MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique
Département de Mathématiques et d'Informatique
Filière : Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : **Ingénierie des Systèmes d'Information**

THEME :

Une Transformée Opérationnelle pour l'édition collaborative dans les réseaux P2P.

Etudiant(e) : Hachelaf Houria Salima.

Encadrant(e) : Mechaoui Moulay Driss.

Année Universitaire 2015/2016

Résumé :

Les réseaux P2P utilisés aujourd'hui peuvent regrouper des millions d'utilisateurs à la fois. Pour faciliter les échanges de données entre ces derniers, ces systèmes s'appuient sur des approches de partage permettant de dupliquer les objets partagés sur le réseau. Cependant, les modifications intervenues sur une copie ne peuvent être propagées correctement étant donné que les mises-à-jour se font en concurrence sur l'ensemble des répliques et que les utilisateurs peuvent joindre ou quitter le réseau à n'importe quel moment. L'approche des Transformées Opérationnelles TO développée dans ce contexte permet d'assurer la cohérence des copies en imposant la vérification de certaines conditions de convergence. Dans ce mémoire, nous allons étudier les méthodes de l'approche TO qui permettent de contrôler les mises-à-jour concurrentes dans un système d'édition collaborative et proposer une nouvelles structure de données capable d'assurer la cohérence des données partagées et de supporter les propriétés des réseaux P2P tels que les déconnexions fréquentes et les délais de propagation des messages.

Sommaire

I. Introduction générale

I.1. Contexte :

Ces vingt dernières années ont vues une évolution majeure des systèmes et du matériel informatique, à savoir le passage d'une architecture centralisée vers une architecture distribuée. Le progrès continu de ces technologies informatique donne naissance à une nouvelle forme de communication celle du « Peer-to-Peer ». Parallèlement, les applications basées essentiellement sur les réseaux tels que les systèmes d'édition collaborative essayent de s'adapter aux nouvelles propriétés de cet environnement **[1]**.

Ces systèmes ont été conçus pour permettre à un ensemble d'utilisateurs de travailler sur un même document à partir de plusieurs sites. Ce travail collaboratif vise à faciliter le partage des données et permet de bénéficier des connaissances et des compétences de plusieurs utilisateurs. Les échanges d'informations entre ces derniers peuvent suivre différentes formes : synchrone ou asynchrone. Dans le cadre d'un échange rapide et correcte, il est préférable de répliquer le document partagé sur un réseau P2P afin qu'il soit facilement accessible. Or, les modifications parallèles du contenu de ces copies peuvent causer d'éventuelles incohérences. L'un des principaux problèmes dans l'édition collaborative est donc le maintient de la cohérence du document partagé.

Les solutions proposées dans ce contexte peuvent suivre deux approches: optimiste et pessimiste. L'approche pessimiste donne l'impression aux utilisateurs de travailler sur une même copie, elle est également capable d'assurer une forte cohérence grâce au principe de verrouillage tel que dans la transaction de base de données. Cependant, il s'est avéré que ce mécanisme est impropre à l'édition collaborative en raison du temps de réponse très élevé dû au délai entre la demande et l'acquisition d'un verrou **[2]**.

En revanche, l'approche optimiste est plus convenable pour une édition collaborative en autorisant les opérations des mises-à-jour simultanées. Le système tolère la divergence entre les copies au cours des modifications mais veille à ce que les copies convergent dans une date ultérieure. En particulier, une approche optimiste dite « Transformée Opérationnelle » ou plus brièvement TO a été spécialement conçue pour répondre aux exigences de l'édition collaborative. Dans cette approche, des transformations sont réalisées sur les opérations de mises-à-jour d'une manière qui peut assurer la convergence des copies. **[2]**

Les fonctions de transformation déjà développées pour les systèmes centralisés imposent un ordre de sérialisation unique c'est-à-dire que les opérations échangées doivent être exécutées dans le même ordre sur tous les sites. Dans le contexte distribué, le modèle TO n'exige aucun ordre total mais impose en contrepartie aux fonctions de transformation de satisfaire des propriétés supplémentaires pour garantir la convergence. **[3]**

De nombreux algorithmes ont été développés depuis les premiers travaux d'Ellis et al, un algorithme a été développé en 1996 principalement par Ressel **[4].** La première propriété « TP1 » à satisfaire dans cet algorithme définit une identité d'états. Elle stipule que, pour deux opérations O1 et O2, l'état produit en exécutant O1 avant O2 est le même que celui

résultant de l'exécution de O2 avant O1. La propriété « TP2 » définit une identité d'opérations. Elle stipule que le résultat de la transformation d'une opération par rapport à une séquence d'opérations concurrentes ne dépend pas de l'ordre selon lequel les opérations de cette séquence ont été transformées. Les autres procédures proposées dans ce cadre sont fortement dépendantes du type de données partagées. Si un développeur d'éditeurs collaboratifs veut fournir le partage d'un type de données spécifique, il doit créer les fonctions de transformation adéquates et prouver par la suite qu'elles sont conformes aux deux propriétés « TP1 » et « TP2 ».

I.2. Problématique :

L'analyse des fonctions de transformations par les chercheurs démontre que les deux propriétés « TP1 » et « TP2 » sont suffisantes pour assurer la cohérence des répliques. Bien que ces deux propriétés soient clairement formulées, il est difficile de concevoir des fonctions de transformation les respectant. En effet, le nombre élevé de nœuds et d'opérations parallèles que peut supporter un système P2P ainsi que leur mouvement instable sur le réseau représente un obstacle devant la satisfaction de la propriété TP2. Certaines solution proposent de fixer la taille du réseau ce qui est incompatible avec les caractéristiques P2P, d'autres solutions nécessitent l'utilisation d'un vecteur d'horloge pour maintenir un ordre total sur les opérations ce qui est d'un coût élevé et limite par ailleurs le passage à l'échelle. **[2, 3]**

Jusqu'à l'heure actuelle, toutes les fonctions de transformation proposées ne répondent pas sur la propriété « TP2 », toutefois, une fonction de transformation qui assure la convergence sera d'un grand intérêt pour le développement d'applications dans les environnements P2P **[2]**.

I.3. Objectif et contributions :

Dans le contexte de ce projet, le principal objectif est de fournir un nouveau type de données capable d'assurer la cohérence des répliques tout en supportant la mobilité des nœuds du réseau et pouvant passer à l'échelle en nombre d'utilisateurs. Pour ceci nous devons d'abord étudier les techniques déjà proposées pour la réplication des données dans les systèmes P2P.

Nous allons dans un premier temps essayer de présenter les situations de divergences rencontrées dans une édition collaborative décentralisées. Nous allons également prouver, à travers le déroulement de plusieurs scénarios, que les procédures proposées dans l'approche TO ne permettent pas de résoudre ces conflits.

La seconde contribution peut être une solution efficace pour le problème de la convergence dans le cadre de l'approche TO. Cette solution s'appuie sur un nouveau type de données pour lequel nous allons développer les fonctions de transformations nécessaires pour assurer la convergence. On essaiera d'apporter des améliorations sur les fonctions de transformation

déjà publiées et en dernier lieu nous allons appliquer cette nouvelle fonction de transformation sur un algorithme d'intégration dans le but d'évaluer son fonctionnement.

I.4. Organisation du document :

Ce mémoire s'organise de la manière suivante :

Dans le chapitre 1, Nous allons fournir un aperçu général sur les éditeurs collaboratifs basés sur l'approche TO qui existent aujourd'hui, on passe ensuite à leur définition ainsi que leurs modèles et leurs caractéristiques sur les réseaux P2P. On introduit également la notion de la réplication avec ses différents types et les restrictions qu'elle mette sur les interactions des collaborateurs dans les réseaux P2P et nous déterminons en dernier lieu les critères qu'un éditeur collaboratif massif doit obéir pour assurer la cohérence (modèle CCI) ainsi que les approches permettant de réconcilier la divergence des copies.

Dans le chapitre 2, nous allons présenter l'approche TO sur laquelle s'articule ce projet, nous évaluons chacun de ses composants ainsi que les fonctions de transformation respectant les propriétés « TP1 » et « TP2 » pour enfin passer à la description des algorithmes d'intégrations les plus utilisés dans cette approche.

Nous verrons dans le chapitre 3 l'algorithme d'intégration dit « ADOPTED » qui semble le plus approprié pour atteindre notre objectif. Nous allons détailler chacune de ses procédures, décrire son fonctionnement pour l'intégration des opérations distantes et présenter finalement les limites que rencontre cet algorithme lors de la gestion des différentes opérations.

Dans le dernier chapitre nous proposons une nouvelle structure capable d'assurer la convergence des données dans un environnement d'édition collaborative P2P indépendamment de l'ordre d'exécution des opérations d'édition sur les différents sites, nous décrivons également son fonctionnement et nous présentons en dernier lieu l'outil que nous avons élaboré permettant l'utilisation de ce nouveau mécanisme de réplication.

I.5. Etat de l'art:

Les plateformes d'édition collaboratives qui existent aujourd'hui ont été construites pour faciliter le partage des données et bénéficier des connaissances et des compétences de plusieurs d'utilisateurs, à titre d'exemple le système de control de version SVN (abrégé de Subversion) qui a été conçu pour permettre l'édition collaborative des logiciels, et la plateforme d'édition collaborative IceCube qui se base sur une approche de convergence. GoogleDoc représente à son tour un système d'édition collaborative en ligne permettant de créer, éditer et partager rapidement des documents à partir de n'importe quel navigateur en utilisant un compte email GMAIL **[5, 6, 7].**

I. Introduction générale

Le Wiki représente l'un des outils les plus utilisés pour l'édition des documents où les utilisateurs ont toujours le droit de modifier le contenue de n'importe quelle page **[3].** Cependant, l'étude et l'analyse de ces plateformes montre que l'utilisation d'une architecture client serveur dans ces systèmes est un point de faiblesse qui les rend vulnérable aux pannes. L'architecture Peer-to-Peer permet de pallier les limites des systèmes centralisés et garantit le passage à l'échelle, or le mouvement dynamique des paires pose un problème lors des mises-à-jour des copies distribuées.

Les approches développées pour résoudre ce problème sont nombreuses, chacune de ces approches fut développée pour un type de données spécifique : documents textes, documents graphiques (image), fichiers XML…, elles s'appuient sur différentes techniques de réplication et doivent suivre des mécanismes pour garantir la convergence des copies. Ces approches sont principalement : l'approche multi-versions **[2],** l'approche SRC (sérialisation / Résolution de conflits) **[2],** l'approche CRDT **[8]** (types de données commutatives répliquées) et l'approche TO (Transformée Opérationnelle) qui fait l'objet de notre étude **[2].**

L'approche TO exige des transformations des opérations d'édition à l'aide d'un ensemble de procédures. Cette approche est applicable sur un réseau P2P. Le critère de la convergence ne sera respecté que si ces procédures de transformation satisfassent les deux propriétés TP1 et TP2 **[2].** Cependant, tous les algorithmes de transformation proposés dans la littérature se heurtent à la satisfaction de la condition TP2. A titre d'exemple, l'algorithme SOCT4 conserve la généralité de l'approche TO et propose un ordre total sur l'intégration des opérations pour remplir TP2 **[6].** Le maintient de l'ordre total nécessite la présence d'un site central ou un ensemble stable des sites. Ces contraintes empêchent le SOCT4 d'être utilisé dans un environnement décentralisé **[3].**

Les solutions proposées **[4]** comme l'algorithme d'Ellis, l'algorithme de Ressel, l'algorithme de Du Li, l'algorithme de Sun **[2],** l'algorithme de Suleiman sont tous révélés inefficaces face à la grande mobilité des clients et aux traitements d'un très grand nombre d'opérations parallèles augmentant ainsi les coûts de gestion des documents partagés d'une manière indéfinie et limitant par conséquent le passage à l'échelle **[3].**

Liste des figures

II. Chapitre I : L'édition collaborative

II.1. Introduction :

Les applications d'édition collaborative connaissent un grand essor grâce au développement rapide des technologies informatiques ainsi qu'à l'expansion des services du Web 2.0 qui ont rendu l'Internet plus interactive et plus collaborative dans les dernières années. Cependant, les applications d'édition collaborative classiques ne permettent pas de supporter ce passage à l'échelle puisqu'elles ne sont pas conçues pour soutenir une telle charge de données et d'utilisateurs.

