

Un modèle sûr et générique pour la synchronisation de données divergentes

Gérald Oster, Pascal Molli, Hala Skaf-Molli, Abdessamad Imine

► **To cite this version:**

Gérald Oster, Pascal Molli, Hala Skaf-Molli, Abdessamad Imine. Un modèle sûr et générique pour la synchronisation de données divergentes. Premières Journées Francophones : Mobilité et Ubiquité - UbiMob'04, Jun 2004, Nice, France, 9 p, 2004. <inria-00108119>

HAL Id: inria-00108119

<https://hal.inria.fr/inria-00108119>

Submitted on 19 Oct 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Un modèle sûr et générique pour la synchronisation de données divergentes

Gérald Oster, Pascal Molli, Hala Skaf-Molli, Abdessamad Imine
Projets ECOO & CASSIS
LORIA-INRIA Lorraine, FRANCE
{oster,molli,skaf,imine}@loria.fr

ABSTRACT

La réconciliation de données divergentes est un des problèmes clefs de l'informatique mobile, ainsi que des systèmes de gestion de configuration. Malgré le nombre important de synchroniseurs et d'outils de fusion qui traitent ce problème, aucun critère de correction d'un processus de synchronisation n'existe.

Dans cet article, nous proposons d'utiliser le modèle des transformées opérationnelles pour raisonner sur la synchronisation de données divergentes. Nous présentons un algorithme et des fonctions de transformation qui réalisent la réconciliation d'un système de fichiers. Contrairement aux autres synchroniseurs, notre système garantit des propriétés bien définies telles que *la convergence* et *le respect de la causalité*. Il est également extensible à d'autres types de données.

Categories and Subject Descriptors

D.2 [Software Engineering]: Distribution, Maintenance, and Enhancement

General Terms

Algorithms, Reliability

Keywords

Synchronisation, divergence, transformées opérationnelles

Introduction

La mobilité se caractérise par une alternance de phases de travail en mode connecté et en mode déconnecté. Pour supporter le travail en mode déconnecté les données sont répliquées sur le support mobile. Le travail est alors rythmé par des phases de divergence (en période de déconnexion) où les différentes copies évoluent en parallèle, et des phases de convergence durant lesquelles les copies devront être réconciliées. Cette réconciliation est généralement déléguée à un synchroniseur.

Un synchroniseur est une application critique. En effet, si la correction de la synchronisation n'est pas assurée, les utilisateurs peuvent perdre des données, voire lire des données incohérentes.

De nombreux systèmes existent pour réconcilier des données divergentes : synchroniseurs de fichiers [1], outils livrés avec les assistants personnels, outils de fusion dans les systèmes de gestion de configuration [2], algorithmes optimistes de réplification dans les bases de données [8, 16] et les systèmes distribués [13, 15, 19]. Cependant, ces systèmes présentent encore de nombreux défauts.

Tout d'abord, ils ne garantissent pas une convergence des copies dans tous les cas. La plupart des systèmes se contentent de propager les modifications non conflictuelles et de déléguer la résolution des conflits aux utilisateurs [1]. Ainsi, en cas de mises à jour concurrentes conflictuelles, les copies demeurent divergentes après synchronisation.

Ensuite, le niveau de granularité du processus de synchronisation est un autre point critique. Un niveau de granularité trop grossier engendre un système de détection de conflit superficiel, qui découvre de fausses situations conflictuelles. Par exemple, deux modifications concurrentes d'un même fichier texte seront détectées comme un conflit par un synchroniseur de fichiers, parce que les deux copies du fichier ont été modifiées en concurrence, et ce même si les deux modifications opèrent à des endroits différents dans le fichier. Ainsi, si deux utilisateurs travaillent sur une copie du même document, l'un rajoute un paragraphe, tandis que l'autre corrige les fautes d'orthographe. Au moment de la synchronisation, ils devront choisir entre une des deux versions du document. Ce problème survient du fait que le synchroniseur ne travaille pas au bon niveau de granularité. Pour être plus efficace, un synchroniseur doit pouvoir travailler à tous les niveaux de granularité : au niveau du système de fichiers, mais aussi au niveau du contenu du fichier, voire même au niveau du caractère dans le cas d'un fichier texte.

La plupart des approches souffrent d'un manque de généralité. L'algorithme de réconciliation de deux valeurs numériques divergentes est-il le même que celui qui réconcilie des paragraphes d'un document texte, des arbres XML, ou encore des bases de données ? Pour le moment, dans les systèmes de gestion de configuration, différents outils sont utilisés à différents niveaux de granularité. Un outil réconcilie le système de fichiers, un autre s'occupe de fusionner le contenu des fichiers textes. Malheureusement, ces outils n'appliquent pas la même stratégie. Par exemple, dans CVS [2], si un conflit est détecté au niveau du système de fichiers, la résolution de ce conflit est délégué à l'utilisateur. Par contre, si un conflit est détecté au niveau du contenu du fichier, une fusion automatique est effectuée ; les utilisateurs pouvant compenser la décision du système après la fin de la synchronisation.

La convergence n'étant pas assurée en cas de modifications conflictuelles, les outils existants utilisent des méthodes propres de résolution de conflit. Deux problèmes se posent. Le défaut le plus visible est relatif à l'interaction avec l'utilisateur : lorsqu'un conflit est détecté, l'outil demande l'intervention de l'utilisateur pour le

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Mobilité & Ubiquité 2004 June 1-3, 2004, Nice, France.

Copyright 2004 ACM X-XXXXXX-XX-X/XX/XX ...\$5.00.

résoudre. Plus le nombre de conflit est élevé et plus l'utilisateur sera sollicité. Le second concerne la correction du processus de synchronisation : ces méthodes de résolution étant ad hoc, il est difficile de connaître leur impact sur la convergence.

Comme la synchronisation est une procédure qui rend identiques deux copies divergentes alors plusieurs états de convergence sont possibles. Il faut donc être capable de caractériser l'état de convergence recherché de manière déterministe dans toutes les situations possibles.

Dans cet article, nous proposons d'utiliser le modèle des transformées opérationnelles [6, 25, 21, 24] comme fondement théorique au développement d'un synchroniseur *sûr* et *générique*. Ce synchroniseur assure la convergence *dans tous les cas*, en réconciliant les données divergentes à *tous les niveaux de granularité*. Il calcule un état de convergence où les conflits sont représentés ; les utilisateurs peuvent les résoudre plus tard. Les mêmes propriétés de correction sont assurées à tous les niveaux de synchronisation. Il peut être étendu pour supporter de nouveaux types de données. Pour valider notre approche, nous avons développé un synchroniseur de fichier capable de réconcilier avec ce même algorithme un système de fichiers, et leurs contenus. Les fichiers supportés sont de types texte ou XML.

