: Algorithm of the compensation

operation. In OT approach, correctness is ensured by a set of properties that must be satisfied by the transformation functions. Since integration algorithms make no distinction between compensating operations and other operations, compensating operations have also to satisfy the correctness properties.

3.1 Correctness of the compensation approach

The compensation approach does not depend of the integration algorithm. It is defined at the transformation functions level. In order to be correct, the transformation functions defined for regular and compensation operations have to ensure:

- at least TP_1 or TP_1 and TP_2 if TP_2 is required.
- TP_c to ensure the respect of compensation. These property ensures that the compensation effect will always 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, 21, 22].

  The compensation effect is defined on the state obtained by the execution of the compensated operation. This effect should be preserved if the operation we want to compensate is not the last. The property

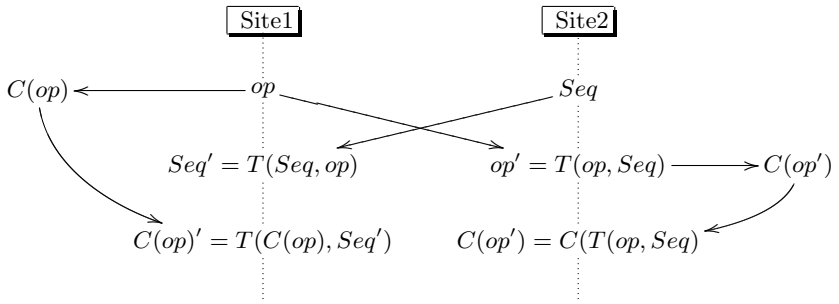

Figure 5: Respect of the compensation effect

TP_C expresses that the effect defined on a state will not be modified by concurrent operation. The condition Tp_c is formally defined as:

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

Figure 5 explains the TP_c 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 generate $C(op)$ and transform it through following operations $T(seq, op)$. Site2 compensate the last received operation which is $T(op, seq)$. These two compensating operations are defined on the same state, they compensate the same operation, so they 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 three – or two – 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 [23]).

On an other hand, each of the cases to check can be easily handle by a automated formal theorem prover. Consequently, we choose to use the proof environment VOTE [18] based on the theorem prover Spike [24, 25] which generate all the cases and ensure the verification of all properties. If property are violated, the environment return the found counter-example as a scenario. Such scenario are very useful to correct the problematic transformation function. If the theorem prover succeeds, the obtained proof is theoretically indisputable. But this proof is based on a hand-written specification. Thus, a claim that a system is automatically proved must be associated with the specification for careful reading.

Of course, the usage of the VOTE environment in particular is not mandatory to validate the properties, and this verification can be done by any other suitable formal method or either by a hand-proof if the system designer is enough confident.

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.

3.2 Integrating the compensation in existing integration algorithm

In the OT framework, integration algorithm (SOCT2, SOCT4, GOTO, COT) are defined for operations (with no assumption about the number or the kind of operation) 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 to obtain a new set of operations and transformation functions. Consequently, the resulting set can easily be used with existing integration algorithm.

Our framework also requires a compensation algorithm. To compensate an operation *op*, we generate a compensating operation $C(op)$. The compensation algorithm transforms $C(op)$ with 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 algorithms is to transform an operation against a set of concurrent operations. For instance :

- In the adOPTed [14] algorithm, we can call the *Translate Request* method with arguments $C(op)$ and the state vector of *op* to obtain $C(op)'$. We then need to remove $C(op)$ and its transformed versions from the model and send $C(op)'$ with the current state vector incremented as for a new operation.
- In the SOCT4 [16] algorithm, since the history of each site is ordered using a continuous global order, we just have to call the *Transpose Forward* method on $C(op)$ and all the operations following *op* in the history to obtain $C(op)'$.
- In the COT [12] algorithm, we can call, as the authors do for their undo mechanism, the *COT-DO* method with arguments $C(op)$, with the same "context vector" as *op*, and the current document state. On the other hand, we do not have to include $C(op)$ in the context vector of *op* and we send $C(op)'$ with the context vector of a new operation.