Dans ce chapitre, nous allons présenter un état de l'art relatif à la notion de l'édition collaborative ainsi que les aspects à mettre en œuvre pour supporter l'engagement de plusieurs utilisateurs. Tout d'abord, nous présenterons de manière détaillée la notion du travail collaboratif, les grandes catégories des systèmes élaborés dans ce contexte ainsi que les solutions proposées dans la littérature pour élargir les possibilités actuelles de la collaboration sur les différents types de réseaux. Dans un second temps, nous allons voir les techniques utilisées pour traiter les problèmes de la réconciliation lors d'une édition concurrentielle sur les infrastructures de communication.

II.2. Les systèmes d'édition collaborative :

Le développement des réseaux et du matériel informatique favorise la collaboration entre les utilisateurs même s'ils sont distribués dans l'espace ou dans le temps. Les formes de collaboration qui apparaissent sont variées mais s'appuient toujours sur le même principe, celui de faire collaborer un groupe de personnes situées dans des lieux différents et leurs permettre de partager et de travailler simultanément sur un ensemble de documents de différents types : texte, graphe, dessins, figure, vidéo, programme, XML…. **[1, 2, 7]**

Les personnes connectées travaillent sur un projet commun en interagissant à distance et à n'importe quel moment grâce aux systèmes d'édition collaborative. Ces derniers ont été construits pour faciliter la coopération du travail et rendre l'ensemble d'objets partagés visibles et accessibles à tout moment. Ils doivent également fournir aux participants un environnement simple, réactif et de haute disponibilité ce qui permet de réduire remarquablement le temps d'un projet. **[1, 2, 7]**

Dans les années récentes, les systèmes d'édition collaborative connaissent un usage courant que ce soit dans le domaine personnel tel que dans les forums interactifs de discussion, le domaine professionnel qui inclut le développement des logiciels « open source », la rédaction d'articles scientifiques par plusieurs chercheurs, la régulation de trafic visant à détecter les embouteillages ou les perturbations en temps réel par communication entre véhicules et dans beaucoup d'autre domaine visant à éditer des documents numériques (texte, image, son …) par plusieurs utilisateurs.**[8]**

II.2.1. Les modèles d'édition collaborative :

Il existe plusieurs modèles pour la mise en œuvre d'un système d'édition collaborative. On distingue généralement deux modèles élémentaires pour ces systèmes, le premier appelé centralisé qui est à l'origine la principale organisation pour les réseaux informatiques, le deuxième appelé décentralisé ou Peer-to-Peer qui est apparu pour pallier les insuffisances du premier.

II.2.1.1. Le modèle centralisé :

Un système est dit centralisé lorsque la communication entre les membres passe nécessairement par une entité appelée la tête du réseau. Cette entité permet d'initier et d'administrer les ressources informatiques qui circulent sur le réseau de données tout au long de leur chemin aux utilisateurs **[9]**.

Dans les réseaux informatiques, cette communication fonctionne selon un environnement appelé Client/ Serveur. Cela signifie que des machines clientes (des machines faisant partie du réseau) contactent une machine généralement très puissante en termes de capacités d'entrés-sorties appelée « Serveur » et que seule cette entité permet de gérer les ressources matérielles, les espaces de stockage, les traitements de données et les puissances de calcul. **[10]**

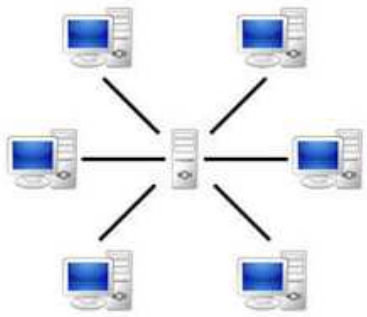


Figure 1: Le modèle centralisé (Client/ Serveur).

Le modèle Client/serveur représente un modèle de fonctionnement logiciel standard c'est-à-dire qu'il peut s'adapter sur tout type d'architecture matérielle une fois les appareils interconnectés entre eux. Les communications dans cet environnement ne sont que des successions d'opérations dites « requêtes-réponses » où chaque requête est définie par le message transmis d'un client vers un serveur et décrivant l'opération à exécuter pour le compte de ce client et dont chaque réponse représente le message transmis d'un serveur vers un client suite a l'exécution d'une opération et contenant le résultat de cette opération. **[10]**

Le processus client est à l'initiative de cette communication. Il représente une machine, généralement un ordinateur personnel ou des appareils individuels (Smartphones, tablettes…) équipé avec des applications du réseau qui sont conçues pour demander et recevoir des données sur la durée du réseau. A la phase de réception: le client doit attendre les résultats et assurer la bonne présentation de ces derniers. **[11]**

Le processus serveur représente un entrepôt de fichiers, de dossiers, de bases de données et un ensemble d'applications encore plus compliquées qui doit être actif et doit attendre les demandes imprévisibles des clients. A la demande d'un client, le serveur doit récupérer les données pour effectuer le traitement et doit également rendre le service et les résultats au client concerné. Le serveur étant l'élément de base dans ce système doit être à l'écoute des clients et capable de traiter le grand nombre des demandes reçues à tout moment. **[11]**

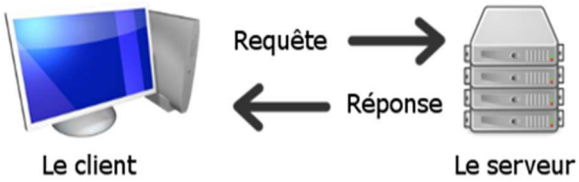


Figure 2: La communication Client/ Serveur.

Si le nombre de collaborateurs augmente lors d'une session collaborative, plusieurs opérations vont être exposées au serveur au même moment. Dans ce cas, le serveur risque de perdre ses capacités de fonctionnement face au traitement d'un très grand nombre de requêtes simultanées **[5]**. La prise en charge d'une énorme quantité de client sans être submergé par la charge nécessite une machine robuste et rapide, qui fonctionne 24h/24 ainsi que la présence d'administrateurs systèmes et réseaux pour la gestion des serveurs. En outre, l'architecture Client/ Serveur est d'un coût élevé lié à la maintenance et à la technicité du serveur car ce dernier est le seul maillon faible du réseau étant donné que tout le réseau est architecturé autour de lui. Le serveur doit donc être capable d'assurer une grande tolérance aux pannes et doit protéger au maximum les données pour garantir un fonctionnement au moins minimum en cas de panne puisque toute panne du serveur entraine l'arrêt total du fonctionnement du système dans sa globalité.

Pour couvrir ces limites, des solutions proposent de distribuer les tâches de calcul et de traitements sur plusieurs entités dans le réseau afin de préserver les propriétés nécessaires pour son fonctionnement. C'est d'où vient l'idée de créer une architecture décentralisée **[8]**.

II.2.1.2. Le modèle décentralisé:

Un système décentralisé (appelé aussi Pair-à-Pair, Poste-à-Poste, égale-à-égale) est constitué d'un ensemble de nœuds (ordinateurs, machine, mobiles, …) interconnectés par l'intermédiaire d'un réseau. Dans un system pair-à-pair, chaque nœud est une entité réseau complète qui remplit à la fois le rôle du client et celui du serveur lors des échanges de fichiers. De ce fait, chaque membre peut envoyer et recevoir des informations à un instant donné **[9]**.
L'ensemble de pairs est capables de s'auto-organiser au sein du réseau dans le but d'échanger ou partager des services et des ressources telles que les données de différents types, des programmes, des capacités de stockage ou de calcul …etc. **[12]**

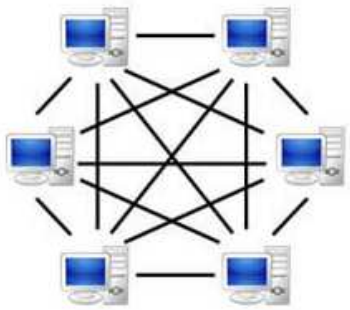


Figure 3: Le modèle décentralisé (P2P)

Le but avoué de cette architecture est d'améliorer les performances et d'augmenter la fiabilité du système. En effet, le P2P offre une communication directe entre les postes ce qui permet de moins solliciter le serveur et par conséquent d'alléger sa charge de travail. Par ailleurs, le P2P permet de mutualiser les puissances de calculs non-exploités par les ordinateurs des collaborateurs permettant ainsi de diviser et d'équilibrer la charge de travail entre ces derniers **[13]**.

II.2.2. Caractéristiques P2P :

Les systèmes P2P sont considérés comme étant des systèmes ouverts, autonomes, tolérants aux pannes et distribués à grande échelle. A vrai dire, chaque membre peut gérer par lui-même un ensemble d'applications et de ressources (autonomie), il a également la capacité d'effectuer ses traitements indépendamment des autres pairs (décentralisation) et peut fournir à tout moment des ressources ou des services aux autres nœuds afin de pouvoir joindre et utiliser le système (coopération). En cas de panne d'un pair, le dysfonctionnement reste localisé et le système continu tout de même à fonctionner, la mise en place d'une grande entité de sauvegarde avec les risques de perte d'information n'est donc pas nécessaire puisque les données sont réparties sur le réseau. De ce fait si un nœud tombe en panne le système reste toujours fonctionnel (tolérance aux pannes).

Par ailleurs, ce modèle assure une meilleure disponibilité de données en considérant que les sauvegardes sont réparties dans le réseau évitant ainsi toute perte d'informations (disponibilité). On peut dire alors que le système reste toujours fonctionnel et capable de gérer les données et les utilisateurs quelque soit leur nombre et leur comportement sur le réseau (passage à l'échelle). **[12]**

Ces caractéristiques sont donc très utiles pour les systèmes d'édition collaborative. Lors d'une session collaborative, le nombre élevé des échanges de fichiers ne peut influencer sur les performances du système grâce aux traitements répartis des données car, contrairement au modèle centralisé, la qualité et la quantité des données sont proportionnelles au nombre d'utilisateurs **[5]**.

Cependant, Les points négatifs des systèmes P2P tels que le dynamisme posent certains problèmes. En réalité, les nœuds du réseau peuvent joindre ou quitter le système dynamiquement sans contrôle global, ce qui implique l'instabilité de l'ensemble des ressources et des services du système. Il n'est donc pas possible de connaitre préalablement et précisément la structure du réseau à chaque instant et l'ensemble des données et des services disponibles. **[13]**

II.2.3. La réplication :

Afin d'assurer la résistance aux pannes et d'avoir un système robuste il est possible de répliquer les documents partagés auprès des nœuds du réseau. Ceci permet de faciliter et d'accélérer considérablement l'accès aux données. D'après sa définition, un réseau P2P peut supporter un très grand nombre d'utilisateurs se trouvant dans des zones géographiques distantes. Lors du lancement d'une opération (lecture ou écriture du document partagé) par un utilisateur, il est possible de passer par un nombre important de pairs pour atteindre le nœud souhaité. D'autre part, l'information détenue par ce nœud peut être de taille non négligeable ou encore stockée sur plusieurs postes du réseau. Cependant, la latence impliquée (temps de réponse sur la requête) et les risques qui menacent la continuité et la rapidité d'accès aux données peuvent augmenter considérablement. L'infrastructure de communication existante limite donc les possibilités d'accès aux données.

Le besoin d'améliorer les performances du réseau a conduit à la conception d'une infrastructure distribuée et partageable mettant de partager et de dupliquer les données dans le système. Il s'agit d'un processus de réplication où tous ou bien certains objets partagés seront copiés et sauvegardés auprès de chaque site à travers le monde. Ces copies sont dites répliques **[2]**.

La manipulation parallèle des répliques ne doit pas provoquer des pertes d'informations, le système doit donc être capable de garantir la persistance des données partagées quelque soit les conditions de communication. Les procédures d'organisation et de gestion des répliques deviennent ainsi compliquées à établir car pour chaque requête générée sur un dossier (une réplique), il est absolument indispensable de reproduire et de propager cette opération sur toutes les autres répliques.

