Un modèle générique de Garbage Collection pour les éditeurs collaboratifs basé sur l'approche TO dans les environnements P2P et mobiles

Moulay Driss Mechaoui
Université de Mostaganem
Algérie
moulaydrissnet@yahoo.fr

Abdessamad Imine
Inria Lorraine et Université Nancy 2
France
imine@loria.fr

Fatima Bendella
Université USTO d'Oran
Algérie
Bendella_fatima@yahoo.fr

***Résumé*— L'approche de transformée opérationnelle (TO) est l'une des meilleurs techniques qui permet de supporter la collaboration dans les environnements mobiles et distribués. Les éditeurs collaborative en temps réel utilisent cette technique pour la réconciliation des données modifiées simultanément par plusieurs utilisateurs. Cependant, l'approche TO utilise un log qui peut atteindre une taille gigantesque au fur et à mesure que le nombre des opérations effectuées, et des utilisateurs augmente, cela épuise l'espace de stockage surtout si la collaboration est sur un dispositif mobile et diminue la performance de l'application de collaboration en terme de temps d'intégration des modifications. Pour résoudre ce problème, on propose un modèle de Garbage Collection GC distribué pour les éditeurs collaboratifs en temps réel dans les environnements paire à paire (P2P) et mobiles, ce modèle permet de nettoyer le log en conservant les propriétés de cohérence et de convergence des données partagés.**

Mots-clé: Editeur collaboratif en temps réel, Garbage Collection, transformée opérationnelle.

I. INTRODUCTION

Les éditeurs collaboratives en temps réel est une classe des systèmes distribués basée sur l'interaction de plusieurs utilisateurs tentant de modifier simultanément un objet partagé (texte, image, graphique), tels que des articles scientifiques, pages wiki, agenda distribué, et le code source d'un programme [1]. Ces éditeurs permettent d'établir des collaborations afin de parvenir à une tâche commune. Chaque site dispose d'une copie de l'objet partagé qui peut le modifier à volonté, et les modifications sont propagées pour être exécuter sur d'autres copies. Pour intégrer les changements effectués dans les autres sites, les éditeurs collaboratifs utilisent une approche asynchrone des transformées opérationnelle (TO) pour la sérialisation des transactions concurrentes [2].

L'approche de transformée opérationnelle TO est une technique optimiste qui a été proposée pour résoudre le problème de divergence [3]. Le modèle TO considère N sites, chaque site dispose d'une copie des objets partagés, quand un objet est modifié sur un site, l'opération est exécutée immédiatement et se propage vers d'autres sites pour être exécutée à nouveau. Chaque site sauvegarde toutes les opérations exécutées dans un tampon aussi appelé un journal (Log). Chaque opération est traitée en quatre étapes: (i) la génération sur le site local, (ii) la diffusion vers d'autres sites, (iii) la réception par d'autres sites, (iv) l'exécution sur d'autres sites. Le contexte d'exécution d'une opération *o* reçue peut être différent de son contexte de génération.

L'objet partagé est une suite finie d'éléments de tout type de données, il est représenté par une structure linéaire de données, un élément peut être considéré comme un caractère, un paragraphe, une page, un nœud XML, etc. Cette structure linéaire de données peut être facilement étendue à une série de documents multimédia [4]. Il est supposé que l'objet partagé ne peut être modifiée que par les opérations primitives suivantes: (i) Ins (p; e) insère l'élément e à la position p; (ii) Del (p) supprime l'élément à la position p.

Motivation. La technique de transformée opérationnelle se base sur le contenu de leur log pour définir le contexte d'exécution des opérations effectuées, pour cela toutes les opérations générées localement ou reçus sont sauvegardés dans le log pour assurer la convergence des données partagées. Par conséquent, la taille du log augmente au fur et à mesure que le nombre d'utilisateurs du groupe de collaboration et les opérations échangées augmentent, et aussi lors de l'exécution d'une collaboration intense sur une longue durée de temps. Si la taille de log augmente alors le temps de génération et d'intégration des opérations augmente aussi et par conséquent la performance de l'éditeur diminue significativement.

