Compensation in Collaborative Editing

Stéphane Weiss, Pascal Urso, Pascal Molli

Compensation in Collaborative Editing

Stéphane Weiss, Pascal Urso and Pascal Molli
Nancy-Université
Loria
{*weiss,urso,molli*}*@loria.fr*

Abstract. In order to support users to recover from erroneous changes or to explore previously executed modifications, collaborative editing systems offer the undo mechanism. Providing an undo feature in fully decentralised collaborative editors is challenging as users are allowed to revert any changes performed by any user. 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 operational transformation algorithm.

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 (Tammaro, Mosier, Goodwin and Spitz, 1997; Noël and Robert, 2004).

Undo/Redo has been recognized as an important feature of collaborative editing systems (Abowd and Dix, 1992; Sun, Jia, Zhang, Yang and Chen, 1998). 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) (Ellis and Gibbs, 1989; Sun et al., 1998) 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.

The OT framework allows to build high responsiveness systems. To preserve this advantage, we need to build fully-decentralized editors as using a central server is obviously a bottleneck for global performance.

Many undo algorithms have been proposed in the OT framework (Ressel, Nitsche-Ruhland and Gunzenhäuser, 1996; Ferrié, Vidot and Cart, 2004; Sun and Chen, 2002; Sun and Sun, 2006). These algorithms are complex and assume underlying transformation functions satisfy certain properties. Unfortunately, such transformation functions have never been published. The COT algorithm (Sun and Sun, 2006) proposes a simpler solution but requires a total order on operations. This is a serious bottleneck for global performance. To our knowledge, none of the existing approaches allow to build fully-decentralized collaborative editors supporting undo features.

In this paper, we propose a new approach for undo based on compensation. All previous algorithms try to cancel an operation by restoring the state before the operation to be cancelled was executed. These algorithms can be considered as backward recovery approaches. On the contrary, we propose to undo an operation by executing a compensating operation. We do not restore the initial state but we compute a new state where the operation to be cancelled is semantically undone. Our compensation mechanism can be considered as a forward recovery approach. The advantages of a forward mechanisms are presented in what follows:

- Prior undo approaches suppose that an operation followed by its undo operation is neutral, i.e. if S is the initial state and op an operation executed on this state then $S \circ op \circ undo(op) = S$. In the compensation approach, this assumption is relaxed. For example, consider a string initialized to the empty string $[""]$ and an operation $ins(p, c)$ that inserts the character c at position p. In the backward recovery approach, it is mandatory that:

  $$[""] \circ ins(1,'a') \circ undo(ins(1,'a')) = [""]$$

  In the compensation approach, the following result is considered to be correct:

  $$[""] \circ ins(1,'a') \circ undo(ins(1,'a')) = ["\not{a}"]$$

  $["\not{a}"]$ represents a string where 'a' has been inserted then deleted. Note, that if from a system point of view, the two above solutions are different, they are equivalent from a user point of view as the final visualized state is the same – character is invisible. We can give another example in the bookkeeping domain. When a bookkeeper validate 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. We can consider that backward recovery is a particular case of compensation.

- A compensating operation is treated as a normal operation. It does not require any adaptation of the integration algorithm. Compensation approach can be used with all existing algorithms including GOTO, COT, adOPTed. It will provide undo capabilities even to algorithms that do not provide this feature natively as SOCT2, SOCT4.
- Compensation approach uses the naive undo algorithm. According to a naive undo algorithm, when an undo operation is performed the system generates the corresponding compensation operation, transforms it according to next executed operations and executes it.

In this paper, we describe our framework for compensation.We present the undo algorithm and the properties required on the transformation functions. We show how this framework was applied on the tombstone transformation approach (Oster, Urso, Molli and Imine, 2006). This implies to define compensating operations, write new transformation functions including these new operations and prove expected properties. The result is a correct set of transformation functions including compensating operations. This set can be used to build a fully decentralized collaborative text editor with undo capabilities. Finally we compare our approach with existing backward recovery approaches.

The OT approach

In the OT approach, shared documents are replicated. It allows any user to modify at any time his own copy of the document. Therefore, different copies of the same document can be modify in parallel. In the OT model, a modification is represented as an operation and a user is supposed to work at one site. Each site sends all its operations to the other sites. Remote operations have to be executed locally on each site. When all sites have received and integrated all operations, their copies of the document must be the same.