Les contraintes sur les données partagées sont aussi différentes et suivent dans la plus part du temps le type de l'application collaborative c'est-à-dire le mode de collaboration et l'infrastructure de communication. Pour les applications qui exigent de voir rapidement les mises-à-jour comme les compagnies aériennes, les banques, les sociétés de bourse et la plus part des jeux multi-joueurs où les interactions entre les membres sont nombreuses, ce type d'applications nécessite une forte synchronisation pour les données qui sont modifiées fréquemment, on privilégiera donc un mode de réplication synchrone. Par contre, dans les applications où les données sont faiblement structurées comme dans les forums, les données

échangées sont de natures différentes : date, auteur, sujet, corps d'un message… ou comme dans les communautés professionnelles : base de données, agenda de groupe, édition collaborative d'un document quelconque… on peut autoriser une collaboration entre des utilisateurs distants et potentiellement déconnectés c'est-à-dire asynchrone. **[7, 14]**

II.2.3.1. La réplication synchrone vs asynchrone :

Dans un mode de réplication asynchrone, les transactions effectuées sont regroupées et sauvegardées suivant leurs ordre d'exécution sur un site et sont diffusées plus tard aux autres pairs du réseau à l'aide d'un processus de synchronisation. Dans ce mode de réplication, les utilisateurs ont toujours accès au document partagé même s'ils possèdent d'une version plus ancienne. Cependant, les données modifiées ne sont pas assez sécurisées **[14]**.
A vrai dire, la séquence de transactions effectuée sur un site est stockée dans une file locale afin qu'elle soit diffusée ultérieurement aux autres nœuds distants. En cas de panne du site serveur, les transactions seront perdues et ne peuvent jamais être récupérées **[14, 15]**.

Pour la réplication synchrone, les répliques sont traitées en temps réel. Le système effectue des mises–à-jour immédiates sur toutes les copies suite au lancement de la moindre transaction. Les requêtes sont déployées sur l'ensemble des bases de données avant leur validation sur le poste la où elles ont été générées **[2]**. Ces opérations sont aussitôt propagées qu'elles peuvent être exécutées dans un ordre quelconque sur les sites distants provoquant ainsi des défaillances de certaines répliques. On peut dire alors que les répliques ne se trouvent pas dans le même état à un moment donné, de même, les répliques ne gardent pas leurs cohérences par rapport à la dernière version du document **[14, 15]**.

Pour couvrir ces problèmes de mises-à-jour, plusieurs solutions envisageables ont été proposées, certaines utilisent le principe de verrouillage des écritures tel que dans les transactions de base de données, une fois les mises-à-jour propagées (les données atteignent les sites distants) et enregistrées le système autorise les écritures à nouveau sur les répliques. D'autres solutions utilisent des mécanismes d'ordonnancement sur les transactions distantes, à chaque opération est attribué un identifiant unique indiquant son ordre d'exécution sur les nœuds distants. Cette opération risque d'attendre indéfiniment si les opérations qui la précèdent n'ont pas été reçues **[4, 14]**.

Cependant, les verrous sur un ensemble de nœuds dynamiques sont d'un coût très élevé et le maintient d'un ordre total nécessite la présence d'un serveur central ce qui est incompatible avec les caractéristiques pair-à-pair. Même si ces solutions permettent au système de résister aux pannes, la gestion d'un très grand nombre d'utilisateurs et d'opérations dans un intervalle de temps rétréci ne sera pas garantie dans toutes les situations et le système passe ainsi difficilement à l'échelle **[3]**. Ces solutions sont étroitement liées aux deux mécanismes de réplication dites pessimiste et optimiste **[2]**.

II.2.3.2. Le mécanisme de réplication pessimiste :

Dans un mécanisme de réplication pessimiste les utilisateurs ont l'impression de travailler sur une seule et même copie. A vrai dire, les écritures sur les répliques sont effectuées d'une manière séquentielle, le système tolère les opérations de lecture indépendantes et simultanées, cependant une seule opération d'édition est autorisée à un moment donné. En d'autres termes, si un utilisateur effectue la mise à jour d'une réplique, le système doit interdire ou bloquer le reste des utilisateurs à effectuer simultanément des opérations de modifications. Ceci implique que le système doit utiliser un mécanisme de verrouillage pour bloquer les opérations d'édition parallèles et par conséquent assurer la cohérence des répliques **[4]**. Les opérations d'écritures sont faites simultanément et ne doivent pas êtres interrompues ce qui exige la mise en œuvre d'un ensemble de techniques de synchronisation par un site central et une connexion stable et régulière vers ce site.

Cette approche ne peut donc être utilisée sur les réseaux P2P vu le mouvement dynamique des entités qui perturbe le processus de verrouillage. Par ailleurs, ce mode de réplication nécessite des moyens très coûteux en raison principalement de la bande passante nécessaire pour assurer un échange parfait entre les différents membres du réseau lors des opérations d'écritures.

II.2.3.3. Le mécanisme de réplication optimiste :

Dans l'approche optimiste, chaque utilisateur manipule sa copie indépendamment des autres. Ce mode de réplication est considéré comme étant un moyen efficace facilitant le passage à l'échelle ainsi que la tolérance aux pannes en s'appuyant sur le principe de l'exécution, la diffusion et la réexécution. Les opérations d'écriture peuvent être effectuées en parallèles sur l'ensemble des répliques. Les modifications effectuées sur un site ne sont pas envoyées immédiatement vers les autres cependant, lorsqu'elles sont diffusées, elles peuvent s'exécutées en parallèle engendrant des situations de conflit **[16].**

Dans cette approche les états des copies ne sont pas stables au cours des modifications mais doivent finir par avoir la même valeur une fois les modifications seront effectuées sur toutes les répliques **[4]**. Ce mécanisme de réplication peut donc s'appliquer facilement dans un environnement P2P et semble plus convenable pour une édition collaborative que le mécanisme précédant. **[5]**

Afin d'assurer la cohérence entre les répliques, il est certain que ces systèmes utilisent des mécanismes pour résoudre les conflits et permettre à l'ensemble des répliques de converger vers la même valeur une fois le système au repos. Plusieurs modèles de cohérence ont été élaborés parmi lesquels on peut citer le modèle CCI **[4]**.

II.3. Le modèle de cohérence CCI :

II.3.1. Respect de la causalité :

Le terme « causalité » désigne une relation de précédence entre deux évènements ou plus. Dans une édition collaborative, on parle de causalité lorsqu'il existe une relation de précédence entre les différentes opérations. Le respect de la causalité signifie que l'ordre des opérations reçues peut être maintenu, on dit que les opérations sont reçues dans un ordre causal.

Afin d'assurer une relation de précédence causale entre n'importe quel pair d'opérations (O1, O2), on doit assurer que O1 doit toujours s'exécuter avant Op2 sur tous les sites. Supposons les deux opérations O1 et O2 générées respectivement sur les deux répliques A et B.

L'opération O1 précède causalement l'opération O2 (noté *O1 -> O2*) dans seulement l'une des situations suivantes **[1]** : soit A et B deux répliques,

- Si A et B sont identiques (le même site) et l'opération O1 a été générée avant l'opération O2.
- Si A et B ne sont pas identiques (deux sites différents) et l'opération O1 a été exécutée sur B avant que l'opération O2 soit généré sur ce site.

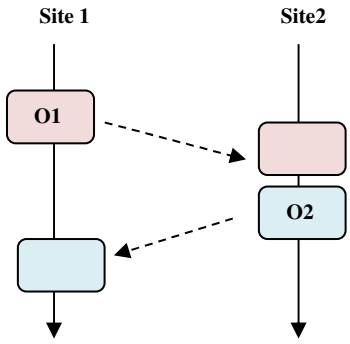


Figure 4: Principe de la causalité

Pour expliquer cet ordre d'apparition des opérations, on suppose que l'opération O2 dépend des effets de l'opération O1. Dans ce cas, si le respect de la causalité est préservé, les mises-à-jour seront exécutées dans le même ordre sur tous les sites du réseau et par conséquent les répliques vont finir par avoir la même valeur.

II.3.2. Préservation de l'intention :

Lorsque les opérations s'exécutent en concurrence sur les répliques, elles risquent d'être exécutées dans un ordre quelconque. Dans ce cas, si une opération distante est reçue sur un site, elle doit tenir compte des effets des opérations locales qui viennent d'être exécutées sur ce site avant son arrivée. L'effet qu'apporte cette opération sur l'état de la réplique désigne en quelque sorte « l'intention » de cette opération **[1, 3]**. Aucune méthode formelle n'est adaptée pour préserver l'intention car cela dépend dans la plus part du temps du type et de la structure de données utilisés par le système **[8]**.

II.3.3. La convergence :

Dans une architecture distribuée, le terme convergence est utilisé pour désigner que le système est capable de refléter correctement les modifications intervenues sur un document vers toutes les autres copies de ce document. Ceci revient à dire que le système doit permettre d'établir une cohérence entre les répliques ce qui est difficile à assurer dans certaines situations **[6]**. Dans un mode de réplication pessimiste ou asynchrone, le problème de la convergence ne se produit pas souvent grâce au mécanisme du verrou et celui de la file de références locale qui permettent un déroulement séquentiel des opérations de modifications, contrairement au mode de réplication optimiste ou synchrone où les transactions sont effectuées d'une manière concurrente sur l'ensemble de sites engendrant ainsi des conflits au niveau des répliques. En réalité, ce désordre d'exécution implique que les répliques ne finissent pas par avoir la même valeur. On dit alors qu'elles sont divergentes. **[8]**

II.3.3.1. Types de la convergence :

La convergence se présente également sur différents types qui sont **[1, 8]**:

- **La convergence forte :** dans ce type de convergence les répliques doivent être identiques sur tous les sites suite à la moindre modification. Le système doit donc garantir les mises-à-jour de l'ensemble des copies à chaque fois qu'un changement se produit.

- **Divergence limitée :** ce type de convergence n'est pas adaptée aux systèmes pair-à-pair car il tolère quelques défaillances des répliques. En effet, cette approche permet une divergence bornée en limitant généralement le nombre d'opérations à exécuter sur un site par un seuil.

- **Convergence à terme** : cette approche permet la divergence des répliques au cours des modifications à condition qu'elles finissent par converger. Les copies peuvent avoir des valeurs différentes tant que le système est actif, une fois les opérations de modification sur l'ensemble de nœuds sont propagées, le système doit conduire ces copies vers un état stable (les répliques sont identiques sur tous les sites). On peut dire alors que ce type de convergence est le plus convenable pour une édition collaborative.

II.4. Approches de gestion de la cohérence dans les éditeurs collaboratifs :

Nous avons vu précédemment que la cohérence des répliques représente le plus grand défi des systèmes d'édition collaborative. Pour garantir une édition cohérente, ces systèmes doivent vérifier le critère de la convergence tout en traitant des documents textuels, à savoir une suite de caractères ou plus généralement une suite d'objets finie ainsi qu'une suite d'opérations d'insertions et de suppressions qui peuvent s'exécuter en concurrence engendrant des situations de conflits.

Plusieurs algorithmes de convergence et de détection de conflits issus des opérations concurrentes ont été élaborés. Ces algorithmes peuvent suivre différentes approches dont certaines détectent les conflits en imposent un verrouillage sur les ressources ou un ordre total sur les opérations **[8]**. Ces modèles sont très efficaces pour assurer la cohérence des répliques, malheureusement ils ne sont adaptés aux architectures P2P dans lesquelles aucune entité centrale n'est présente pour le maintient d'un ordre sur les opérations.