Toutefois, si la collaboration est mobile; de nombreuses opérations peuvent s'accumuler, défiant la capacité de l'algorithme basé sur TO de traiter toutes les opérations, même si le périphérique mobile a assez d'espace mémoire, le temps d'intégration d'une opération distante reste toujours coûteux, cela est dû à la modeste capacité de calcul des dispositifs mobiles.

Contribution. Dans ce papier, on propose un modèle de Garbage Collection (GC) pour les éditeurs collaboratifs en temps réel basé sur l'approche des transformées opérationnelle (TO), ce modèle est indépendant du modèle de contrôle de concurrence, les deux modèles peuvent fonctionner simultanément toute en préservant la cohérence et convergence de l'objet partagé.

Le modèle de contrôle de concurrence de l'éditeur collaboratif utilise la diffusion générale des messages entre les différents sites de collaboration, par contre, le modèle de contrôle de Garbage Collection est structuré sous forme d'un anneau, et les messages de Garbage Collection passent d'un site vers un autre en direction unique. La structure d'anneau est utilisée juste pendant la session de Garbage Collection, et cette structure permet l'exécution de processus de contrôle de concurrence et de Garbage Collection en parallèle.

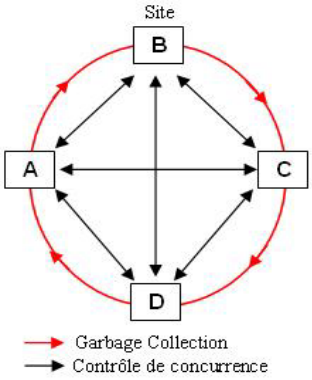

Figure 1. Figure. Structure générale de modèle de contrôle de concurrence et de Garbage Collection

Notre modèle de Garbage Collection (GC) permet de (i) nettoyer tous les journaux du groupe de collaboration, (ii) la conservation de la cohérence et la convergence des objets partagées, (iii) capable de fonctionner sur des dispositifs mobiles avec le respect de leurs conditions particulières et (iv) de répondre aux exigences de Garbage Collection mentionnées dans [5].

Dans ce travail, nous nous sommes basés sur des travaux précédents [5,8,9] sur les éditeurs de collaboration, nous visons à étendre le modèle de collaboration pour qu'il soit supporté sur les dispositifs mobiles et d'améliorer la performance des modèles éditeurs de collaboration existants par l'ajout d'un modèle de Grabage Collection.

II. LE MODELE DE GARBAGE COLLECTION (GC)

Nous avons conçu un modèle de Garbage Collection (GC) placé comme une super couche de l'algorithme de contrôle de concurrence [6,7] désigné pour gérer toutes les interactions simultanées qui se produisent dans l'éditeur collaborative en temps réel. Cet algorithme [6,7] repose sur (i) la réplication des documents partagés afin de fournir l'accès aux données sans contraintes, et (ii) un modèle de cohérence basée sur la dépendance causale.
Nous supposons que tous les messages arrivent à destination, et aucun site n'est défaillant.

A. Structure de log

Pour supprimer un log par une procédure de GC, il faut que toutes les opérations du log soient intégrées dans tous les sites. Pour vérifier cette condition, nous comparons la totalité du log de l'initiateur de GC et les logs des autres sites. Cependant, comparer un journal est une tâche coûteuse surtout si nous manipulons un log de grande taille. Pour surmonter ce problème, nous proposons la solution suivante. Chaque site maintien son propre journal qui peut être vu comme un arbre de dépendance en utilisant les relations de dépendance sémantique.