For example, we consider two sites sharing the same text document (Figure 1). The initial state of the document is “Compnsation”. On Site 1, a user wants to insert a character ’e’ to obtain “Compensation”. Concurrently, on Site 2, another user wants to insert a ’s’ at the end of the word. Each site sends its operation to the other. After the execution of the remote operation, site 1 and site 2 do not have the same document. On Site 1, the character ’s’ should have been inserted at position 12 instead of 11. In the OT model, a set of transformation functions T has to be devised in order to transform remote operations regarding concurrent operations.

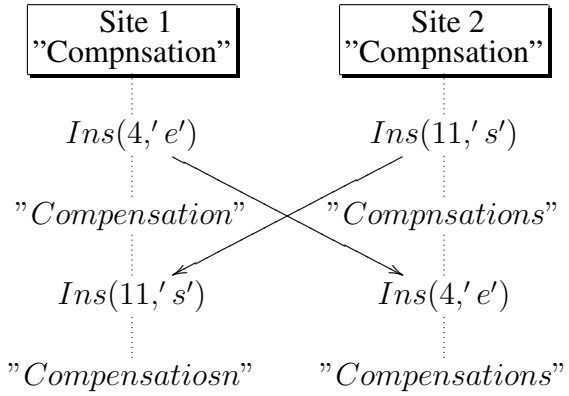

Figure 1. Divergence scenario

Compensation approach

In the compensation approach, the system does not return to a previous state but to a state semantically equal to a 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 have chosen $C(Ins(p, c)) = Del(p)$.

We define the effect of compensating operations on the state obtained after the execution of the operation we want to compensate. To compensate any operations, we use a simple algorithm.

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. Now, we want to compensate op if it is not the last executed 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 execution of the operation $C(op_1)$ deletes the characters ’b’.

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

To compute the operation $C(op)'$, we need the following algorithm (also illus-

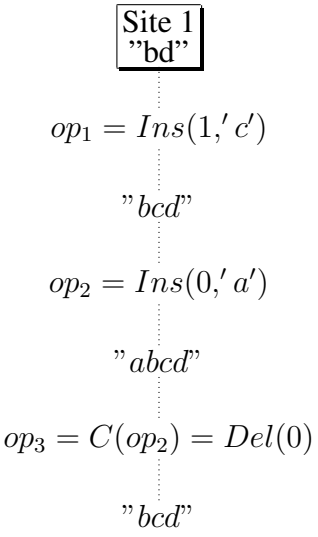

Figure 2. Compensation of a last executed operation

trated 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.

This algorithm is known as the naive algorithm for undo.

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

Correctness of the compensation approach

We consider a compensation correct if:

- it does not provide divergence,
- the effect defined for the compensation of the last executed operation is the same for any operation we want to compensate.

In the compensation approach, the correctness is based on the two standard properties TP_1, TP_2 and on a new property we called TP_C needed to ensure the respect

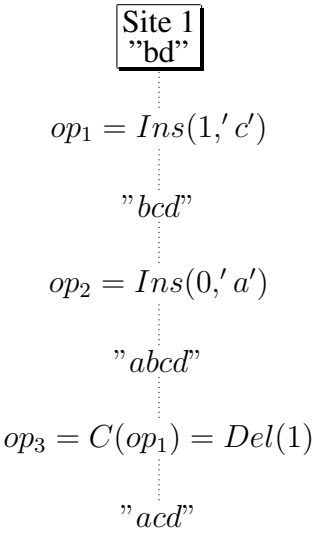

Figure 3. Naive compensation of a non-last executed operation

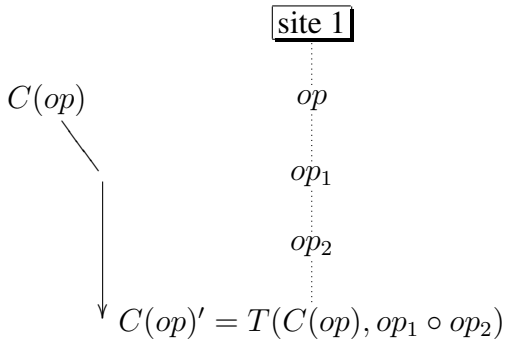

Figure 4. Algorithm of the compensation

of the compensation. Finally, we present a solution to verify that our transformation functions satisfy these properties.