L'approche des types de données commutatives répliquées (CRDT) exige que toutes les opérations d'édition soient commutatives afin d'être exécutées dans un ordre quelconque ce qui représente la plus grande difficulté à laquelle CRDT se bute. Ramsey et Csirma ont réussis à réaliser en 2001 un algorithme qui utilise une façon algébrique pour résoudre les conflits causés par l'exécution concurrente des opérations **[8]**. L'algorithme effectue les opérations concurrentes en utilisant un ensemble de règles de détection d'incohérences et garde le reste d'opération jusqu'à pouvoir résoudre les conflits. Il se peut alors que des opérations restent en suspens pendant un temps illimité. **[8]**

Une autre approche basée sur la transformation des opérations propose une transformation préalable des opérations distantes reçues par rapport aux opérations concurrentes au niveau d'un site à l'aide d'un algorithme de transformation **[2].**

Dans sa thèse, M. Ressel a démontré que cet algorithme de transformation peut assurer la convergence à condition que la transformée satisfasse deux propriétés. **[17]**

Cette approche dite Transformée Opérationnelle permet de garantir la convergence des documents sans imposer de contraintes de verrouillage ni d'ordre total, elle assure également que toutes les mises à jour soient faites sur l'ensemble des répliques même si ces opérations ne sont pas initialement commutatives, c'est pourquoi, dans la partie qui suit nous allons nous pencher plus vers cette approche qui semble beaucoup plus convenable pour une édition collaborative sur un réseau P2P. **[17]**

II.5. Conclusion :

L'édition collaborative devient de plus en plus massive avec l'émergence du web 2.0. Ce changement d'échelle met à mal les approches existantes qui n'ont pas été conçues pour soutenir une telle charge.

Afin de répartir la charge et d'améliorer le passage à l'échelle, les développeurs se penchent vers un nouvel environnement de communication celui du P2P. Les applications de collaboration basées sur les systèmes P2P sont capables d'assurer le passage à l'échelle, malheureusement ils amènent de nouveaux défis tels l'incohérence des documents partagés due à la quantité énorme d'utilisateurs et de données ainsi que leurs mouvement dynamique sur le réseau. Il est donc indispensable de fournir des mécanismes permettant de déployer des systèmes d'édition collaborative qui supportent les environnements P2P.

II. Chapitre I : L'édition collaborative

Dans la partie qui suit nous allons présenter une approche formelle appelée « Transformée Opérationnelle » pour obtenir la cohérence dans les systèmes d'édition collaborative et qui est susceptible d'assurer les caractéristiques P2P.

III.1. Introduction :

Afin de supporter une collaboration massive, plusieurs applications collaborative utilisent un modèle de réplication optimiste décentralisé basé sur l'approche des transformées opérationnelles (TO).

L'objectif de ce chapitre consiste à discuter la capacité de ce mécanisme à tolérer les caractéristiques des réseaux pair-à-pair. Pour cela, nous nous intéressons dans cette partie à étudier le fonctionnement et les composants de ce mécanisme qui doit assurer les trois critères du modèle CCI vues dans le chapitre précédent : respect de la causalité, préservation de l'intention ainsi que la convergence. Cette approche doit de plus être compatible avec les caractéristiques des environnements P2P qui perturbent la continuité et la rapidité du fonctionnement du réseau tels que la dynamicité des pairs, le grand nombre d'utilisateurs, de données et de modifications.

III.2. Définition de l'approche des Transformées Opérationnelles (TO):

L'approche des **Transformées Opérationnelles** (plus brièvement TO) est un mécanisme de réplication optimiste proposé par la communauté des éditeurs collaboratifs synchrones pour pallier les problèmes de l'incohérence des copies causés par les opérations concurrentes.
Cet ensemble de techniques qui s'applique sur un ensemble des nœuds répartis sur le réseau et possédant chacun une copie du document partagé c'est-à-dire réplique a été élargi aux éditeurs collaboratifs asynchrones ainsi que sur les structures linéaires et hiérarchiques pour enfin l'employer dans l'édition des pages Wiki **[5]**. Chaque réplique est modifiée par l'exécution d'une opération générée localement ou reçue d'un site distant **[2].**

Cette approche permet au système de résister aux mises-à-jour simultanées de l'objet partagé même si ces mises à jour sont diffusées dans un ordre quelconque vers toutes les répliques. Ceci est réalisé grâce à la journalisation des différentes opérations au niveau de chaque copie. Les journaux des différentes copies ne sont pas identiques cependant ils sont considérés comme étant équivalents puisque l'ensemble des opérations exécuté à partir d'un même état de départ donne le même état d'arrivée mais que l'ordre d'exécution des opérations concurrentes est différent d'un journal à un autre.

Les transformations sont effectuées sur les opérations reçues des autres sites par rapport aux opérations locales avant de les exécuter. Le système est donc mené vers un état stable où toutes les mises-à-jour sont effectuées correctement et par conséquent toutes les répliques sont identiques et convergent vers la même valeur **[4]**.

L'opération d'édition est concrètement réalisée grâce à un mécanisme basé principalement sur deux composants qui sont:

- **Une procédure d'intégration :** son rôle est de garantir la diffusion, la réception et l'exécution des différentes opérations de mises-à-jour. Chaque opération générée ou exécutée localement doit être stockée dans un journal (appelé aussi histoire ou Log).
L'algorithme d'intégration doit assurer la gestion des opérations locales et distantes sur lesquelles les transformations vont être effectuées à l'aide d'une fonction de transformation **[4, 5]**.

- **La fonction de transformation :** cette fonction est définie par l'ensemble des procédures assurant la fusion des opérations de mises-à-jour concurrentes en prenant en considération l'historique des opérations précédentes. En d'autres termes, cette fonction doit prendre en compte l'impact des opérations déjà exécutées sur celle qu'on veut appliquer au niveau d'une réplique afin d'appliquer les bonnes transformations pour l'obtention d'une édition cohérente. Cet ensemble de procédures est spécifique pour chaque type de données **[4]**.

L'algorithme d'intégration est défini comme correct si il assure la causalité, la convergence et la préservation de l'intentions c'est-à-dire le modèle CCI. Cette approche est susceptible d'assurer le modèle CCI ainsi que les caractéristiques des systèmes P2P car en assurant le critère de la causalité, la possibilité d'avoir des répliques identiques augmente.

Cependant dans certaines situations la dépendance entre les opérations n'est pas suffisante pour assurer la cohérence. Par exemple lorsque les opérations sont reçues en concurrence, le principe de la causalité ne sera plus valide pour assurer la convergence puisque aucune opération ne dépend causalement de l'autre **[1, 4]**.

Dans le but de montrer la divergence entre les répliques qui exécutent les opérations de modification d'une manière concurrente, le scenario de la Figure 4 fait intervenir deux opérations sur la même chaine initiale « Date ».

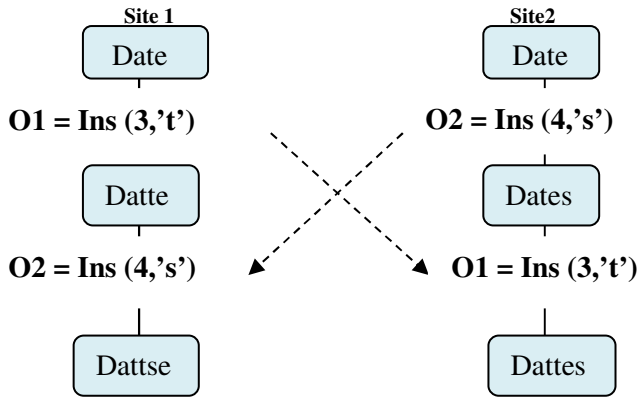


Figure 5: Exécution concurrente de deux opérations.

Cette figure illustre l'exécution parallèle des deux opérations d'insertion O1= Ins (3,'t') et O2= Ins (4,'s') sur le même état initial « Date ». Cette chaine est tout d'abord répliquée sur les deux sites « site1 » et « site2 ». Au moment ou le site 1 exécute sur sa réplique locale l'opération Ins (3,'t') pour insérer le caractère « t » à la troisième position, le site2 insère le caractère « s » a la position 4 en exécutant l'opération Ins (4,'s'). Après l'intégration mutuelle de ces opérations sur les deux sites, une situation de divergence entre les répliques finales est obtenue.

Cette incohérence est due à l'intégration directe des opérations d'insertion sur les répliques distantes. Lorsque ces opérations sont envoyées vers le site distant elles seront directement exécutées sur ce site sans prendre en considération l'effet de l'opération locale sur l'état de la réplique. Puisque ces opérations sont appliquées sur deux états différents « Dates » et « Datte » le résultat de leur exécution sera automatiquement différent.

On remarque qu'il est possible de recalculer la nouvelle position d'insertion d'un caractère sur une chaine avant l'exécution de l'opération distante. On parle d'une phase de transformation des opérations reçues au niveau des sites distants. Cette transformation est réalisée par un algorithme connu sous l'appellation de transposition en avant ou encore transformation inclusive noté TI **[4]**.

III.2.1. La transformation inclusive (TI):

L'algorithme de la transformation inclusive est une fonction définie pour tout couple d'opérations (O1, O2) telles que ces deux opérations sont nécessairement concurrentes et définies sur un même état initial.

D'une manière générale, cette procédure permet la transformation d'une opération distante suivant les modifications qui ont été apportées juste avant son arrivée (qu'elle n'a pas vue sur con site originel). L'objet partagé est présenté par une suite finie d'éléments de tout type de données. Les fonctions de transformation doivent être spécifiques pour un type de données particulier : chaîne de caractères, document XML, fichiers… **[4]**

La première fonction de transformation a été introduite par Ellis et Al et reprise dans de nombreuses autres publications. Dans la littérature, cette fonction est notée par « T » sur certain document ou par « TI » sur d'autre, elle admet comme paramètres deux opérations concurrentes O1 et O2 et fournit comme retour une nouvelle opération O1' qui est adéquate à O1 mais définie sur le nouvel état du site 2 la où l'opération O2 a été appliquée. L'opération O1' doit garder la même intention que l'opération O1 mais à partir de l'état résultant de l'exécution de l'opération O2. **[4]**

Cet algorithme prend différentes formes, la plus générale qui est principalement développée par Ellis et Al et se présente comme suit **[4]:**

```
IT (Ins (p1, c1), Ins (p2, c2)) :

        if (p1 < p2)   then
                return Ins(p1, c1)
        else
                return Ins(p1+1, c1)
        endif;
```

Figure 6 : Algorithme de transformation Ins-Ins

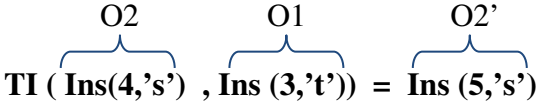
Tels que : Ins1 (p1, c1) et Ins2 (p2, c2) sont deux opérations d'insertion générées respectivement sur les deux sites 1 et 2 et "p" et "c" représentent la position d'insertion et le caractère à insérer dans cette position. Ainsi, le site 1 exécute l'histoire [Ins1, Ins2] qui veut dire l'opération « Ins1 » suivie de « Ins2 » et le site 2 exécute l'histoire [Ins2, Ins1].

La fonction TI se charge de la transformation de « Ins1 » par rapport à l'effet apporté par l'opération « Ins2 » au niveau du site 2.

La position d'insertion de l'opération « Ins1 » sera incrémentée par la valeur de « 1 » si la position d'insertion de l'opération « Ins1 » est supérieure à celle de « Ins2 » : O1' = T (O1, O2) = Ins(p1+1, c1) . Sinon, l'opération « Ins1 » sera conservée telle qu'elle : O1' = T (O1, O2) = Ins(p1, c1).

En résumé, lorsqu'une opération distante arrive sur un site quelconque, celle-ci sera transformée pour inclure les effets des opérations qui la précèdent.

La Figure 5 illustre le même scenario présenté dans la Figure précédente mais cette fois-ci avec l'intégration de la nouvelle fonction de transformation TI entre les deux opérations d'insertions. Contrairement à l'opération O1 qui s'exécute de la même façon sur les deux sites, l'opération O2 est exécutée sur site 1 après avoir incrémente la position d'insertion du caractère « s » à cinq car sa nouvelle position d'insertion est supérieure à la position de l'insertion précédente.



La transformation au niveau de la position de l'opération O2 a aboutie à un résultat identique dans les deux sites, la chaine **Dattes** est obtenue sur le site 1 et le site 2 malgré l'exécution concurrente des opérations. La convergence est donc assurée même si l'ordre d'exécution des deux opérations sur les deux sites est différent.