La suite de ce papier est organisée comme suit : la section 1 présente brièvement le modèle des transformées opérationnelles. La section 2 décrit l'algorithme générique d'intégration. Les sections 3 et 4 détaillent les fonctions de transformation relatives à un système de fichiers et des fichiers de contenu texte. La section 6 donne un aperçu de l'état de l'art. Enfin, la dernière section conclue en exposant certains sujets de recherche future.

1. LE MODÈLE DES TRANSFORMÉES OPÉRATIONNELLES

Le modèle des transformées opérationnelles est issu de l'étude des environnements collaboratifs synchrones [6]. Le modèle considère n sites. Chaque site possède une copie des objets partagés. Quand un objet est modifié sur un site, l'opération correspondante est exécutée immédiatement sur ce site, puis diffusée aux autres sites pour y être également exécutée. Autrement dit, une opération est traitée en quatre étapes : (a) génération sur un site ; (b) diffusion aux autres sites ; (c) réception par les autres sites ; (d) exécution sur ces autres sites.

Lorsqu'une opération op est reçue, son contexte d'exécution peut être différent de celui sur lequel elle a été générée. Dans ce cas, l'intégration de op sur les autres sites peut conduire à des incohérences entre les différentes copies. La figure 1(a) illustre un tel cas. Soient deux sites $site_1$ et $site_2$ partageant un objet de type *chainedecaracteres*. Un objet de ce type peut être modifié par une opération $Ins(p,c)$ qui a pour effet d'insérer le caractère c à la position p dans la chaîne (0 étant la position du premier caractère). Les utilisateurs $user_1$ et $user_2$ génèrent deux opérations concurrentes $op_1 = Ins(2,f)$ et $op_2 = Ins(5,s)$. Quand op_1 est reçue et exécutée sur $site_2$, elle produit l'état attendu "effects". Mais, lorsque op_2 est reçue sur $site_1$, comme celle-ci ne tient pas compte de l'exécution de op_1 avant elle, l'état obtenu est "effecst". Les deux copies n'ont pas convergé.

Dans le modèle des transformées opérationnelles, les opérations reçues doivent être transformées par rapport aux opérations locales concurrentes avant d'être exécutées. Cette transformation est effectuée en utilisant des fonctions de transformation. Cette fonction notée T prend en paramètre deux opérations concurrentes op_1 et op_2 définies sur un même état s de l'objet partagé et calcule en résultat une opération op'_1 . Cette opération est équivalente à op_1 en terme

d'effet, mais elle est définie sur l'état s' résultant de l'exécution de op_2 sur l'état s . La figure 1(b) illustre l'effet d'une telle transformation. Ainsi, dans l'exemple précédent, lorsque op_2 est reçue par $site_1$, elle doit être transformée par rapport à op_1 . Dans ce cas, la transformation se fait de la manière suivante :

$$T(\overbrace{Ins(5,s)}^{op_2}, \overbrace{Ins(2,f)}^{op_1}) = \overbrace{Ins(6,s)}^{op'_2}$$

La position d'insertion de op_2 est alors incrémentée puisque l'opération op_1 a inséré le caractère 'f' avant le point d'insertion de 's' sur l'état "effect". Ensuite, op'_2 est exécutée sur $site_1$. De la même façon, lorsque op_1 est reçue sur $site_2$, la transformation suivante est effectuée :

$$T(\overbrace{Ins(2,f)}^{op_1}, \overbrace{Ins(5,s)}^{op_2}) = \overbrace{Ins(2,f)}^{op'_1}$$

Dans ce cas, la fonction de transformation retourne l'opération $op'_1 = op_1$ puisque, la lettre 'f' est insérée avant la lettre 's'.

Intuitivement, on peut écrire cette fonction de transformation de la façon suivante :

$T(Ins(p_1, c_1), Ins(p_2, c_2)) :-$	
if ($p_1 < p_2$) then	2
return $Ins(p_1, c_1)$	
else	4
return $Ins(p_1 + 1, c_1)$	
endif	6

Cet exemple montre que le modèle des transformées opérationnelles met en jeu deux composants majeurs : un *algorithme d'intégration* et un ensemble de *fonctions de transformation*. L'algorithme d'intégration est responsable de la réception, de la diffusion et de l'exécution des opérations. Il est indépendant du type de données manipulées. Si nécessaire, il fait appel aux fonctions de transformations. Ces fonctions ont la charge de la fusion des modifications en transformant deux opérations concurrentes définies sur un même état. Elles sont spécifiques à un type de données particulier (*chainedecaracteres* dans notre exemple).

Une description plus théorique du modèle est présente dans [25, 21, 24, 23]. Il a été montré que pour être correct, un algorithme d'intégration doit assurer deux propriétés générales :

Convergence Lorsque le système est *stable*, c'est à dire que toutes les opérations ont été diffusées et intégrées, toutes les copies doivent être identiques.

Causalité Si sur un site, une opération op_2 a été exécutée après op_1 , alors op_2 doit être exécutée après op_1 sur tous les sites.

Plusieurs algorithmes d'intégration ont été proposés tels que SOCT 2,3,4 [21, 27], dOPT [6], adOPTed [18], GOTO [25]. Il a été prouvé que leur correction [25, 21] repose uniquement sur les fonctions de transformation qui doivent satisfaire les deux conditions suivantes :

1. La condition C_1 définit une *égalité d'état*. L'état obtenu par l'exécution de op_1 suivi de l'exécution de $T(op_2, op_1)$ doit être le même que celui obtenu par l'exécution de op_2 suivi par $T(op_1, op_2)$:

$$C_1 : op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$$

2. La condition C_2 garantit que la transformation d'une opération par rapport à une séquence d'opérations concurrentes ne dépend pas de l'ordre dans lequel les opérations de la séquence ont été transformées :

$$C_2 : T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$$

En résumé, pour utiliser le modèle des transformées opérationnelles, nous devons suivre les étapes suivantes :

1. Choisir un algorithme d'intégration. Certains algorithmes, de

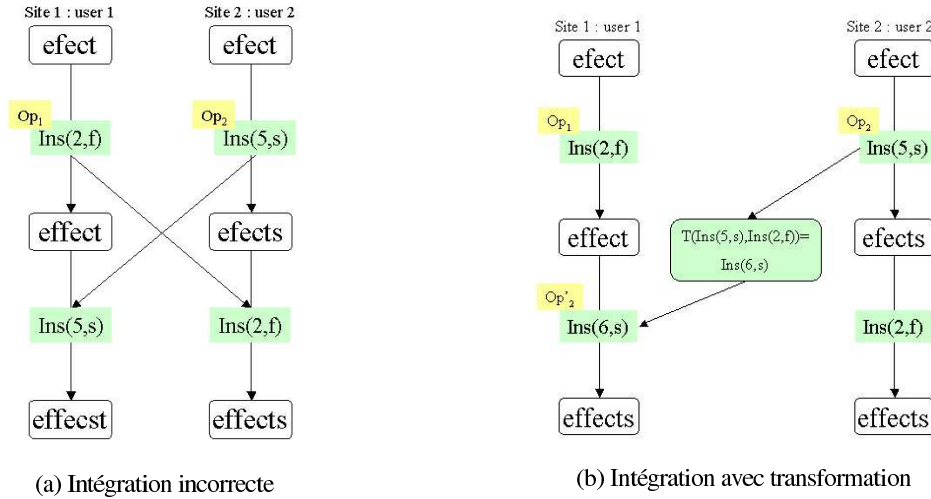


FIG. 1 – Intégration de deux opérations concurrentes

par leur construction, ne requièrent que la satisfaction de la condition C_1 par l'ensemble des fonctions de transformation.