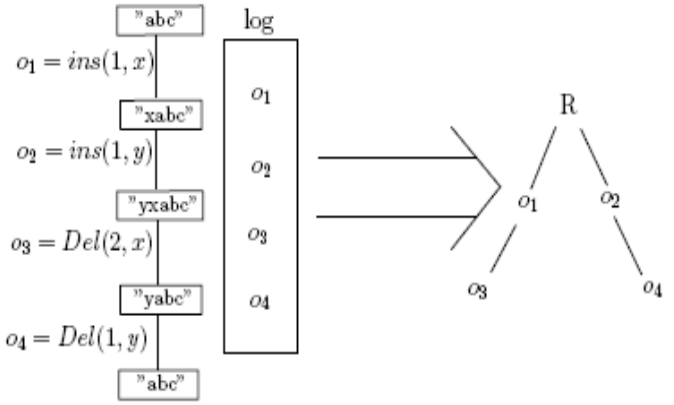

Figure 2. L'arbre de dépendance construite à partir du journal

Les feuilles de cet arbre représentent un résumé de ce qu'un site a reçu comme opérations. Soit *L* l'ensemble des feuilles maintenues par chaque site s. Dans l'exemple de la figure1, L = {o_3; o_4}. La racine de l'arbre de dépendance notée *R* représente l'identifiant de l'opération racine générée par le site qui a effectué la procédure de GC. Cette racine R sert comme un point de rupture (breakpoint) indiquant l'initiation de GC.
La suppression d'un élément dépend de l'opération qui a inséré cet élément. o_3 dépend sémantiquement de o_1 et o_4 dépend de o_2 . Supposons que o_1 et o_2 dépendent de la racine *R*, le journal est considéré comme l'ensemble des feuilles {o_3, o_4} Toutes les opérations concurrentes dépendent systématiquement de l'opération de racine *R*. (*pour plus de détails, le lecteur peut se référer à* [6,7]).

B. Le modèle de collaboration

Dans notre solution, un éditeur collaborative se compose d'un groupe de collaboration de *N* sites (où *N* est variable dans le temps) commencent une session de collaboration à partir d'un

même état initial *St*. Chaque site stocke toutes les requêtes exécutées dans un journal (log) canonique *L*. Pendant une session de Garbage Collection les sites utilisent le log L_{GC}, ce log assure les même propriétés de log *L*.
Dans le modèle utilisé, nous définissons la requête *q* comme un quadruplet (*c*, *r*, *a*, *o*) où *c* est l'identifiant du site collaborateur (ou l'utilisateur) émetteur de la requête et *r* est son numéro de série. Il est à noter que la concaténation de *c* et de *r* est définie comme l'identifiant de *q*. La composante *a* est l'identifiant de la requête de précédence, et enfin *o* est l'opération à être exécutée sur l'état partagé.
Étant donné un journal (log) *L*, *L* [i] désigne la i ième requête de *L*; | *L* | est la longueur de *L*, L_{GC} est le log candidat de suppression utilisé pendant la session de GC.
Un état stable dans un éditeur collaboratif en temps réel est atteint lorsque toutes les opérations générées ont été effectuées sur tous les sites. Pour cela les critères suivants doivent être assurés [6,7]:
Définition 1. (Critères de cohérence) Un éditeur collaboratif en temps réel est cohérent si est seulement si, il satisfait les propriétés suivantes:
1) la préservation de la dépendance: si o_1 dépend de o_2 alors o_1 est exécuté avant o_2 sur tous les sites.
2) la convergence: lorsque tous les sites ont effectué le même ensemble d'opérations, les copies des documents partagés sont identiques.
A l'état stable, les logs des sites ne sont pas nécessairement identiques parce que les opérations simultanées peuvent être exécutées dans un ordre différent. Néanmoins, ces journaux doivent être équivalents dans le sens où elles doivent aboutir au même état final.
Définition 2. (Journal Equivalent) Deux journaux sont équivalents si et seulement si ils produisent le même état lorsqu'il est appliqué à un état donné *St*.
Pour éviter le problème TP2 puzzle [6,7], on définit une classe de journaux qui nous permettent de construire des chemins de transformation menant à une convergence des données.
Définition 3. Un log *L* est canonique si et seulement si *L* est la concaténation de deux sous-journaux L_i (ensemble d'opérations ins insertion) et L_d (ensemble de opérations supprimer del) de telle sorte que L_i ne contient pas de requêtes de suppression et L_d ne contient pas de requêtes d'insertion.
Dans un journal canonique, on impose un ordre sur les requêtes d'insertion et de suppression: une requête d'insertion doit toujours être avant les requêtes de suppression.