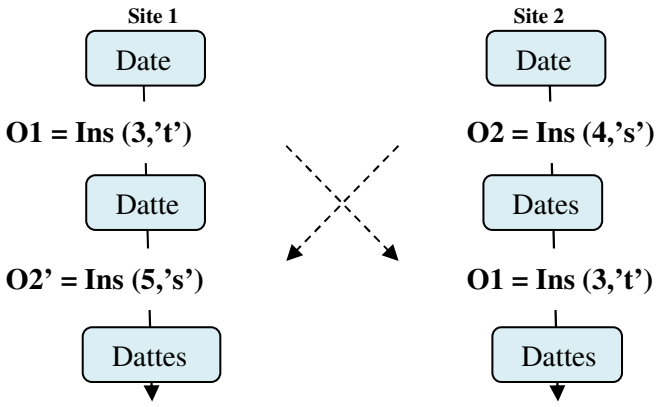


Figure 7: Transformation de deux opérations concurrentes.

Bien évidement, la fonction de transformation est extensible sur le reste des opérations de mises-à-jour telle que la suppression, le raisonnement reste éventuellement le même cependant le calcul des positions subit quelques modifications **[4]**.

```
IT(Ins(p1,c1), Del(p2,c2)) :

        if (p1 <= p2)   then
                return Ins(p1,c1)
        else
                return Ins(p1-1,c1)
        endif;
```

Figure 8 : Algorithme de transformation Ins-Del

```
IT(Del(p2,c2), Ins(p1,c1)) :

        if (p2 >= p1)    then
                return Del(p2+1,c2)
        else
                return Del(p2,c2)
      endif;
```

Figure 9 : Algorithme de transformation Del-Ins

Pour que cet ensemble de fonctions de transformations soit correctement utilisé, il est important de vérifier si les deux opérations concurrentes ont été générées sur le même état initial. Dans le cas contraire, les intentions des utilisateurs peuvent facilement êtres violées et par conséquent la convergence ne sera pas garantie. Ce problème est dit « concurrence partielle ».

III.3. La concurrence partielle :

Dans une édition collaborative, la concurrence partielle peut survenir lorsque deux opérations concurrentes sont définies respectivement sur deux états S1 et S2 tels que S1≠S2. Cette problématique peut occasionner des situations de divergence dues à l'utilisation inappropriée de la transposition en avant. **[3, 4]**

On peut illustrer une concurrence partielle par une situation dans laquelle une opération, dénotée Op3 par exemple, est concurrente à une séquence d'opérations Op1 et Op2. Sur la Figure suivante on remarque que Op1 précède causalement Op2, de plus, Op3 et Op1 sont concurrentes et générées sur le même état « comédy », cependant, l'opération Op2 est partiellement concurrente à Op3 du fait que Op2 et Op3 ne sont pas définies sur le même état. **[3]**.

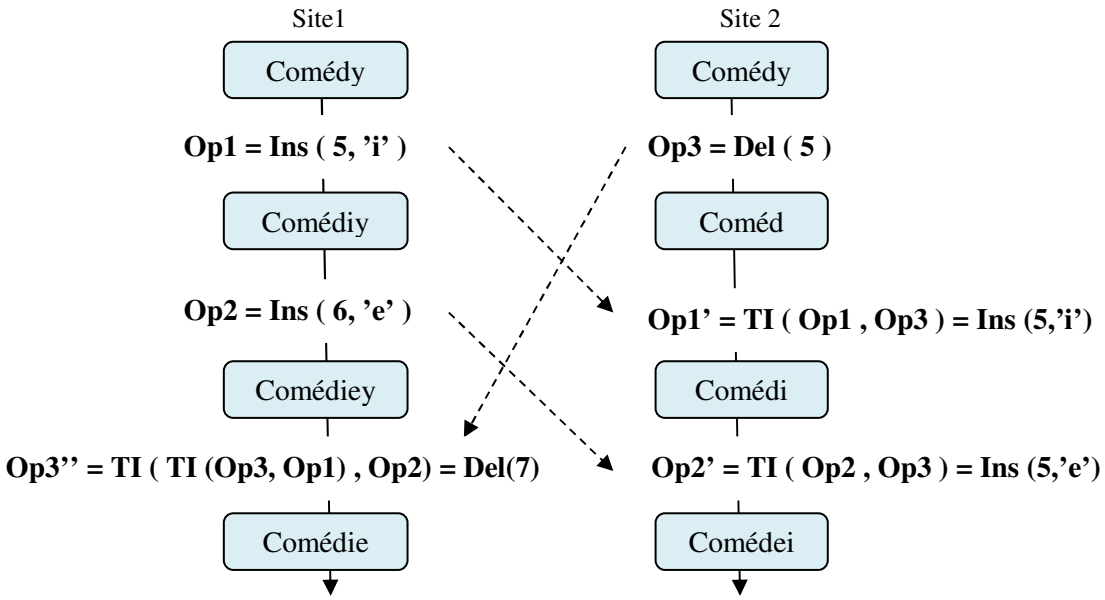


Figure 10 : Transformation incorrecte dans une situation de concurrence partielle.

Dans l'exemple de la Figure 10 on considère que les deux sites tentent de corriger la chaîne partagée « Comédy ». Le site 1 génère deux opérations consécutives qui sont Op1 = Ins(5, 'i') et Op2= Ins(6, 'e'). En parallèle, le site 2 génère l'opération Op3 = Del(5). Les opérations Op2 et Op3 ont été générées respectivement sur les états « Comédiy » et « Comédy ». Ces deux états sont différents et donc Op2 et Op3 sont partiellement concurrents.

Lorsque Op3 arrive sur le site 1, elle est transformée consécutivement par rapport à Op1 et Op2. On obtient donc Op3'' = IT (IT (op3, op1), op2) = Del(7). En exécutant cette opération on obtient la chaîne souhaitée « Comédie ».

Au niveau du site 2, la transformation de Op1 par rapport à Op3 produit op1' = IT (Op1, Op3). Selon la fonction de transformation Ins-Del, lorsque les deux opérations impliquent la même position, l'opération d'insertion sera conservée telle qu'elle est, on obtient dans ce cas Op1'= Ins(5,'i'). Quant à la transformation d'Op2 par rapport à Op3, elle donne Op2'=IT (Op2, Op3) = Ins (5, 'e') dont l'exécution produit la chaîne « Comédei ».

Il est clair que les deux sites 1 et 2 divergent puisque leurs états sont différents. Cette divergence est causée par l'application inappropriée de la fonction IT.

En réalité, IT (Op2, Op3) est une transformation incorrecte selon la définition car Op2 et Op3 sont définies sur deux états différents. Dans ce cas Op2 ne doit pas être directement transformée par rapport à Op3 car Op2 dépends causalement de Op1. D'une autre part, au niveau du site 2, l'intégration de l'opération Op2 doit tenir compte des intentions de Op3 et Op1'= IT (op1, op3).

Pour éviter le problème de la concurrence partielle et permettre l'utilisation correcte de la transformation inclusive, beaucoup de solutions ont été proposées. Ces solutions dépendent généralement de l'algorithme d'intégration utilisé par le système. **[3, 4]**.

Dans le SOCT2 par exemple, une autre fonction de transformation est employée pour changer l'ordre de chaque couple d'opérations exécutées. Cette nouvelle fonction est appelée « transposition en arrière » et a été également reprise sous l'appellation de « transpose » dans l'algorithme GOTO assure l'utilisation correcte de la fonction TI sans que les intentions soient violées. **[18, 19]**

Dans le GOT algorithme, une fonction de transformation inverse à la TI appelée Transformation Exclusive permet la transformation des opérations partiellement concurrentes (Op2 et Op3 dans notre exemple) de façon à ce qu'elles soient définies sur le même état et qu'on puisse appliquer ensuite la transposition en avant. **[19]**

Les procédures utilisées dans l'algorithme ADOPTED sont conçues pour conserver les opérations reçues telles qu'elles ont été générées sur leurs sites d'origines. Les résultats des transformations précédentes ainsi que les transformations intermédiaires sont à leurs tours sauvegardés. Cela rend possible l'obtention d'un même état sur lequel les deux opérations qui sont en concurrence partielle seront appliquées. **[17, 18, 19]**

Si on applique cette solution sur l'exemple précédent, on doit d'abord transformer Op3 par rapport à Op1 pour obtenir une nouvelle opération définie sur le même état que celui d'Op2 et par rapport à laquelle on peut transposer Op2 en avant.

L'intégration correcte de l'opération Op2 est illustrée dans la Figure 11.

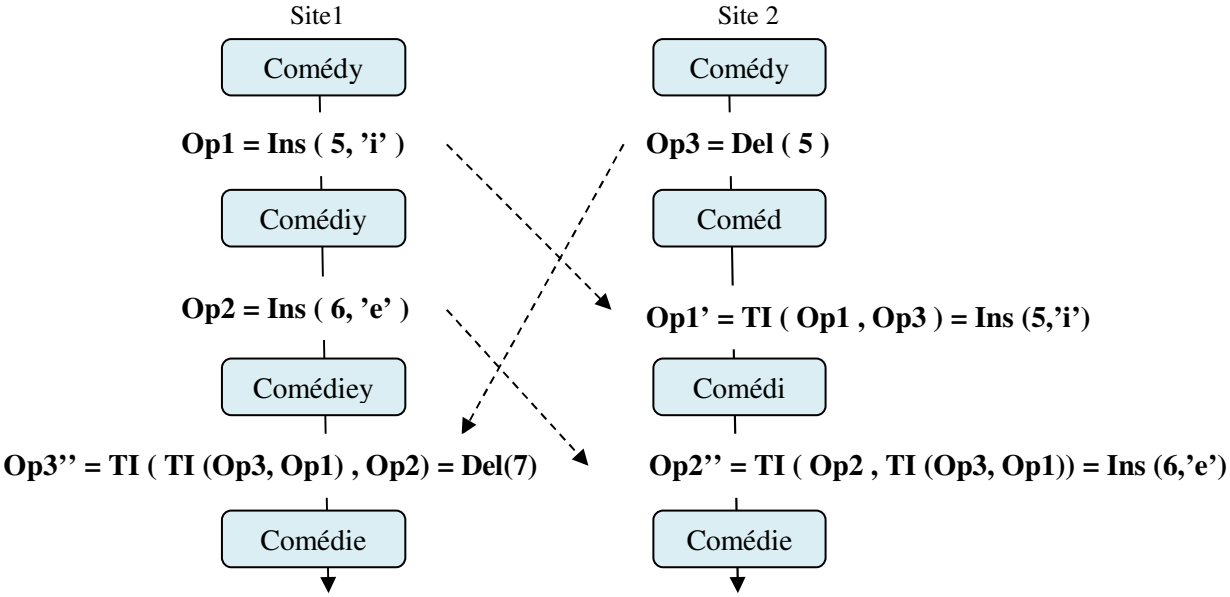


Figure 11 : Transformation correcte dans une situation de concurrence partielle.