- Définir les types des données partagées et leurs opérations associées.
- Écrire les fonctions de transformation pour tous les couples d'opérations. Par exemple, sur un objet de type chaîne de caractère, munis de deux opérations $Ins(p,c)$ et $Del(p)$, nous devons écrire les fonctions suivantes :

$T(Ins(p_1, c_1), Ins(p_2, c_2)) :-$	
$T(Ins(p_1, c_1), Del(p_2)) :-$	2
$T(Del(p_1), Ins(p_2, c_2)) :-$	
$T(Del(p_1), Del(p_2)) :-$	4

- Prouver que les conditions C_1 et C_2 sont vérifiées par les fonctions de transformation.

2. UN ALGORITHME D'INTÉGRATION POUR LA SYNCHRONISATION

Dans le modèle des transformées opérationnelles, l'algorithme d'intégration est responsable de la diffusion, de la réception, de l'intégration et de l'exécution des opérations. Parmi les algorithmes existants, SOCT4 [27] avec son processus de diffusion différée est l'algorithme le plus proche de nos besoins pour la synchronisation. SOCT4 est basé sur un ordre global continu des opérations et requiert uniquement la satisfaction de la condition C_1 . Chaque opération est envoyée avec une estampille unique. Une opération munie de son estampille ne peut être envoyée que si toutes les opérations qui la précèdent selon l'ordre des estampilles ont été reçues et exécutées sur ce site.

La Figure 2 présente notre algorithme basé sur SOCT4. Le processus de synchronisation d'un site S nécessite deux paramètres: (a) log est la séquence d'opérations locales qui a été exécutée depuis la dernière synchronisation. (b) N_s est l'estampille de la dernière opération reçue ou envoyée par le site S . Nous définissons deux fonctions :

1. $getOp(int\ ticket) \rightarrow op$: retourne l'opération associée à l'estampille $ticket$. Si aucune opération n'est associée, $getOp$ retourne \emptyset

2. $send(Operation\ op, int\ ticket) \rightarrow boolean$: diffuse l'opération locale op en lui associant l'estampille $ticket$. Si $ticket$ est déjà utilisée,

```

sync(log, N_s) :-
  while ((op_r = getOp(N_s+1)) != 0)
    for (i=0; i < log.size(); i++)
      op_l = log[i];
      log[i] = T(op_l, op_r);
      op_r = T(op_r, op_l);
    endfor
    execute(op_r)
    N_s = N_s + 1
  endwhile

for (i=0; i < log.size(); i++)
  op'_i = log[i];
  if send(op'_i, N_s+1) then
    else
      error 'need to synchronize'
    endif
  endfor

```

FIG. 2 – Algorithme générique de synchronisation

alors cela signifie qu'une synchronisation concurrente est en cours. Dans ce cas, l'opération $send$ retourne $false$. L'état courant de la copie locale n'est pas corrompu. Pour continuer il suffit de relancer une nouvelle synchronisation.

$site_1$	$site_2$
op_1	op_3
op_2	op_4
$s_1 = synchronise$	$s_2 = synchronise$
$s_3 = synchronise$	

FIG. 3 – Scénario d'intégrations

Supposons que l'on souhaite synchroniser deux sites comme illustré par la Figure 3. Au départ, chaque site à sa valeur $N_s = 0$. Le site $site_1$ exécute deux opérations locales op_1 et op_2 , pendant que le site $site_2$ exécute en concurrence deux autres opérations op_3 et op_4 .

- À l'étape s_1 , $site_1$ se synchronise. Il effectue donc $sync([op_1, op_2], 0)$. Comme il n'y a pas d'opérations concurrentes disponibles,

il diffuse juste ses opérations op_1 et op_2 au site $site_2$. Sur $site_1$, $N_s = 2$.

2. À l'étape s_2 , $site_2$ se synchronise. Il effectue donc $sync([op_3, op_4], 0)$. Comme il a reçu des opérations concurrentes provenant de $site_1$, il appelle les transformation suivantes :

$$\begin{aligned} op'_1 &= T(op_1, op_3) \\ op'_3 &= T(op_3, op_1) \\ op''_1 &= T(op'_1, op_4) \\ op'_4 &= T(op_4, op'_1) \\ op'_2 &= T(op_2, op'_3) \\ op''_3 &= T(op'_3, op_2) \\ op''_2 &= T(op'_2, op'_4) \\ op''_4 &= T(op'_4, op'_2) \end{aligned}$$

op'_1, op'_2 sont exécutées sur $site_2$. op''_3, op''_4 sont envoyées aux autres sites, en l'occurrence $site_1$. A ce moment là, $N_s = 4$ sur $site_2$.

3. À l'étape s_3 , $site_1$ se synchronise à nouveau, en appelant $sync([1, 2])$. Il n'a pas produit de nouvelles opérations locales, il peut donc juste intégrer les opérations reçues sans les transformer. Maintenant, $N_s = 4$ sur $site_1$.

4. Après l'étape s_3 , $site_1$ a donc exécuté la séquence d'opérations suivante :

$$\begin{aligned} op_1 \\ op_2 \\ op''_3 &= T(T(op_3, op_1), op_2) \\ op''_4 &= T(T(op_4, op'_1), op'_2) \end{aligned}$$

tandis que $site_2$ a exécuté la séquence suivante :

$$\begin{aligned} op_3 \\ op_4 \\ op''_1 &= T(T(op_1, op_3), op_4) \\ op''_2 &= T(T(op_2, op'_3), op'_4) \end{aligned}$$

L'équivalence d'état est garantie par la satisfaction de la condition C_1 par les fonctions de transformation. On notera dans cet exemple, que ce sont les fonctions de transformation qui ont la charge de la détection et de la résolution des conflits. Pourtant, le problème est plus simple. En effet, une fonction de transformation ne détecte et ne résout les conflits que pour *une combinaison de deux opérations concurrentes* définies sur le même état. Si la transformation d'une opération a un effet sur l'opération suivante, les effets de cascades sont pris en compte automatiquement par l'algorithme d'intégration.