C. La structure d'anneau

Nous avons utilisé une structure d'anneau pour l'exécution du processus de Garbage Collection, cette structure est formée d'une manière dynamique, elle est mise à jour à chaque fois qu'un site joint ou quitte le groupe de collaboration.
Le principe de fonctionnement est basé sur l'envoi de message d'initialisation de Garbage Collection GCI d'un site de l'anneau vers son successeur en direction unique, chaque site reçoit le message GCI applique la procédure de Garbage Collection, la procédure de GC sera terminée quand le site initiateur de GC reçoit le message GCI.
Après le réception de GCI, le site initiateur envoi un message d'ordre GCO au groupe de collaboration pour la suppression du log candidat.
Il faut rappeler que pendant l'exécution de cette procédure, le travail de collaboration (génération/intégration des opérations) est toujours maintenu sans problème.

D. Algorithme de Garbage Collection

Initialement, la valeur de la racine *R* de l'arbre de dépendance conçu à partir de log *L* est nulle (*R* = null), et toutes les opérations indépendantes (concurrentes) du log *L* dépendent de la racine *R*.
Pour lancer une procédure de Garbage Collection, le site initiateur génère une nouvelle racine *R = R',* cette dernière (*R'*) devient la nouvelle racine du log L_{GC}.

Notre algorithme de contrôle d'accès concurrent avec le modèle de Garbage Collection est donné dans ce qui suit (l'algorithme 1).

```
1: Main:
2: INITIALIZATION
3: while not aborted do
4: if there is an input o then
5: GENERATE REQUEST (o)
6: else
7: RECEIVE REQUEST
8: INTEGRATE REMOTE REQUESTS
9: end if
10: end while

11: INITIALIZATION:
12: W ← [ ]
13: L ← [ ]
14: L_GC ← [ ]
15: Q ← [ ]
16: GC_State ← false
17: Nxt ← null

18: RECEIVE REQUEST:
19: if there is a request q from a network then
20: Q ← Q + q
21: else
22: if there is a message m from a network then
23: RECEIVE_MESSAGE (m)
24: end if

25: INTEGRATE REMOTE REQUEST:
26: if there is q in Q that is causally-ready then
27: Q ← Q − q
28: INTEGRATION (q)
29: end if
```

Algorithme1. Contrôle de la concurrence avec le modèle GC

Dans notre solution de Garbage Collection basé sur la structure d'anneau, nous utilisons deux jetons (token), le premier est utilisé pour assurer la procédure de Garbage

Collection, et l'autre pour quitter l'anneau en toute sécurité en assurant le maintien de la structure d'anneau. Les deux jetons circulent dans l'anneau simultanément, et chaque jeton est utilisé exclusivement par un seul utilisateur. Par exemple, si un utilisateur veut lancer une procédure de GC, il attend le passage de jeton de GC, si le jeton est libre alors il peut injecter dans le jeton les informations nécessaire pour effectuer le GC tel que les feuilles *F*, la nouvelle racine *R'*. Après que le jeton fait un tour complet dans l'anneau, il sera intercepté par l'initiateur de GC, ce dernier réinitialise le jeton en état initial et le relance de nouveau dans l'anneau pour une autre réutilisation par d'autres utilisateurs. Il faut mentionner que les utilisateurs ne peuvent pas changer le jeton utilisé pendant la session de GC jusqu à ce que la procédure de GC sera terminée.