Le scénario se déroule comme suit :

- Quand Op3 arrive sur le site1, elle est d'abord transformée par rapport à Op1. L'opération obtenue, disons Op3', est transformée et sauvegardée sous la forme suivante:

$$\underbrace{TI}_{Op3}\ (\ Del\ (\ 5\ )\ ,\ \underbrace{Ins\ (\ 5,\ 'i'\ )}_{Op1}\ )\ =\ \underbrace{Del\ (6)}_{Op3'}$$

- L'opération résultante Op3' est ensuite transformée par rapport à Op2 au niveau du même site :

$$\underbrace{TI}_{Op3'}\ (\ Del\ (\ 6\ )\ ,\ \underbrace{Ins\ (\ 6,\ 'e'\ )}_{Op2}\ )\ =\ \underbrace{Del\ (7)}_{Op3''}$$

- Ensuite, le site2 intègre l'opération distante Op1. Celle-ci sera transformée par rapport à Op3 de la manière suivante :

$$\underbrace{TI}_{Op1}\ (\ Ins\ (\ 5,\ 'i'\ ),\ \underbrace{Del\ (\ 5\ )}_{Op3}\ )\ =\ \underbrace{Ins\ (\ 5,\ 'i'\ )}_{Op1'}$$

- Finalement, le site2 intègre Op2. Il faut d'abord transformer Op2 par rapport à Op3'= (Op3, Op1) et qui est déjà sauvegardée pour éliminer l'effet de l'opération Op1 sur le site2. On obtient une opération Op2'' définie sur le même état que celui de Op2.

$$\underbrace{TI}_{Op2}\ (\ Ins\ (6,'e'),\ \underbrace{Del\ (6)}_{Op3'}\ )\ =\ \underbrace{Ins\ (6,'e')}_{Op2''}$$

A partir de ces transformations on peut déduire que l'exécution des opérations obtenues en tenant compte des l'intentions des utilisateurs permet de conduire au bon résultat. Cependant cette prise en compte du critère de la causalité ainsi que celui de la préservation de l'intention n'est parfois pas suffisante pour assurer que tous les sites convergent vers la même valeur. Il arrive souvent que deux utilisateurs tentent d'insérer par exemple en même temps et à la même position un caractère quelconque. **[4]**

Considérons les deux opérations concurrentes Op1 et Op2, générées et exécutées sur le même état après avoir été transposées correctement sur les deux sites.

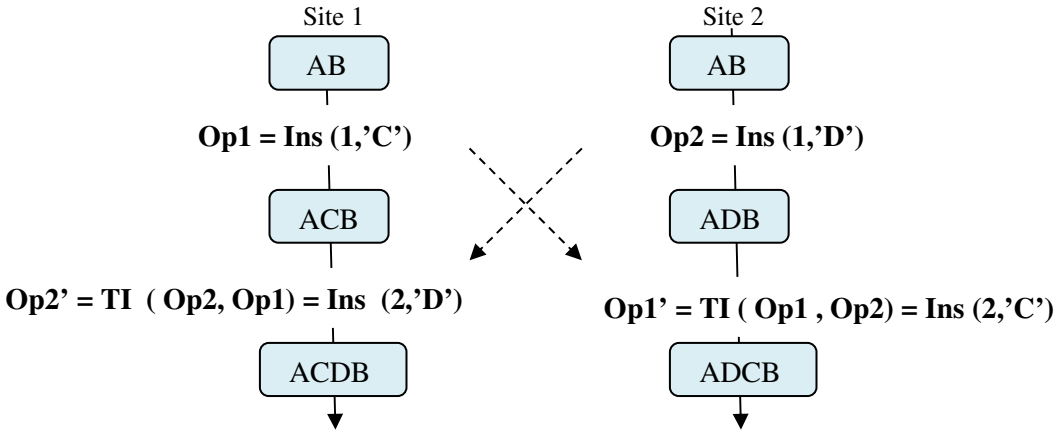


Figure 12: Situation de divergence des répliques.

Dans cet exemple, les deux utilisateurs exécutent leurs opérations sur le même état initial au même moment. Ensuite, chacune des ces opérations est envoyée vers le 2ème site pour garantir la mise-à-jour de cette réplique. On applique automatiquement la fonction de transformation inclusive afin d'éviter tout conflit. Dans ce cas, chaque réplique doit faire face à une séquence d'opération avant que le système se met au repos. Cette séquence est dénotée comme suit : O1 ○ O2 … ○ On où le symbole « ○ » représente la concaténation entre 2 opérations successives **[6]**.

Comme le montre la Figure 12, la séquence d'opérations op1○ TI(op2, op1) se termine par l'état "**ACDB**" tandis que l'exécution de la séquence op2 ○ TI(op1, op2) donne l'état "**ADCB**". La situation de divergence se présente clairement même si la préservation de l'intention et l'utilisation de la TI ont été établies convenablement **[6]**.

III.4. Les propriétés de la transformation :

Pour assurer la convergence des répliques, la fonction de transformation inclusive (TI) doit satisfaire deux propriétés TP1 et TP2 :

III.4.1. La propriété TP1 :

Cette propriété exige que, pour deux opérations O1 et O2 définies sur un même état initial, l'exécution de la séquence d'opérations O1 ○ T(O2, O1) produit le même état final que celui après l'exécution de O2 ○T(O1, O2).

Donc, quelque soit l'ordre d'exécution des deux opérations concurrentes, le résultat reste le même ce qui implique l'équivalence d'état après avoir exécuté chacune des séquences à partir du même état. Cette propriété est formulée de la manière suivante **[20]**:

$$O1 \circ T(O2, O1) \equiv O2 \circ T(O1, O2)$$

Dans l'exemple précédent (Figure 12), les deux états finaux **"ACDB"** et **"ADCB"** sont divergent, mais il parait difficile de préciser lequel d'entre eux est plus correcte par rapport à l'autre. Dans ce cas, une certaine logique devra-t-être poursuite puisque les insertions peuvent se faire mutuellement et que nul ne peut déterminer si l'un des états est meilleur que l'autre.

Etant donné que rien ne peut permettre de décider lequel des deux caractères doit être placé devant l'autre, il faut donc prendre un choix arbitraire. Aussi, il faut garantir que ce choix sera appliqué de la même manière sur toutes les répliques.

Les solutions proposées dans ce contexte visent soit à établir un ordre de priorité sur les opérations ou sur les identifiants des sites, soit à imposer un ordre par rapport aux lettres alphabétiques.

La fonction de transformation TI satisfait parfaitement la condition TP1 sauf dans le cas d'une insertion à la même position, Ressel et al on montré qu'il est possible d'intégrer cette condition en ajoutant un ordre de priorité sur les identifiants des sites afin de faire converger l'ensemble des répliques **[4]**:

```
IT (Ins (p1, c1, id1), Ins (p2, c2, id2)):
        if ((p1<p2) || (p1 = p2 et id1<id2)) then
                return Ins (p1,c1)
        else
                return Ins (p1+1,c1)
        endif;
```

Figure 13 : Algorithme de transformation qui assure la propriété TP1

Ainsi, pour deux insertions à la même position, le caractère correspondant à l'identifiant le plus petit sera inséré avant celui de l'autre opération au niveau de la chaine de caractère. La fonction de transformation prend un troisième paramètre indiquant cette priorité d'insertion.

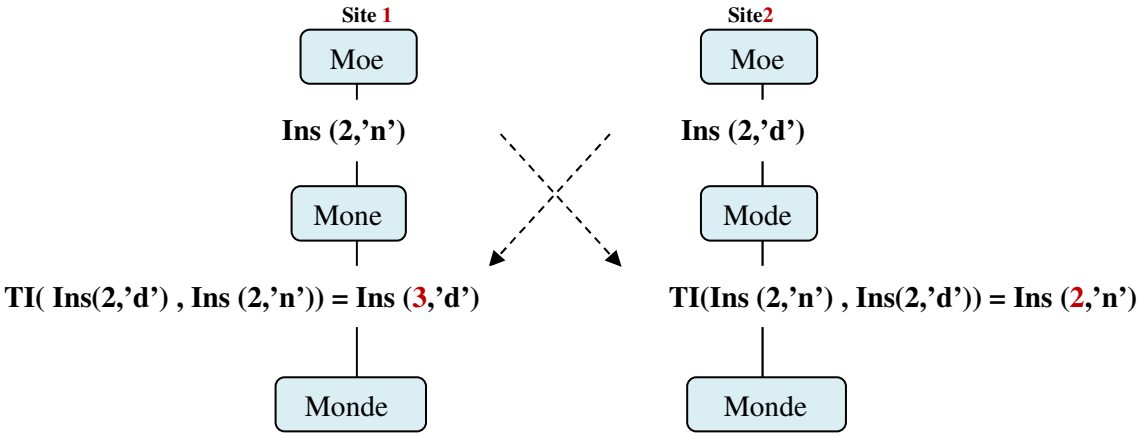


Figure 14: Convergence de copies avec la satisfaction de la propriété TP1

Dans l'exemple ci-dessus, deux opérations d'insertion sont effectuées au même moment et sur le même état initial. Le site 1 génère l'opération **Ins (2,'n')** pour obtenir le mot **« Mone »,** en parallèle le site 2 génère l'opération **Ins (2,'d')** pour obtenir **« Mode ».**

Les améliorations apportées par Ressel sur l'algorithme de transformation permettent aux deux répliques de converger vers le même état final. En appliquant cette nouvelle fonction de transformation sur l'exemple de la Figure 9, la position d'insertion du caractère « **d** » qui est générée par le site ayant l'indice 2 sera incrémentée par 1 au niveau du site ayant l'indice 1.

Il a été démontré, dans l'ensemble des travaux publiés, que la propriété TP1 est nécessaire pour assurer la convergence. Seulement, quand plus de deux opérations concurrentes sont présentes, cette propriété n'est pas suffisante pour garantir une cohérence. La Figure suivante décrit un scénario possible pour créer une situation de divergence. Dans ce scénario, la fonction de transformation satisfaisant la propriété TP1 présentée précédemment est utilisée.

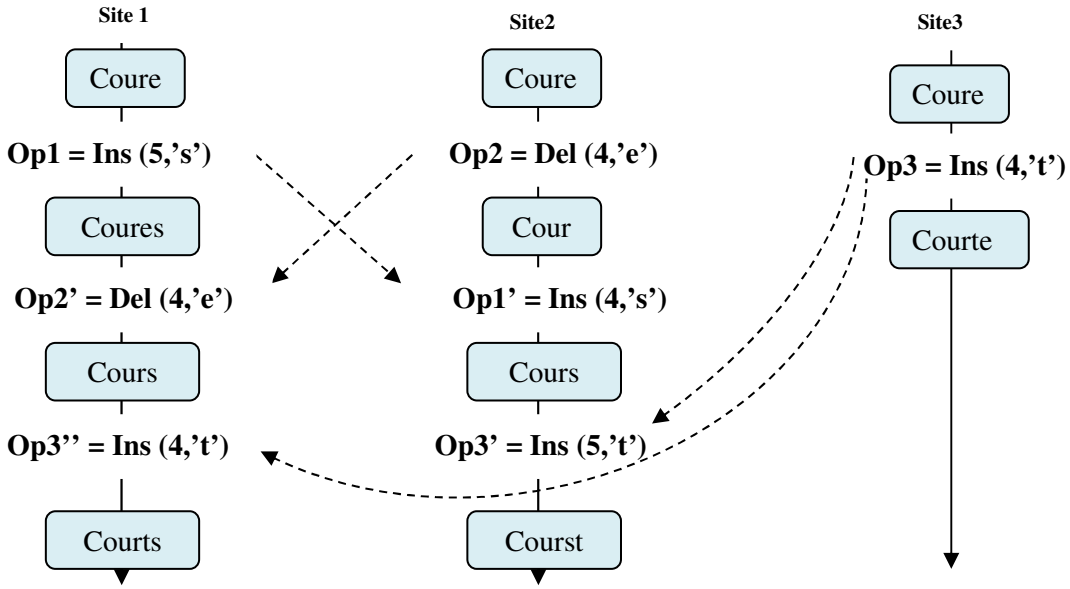


Figure 15: Divergence de copies pour trois opérations concurrentes.

Dans cet exemple, les trois utilisateurs essayent d'exécuter concurremment les trois opérations Op1, Op2 et Op3. Lorsqu'on applique l'algorithme de transformation Ins-Del et Del-Ins sur les deux opérations Op1 et Op2 on obtient le même état « Cours ». L'opération Op3 permet d'insérer en parallèle le caractère t à la position 4, cette opération est ensuite envoyée vers les autres répliques. Sur le site 1, l'opération Op3 doit être transformée par rapport à Op2' : **Op3'' = Ins (4,'t')** ce qui donne l'état « Courts » en appliquant l'algorithme Ins-Del. Cependant, sur le site 2 l'opération Op3 doit prendre en considération l'effet de Op1' qui est une opération d'insertion, dans ce cas la position d'insertion sera incrémentées selon l'algorithme Ins-Ins : **Op3' = Ins (5,'t').** Puisqu'on applique deux opérations différentes sur le même état on ne peut pas s'attendre à un résultat identique.