Since compensating operations are treated exactly as normal operations, we simply have to verify TP_1 and TP_2 on the complete set of operations (Ressel et al., 1996).

(1) The transformation property TP_1 define 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 $T(op_1, op_2)$:

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

(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.

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

The verification of these properties will ensure convergence, we have now to be sure that the compensation effect we have defined will always be the same, even if the operation compensated is not the last executed one.

The compensation effect has been defined on the state obtained by the execution of the operation we want to compensate. This effect should be preserved if the operation we want to compensate is not the last. The property TP_C ensures that the effect defined on a state will not be modified by concurrent operation.

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

Figure 5 explains this property. Two site make concurrent operations. Site 1 generates op while site2 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. Site 1 generate $C(op)$ and transform it through following operations $T(seq, op)$. Site 2 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.

This condition is similar to the condition $C4$ (Ferrié et al., 2004) and the condition $IP3$ (Sun, 2000; Sun, 2002; Sun and Sun, 2006).

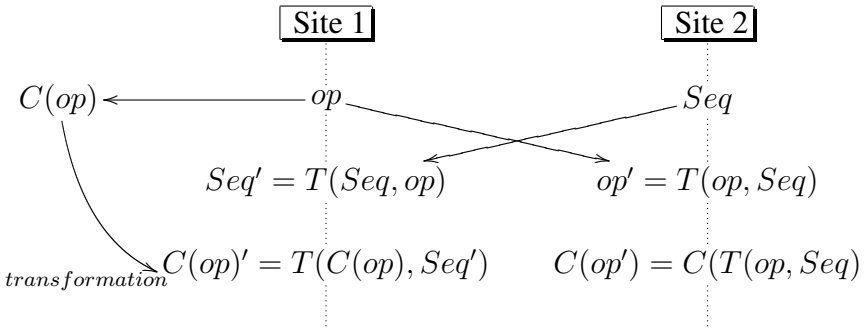

Figure 5. Respect of the compensation effect

The verification of this property ensures that whenever an operation is compensated, the compensation effect remains the same.

Now, we have three properties we need to verify to ensure a correct compensation system. We need an efficient way to prove that our transformation functions verify those properties.

To make a correct approach using compensation, we need to verify three properties. One of the particularity of the OT approach is the huge numbers of cases

to check. According to this remark, a hand proof is not the best solution. Consequently, we use the proof environment VOTE (Imine, Molli, Oster and Urso, 2003) based on the Spike theorem prover (Nieuwenhuis, 2003; Stratulat, 2001) to ensure 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. In the following section, we apply the compensation approach to the TTF functions.

Compensation in the TTF approach

We have designed the compensation framework and now, we want to instantiate it. This framework is more general than the undo approach. The undo approach could be applied to transformation functions which satisfy the properties TP_1 and TP_2 and which are defined for inversible operations. The compensation approach could be applied to the same functions as the undo approach and for transformation functions defined for 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 undo approach could not be used.

To apply the compensation to the TTF functions, we need first to define compensating operations. For each operation, we need to 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.

The Tombstones Transformation Functions

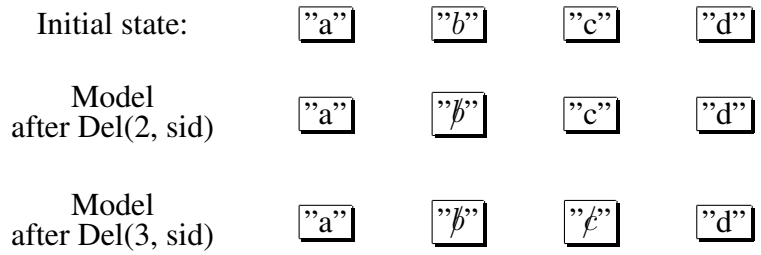

Figure 6. Model in the TTF approach.