Consequently, any OT integration algorithms can determine the compensating operation. The resulting operation is treated as a normal operation. Thus, we can use the compensation algorithm with any integration algorithm.

In the following section, we instantiate the compensation approach on the Tombstone Transformation Functions (TTF). The TTF have the two particularities to handle non-inversible operations and to be the only transformation functions that ensure TP_1 and TP_2 [17].

4. COMPENSATION IN THE TTF APPROACH

The compensation approach could be applied to the same functions as the undo approach and for transformation functions defined on operations which are not inversible. In order to illustrate the compensation approach, we choose to apply it on the TTF functions. In the TTF approach, the operation "Ins" is not inversible and consequently, the traditional undo approaches could not be used.

To apply the compensation to a transformation function, we first need to define compensating operations. For each operation, we define an operation which compensate its effect. A compensating operation does not necessary exist in the initial set of operation. Using the definition of compensating operation, we can easily write the function C.

Afterward, we write the transformation functions for all operations. Finally, we must prove that our transformation functions verify the properties TP_1, TP_2 and TP_C.

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 [17].

The main idea of the model is to keep deleted characters as tombstones. The document's view only shows visible characters and tombstones are hidden. Consequently, the model differs from the view. Figure 6 illustrates this. Assume that a document is in a state "abcd". Now, 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$\not{b}$cd". Since tombstones are necessary to achieve consistency, they cannot be removed and thus, the operation "Ins" is not inversible.

The TTF transformation functions (Figure 7) can only be used with the TTF model. These functions are defined for two kind of operations: an operation "Ins(p, c, sid)" which inserts a character and an operation "Del(p, sid)" which deletes a character. Parameters of an insertion are the position p, the character c and the site identifier sid. Parameters of a deletion are the position p and the site identifier sid. Operations are computed according the model. Consequently, on Figure 6, on state "*a$\not{b}$cd*", to delete the character 'c' which has the position 3 in the model, the generated operation is "Del(3, sid)".

In other OT approaches, the deletion of a character decreases the position of all following characters. The TTF model's particularity is that a character's position can only grow. Therefore, transforming an operation with any operations "Del" will never modify it.