Une deuxième propriété TP2 doit être respectée pour couvrir l'insuffisance de TP1.

III.4.2. La propriété TP2 :

Pour chaque trois opérations simultanées O1, O2 et O3 définis sur le même état, la transformation de O3 par rapport à la séquence composée de O2 suivie de T (O1, O2) doit donner le même état que celui de la transformation de O3 par rapport à la séquence O1 suivie par T (O2, O1). Cette propriété est formulée comme suit **[6]** :

$$T(O3, O1 \circ T(O2, O1)) = T(O3, O2 \circ T(O1, O2))$$

La fonction de transformation TI citée précédemment doit garantir que quelque soit la séquence d'opérations à exécuter, le résultat de cette transformation ne doit pas dépondre de l'ordre selon lequel ces opérations ont été transformées **[20]**. Ceci revient à prouver une égalité entre des séquences d'opérations obtenues après la transformation d'un ensemble d'opérations dans différents ordres.

Ressel et al ont démontré que ces deux propriétés TP1 et TP2 sont suffisantes pour que la fonction de transformation assure la convergence des copies indépendamment de l'ordre dans lequel les opérations simultanées sont transformées et exécutées **[4]**. Cependant, pour une édition collaborative, le nombre de paires étant variable, les algorithmes d'intégration développés dans l'approche TO doivent permettre de passer facilement à l'échelle. Jusqu'à l'heure actuelle, plusieurs algorithmes d'intégration ont été proposés mais aucun d'eux ne satisfait parfaitement la propriété TP2 **[17]**.

III.5. Les algorithmes d'intégration :

Comme nous l'avons vu dans sa définition, la procédure d'intégration doit être capable d'assurer l'envoi et la réception des différentes opérations échangés entre les sites du réseau ainsi que leur exécution. L'intégration d'une opération distante « Op » générée sur un site « 1 » à partir d'un état courant « S » a pour but de permettre son exécution sur le site distant « S' ». Ceci consiste à calculer sa forme transformée « Op ' » qui sera exécutée sur l'état « S' » du site « 2 ». L'opération résultante « Op' » doit réaliser la même intention que celle de « Op » générée sur un site distant « 1 ». Dans ce cas, l'opération reçue « Op » sera transformée par rapport à toutes ses opérations concurrentes exécutées sur le site « 2 » avant son arrivée. **[2, 4]**

Pour que la transformation soit faite convenablement, plusieurs algorithmes d'intégration ont été développés comme ADOPTED, GOTO, SOCT2, SOCT4... Le choix de ces algorithmes dépond dans la plus part du temps du type du réseau sur lequel cet algorithme peut être exploitable ainsi que la structure de données utilisée **[21]**.

Dans une intégration centralisée, des algorithmes tels que SOCT4 et COT permettent la gestion des opérations concurrentes par l'intermédiaire d'un site central en exigeant que l'exécution de ces opérations soit faite dans le même ordre sur tous les sites **[21]**.

Pour une intégration décentralisée, les procédures fréquemment utilisées sont ADOPTED, SOCT2, GOT. Ces procédures ne nécessitent aucun nœud central et peuvent également supporter un nombre important d'opérations sans imposer un ordre total sur leur exécution. Dans ce rapport on s'intéresse plus à ce type d'intégration car il est plus approprié aux systèmes P2P **[19, 22]**.

Le SOCT2 est un algorithme dédié aux systèmes collaboratifs P2P qui se limite aux structures linéaires telles que les documents textuels. Cet algorithme permet d'assurer au

maximum le modèle CCI grâce à l'utilisation des vecteurs d'états qui garantissent le critère de la causalité et en utilisant des transpositions en avant et en arrière qui permettent de préserver les intentions. **[18]**

L'algorithme ADOPTED, qui a été développé principalement par Matthias Ressel, représente une extension de DOPT-algorithme d'Ellis et Gibbs. L'idée de base est de prendre une opération déjà exécutée dans un état passé et de la transformer de façon à ce qu'elle peut être appliquée sur l'état actuel. Le respect de la causalité dans cet algorithme est garanti de la même façon que dans l'algorithme précédant c'est-à-dire en utilisant les vecteurs d'états, seulement, pour la préservation de l'intention une seule fonction de transformation est utilisée pour permettre à l'ensemble de sites de converger vers la même valeur. **[17, 18, 19]**

III.6. Conclusion :

Dans ce chapitre, nous avons donné une vue générale sur l'approche des transformées opérationnelles qui permet de garantir la convergence des répliques et la préservation des intentions pour des données structurées linéaires sans nécessité de site central ou d'ordre global ni de vecteurs d'horloges.

Pour garantir la convergence des répliques, l'algorithme de transformation utilisé dans cette approche doit satisfaire les deux conditions TP1 et TP2. Etant donné que le nombre de cas à considérer pour garantir ces propriétés est infini et que les vérifications des incohérences sont souvent très difficiles (voire même impossibles) à effectuer à la main, le développement d'un algorithme de transformation devient une tâche fastidieuse. D'autre part, cette difficulté varie selon la nature de l'objet collaboratif partagé. Ainsi, la transformation pour des objets ayant une structure linéaire demeure toujours un problème ouvert.

IV. Chapitre III : L'algorithme d'intégration ADOPTED

IV.1. Introduction :

L'approche de la transformée opérationnelle (OT) a été proposé pour résoudre la divergence dans les environnements de réplication optimiste. Dans ce cadre, chaque opération doit s'exécutée localement est sera ensuite diffusée aux autres sites pour qu'elle soit intégrée.

Dans le chapitre précédent, nous avons vu que les algorithmes d'intégration dépondent du type du réseau sur lequel la collaboration s'effectue. Le grand défi pour ces algorithmes est de détecter la concurrence des opérations.

Dans ce chapitre, nous avons choisi de présenter l'algorithme d'intégration décentralisé appelé « ADOPTED » qui semble le plus approprié pour notre étude. Pour cela nous allons analyser les différentes structures sur lesquelles il est basé pour gérer les incohérences causées par les opérations concurrentes.

IV.2. Présentation de l'algorithme ADOPTED :

Les opérations traitées par ADOPTED sont, comme pour toutes les autres procédures d'intégration, classées soit comme étant locales ou bien distantes. Pour chaque opération locale, l'ensemble des procédures doit assurer l'exécution immédiate de cette opération sur le site la où elle a été générée, ceci permet de réduire le temps de réponse aux requêtes de l'utilisateur, de plus, cette exécution sera impérativement suivie de la diffusion de cette même opération accompagnée par son vecteur d'état vers tous les sites présents sur le réseau.
Pour les opérations distantes, celles-ci doivent être transformées (s'il est nécessaire) à partir de l'état courant du site et par rapport aux opérations qui leurs sont concurrentes sur le Log à l'aide d'une fonction de transformation avant qu'elles soient exécutées. **[19]**

La fonction de transformation utilisée est notée « Translate Request » dans ADOPTED, elle permet de déterminer quelles opérations devrait-on transformer et par rapport à quelles séquence d'opérations en effectuant un appel récursif, ensuite la fonction devrait appliquer une transposition des opérations sélectionnées à l'aide d'une autre procédure nommée L-transformation dans ADOPTED qui est similaire à la fonction de transformation inclusive TI vue précédemment. **[19]**

Cet algorithme représente une extension de DOPT-algorithme en ajoutant un graphe d'interaction multidimensionnel (appelé aussi N-dimensionnel où N est le nombre de sites de coopération) qui maintient tous les chemins valides des transformations de toutes les opérations exécutées au niveau d'un site. **[23]**

IV.2.1. Le modèle d'interaction (graphe d'exécution) :

Ce graphe est composé d'un ensemble de sommets E qui représentent les différents états par lesquels le système (l'utilisateur) est passé ainsi qu'un ensemble d'arcs A qui représentent les opérations dont l'exécution a provoquée la transition d'un état vers un autre. La valeur du

vecteur d'état correspond à une coordonnée dans un repère orthonormé. Chacune des dimensions de ce repère est associée à un utilisateur et toutes les opérations qu'il a générées sont représentées par des vecteurs sur cette dimension. **[17, 23]**

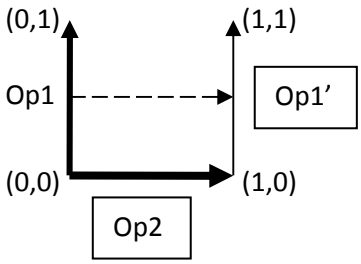


Figure 16 : Graphe d'exécution du site2

Chaque site possédant une copie du document partagé doit gérer un graphe représentant l'histoire multidimensionnelle des opérations locales et distantes exécutées sur le site. Par exemple, sur la Figure 16 les deux utilisateurs ont générés en concurrence chacun une opération : « Op1 » sur le site « 1 » et « Op2 » sur le site « 2 ». Les opérations concurrentes générées sur le même état initial (0,0) sont représentées en gras, la transformation de l'opération distante « Op1 » est représentée en pointillés et les opérations réellement exécutées sur le site2 sont encadrées. Lorsque « Op1 » atteint le site2, elle doit être transformée par rapport à « Op2 » sur l'état courant du site2 qui est (1,0). On obtient une nouvelle opération « Op1' » qui doit réaliser la même intention que celle de « Op1 » exécutée sur le site1. **[17]**

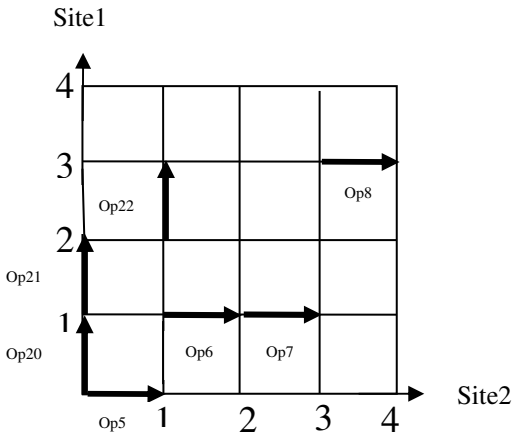


Figure 17 : Modèle d'interaction décrivant un ensemble des requêtes générées par deux utilisateurs

La Figure 17 montre un ensemble de requêtes pour les deux sites « 1 » et « 2 », les quatre requêtes (Op5, Op6, Op7, Op8) sont générées par le site2 et les trois requêtes (Op20, Op21,

Op22) sont générées par le site1. L'origine de chaque flèche indique le vecteur d'état correspondant à la requête représentée par cette flèche. Encore, chaque ligne et chaque colonne peut contenir au plus une seule flèche car au niveau de chaque site les opérations locales sont exécutées séquentiellement (dans un ordre causal).

Les opérations sont stockées sous leurs différentes formes possibles : l'opération originale, l'opération intermédiaire et l'opération exécutée. Donc puisque on conserve les opérations sous leur forme originale on peut toujours retrouver l'état (le sommet) sur lequel est définie une opération. De plus, puisque on conserve les opérations exécutées on peut toujours trouver, pour une opération reçus, la séquence d'opérations concurrentes par rapport à laquelle elle doit être transposée. Sur le graphe cette séquence est déterminée par le chemin qui conduit de l'état à partir duquel l'opération a été définie jusqu'à l'état actuel du site (Figure 18).