Cet algorithme est donc un synchroniseur sûr et générique si les fonctions de transformation sous-jacentes satisfont la condition C_1 . Il garantit la convergence et respecte la causalité.

3. FONCTIONS DE TRANSFORMATION POUR UN SYSTÈME DE FICHIERS

Nous avons défini des fonctions de transformation pour synchroniser un système de fichiers et leurs contenus. Les fichiers supportés sont des documents textes ou XML. Nous limitons notre description aux fichiers de type texte.

Écrire des fonctions de transformation *correctes* n'est pas une tâche facile. Il faut écrire des fonctions qui assurent la convergence en vérifiant la condition C_1 , tout en préservant l'effet de l'opération originale. La stratégie générale que nous avons adopté lors de l'écriture de nos fonctions de transformation, est *de converger vers un état où les conflits sont représentés*. Un outil de fusion fait la même chose lorsqu'il fusionne deux fichiers textes. Par exemple, `rcsmerge` [26] fusionne deux modifications conflictuelles en produisant le contenu suivant :

```
<<<<<< testfile.txt
```

```
std::string LineReader::readLine()
{
    return std::read_line( cin );
}=====
CString LineReader::ReadLine()
{
    CString line;
    m_archive >> line;
    return line;
}
>>>>>>> 1.1.1.1.2.1
```

Ce contenu représente dans le fichier texte ces deux modifications conflictuelles en les délimitant avec des marqueurs spécifiques. Les utilisateurs peuvent ainsi résoudre le conflit en éditant le fichier.

Nous appliquons le même principe pour un système de fichiers. Nous gérons les conflits en renommant les fichiers ou les répertoires impliqués dans le conflit. Par exemple, si deux utilisateurs créent en concurrence le même fichier dans le même répertoire, nous convergeons vers un état où nous avons renommé un des deux fichiers. Les utilisateurs peuvent compenser ce choix après la synchronisation et se synchroniser ensuite.

La correction du modèle des transformées opérationnelles repose sur la correction des fonctions de transformation. Si ces fonctions de transformation ne vérifient pas la condition C_1 aucune garantie n'est assurée. Vérifier la satisfaction de la condition C_1 est une tâche laborieuse qui consomme du temps. Il est quasi impossible de réaliser cette tâche à la main. Nous avons effectué cette preuve en utilisant le prouveur automatique de théorème SPIKE [20, 11, 10]. La spécification prise en entrée par SPIKE correspond en tout point aux fonctions de transformation données dans cet article.

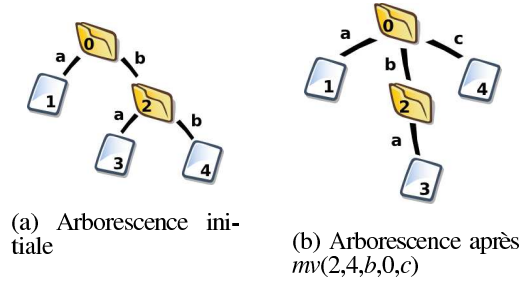


FIG. 4 – Représentation d'un système de fichiers

Nous considérons un système de fichiers comme un arbre où les nœuds sont les répertoires et les feuilles sont les fichiers. Nous définissons les opérations suivantes :

1. $mf(int id, int pid, String name)$: mf crée un fichier identifié par un identifiant unique id . pid est l'identifiant du répertoire parent. id est référencé avec le nom $name$ dans le répertoire pid . mf possède les pré-conditions suivantes : id n'existe pas. pid existe et référence un répertoire. $name$ n'est pas utilisé comme nom dans le répertoire pid

2. $md(int id, int pid, String name)$: md crée un répertoire. Afin de représenter la racine du système de fichiers, nous considérons que l'identifiant unique 0 existe toujours et représente cette racine. La séquence d'opérations $mf(1,0,a); md(2,0,b); mf(3,2,a); mf(4,2,b)$ construit l'arborescence illustrée par la Figure 4(a):

3. $mv(int pid1, int id1, String name1, int pid2, String name2)$: mv déplace l'objet identifié par $id1$ référencé dans le répertoire $pid1$ par le nom $name1$ sous le répertoire $pid2$ avec $name2$ comme nouveau nom. mv exige comme pré-conditions : $pid1, id1, pid2$ existent. $id1$ est référencé avec le nom $name1$ dans le répertoire $pid1$. $name2$ n'est pas un nom utilisé dans $pid2$.

Si nous appliquons l'opération $mv(2,4,b,0,c)$ sur l'arborescence précédente illustrée par la Figure 4(a), nous obtenons l'état décrit par la Figure 4(b).

Nous n'avons pas défini d'opération de suppression. En effet, nous considérons que supprimer un objet est équivalent à déplacer cet objet vers un répertoire représentant une corbeille.

```

T(mf(id1, idp1, n1, s1), mf(id2, idp2, n2, s2)) =
if (idp1 == idp2) then
  if (n1 == n2) then
    if (id1 < id2) then return
      mf(id1, idp1, max(s1) ⊙ id1,
        s1 ∪ {max(s1) ⊙ id1})
    else return
      mv(idp2, id2, n2, idp2, max(s2) ⊙ id2,
        s2 ∪ {n2} ∪ {max(s2) ⊙ id2})
      ⊔ mf(id1, idp1, n1, s2 ∪ {max(s2) ⊙ id2})
    endif
  else
    return mf(id1, idp1, n1, s2 ∪ {n1})
  endif
else
  return mf(id1, idp1, n1, s1)
endif;

```

FIG. 5 – Fonctions de transformation pour mf-mf

```

T(mf(id1, idp1, n1, s1),
  mv(opid2, id2, nb, idp2, n2, s2)) =
if (idp1 == idp2) then
  if (n1 == n2) then
    if (id1 < id2) then return
      mf(id1, idp1, max(s1) ⊙ id1,
        s2 ∪ {max(s1) ⊙ id1})
    else return
      mv(idp2, id2, n2, idp2, max(s1) ⊙ id2,
        s2 ∪ {n2} ∪ {max(s1) ⊙ id2})
      ⊔ mf(id1, idp1, n1, s2 ∪ {max(s1) ⊙ id2})
    endif
  else return
    mf(id1, idp1, n1, s2 ∪ n1)
  endif
else return
  mf(id1, idp1, n1, s1)
endif

```

FIG. 6 – Fonctions de transformation pour mf-mv

La Figure 5 décrit la fonction de transformation pour le couple mf-mf. Nous expliquons maintenant la stratégie employée pour le re-nommage des objets :