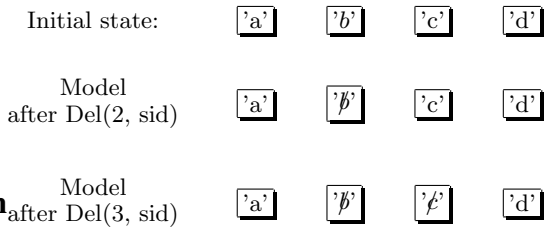

Figure 6: Model in the TTF approach.

```
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 7: TTF transformation functions

4.2 Defining compensating operations

As the TTF approach is defined for two operations: "Ins(p, c, sid)" and "Del(p, sid)", we need to find two operations to compensate them. The compensating effect required is to obtain that visible characters are the same (Figure 8).

Fortunately, it is obvious that there is no difference between compensating an insertion and deleting a character. So we can use the operation "Del" for compensating "Ins" since it has the desired effect which is removing the character in the user's view.

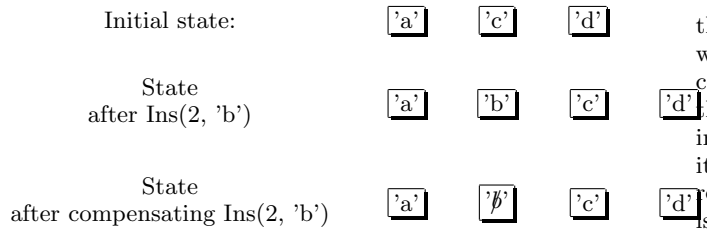

Figure 8: Compensating the operation Ins.

If we use the operation "Ins" to compensate an operation "Del", the transformation functions do not satisfy the property TP_C as shown in (Figure 9). This scenario was obtained with the proof environment VOTE.

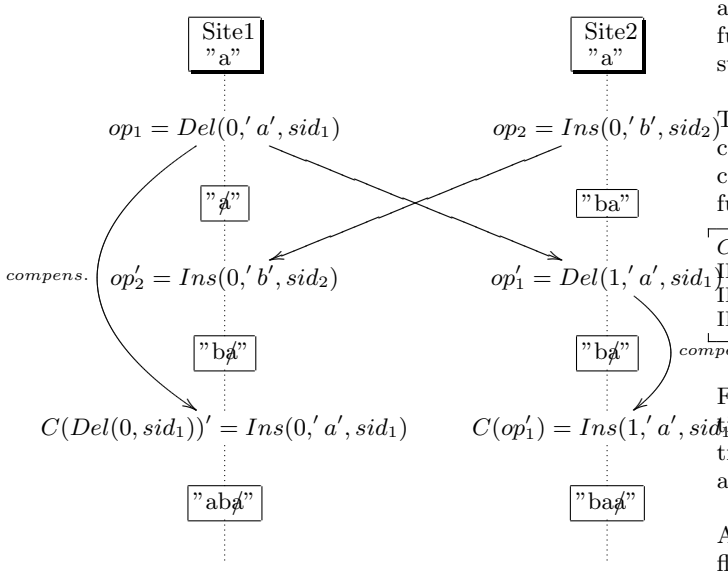

Figure 9: TP_C counter-example

Two sites share the same document "a" (Figure 9). Site1 deletes the character 'a' while Site2 inserts a character 'b' before the 'a'. After the execution of remote operations, each site owns the same document "b". As they are in the same state, they must generate the same compensating operation to compensate the same operation. Site2 compensates op'_1 which is the last executed operation. Site1 compensates op_1. Consequently, it generates the operation "$Ins(0,'a', sid_1)$" and, according to the compensation algorithm, transform it with the operation op_2. As $sid_1 < sid_2$, the resulting operation is "$Ins(0,'a', sid_1)$" which is not the same operation as the operation obtained by site 2. Consequently, the transformation functions do no verify the property TP_C[1].

Compensating the operation "Del" with the operation "Ins" inserts a new character in spite of the existing tombstone for this character. Thus, we define a new operation "Undel(position, sid)" to compensate the operation "Del". This operation replace the tombstone of a deleted character by the original character. This operation "Undel" allows to build transformation function which satisfy TP_C and to reuse tombstones.

The function $C(op)$ links normal operations to compensating operations. As we have defined compensating 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 $C(op) := Del(p, sid)$

Finally, we can write the transformation functions for all operations. The definition of the transformation functions for the operations "Ins" and "Del" are the same as presented in 7.

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 with any "Undel" (or "Del") operations is not modified.

[1]Consistency is not violated by the scenario even if the sites do not share the same state. Indeed, when local compensation operations will be sent and integrated, the sites will finally share the state "aba".

```
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) then
      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 the transformation functions are bijective, they can easily be reversed (Figure 10) and consequently allow us to apply our approach with integration algorithms as SOCT2, GOTO which require reversible transformation functions.

4.3 Correction of the approach

In the TTF approach, the transformation functions are written in order to satisfy the property TP_2. The property TP_1 is defined by a state equality, then we have to define the effect of the operation "Undel" on the state. If the effect of "Undel" is simply to make the character visible, the property TP_1 is violated (see Figure 11)