The TTF approach is divided in two parts: the model and the transformation functions.

```
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

The main idea of the model is to keep deleted characters as tombstones. In the view of the document, we only show visible characters, tombstones are hidden. Consequently, the model differs from the view. Figure 6 illustrates this. Assume that a document is in a state "abcd". Now, we delete the character 'b'. In the TTF model, the character is replaced by a tombstone. The view differs from the model as the view only contains "acd" while the model contains "$a\not{b}cd$".

The TTF transformation functions (Figure 7) can only be used with the TTF model. The parameters of an insertion are the position p, the character c and the site identifier sid and the parameters of a deletion are only the position p and the site identifier sid. Operations are computed according the model. Consequently, on Figure 6, after the deletion of the character 'b', we need to generate the operation "Del(3, sid)" to delete the character 'c' as, in the model, it is in position 3.

Defining compensating operations

The TTF approach is defined for two operation: "Ins(position, character, sid)" and "Del(position, sid)". To apply the compensation approach, we need to define two operations to compensate operation Ins and Del. The compensating effect we want to obtain is 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.

If we use the operation Ins to compensate an operation Del, the transformation

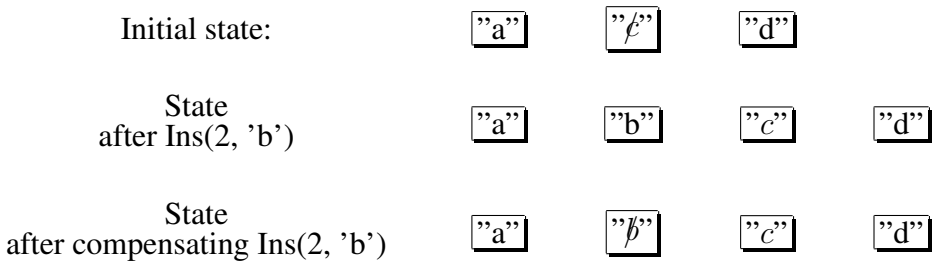

Figure 8. Compensating the operation Ins.

functions do not satisfy the property TP_C as shown by the scenario given by the proof environment VOTE (Figure 9).

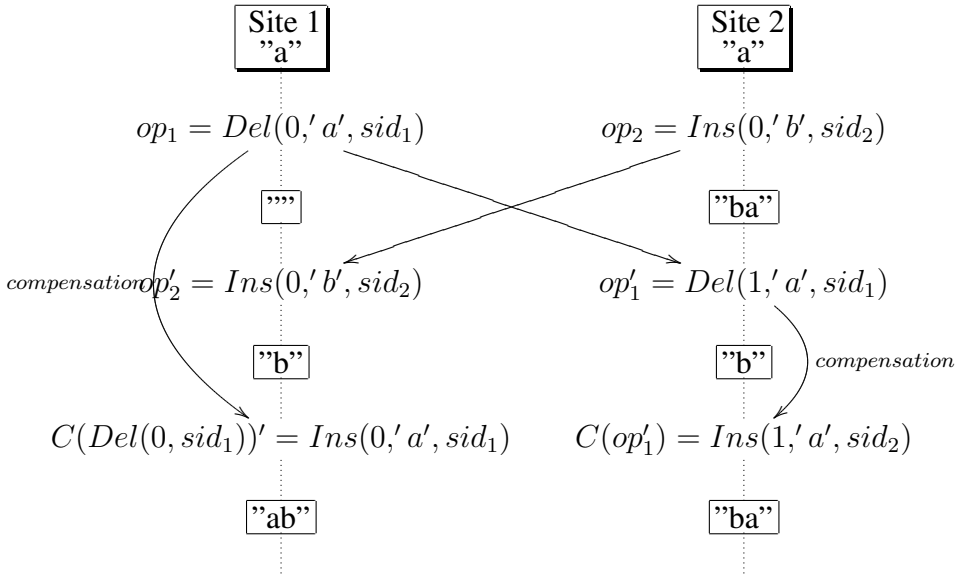

Figure 9. TP_C counter-example

Two sites share the same document "a" (Figure 9). Site 1 deletes the character 'a' while Site 2 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. Site 2 compensates op_1' which is the last executed operation. Site 1 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.

We define a new operation "Undel(position, sid)" to compensate the operation Del. Compensating the operation Del with the operation Ins inserts a new character in spite of the existing tombstone for this character. The operation "Undel" will al-

low us 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 operation Ins and Del is the same as defined in Oster et al. (2006).