1. Comment calculer un nom unique dans un répertoire? Afin de pouvoir représenter les conflits en renommant les fichiers, il nous faut pouvoir calculer des noms d'objets uniques dans les fonctions de transformation. Ce calcul doit pouvoir s'effectuer en utilisant uniquement les informations d'états disponibles concernant l'état sur lequel les deux opérations concurrentes sont définies. Chaque opération modifiant le contenu d'un répertoire possède un paramètre qui contient l'ensemble des noms utilisés dans ce répertoire après exécution de l'opération. Par exemple, dans un répertoire d'identifiant l et contenant des objets nommés $\{a,b,c\}$, l'opération $mf(2,1,d)$ possède en fait un quatrième paramètre s contenant l'ensemble $\{a,b,c,d\}$. Sur cet ensemble s , nous définissons un nouvel opérateur $max(s)$. Cet opérateur retourne le nom possédant la

```

T(mv(opid1, id1, na, idp1, n1, s1),
  mf(id2, idp2, n2, s2)) =
if (idp1 == idp2) then
  if (n1 == n2) then
    if (id1 < id2) then return
      mv(opid1, id1, na, idp1, max(s2) ⊙ id1,
        s2 ∪ {max(s2) ⊙ id1} \ {na})
    else return
      mv(idp2, id2, n2, idp2, max(s2) ⊙ id2,
        s2 ∪ {max(s2) ⊙ id2} \ {n2})
      ⊔ mv(opid1, id1, na, idp1, n1,
        s2 ∪ {max(s2) ⊙ id2} \ {na})
    endif
  else return
    mv(opid1, id1, na, idp1, n1, s2 ∪ {n1} \ {na})
  endif
else return
  mv(opid1, id1, na, idp1, n1, s1)
endif;

```

FIG. 7 – Fonctions de transformation pour mv-mf

valeur la plus élevée selon l'ordre lexicographique. Nous pouvons ainsi obtenir un nom unique par la formule $max(s) \odot id$, où \odot représente l'opérateur de concaténation.

2. Quel est l'objet à renommer? Afin de garantir la convergence, nous devons prendre le même choix déterministe sur tous les sites. Comme chaque objet est identifié par un identifiant unique, nous avons fait le choix de renommer l'objet possédant le plus petit identifiant. Donc, si nous intégrons deux opérations créant le même fichier dans le même répertoire, deux cas peuvent se produire : (a) nous sommes en train de transformer l'opération avec le plus faible identifiant (cf. ligne 5 de la Figure 5). Dans ce cas, nous renommons juste le fichier avec le nom $max(s1) \odot id1$. (b) nous sommes en train de transformer l'opération avec l'identifiant le plus élevé (cf. ligne 8 de la Figure 5). Dans ce cas, nous renommons le fichier avec le plus faible identifiant, et nous créons le fichier avec le plus grand identifiant sans le renommer. Cette transformation retourne donc une séquence de deux opérations. \boxplus dénote l'opérateur de construction d'une séquence.

Les Figures 6 et 7 décrivent les fonctions de transformation pour les couples mf-mv et mv-mf. La section 5 illustre un exemple d'utilisation de ces fonctions.

4. FONCTIONS DE TRANSFORMATION POUR DES FICHIERS TEXTES

Nous définissons les opérations suivantes sur un fichier de type texte :

1. $ab(id1, s1, os1, v1)$. Ajoute un bloc de texte $v1$ au fichier d'identifiant $id1$ au point d'insertion $s1$. Le paramètre $os1$ est utilisé [10] pour résoudre des situations ambiguës [22] de faux conflit. Ce paramètre conserve la valeur original de la ligne d'insertion. Donc à la génération d'une opération ab , on a $os1 = s1$. Lors des transformations, $os1$ ne varie pas. Afin de simplifier l'explication des fonctions de transformation, $l1$ dénotera le nombre de ligne du bloc $v1$. L'opération ab a pour pré-condition : $id1$ et $s1$ doivent exister.

2. $db(id1, s1, ov1)$. Détruit le bloc de texte $ov1$ contenu dans le fichier d'identifiant $id1$ à la ligne $s1$. On utilisera également $l1$ pour représenté le nombre de ligne du bloc $ov1$. L'opération db possède les pré-conditions suivantes : $id1$ et $s1$ doivent exister.

La Figure 8 présente la fonction de transformation pour le couple d'opération ab-ab. Comme pour le système de fichiers, notre stra-

```

T(ab( $id_1, s_1, os_1, v_1$ ) , ab( $id_2, s_2, os_2, v_2$ )) :-
if ( $id_1 \neq id_2$ ) then
return ab( $id_1, s_1, os_1, v_1$ )
else
if ( $s_1 < s_2$ ) then
return ab( $id_1, s_1, os_1, v_1$ )
elseif ( $s_1 > s_2$ ) then
return ab( $id_1, s_1 + l_2, os_1, v_1$ )
else
if ( $os_1 < os_2$ ) then
return ab( $id_1, s_1, os_1, v_1$ )
elseif ( $os_1 > os_2$ ) then
return ab( $id_1, s_1 + l_2, os_1, v_1$ )
else
if ( $v_1 == v_2$ ) then
return Id(ab( $id_1, s_1, os_1, v_1$ ))
else
return db( $id_2, s_2, l_2, v_2$ )
⊕ ab( $id_1, s_1, os_1, v_1 \odot v_2$ )
endif
endif
endif
endif

```

FIG. 8 – Fonctions de transformation pour ab-ab

tégie générale est de converger vers un état où les conflits sont représentés.

Un conflit survient lorsque les effets de deux opérations concurrentes se chevauchent. Par exemple, lorsqu'une opération *db* efface un bloc de texte dans lequel se situe la ligne d'insertion d'une opération *ab* concurrente. Pour le couple ab-ab, un conflit est possible uniquement si les deux opérations concurrentes insèrent un bloc de texte à la même ligne. Dans ce cas, la fonction de transformation doit générer une séquence d'opérations qui effectue les actions suivantes :

1. on détruit le bloc qui a déjà été inséré localement,
2. on insère un bloc qui représente le conflit des deux opérations. La représentation choisie est la même que celle utilisée dans *rcsmerge* [26].

Au cas où il n'y a pas de conflit, on décale juste la position d'insertion du bloc. La même stratégie est appliquée pour écrire les fonctions de transformation pour les couples ab-db (cf. Figure 9) et db-ab (cf. Figure 10).

```

T(ab( $id_1, s_1, os_1, v_1$ ) , db( $id_2, s_2, ov_2$ )) :-
if ( $id_1 \neq id_2$ ) then
return ab( $id_1, s_1, os_1, v_1$ )
else
if ( $s_1 < s_2$ ) then
return ab( $id_1, s_1, os_1, v_1$ )
elseif ( $s_1 > s_2 + l_2 - 1$ ) then
return ab( $id_1, s_1 - l_2, os_1, v_1$ )
else
return ab( $id_1, s_2, s_2, ov_2 \odot v_1$ )
endif
endif
endif