To ensure TP_1, we define the effect of "Undel" replacing the visibility flag associated to characters by a visibility level. This visibility level is an integer. Initially, a character inserted have a visibility level of 1. Each time an operation deletes this character, its visibility level is decreased. Each time an operation undeletes this character, we increase its visibility level.

```
T^-1( Ins(p1, c1, sid1), Undel(p2, sid2)):
   return Ins(p1, c1, sid1):
end

T^-1( Undel(p1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) then
      return Undel(p1, sid1)
   else return Undel(p1 − 1, sid1)
end

T^-1( Del(p1, sid1), Undel(p2, sid2)):
   return Del(p1, c1, v1, sid1)
end

T^-1( Undel(p1, sid1), Del(p2, sid2)):
   return Undel(p1, sid1)
end

T^-1( Undel(p1, sid1), Undel(p2, sid2)):
   return Undel(p1, sid1)
end
```

Figure 10: Reverse transformation functions TTF with compensation

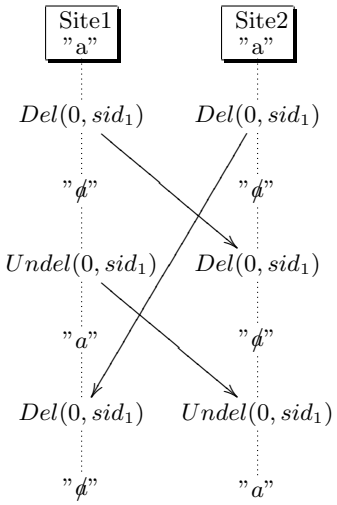

Figure 11: Violation of the property TP_1

A character is said "visible" and appears in the document's view if its visibility level is at least 1. And a character is said "invisible" and

does not appear in the document's view if its visibility level is less than 1.

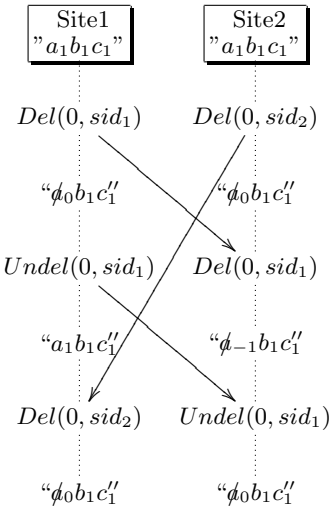

Figure 12: Visibility level

Figure 12, we assume that two sites share the same document containing a string "abc". We assume that the visibility level associated to each characters is 1. Site1 and Site2 generate concurrently an operation to delete the character "a". The visibility level is decreased on both sites. As the visibility level is less than 1, the character is not visible. When site2 receives site1's operation, site2 decreases the visibility level associated to the character "a". Site1 cancels his deletion by generating an operation "$Undel(0, sid_1)$". The execution of this operation increases the visibility level and the character 'a' is now visible. After integrating remote operations, site1 and site2 are in the same state and the visibility level associated to each characters is the same on both sites. This behavior is similar to the undo effect defined in [14].

Using the proof environment VOTE [18], we have proven that our transformation functions verify the properties TP_1, TP_2 and TP_C. The system specification given to the theorem prover Spike can be reviewed and tested at the following url : `http://potiron.loria.fr/projects/graveyard`.

4.4 Implementation

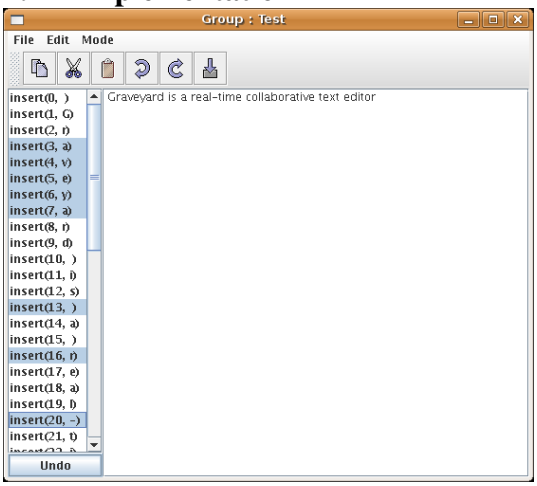

Figure 13: Graveyard real-time editor

In order to validate our approach, we have built the Graveyard prototype. Graveyard is a real-time collaborative text editor (cf. figure 13). It relies on the SOCT2 algorithm for 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 14

For this implementation, we used SOCT2 but we can replace SOCT2 by SOCT4, adOPTed or GOTO and obtain the same result.