1) Lancement de processus de GC

Le site initiateur de GC génère une nouvelle racine *R'*, cette racine est une variable qui prend une valeur unique générée par l'initiateur, cette valeur est obtenue par la concaténation de l'identifiant de site *c* et un numéro séquentiel des opérations *k* déjà appliquées dans le même site.

Il rend l'état de GC actif par le changement de la variable STATE_GC à vrai, et transfère le contenu de log *L* au log candidat pour la suppression L_{GC}, et envoi à son voisin direct dans l'anneau le jeton de GC qui va contenir les feuilles de dépendances *F* déduites à partir de log *L* et la nouvelle racine *R'*.

2) La génération locale des opérations

Les opérations locales générées dans le site initiateur de GC par la fonction GENERATE REQUEST seront appliquées sur le log vide *L*, et les opérations concurrentes vont dépendre de la nouvelle racine *R'*.

```
1: GENERATE REQUEST (o):
2: l ← Do (o, l)
3: q ← (c, r, null, o)
4: q0 ← COMPUTEBF (q, L)
5: L ← CANONIZE (q, L)
6: broadcast q0 to other users
```

Algorithme 2. La génération locale des opérations.

L'algorithme 2 utilise la fonction Do(*o*, *L*) pour effectuer l'effet de l'opération *o* sur l'état courant de document *St*, et utilise ComputeBF(o) pour calculer les relations de dépendance sur le log *L*, la fonction CANONIZE permet d'assure que a chaque intégration ou génération d'une opération le log restera toujours canonique (pour plus de détails voir [6,7]).

3) Intégration des opérations distantes

Si le site reçoit une opération distante qui dépend de l'ancienne racine *R* ou d'une opération qui se trouve dans L_{GC}, cette opération distante sera transformée sur le log L_{GC} puis sur le log *L* par la fonction COMPUTEFF avant d'être appliqué sur l'état de document partagé (ligne 5-7 algorithme 3).

```
1: INTEGRATION (q):
2: if not GC_State then
3: q0 ← COMPUTEFF (q, L)
4: else
5: if the dependence of q is not in L then
6:     q0 ← COMPUTEFF (q, LGC)
7:     q0 ← COMPUTEFF (q0, L)
8: else
9:     q0 ← COMPUTEFF (q, L)
10: end if
11:   if there is q in W then
12:   W ← W – q
13:   Apply_GC
14: end if
15: end if
16: l ← Do (q0.o, l)
17: L ← CANONIZE (q0, L)
18: end if
```

Algorithme 3. Intégration des opérations distantes.

La fonction d'intégration utilisée INTEGRATION (voir algorithme 3) permet d'assurer le fonctionnement de processus de réconciliation et de Garbage Collection en parallèle sans perdre de dépendance entre les opérations générées avant et pendant le processus de GC, et permet aussi l'intégration et la génération des opérations sans blocage chose qui n'était pas possible dans [5].

4) Procéder le Garbage Collection

A la réception de jeton GCI, chaque site vérifie si les opérations contenues dans ce message sont déjà exécutées ou pas. Pour faire cette vérification, le site calcule la différence entre l'ensemble des feuilles reçues par le jeton GCI et les feuilles locales du site, le résultat de cette différence représente les opérations non vues par le site, et ils sont ajoutés à la liste de feuilles non encore vues *W* (ligne 5 algorithme), cette liste est mise à jour à chaque réception d'une opération distante par la suppression de cet opération de la liste *W* (ligne 11-12 algorithme 3).