```

FIG. 9 – Fonctions de transformation pour ab-db

Par manque de place, nous ne présentons pas dans cet article les fonctions de transformation pour les couples db-ab et mv-mv. Ainsi que toutes les autres fonctions de transformation $T(op_1, op_2)$ (par exemple $T(op_1 = mv, op_2 = ab)$) qui retournent l'opération non transformée op_1 .

```

T(db( $id_1, s_1, ov_1$ ) , ab( $id_2, s_2, os_2, v_2$ )) :-
if ( $id_1 \neq id_2$ ) then
return db( $id_1, s_1, ov_1$ )
else
if ( $s_1 > s_2$ ) then
return db( $id_1, s_1 + l_2, ov_1$ )
elseif ( $s_1 + l_1 - 1 < s_2$ ) then
return db( $id_1, s_1, ov_1$ )
else
return db( $id_2, s_2, v_2$ )
⊕ db( $id_1, s_1, ov_1$ )
⊕ ab( $id_1, s_1, s_1, ov_1 \odot v_2$ )
endif
endif

```

FIG. 10 – Fonctions de transformation pour db-ab

5. EXEMPLE

Soient trois utilisateurs travaillant en parallèle sur les mêmes données partagées. La Figure 12(a) illustre l'état initial résultant de l'exécution de la séquence $\underline{mf}(1,0,a,\{a\})$, $\underline{ab}(1,0,0,\{ "gaspard", "melchior", " balthazar" \})$.

Sur cet état initial, les utilisateurs produisent les opérations décrites par la Figure 11. Après la synchronisation de l'étape s_5 , tous les utilisateurs observent le même état illustré par la Figure 12(b).

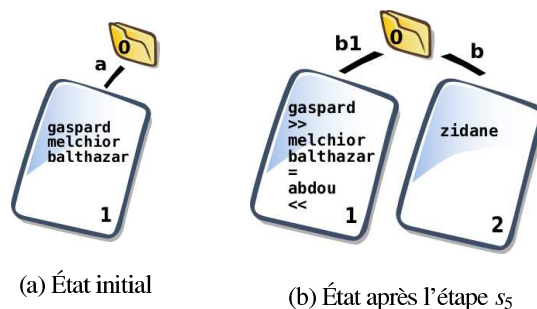


FIG. 12 – États initial et final du scénario

Ce scénario illustre comment notre synchroniseur gère les conflits entre les opérations op_1 et op_3 , et les opérations op_2 et op_5 . Nous explicitons maintenant les effets de chaque synchronisation.

Étape s_1 À cet état, aucune opération concurrente n'a encore été publiée, la synchronisation consiste donc juste à envoyer les opérations op_1 et op_2 aux autres sites.

Étape s_2 Le synchroniseur fusionne les séquences $[op_1; op_2]$ à l'histoire locale du site $[op_3; op_4]$. Pour cela, il calcule les opérations op'_1 et op'_2 équivalentes aux opérations op_1 et op_2 , mais qui prennent en compte l'exécution antérieure de op_3 et op_4 , afin des les exécuter sur l'état courant de la copie. En parallèle, il calcule op'_3 et op'_4 qui sont équivalentes à op_3 et op_4 , mais en tenant compte de l'exécution de op_1 et op_2 . Ces deux opérations sont alors envoyées aux autres sites.

Les transformées suivantes sont donc évaluées :

u_1	u_2	u_3
$op_1 = mv(0,1,a,0,b,\{b\})$ $op_2 = db(1,2,\{''melchior'', ''balthazar''\})$	$op_3 = mf(2,0,b,\{a,b\})$ $op_4 = ab(2,0,0,\{''zidane''\})$	$op_5 = ab(1,3,3,\{''abdou''\})$
$s_1 = synchronise$		
	$s_2 = synchronise$	
		$s_3 = synchronise$
	$s_4 = synchronise$	
$s_5 = synchronise$		

FIG. 11 – Un exemple de scénario d'intégration

Opérations	Résultats
$op'_1 = T(op_1, op_3)$	$mv(0,1,a,0,b1,\{b,b1\})$
$op'_2 = T(op'_1, op_4)$	$mv(0,1,a,0,b1,\{b,b1\})$
$op'_3 = T(op_3, op_1)$	$mv(0,1,b,0,b1,\{b1\})$ $\sqcup mf(2,0,b,\{b,b1\})$
$op'_4 = T(op_4, op'_1)$	$ab(2,0,0,\{''zidane''\})$
$op'_5 = T(op_2, op'_3)$	$db(1,2,\{''melchior'', ''balthazar''\})$
$op'_6 = T(op'_2, op'_4)$	$db(1,2,\{''melchior'', ''balthazar''\})$
$op'_7 = T(op'_3, op_2)$	$mv(0,1,b,0,b1,\{b1\})$ $\sqcup mf(2,0,b,\{b,b1\})$
$op'_8 = T(op'_4, op'_5)$	$ab(2,0,0,\{''zidane''\})$

Étape s_3 Cette étape de synchronisation consiste à transformer les opérations reçues op_1 , op_2 , op'_3 et op'_4 par rapport à l'opération concurrente op_5 . Puis à calculer l'opération op_5''' équivalente à op_5 mais qui prend en compte l'exécution antérieure des opérations reçues. Cette opération est ensuite envoyée aux autres sites.

Étape s_4 Aucune opération locale n'ayant été produite depuis la dernière synchronisation, cette étape se résume à exécuter les opérations reçues, à savoir l'opération op'_5 .

Étape s_5 De la même manière que pour l'étape s_4 sur le site de l'utilisateur u_2 , cette étape consiste à exécuter les opérations reçues depuis la dernière synchronisation, c'est à dire les opérations op'_6 , op'_7 et op_5''' .

Après l'étape s_5 Chaque site a exécuté une séquence d'opération équivalente à la séquence suivante :

$mf(1,0,a,\{a\}),$	
$ab(1,0,0,\{''gaspard'', ''melchior'', ''balthazar''\}),$	2
$op_1 = mv(0,1,a,0,b,\{b\}),$	
$op_2 = db(1,2,\{''melchior'', ''balthazar''\}),$	4
$op_3 = mv(0,1,b,0,b1,\{b1\}) \sqcup mf(2,0,b,\{b,b1\}),$	
$op_4 = ab(2,0,0,\{''zidane''\}),$	6
$op_5''' = ab(1,2,2,\{''>>'', ''melchior'', ''balthazar'', ''='', ''abdou'', ''<<''\});$	8

Ces séquences étant équivalentes à une même séquence, toutes les copies des différents ont donc convergé.

6. ÉTAT DE L'ART

La synchronisation de données divergentes est un problème largement étudié. Pour répondre à ce problème, il existe une multitude d'outils. Nous allons maintenant comparer notre approche avec les outils existants tels que les synchroniseurs de fichiers, les synchroniseurs de PDAs, les outils utilisés dans les systèmes de gestion de configuration, ou encore les résultats issues du domaine des systèmes distribués et des bases de données répliquées.