The TTF transformation functions require to change the data model of the editor. Fortunately, the Model-View-Controller architecture of the Java Swing Framework [2] offers this functionality. Programmers can plug-in their own data model into the abstract document model

[2] `http://en.wikipedia.org/wiki/Swing_(Java)`

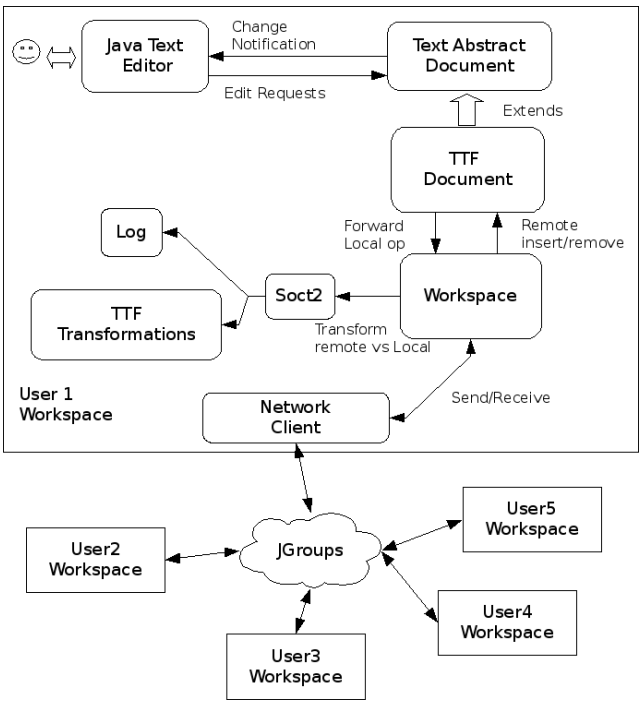

Figure 14: Graveyard architecture

of the text editor. The Eclipse RCP framework [3] offers the same functionalities. 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 others connected editors through the workspace manager.

The workspace manager send local operations to others sites and handle reception of remote operations. When a remote operation is received and is causally ready, this operation is transformed with concurrent operations using the SOCT2 algorithm and TTF transformation. 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 the the JGroups [4] toolkit. This toolkit allow 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 operation executed. The undo panel (see figure 13) contains all operations executed on the local site. The user can select any operations and compensate them by clicking on the button "undo".

[3] `http://www.eclipse.org/`

[4] `http://www.jgroups.org`

The graveyard prototype is available under GPL at `http://potiron.loria.fr/projects/graveyard`.

5. RELATED WORK

In [14], the authors present an undo specific to the adOPTed algorithm by adding two functions called "mirror" and "fold". Unfortunately, this solution cannot allow to undo any operation at anytime. Since the adOPTed algorithm requires transformation function satisfying TP_1 and TP_2, 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 introduce three undo properties called $IP1$, $IP2$ and $IP3$. The property $IP3$ is similar to the property TP_C. 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 TP_1, TP_2 and $IP1$. Unfortunately, transformation functions satisfying these five properties have never been published. TTF functions satisfy TP_1 and TP_2 but not $IP1$ but we still can provide compensation in GOTO. Operations and transformation functions described in this paper can be used in the GOTO integration algorithm to provide an 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.