```
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.

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. In 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 our operation using a compensation level associated to each characters.

This compensation level is an integer. Initially, a character inserted have a compensation level of 1. Each time an operation delete this character, its compensation

```
T^{-1}( Ins(p_1, c_1, sid_1), Undel(p_2, sid_2)):
   return Ins(p_1, c_1, sid_1):
end

T^{-1}( Undel(p_1, sid_1), Ins(p_2, c_2, sid_2)):
   if (p_1 < p_2) then
      return Undel(p_1, sid_1)
   else return Undel(p_1 - 1, sid_1)
end

T^{-1}( Del(p_1, sid_1), Undel(p_2, sid_2)):
   return Del(p_1, c_1, v_1, sid_1)
end

T^{-1}( Undel(p_1, sid_1), Del(p_2, sid_2)):
   return Undel(p_1, sid_1)
end

T^{-1}( Undel(p_1, sid_1), Undel(p_2, sid_2)):
   return Undel(p_1, sid_1)
end
```

Figure 10. Reverse transformation functions TTF with compensation

level is decreased. Each time an operation undelete this character, we increase its compensation level.

A character is said "visible" and appears in the document if its compensation level is at least 1. Obviously, a character is said "invisible" and do not appear in the document if its compensation level is less than 1.

Figure 12, we assume that two sites share the same document containing a string "abc". We assume that the compensation level associated to each characters is 1. Sites 1 and 2 generate concurrently an operation in order to delete the character "a". The compensation level is decreased on both sites. As the compensation level is less than 1, the character is not visible. When site2 receives site1 's operation, site2 decreases the compensation level associated to the character "a". Site1 cancels his deletion by generating an operation "$Undel(0, sid_1)$". The execution of this operation increase the compensation level and the character 'a' is now visible. After integrating remote operations, site1 and 2 are in the same state and the compensation level associated to each characters is the same on both sites. This behavior is similar to the undo effect defined in Ressel and Gunzenhäuser (1999).

Using the proof environment VOTE (Imine et al., 2003), we have proven that our transformation functions verify the properties TP_1, TP_2 and TP_C. We have also validated our approach by implementing the SOCT2 algorithm using these transformation functions.

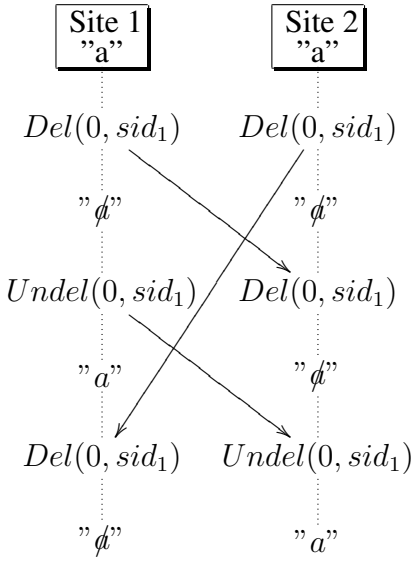

Figure 11. Violation of the property TP_1

Related Work

In Ressel and Gunzenhäuser (1999), 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. Using the transformation functions TTF in association with the compensation approach, we can instantiate adOPTed and provide an "undo" without using the functions "mirror" and "fold".

The ANYUNDO algorithm (Sun and Chen, 2002) 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. Even if our transformation function do not satisfy the five properties required by ANYUNDO algorithm, we can provide compensation in GOTO. All 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 Ferrié et al. (2004), 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 transfor-