Synchroniseurs de fichiers L'objectif général d'un synchroniseur de fichiers est de détecter les mises à jour conflictuelles et de propager les mises à jour non conflictuelles [1].

Pour atteindre cet objectif, la sémantique des primitives d'un système de fichiers doit être clairement définie comme cela a été fait

dans Unison [1]. Pourtant ce type de synchroniseur souffre de deux défauts : sa granularité inadaptée et sa trop grande interactivité. En effet, le synchroniseur n'est pas capable de synchroniser le contenu des fichiers. En cas de mises à jour conflictuelles, c'est à l'utilisateur de choisir si il veut conserver son fichier local ou l'écraser avec le fichier distant. Ainsi, si lors d'une synchronisation 100 conflits sont détectés, l'utilisateur devra interagir avec le système 100 fois. Par opposition, un synchroniseur basé sur des transformées opérationnelles permet de synchroniser le système de fichiers et le contenu des fichiers, avec le même algorithme, en garantissant les mêmes propriétés, et ce sans intervention de l'utilisateur.

Synchroniseurs de données Les synchroniseurs livrés avec les assistants digitaux personnels, tels que ActiveSync, HotSync ou encore I-Sync, permettent de synchroniser plusieurs types de données : un carnet d'adresses, un calendrier, une liste de tâches Cependant, le problème reste inchangé par rapport aux synchroniseurs de fichiers : ils détectent les mises à jour conflictuelles et propagent les mises à jour non conflictuelles.

Le problème le plus aigu avec ces synchroniseurs est le manque de généricité. Ils synchronisent des types simples de données avec des mécanismes de résolution ad-hoc des conflits. Ces solutions restent ponctuelles et ne peuvent être étendues au cas général.

Un synchroniseur à base de transformées opérationnelles permet d'utiliser le même algorithme de synchronisation quel que soit le type des objets manipulés. Par exemple, il est possible de définir un nouveau type *Agenda* muni de trois opérations *AjouterRendezVous*, *SupprimerRendezVous* et *ModifierRendezVous*, d'écrire les fonctions de transformation nécessaires et de prouver qu'elles satisfont la condition C_1 . On obtient alors un synchroniseur d'agenda sûr qui assure la convergence et respecte la causalité.

Gestionnaires de configuration Dans le domaine des gestionnaires de configuration [2, 4, 26, 7], les utilisateurs peuvent travailler en parallèle, diverger et réconcilier leurs copies plus tard en utilisant le paradigme du *copier-modifier-fusionner*. En fait, ce résultat est le produit d'une coopération étroite entre les gestionnaires de version, les outils de fusion et les outils de différenciation. Quand une réconciliation est demandée (généralement quand les utilisateurs re-synchronisent leurs espaces de travail), le gestionnaire de version fournit à l'outil de fusion [14] toutes les versions nécessaires. L'outil de fusion utilise alors un outil de différenciation [3] spécifique au type de données manipulées pour extraire le journal des opérations concurrentes. Les opérations conflictuelles sont marquées en tant que tel, et les opérations concurrentes non conflictuelles sont appliquées.

Dans cette approche, chaque outil de fusion utilise son propre algorithme avec sa propre stratégie de résolution. Un outil fusionne deux systèmes de fichiers divergents, un autre fusionne deux fichiers textes divergents, et encore un autre pour les fichiers XML. Puisqu'ils appliquent chacun leurs propres stratégies de résolution, il se peut que la combinaison de ces outils soit incohérente. Par exemple, dans CVS, le principe de la compensation est utilisé pour résoudre les conflits de mise à jour entre deux fichiers texte, alors

qu'un conflit au niveau du système de fichiers n'est pas résolu, mais délégué à l'utilisateur.

L'approche transformationnelle est plus générale, plus uniforme et plus sûre que le modèle employé par les gestionnaires de configuration. En effet, l'approche transformationnelle ne fait pas coopérer différents outils avec chacun sa propre politique de résolution de conflits et ses propres propriétés garanties. Les mêmes propriétés sont garanties par un seul algorithme qui est appliqué à tous les types de données à synchroniser. Il est donc possible de prouver formellement la correction d'une telle synchronisation.

Systèmes distribués. Le maintien de la cohérence des données partagées au sein d'un système distribué a été largement étudié. Des systèmes connus comme Coda [13], Bayou [15], Ficus [17] permettent aux utilisateurs de travailler de manière déconnectée et utilisent des procédures de réconciliation à la reconnexion.

Bayou utilise un algorithme épidémique pour propager les mises à jour entre les copies en utilisant un critère de cohérence faible. Quand un conflit est détecté, une procédure de réconciliation associée à l'opération conflictuelle est exécutée. Si cette procédure ne peut pas trouver une solution, la résolution de conflit est déléguée aux utilisateurs. Bayou utilise un ordre total sur les opérations. D'autres systèmes [19] utilisent un ordre partiel et tirent avantage de la commutativité des mises à jour.

Les systèmes distribués et l'approche transformationnelle ont des points communs : les deux approches détectent les conflits, les procédures de réconciliation et les fonctions de transformation semblent jouer le même rôle. La commutativité et la condition C_1 se ressemblent : la causalité est utilisée dans les deux approches. Comme, l'approche transformationnelle transforme les opérations, la condition C_1 peut être considérée comme une sorte de "commutativité transformationnelle" et permet de calculer des états de convergence bien plus complexes. Enfin les fonctions de transformation garantissent la convergence dans tous les cas.

IceCube [12] est une approche générique pour réconcilier des données divergentes. **IceCube** ne définit pas un critère général de correction mais utilise des contraintes sémantiques que l'algorithme de réconciliation doit préserver. Deux types de contraintes sont définies : (a) les contraintes statiques qui peuvent être évaluées sans représentation de l'état de la copie. Par exemple, la commutativité des opérations peut être exprimée comme une contrainte statique ; (b) les contraintes dynamiques qui elles nécessitent l'état de la copie pour être évaluées. L'algorithme de **IceCube** prend en entrée deux journaux d'opérations concurrentes et explore toutes les histoires pouvant être construites à partir de ces deux journaux. Les histoires ne vérifiant pas les contraintes statiques sont rejetées. Pour les autres, **IceCube** simule l'intégration sur les copies et rejette celles ne vérifiant pas les contraintes dynamiques. Les histoires restantes sont classées et proposées à l'utilisateur. **IceCube** est une approche intéressante parce qu'elle explore toutes les réconciliations possibles et sélectionne celles qui minimisent les conflits. Sur ce point, **IceCube** peut trouver une réconciliation plus optimale qu'un algorithme basé sur les transformées opérationnelles. Par contre, **IceCube** a d'autres limites : l'étape d'exploration des différentes histoires peut engendrer une explosion combinatoire même si les contraintes statiques restreignent les combinaisons possibles. **IceCube** ne transforme pas les opérations. Dans ce cas, que se passe-t-il si il y a juste 2 opérations concurrentes "md(/a)" et "mf(/a)", toutes les intégrations possibles sont mauvaises, **IceCube** va donc se comporter comme un synchroniseur de fichiers classiques, et proposer les deux alternatives à l'utilisateur.