Après la réception des opérations non vu (*W* = ø), le site peut lancer le processus de GC localement en appelant la procédure Apply_GC qui utilise la fonction ReOrder_Log, cette dernière prend comme paramètre les feuilles *F* envoyées par l'initiateur, le rôle de cette fonction consiste à construire les branches de l'arbre de dépendance locale à partir de feuilles *F* reçues.

```
1: Apply_GC:
2: R← R'.
3: if W is empty then
4: Reorder_Log (F)
5: send message GC (F, R') to the next neighbor
6: end if
```

Algorithme 4. Application de procédure de GC.

Le principe de cette fonction est de lire une feuille de *F* (ligne 3-4 algorithme 4) pour construire une branche.
Les opérations qui forment la branche seront transformées par la fonction Climb_Operation pour être sauvegardé dans le log candidat de suppression L_{GC}.

```
1:ReOrder_Log (F):
2: branch ← []
3: for (i = 0; i ≤ |F| − 1; i + +) do
4: branch ← getBranch (F[i]);
5    for (j = 0; j ≤ | branch | − 1; j + +) do
6:   Climb_Operation (branch[j]);
7:   end for
8: end for
9: delete all processed branches
10: end if
```

Algorithme 5. Procédure de réorganisation de log.

La fonction Climb_Operation permet le transfert des opérations de log *L* vers le log candidat de Garbage Collection L_{GC} par la transformation sémantique des opérations dans *L* en utilisant la fonction PERM qui permet la permutation entre deux opérations en gardant la sémantique de dépendance (pour plus de détails voir [6,7]).

```
1: Climb_Operation (i):
2: j  ← 0
3: q1 ← L[i]
4: q2 ← L [i-1]
5: for (j = i; j>0; j--) do
6: if q1 is not dependent of q2 then
7: < q2, q1 > ← PERM (q1, q2)
8: else
9: LGC ← CANONIZE (q1, LGC)
10: end if
11: end for
```

Algorithme 6. Transformation des opérations vers LGC

Il faut mentionner que la fonction de Reorder_Log ne traite pas toutes les feuilles d'une branche à chaque fois, si on trouve une feuille déjà traitée on peut passer vers une autre branche car les feuilles restantes dans cette branche sont déjà transférées vers L_{GC}, cela rend la fonction plus rapide.
Le site continue la génération locale des opérations en utilisant GENERATE REQUEST et l'identifiant de la nouvelle racine *R'* générée par le site initiateur de la session de GC reçu dans le jeton GCI en tant que nouvelle racine. La fonction INTEGRATION est appelée pour l'intégration des opérations distantes en prenant en considération les opérations distantes reçues pendant la session GC et qui dépend des opérations qui sont dans le log L_{GC}. (ligne 5-7 algorithme 3)
Le site qui possède je jeton GC peut le passer à son voisin direct de l'anneau après l'opération de réorganisation de son log. De cette manière le jeton passe d'un site vers un autre, entre temps les opérations distantes arrivent aux autres sites, cela permet d'éviter le temps d'attente de ces opérations et par conséquent accélère le processus de GC.
Si le jeton GC arrive au initiateur de GC (ligne 2-7 algorithme 7), cela signifie que tous les sites ont effectué le processus de Garbage Collection et peuvent supprimer le log candidat L_{GC} en toute sécurité. Pour cela le site initiateur envoi un message GCO a tous les sites pour supprimer le log L_{GC} (ligne 9 algorithme 7).

```
1: RECEIVE_MESSAGE (m):
2: if m = GCI (F', root) then
3:   if sender GC ≠ initiator GC then
4:   GC_State ← true
5:   W ← F \ F'
6:   Apply_GC
7:   end if
8: else
9: send GCO message to other users.
10: end if
11: if m = GCO then
12: Clean LGC
13: GC_State ← false
14: end if
```

Algorithme 7. Traitement des messages de GC.

Après la suppression de log L_{GC}, tous les site mettent leur état de Garbage Collection GC_State à faux et réinitialise le jeton GC pour être réutilisé dans une autre session de GC.