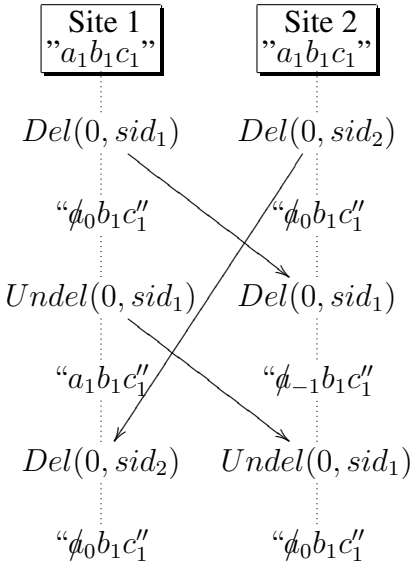

Figure 12. Compensation level

mation functions. This approach is more generic than the compensation approach, as it can be combined with all transformation functions and all integration algorithms. Unfortunately, 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 (Sun and Sun, 2006), undo operations are treated as other usual operations. A correct undo requires the verification of two properties called $IP2$ and $IP3$. As the ANYUNDO algorithm, the COT-UNDO algorithm enforces the verification of these properties. The COT approach requires transformation functions which only satisfy the property TP_1. Unfortunately, achieving consistency using such transformation functions requires a total order on the operations which degrades the global performance. We can use the transformation functions described in this paper to provide compensation in the COT-DO algorithm. This will allow to use the COT-DO approach without a total order.

Some approaches (Sun and Chen, 2002; Ferrié et al., 2004; Sun and Sun, 2006) consider that the verification of properties such as $IP2$ and $IP3$ is difficult. Consequently, they introduce some solutions to enforce the verification of these properties. In COT and ANYUNDO(Sun and Chen, 2002; Sun and Sun, 2006), the algorithm enforces these properties. In Ferrié et al. (2004), the authors introduce a new operation "undo(op)". In the compensation approach, we consider that the property $IP1$ is too restrictive. There is no need that the sequence "operation - compensating operation" is neutral. Consequently, the property $IP2$ makes no sense. The transformation functions only need to satisfy the properties TP_1, TP_2 and TP_C to

ensure a correct undo. We can easily verify these properties by using the proof environment VOTE (Imine et al., 2003).

Conclusions

In all existing OT approaches, undo is viewed as a backward error recovery. In this paper, we introduced our compensation mechanism viewed as a forward error recovery (Abowd and Dix, 1992; Garcia-Molina and Salem, 1987). 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 could apply these functions with COT-DO, SOCT2, 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 a prototype collaborative text editor based on the tombstone transformation approach.

References

Abowd, G. D. and Dix, A. J. (1992): 'Giving undo attention.', *Interacting with Computers* , vol. 4, no. 3, 1992, pp. 317–342.

Ellis, C. A. and Gibbs, S. J. (1989): 'Concurrency control in groupware systems.', in J. Clifford, B. G. Lindsay and D. Maier (eds.), *SIGMOD Conference*, ACM Press, pp. 399–407.

Ferrié, J., Vidot, N. and Cart, M. (2004): 'Concurrent undo operations in collaborative environments using operational transformation.', *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, ODBASE 2004*, Vol. 3290 of *Lecture Notes in Computer Science*, Springer, pp. 155–173.

Garcia-Molina, H. and Salem, K. (1987): 'Sagas.', in U. Dayal and I. L. Traiger (eds.), *SIGMOD Conference*, ACM Press, pp. 249–259.

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

Nieuwenhuis, R. (ed.) (2003): *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, Vol. 2706 of *Lecture Notes in Computer Science*, Springer.

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

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

Ressel, M. and Gunzenhäuser, R. (1999): 'Reducing the problems of group undo.', *GROUP*, pp. 131–139.

Ressel, M., Nitsche-Ruhland, D. and Gunzenhäuser, R. (1996): 'An integrating, transformation-oriented approach to concurrency control and undo in group editors.', *CSCW*, pp. 288–297.

Stratulat, S. (2001): 'A general framework to build contextual cover set induction provers.', *J. Symb. Comput.* , vol. 32, no. 4, September 2001, pp. 403–445.

Sun, C. (2000): 'Undo any operation at any time in group editors.', *CSCW*, pp. 191–200.

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

Sun, C. and Chen, D. (2002): 'Consistency maintenance in real-time collaborative graphics editing systems', *ACM Transactions on Computer-Human Interaction (TOCHI)* , vol. 9, no. 1, Mars 2002, pp. 1–41.

Sun, C., Jia, X., Zhang, Y., Yang, Y. and Chen, D. (1998): 'Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems', *ACM Transactions on Computer-Human Interaction (TOCHI)* , vol. 5, no. 1, Mars 1998, pp. 63–108.

Sun, D. and Sun, C. (2006): 'Operation Context and Context-based Operational Transformation', *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, ACM Press, Banff, Alberta, Canada, pp. 279–288.

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