Bases de données. La gestion de la réplication est un sujet largement étudié dans le domaine des bases de données [8, 16]. Des conflits de réplication peuvent apparaître si le schéma de réplica-

tion permet des mises à jour concurrentes des mêmes données sur des sites multiples. Si deux transactions travaillent sur deux copies différentes et modifient le même tuple concurrentement, un conflit apparaît. Oracle [5] fournit plusieurs méthodes de résolution de conflits. Par exemple, la méthode de la "dernière estampille" résout les conflits en se basant sur la plus récente mise à jour. La méthode d'"addition" résout un conflit de mise à jour sur un champ numérique en ajoutant la différence entre les deux mises à jour conflictuelles. La méthode "overwrite" remplace en cas de conflit la valeur actuelle avec la nouvelle valeur. Si la convergence ne peut être atteinte alors une notification est envoyée à l'administrateur. Certaines de ces méthodes semblent préserver la convergence mais pas pour tous les types de conflits ni pour toutes les configurations de réplicas.

L'approche transformationnelle est plus générale que celle utilisée dans les bases de données. Les méthodes de résolution de conflits peuvent être avantageusement remplacées par des fonctions de transformation sur lesquelles on peut prouver formellement la convergence et le respect de la causalité.

7. CONCLUSION ET PERSPECTIVES

Nous avons proposé d'utiliser le modèle des transformées opérationnelles comme cadre théorique pour la synchronisation de données. Nous avons présenté un synchroniseur sûr et générique qui garantit la convergence dans tous les cas. Ce synchroniseur repose sur des fonctions de transformation spécifiques aux types de données manipulées qui vérifient la condition C_1 .

Nous avons développé et prouvé formellement des fonctions de transformation pour un système de fichiers, des fichiers de contenu texte et des fichiers XML [9]. Nous avons validé notre approche en réalisant le synchroniseur S5 intégré à la plate-forme de travail collaboratif LibreSource¹. Il est possible d'y ajouter de nouveaux types de données à synchroniser. Pour cela, il suffit de définir les opérations primitives, d'écrire les fonctions de transformations et de prouver la satisfaction des conditions. La difficulté se situe au niveau de l'écriture des transformées puisque celles-ci doivent permettre de représenter les conflits au sein de l'état de convergence.

Nous orientons maintenant nos recherches dans plusieurs directions. Nous souhaitons développer des fonctions de transformation pour d'autres type de données comme des calendriers, schéma et données de bases de données. De nombreux travaux se sont intéressés à l'annulation d'opération dans les éditeurs synchrones [23]. Ces travaux nécessitent que les fonctions de transformation vérifient au moins la condition C_2 . Nous avons commencé à améliorer nos fonctions pour satisfaire cette condition. Cette approche pourrait être une alternative à la compensation lors de la résolution de conflit. Nous sommes également en train de modifier le prouveur automatique SPIKE afin de réaliser un environnement de développement intégré de fonctions de transformation. Grâce à cet environnement, un utilisateur pourra développer ses propres fonctions en vérifiant les conditions C_1 et C_2 . Dans le cas où elles ne satisfont pas ces conditions, l'environnement est capable de lui donner des scénarios contre-exemples. Nous pensons que ce genre d'environnement permettra de grandement améliorer le processus de développement de fonctions de transformation.

8. REFERENCES

- [1] S. Balasubramaniam and B. C. Pierce. What is a file synchronizer? In *Mobile Computing and Networking*, pages 98–108, 1998.

1. <http://libresource.loria.fr/>

- [2] B. Berliner. CVS II : Parallelizing Software Development. In *Proceedings of USENIX*, Washington D. C., 1990.
- [3] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 26–37. ACM Press, 1997.
- [4] R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computing Surveys (CSUR)*, 30(2):232–282, 1998.
- [5] D. Daniels, L. B. Doo, A. Downing, C. Elsbernd, G. Hallmark, S. Jain, B. Jenkins, P. Lim, G. Smith, B. Souder, and J. Stamos. Oracle’s symmetric replication technology and implications for application design. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, page 467. ACM Press, 1994.
- [6] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.
- [7] J. Estublier, editor. *Software Configuration Management: Selected Papers of the ICSE SCM-4 and SCM-5 Workshops*, number 1005 in Lecture Notes in Computer Science. Springer-Verlag, October 1995.
- [8] J. Gray, P. Helland, P. O’Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182. ACM Press, 1996.
- [9] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Development of transformation functions assisted by a theorem prover. In *Fourth International Workshop on Collaborative Editing*, New Orleans, Louisiana, USA, November 2002.
- [10] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving correctness of transformation functions in real-time groupware. In *Proceedings of the 8th European Conference on Computer-Supported Cooperative Work*, Helsinki, Finland, September 2003.
- [11] A. Imine, P. Molli, G. Oster, and P. Urso. Vote: Group editors analyzing tool. In *International Workshop on First-Order Theorem Proving*, June 2003.
- [12] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Proceedings of the Twentieth ACM Symposium on Principles of Distributed Computing (PODC)*, Newport RI, USA, August 2001.
- [13] P. Kumar and M. Satyanarayanan. Flexible and safe resolution of file conflicts. In *USENIX Winter*, pages 95–106, 1995.
- [14] J. P. Munson and P. Dewan. A flexible object merging framework. In *Proceedings of ACM CSCW’94 Conference on Computer-Supported Cooperative Work*, Technologies for Sharing I, pages 231–242, 1994.
- [15] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, Saint Malo, France, 1997.
- [16] M. Rabinovich, N. H. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases. In *Extending Database Technology*, pages 207–222, 1996.
- [17] P. L. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the ficus file system. In *USENIX Summer*, pages 183–195, 1994.
- [18] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW’96)*, pages 288–297, Boston, Massachusetts, USA, November 1996.
- [19] Y. Saito and H. M. Levy. Optimistic replication for internet data services. In *International Symposium on Distributed Computing*, pages 297–314, 2000.
- [20] S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32(4):403–445, 2001.
- [21] 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)*, pages 36–45, Orlando, Florida, USA, February 1998. IEEE Computer Society.
- [22] M. Suleiman, M. Cart, and J. Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP’97)*, pages 435–445. ACM Press, November 1997.
- [23] C. Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(4):309–361, December 2002.
- [24] C. Sun and D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(1):1–41, March 2002.
- [25] 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)*, 5(1):63–108, March 1998.
- [26] W. F. Tichy. RCS – A system for version control. *Software-Practice and Experience*, 15(7):637–654, July 1985.
- [27] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW’00)*, Philadelphia, Pennsylvania, USA, December 2000.