For example, the transformation of an operation $undo(op_1)$ with an operation op_2 is defined

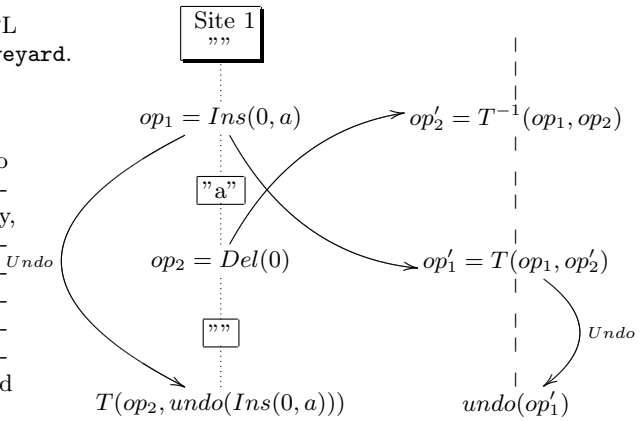

Figure 15: Transformation of an operation undo

by: $T(undo(op_1), op_2) = undo(T(op_1, T^{-1}(op_2, op_1)))$. Figure 15 explains this transformation function. The main idea is to swap the operations op_1 and op_2 and undo the resulting operation op_1'. Unfortunately, the authors do not discuss the case of causally dependent operations. In Figure 15, the operation op_2 is causally dependent of the operation op_1. The result is "Del(-1)" which is incorrect. These operations cannot be swapped. In addition, the inverse transformation of an undo operation cannot be determined without reordering the history buffer. This reordering is costly and depends on the history size. In the compensation approach, even compensating operations are operations and, consequently, the transformation of compensating operations does not have an additional cost.

In the COT approach [12], there is a specific undo mechanism. Undo operation are treated as "do" (i.e. normal) operations since the transformation path for an operation is chosen using "context vectors" that contains information about undone and redone operations. The COT's algorithm enforces the properties TP_2, $IP2$ and $IP3$ using a continuous total order on operations and the "context vectors". Thus, the COT do/undo approach can use transfor-

mation functions which only satisfy the property TP_1[5]. One can use the compensation approach with the COT's integration algorithm. This allows to obtain and undo mechanism with transformation functions only satisfying TP_1 but without using "context vectors" which space complexity is higher than "state vectors" usually used in OT approaches.

Finally, in the above approaches, the use of the compensation approach brings several improvements: it improves the performances, extends the undo functionality or even replaces a non-instanciable undo mechanism. The main characteristic of compensation approach is to realize the undo feature at the transformation function level. It makes the undo feature independent of the integration algorithm. So the compensation approach can be used with any integration algorithm. The approach of Vidot [10] propose the same idea with a generic undo operation. Unfortunately, all cases are not handled with the generic undo operation.

6. CONCLUSIONS

In all existing OT approaches, undo is designed as a backward error recovery. In this paper, we introduced our compensation mechanism designed as a forward error recovery [3, 13]. The compensation approach is more generic than the undo approach: we can apply compensation to all transformation functions even if some operations have no inverse. 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, GOTO and adOPTed. We have a complete solution to build fully-decentralized text editors with undo capabilities. The compensation approach proposed in this paper has been implemented in the Graveyard collaborative text editor based on the tombstone transformation approach.

In future works, we will implement existing integration algorithm in the Graveyard prototype. Thus, we will be able to benchmark all integration algorithms.

[5]Using TTF functions, COT will no longer requires such a total order. However, don't have the TP_2 property to check can be interesting.

7. 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] 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.

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

[6] 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.

[7] 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.

[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 and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 9, no. 1, pp. 1–41, Mars 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] H. Garcia-Molina and K. Salem, "Sagas." in *SIGMOD Conference*, U. Dayal and I. L. Traiger, Eds. ACM Press, 1987, pp. 249–259.

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

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

[17] 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.

[18] 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.

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

[20] 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.

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

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

[23] G. Oster, P. Urso, and P. Molli, "Proving correctness of transformation functions in collaborative editing systems," INRIA, Rapport de Recherche 5795, Dec. 2005.

[24] 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.

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

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