III. JOINDRE ET QUITTER UN GROUPE

La structure d'anneau utilisée se forme d'une manière dynamique, et le jeton de l'anneau circule dans un sens unique d'un site vers un autre et chaque site connaît son voisin direct par la variable *Nxt* qui désigne l'adresse du prochain voisin du site dans l'anneau.
Si un nouveau site veut joindre le groupe de collaboration, **(1)** il contacte un site proche de lui, **(2)** le site contacté envoi l'adresse de son prochain voisin *Nxt* actuellement avec le log et l'état courant de document au demandeur, et met à jour son *Nxt* qui sera l'adresse de demandeur.

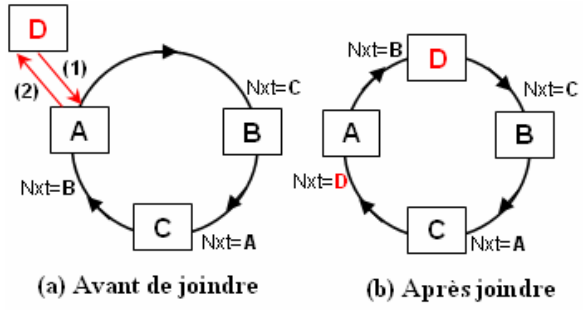

Figure 3. Joindre un groupe de collaboration

Pour quitter le groupe on propose un protocole de sortie. Pour cela on utilise un jeton de sortie dans l'anneau, ce jeton permet de garder la structure de l'anneau, le principe est simple, si un site veut quitter le groupe, il capte le jeton de sortie lors de son passage, il injecte dans ce jeton une demande de sortie avec l'adresse de son voisin direct actuel *Nxt* et relance le jeton dans l'anneau, le site qui possède un *Nxt* égal à l'adresse du site qui demande de quitter le groupe doit modifier son *Nxt* par l'adresse de *Nxt* reçue dans le jeton.

Cette procédure permet de garder la structure de l'anneau en permettant à un site seulement de quitter le groupe à la fois, car si deux sites quittent le groupe simultanément on risque que cassé la structure d'anneau.

Les sites peuvent toujours joindre et quitter le groupe pendant la session de Garbage Collection car les deux jetons de l'anneau peuvent circuler en parallèle. Sauf que dans le cas ou un site, qui possède le jeton GC, veut quitter le groupe il doit le relancer dans l'anneau avant de procéder à la sortie.

Dans le cas ou l'initiateur de Garbage Collection quitte le groupe avant que le jeton de GC fait un tour complet de l'anneau, son voisin direct le remplace automatiquement pour envoyer le message de suppression GCO et termine le processus de Garbage Collection. Pour cela on considère que chaque site reçoit un jeton de GC deux fois, ce qui signifie que l'initiateur de ce jeton a quitté le groupe avant d'achever le processus de GC et de cette façon on évite la circulation infinie d'un jeton GC qui a perdu son initiateur.

IV. TRAVAUX CONNEXES

Le problème posé dans les systèmes distribués, est comment avoir un état global de système à un instant donné ? et pour appliquer un Garbage Collection il faut avoir un état global de groupe de collaboration. Chandy et Lamport [10] proposent un algorithme pour déterminer les états globaux des systèmes distribués par l'enregistrement d'une logique snapshot (ou de causalité) du système. Dans les bases de données, le travail de [12] propose une technique de réduction de taille permettant la suppression des différentes mises à jour qui ne sont plus nécessaires pour le système, aussi [17] propose d'appliquer un Garbage Collection de journal de base de données en utilisant un horodatage de coupure pour déterminer un état global du système, cependant l'utilisation de l'horodatage, est inappropriée dans un contexte dynamique. A notre connaissance, [3] est le seul travail qui a proposé une technique de Garbage Collection afin de réduire la taille du journal pour les éditeurs collaboratifs distribués.

V. CONCLUSION

Dans ce papier, nous proposons un modèle générique de Garbage Collection pour l'édition collaborative basée sur l'approche des transformée opérationnelles dans les environnements mobiles et P2P. Le modèle proposé est dynamique et non bloquant et n'utilise pas un temps d'attente pour les opérations distantes, et indépendant du modèle de collaboration, c'est une amélioration de nos travaux précédentes [5, 8, 9].

Ce modèle fonctionne sur les dispositifs mobiles avec une bonne mesure de performance, mais vu le nombre limité de pages, nous n'avons pas pu détailler la réalisation du prototype développé sur le dispositif mobile Android et les mesures de performance.

REFERENCES

[1] Clarence A. Ellis and Simon J. Gibbs, "Concurrency Control in Groupware Systems". SIGMOD Conference 18 Portland USA, pp. 399–407, 1989

[2] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine," Using the transformational approach to build a safe and generic data synchronizer". Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, pp. 212–220. ACM Press, Florida, USA, 2003.

[3] C. Sun, "Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems". ACM Transactions on Computer-Human Interaction 5, pp. 63–108, 1998

[4] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, " Transparent adaptation of single-user applications for multi-user real-time collaboration". ACM Trans. Comput.-Hum. Interact., 13(4):pp. 531–582, 2006.

[5] M.D Mechaoui, A. Cherif, A. Imine and F. Bendella,"Log Garbage Collector-based Real Time Collaborative Editor for Mobile Devices". In 6th International Conference on Collaborative Computing(IEEE CollaborateCom 2010), Chicago, USA, 2010.

[6] A. Imine, "Coordination Model for Real-Time Collaborative Editors". 11th International Conference COORDINATION, LNCS 5521, pp. 225-246, 2009

[7] A. Imine, "Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair". Phd thesis, University of Henri Poincaré, Nancy, France, 2006.

[8] M.D Mechaoui, A. Imine and F. Bendella," Un modèle de Garbage Collection pour un éditeur collaboratif en temps réel dans les réseaux mobiles et P2P", Colloque sur l'Optimisation et les systèmes d'Information (COSI'2011), Guelma, Algerie, Mars 2011.

[9] M.D Mechaoui, A. Imine and F. Bendella," Distributed Log Garbage Collector-based Real Time Collaborative Editor for Mobile and P2P Environments". In 11th International Conference on New Technologies of Distributed Systems (NOTERE 11), Paris , France, Mai 2011.

[10] K. M, Chandy and L. Lamport," Distributed snapshots: Determining Global States of Distributed System",vol 3, No.1. ACM Transactions on Computer Systems, pp. 63-75, 1985.

[11] Sunil Sarin and Nancy Lynch and A. Lynch, "Discard Obsolete Information In A Replicated Database System". IEEE Transaction on Software Engineering, Vol. SE-13, No. 1, pp. 39-47, 1987.

[12] P. Samarati and P. Ammann and S. Jajodia,"Maintaining replicated authorizations in distributed database systems". Data & Knowledge Engineering journal 18, pp. 55–84, 1996.

[13] Du Li and Rui Li, "An Operational Transformation Algorithm and Performance Evaluation ", Computer Supported Cooperative Work,California USA, pp. 469-508, 2008.

[14] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process". Journal of the ACM, 32(2), pp.374-382, 1985.

[15] B. Shao, D. Li and N. Gu, "A Sequence Transformation Algorithme for Supporting Cooperative Work on Mobile Devices", Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, CSCW 2010, Savannah, Georgia, USA pp.159-168, 2010.

[16] Brad Lushman and Gordon V. Cormack ,"Proof of correctness of Ressel's adOPTed algorithm", Information Processing Letters 86, pp. 303–310, Elseiver B.V, 2003.

[17] Sunil K. Sarin, Charles W. Kaufman, and Janet E. Somers," Using History Information To Process Delayed Database Updates",VLDB '86 Proceedings of the 12th International Conference on Very Large Data Bases 1986. pp. 71-78. San Francisco, CA, USA, 1986.