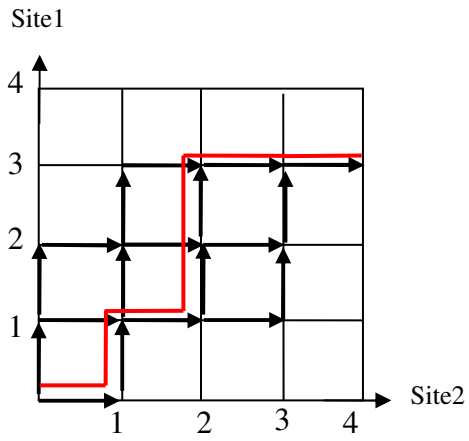


Figure 18 Modèle d'interaction décrivant un chemin d'intégration possible pour le site2

Le graphe d'interaction fournit un modèle très utile pour visualiser les relations de transformation entre les opérations initiales et les opérations transformées. Il serait donc plus facile de déterminer l'opération qui sera exécutée sur l'état courant du document. **[17, 23]**

IV.3. Fonctionnement de l'algorithme ADOPTED :

La fonction principale (main) de cet algorithme permet la gestion des deux types d'opérations : locales et distantes. Le système reste toujours en écoute afin de contrôler les opérations qui circulent sur le réseau. Pour les opérations locales, une fonction notée «Generate Request» est appelée avec comme paramètre l'opération que l'utilisateur vient de générer sur son site. Quant aux opérations distantes, elles sont traitées par une autre fonction dite «Receive Request» (Figure 19).

```
Main:
      Initialize
      while not aborted
        if there is an input o
              Generate Request (o) ;          // locally
              Execute Request
        else
              Receive Request ;               // from remote
              Execute Request
```

Figure 19 : Fonction principale d'ADOPTED

Chaque processus utilisateur gère un ensemble de données permettant la gestion des différentes opérations: l'état du site « s » (1.1), un compteur « k » pour les requêtes générées localement (1.6), le vecteur d'état du site noté « v » (1.5), une file d'attente qui contient l'ensemble des requêtes « Q » (1.3), une autre file de requêtes « L » (1.2) et le modèle d'interaction N-dimensionnel « G » (1.4). La procédure principale de l'algorithme initialise toutes les structures de données avant d'entamer la phase de l'exécution (Figure 20).

```
1. Initialize:

    1.1.   s ← s0 ; the initial application state
    1.2.   L ← empty set
    1.3.   Q ← empty set
    1.4.   G ← initial interaction model
    1.5.   v ← (0, 0, . . . , 0) ; initial state vector
    1.6.   k ← 1 ; initial request id
    1.7.   u ← identification of local participant
```

Figure 20 : Fonction d'initialisation d'ADOPTED

Le vecteur d'état « v » contient le nombre d'exécutions pour chaque utilisateur. La file de requêtes « Q » est utilisée pour stocker les requêtes générées et reçues qui doivent attendre leur exécution. Le modèle d'interaction G est principalement utilisé pour stocker des requêtes susceptibles d'être transformées plus tard. Le journal des requêtes (histoire ou Log) « L » conserve une copie de chaque requête initiale.

Les requêtes des utilisateurs notées « r » seront définies par le quadruplet (u, k, v, o) et seront accessibles en utilisant l'identifiant du site « u » sur lequel elles ont été générées et leur numéro de série « k » (Figure 21).

2. Generate Request (o):

2.1. r ← (u, k, v, o)
2.2. Q ← Q + r
2.3. L ← L + r
2.4. k ← k + 1
2.5. broadcast 'r' to other participants

Figure 21 : Procédure de génération des requêtes

Dès qu'une opération est générée le système doit l'ajouter à la file «Q» contenant l'ensemble des requêtes locales et distantes qui sont en attente (2.2) et à la file « L » pour conserver une copie de l'opération originale dans l'historique (2.3). Cette requête sera ensuite envoyée immédiatement vers l'ensemble des sites présents sur le réseau (2.5).

La fonction « Receive Request » se charge de sélectionner les requêtes distantes à partir du réseau. Lors de la réception d'une opération distante, celle-ci sera rajoutée à son tour aux deux files « Q » et « L » sous sa forme initiale (Figure 22).

3. Receive Request:

3.1. if there is a request from network
3.2. choose request r
3.3. Q ← Q + r
3.4. L ← L + r

Figure 22 : Procédure de réception des requêtes.

Pour exécuter une requête, le système effectue le parcours de la file d'attente des requêtes « Q » pour sélectionner les requêtes exécutables. Une requête ne peut être exécutée que si toutes les requêtes qui lui sont précédentes l'ont été. Il a été prouvé par Ellis et Gibbs que cette précédence causale peut être maintenue si le vecteur d'état de la requête « v(r) » est inférieur ou égal au vecteur d'état actuel « v » (Figure 23).

4. Executable? (r, v):

4.1. logical value of $v(r) \leq v$

Figure 23 : Procédure de sélection des requêtes exécutables

La fonction « Execute Request » (Figure 24) doit donner la priorité aux requêtes locales. Si plus d'une requête est exécutable, les requêtes locales sont préférées dans l'étape (5.2). Cela garantit qu'aucune requête locale ne reste en attente d'exécution lorsque l'utilisateur traite d'autres opérations.

5. Execute Request:

5.1. if there is r in Q satisfying Executable? (r, v), then
5.2. choose executable request r = (j, k_j , v_j , o_j)
5.3. Q ← Q – r
5.4. r″ ←Translate Request (r, v)
5.5. apply operation o(r″) as user j to state s
5.6. increment j^{th} component of v.

Figure 24: procédure d'exécution des requêtes

La requête exécutable choisie sera ensuite supprimée de la file d'attente (5.3) et remise à la fonction « Translate Request » (5.4) avant de l'appliquer sur l'état courant du site « s » (5.5). Cette exécution implique la modification du vecteur d'état du site car la composante qui correspond à l'utilisateur « j » (nombre d'opérations exécutées pour l'utilisateur « j ») devra être incrémentée (5.6).

La fonction de transformation doit assurer l'intégration d'une opération « o » en fournissant une nouvelle opération « o' » définie sur l'état courant du site « s » et dont l'exécution réalise la même intention que celle de « o » lorsqu'elle a été exécutée à partir de l'état sur lequel elle est définie.

Dans l'algorithme ADOPTED la fonction « Translate Request » associe à une opération « o » le vecteur d'état correspondant à l'état sur lequel cette opération a été définie. Cette fonction prend donc comme paramètres la requête « r » et le vecteur d'état courant du site « s ».

Dans le cas de l'exemple illustré dans la Figure 16, lorsque l'opération op1 est reçue, l'appel de « Translate Request » avec comme paramètres l'opération « op1 » et le vecteur d'état (1,0) délivre l'opération « op1' »: Translate Request (op1, (1,0)) = op1' qui est exécutée sur l'objet à partir de l'état (1,0).
La requête « r » sera transformée à partir de son vecteur d'état « v(r) » vers un état spécifié par le vecteur d'état actuel « v » de manière récursive (Figure 25)

```
6.  Translate Request (r, v)

    6.1.        (j, k_j , v_j , o_j) ← r

    6.2.        if  v_j = v then
    6.3.              add request r to model G
    6.4.              return r

    6.5.        else   if G contains r′ for r translated to v
    6.6.              return r′

    6.7.        else
    6.8.            let i be such that
    6.9.                  Reachable? (Decr(v, i)) and
    6.10.                 v_j [i] <= v[i] – 1
    6.11.             v′  ←Decr(v, i)
    6.12.             r_i  ←Request (i, v[i])
    6.13.             r_i' ←Translate Request (r_i, v′)
    6.14.             r′  ←Translate Request (r, v′)
    6.15.             (r″, r_i'' ) ← TF (r′, r_i')
    6.16.             add requests r″ , r_i''  to model G
    6.17.             return r″
```

Figure 25 : Procédure de transposition des requêtes

```
7.  Reachable? (v)
    7.1.    for every i in U:
    7.2.          v[i]=0
    7.3.          or w(Request (i, v[i])) ≤ v
              Avec w(r) = v(r) +1


8.  Decr (v, i):
    8.1.        return copy of v with i^th component decremented


9.  Request (i, j)
    9.1.        return j^th request from user i in L
```

Figure 26: fonctions invoquées par « Translate Request »

Si les deux états sont égaux la requête « r » est retournée telle qu'elle est (6.2 – 6.4). Sinon un prédécesseur immédiat « v' » de « v » dans le modèle d'interaction est déterminée (6.7 - 6.11), de sorte que la requête « r » peut être transformée en « v' ».

La condition (6.9) fait en sorte que « v' » appartient au modèle d'interaction et la condition (6.10) vérifie si « v' » est un successeur de « v_j » (« v' » peut être atteint à partir de «v_j »). Si plus d'un tel prédécesseur existe, il est sûr de choisir l'un, car il a été prouvé dans **[17]** que le chemin le plus long à partir duquel la requête « r » sera traduite n'a pas d'influence sur le résultat.

Si le vecteur « v' » est le prédécesseur de « v » sur l'axe « i » on peut dire que la requête « r_i » sera la v[i]^{ème} requête de l'utilisateur « i » (6.12). La fonction Traduire La requête est appelée récursivement pour traduire les requêtes « r » et « r_i » sur le vecteur d'état « v' » (Figure 27).

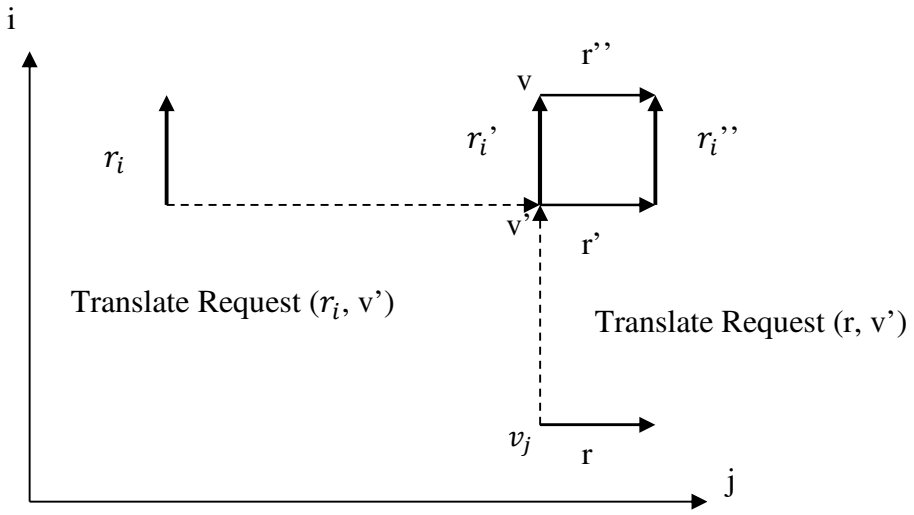


Figure 27: fonctionnement récursif de la fonction « Translate Request »

Les deux requêtes « r ' » et « r_i'» obtenues sont ensuite transformées en utilisant la fonction de transformation FT (6.15). Cette fonction permet la transposition en avant du couple de requêtes « r ' » et « r_i'» et délivre un nouveau couple de requêtes « r'' » et « r_i''» telles que : r '' = Transposition-en-avant (r ', r_i') et r_i'' = Transposition-en-avant (r_i', r ').

Le couple de requêtes « r'' » et « r_i'' » sont ensuit stockées dans le modèle G. Avant de transformer une requête on vérifie d'abord si celle-ci existe déjà dans ce modèle interaction (6.5). De cette façon, la transformée de chaque requête sera calculé au plus une seul fois.

La requête « r'' » sera renvoyée par la fonction « Translate Request » et appliquée finalement sur l'état actuel du site.

IV.4. Limites de l'algorithme ADOPTED :

L'algorithme ADOPTED est soumis aux mêmes contraintes que tous les autres algorithmes d'intégration, il impose donc la vérification des deux propriétés TP1 et TP2. Cependant la deuxième propriété présente jusqu'à présent un grand défi pour les développeurs et toutes les fonctions de transformation déjà publiés ne la font pas satisfaire car elle est très difficile à assurer **[8, 17].**

Dans sa thèse sur la cohérence des données dans les environnements collaboratifs répartis, Gérald Oster **[3]** a essayé de prouver, en fournissant des contres exemples, l'inefficacité de chacune de ces solutions : Proposition d'Ellis et al, Proposition de Ressel et al, Proposition de Suleiman et al, Proposition de Imine et al (2003) **[24]**, Proposition de Imine et al (2004). Ce constat a été renforcé par l'étude de Boucheneb et Imine (2009) **[25]** et par celle d'Aurel Josias Oboubé Randolph (2014) **[2]**. En effet à chaque nouvelle tentative, soit on retrouve une violation de la propriété TP2 (Figure 10), soit on tombe sur une situation de concurrence partielle (Figure 7). **[2]**

L'histoire des opérations (appelée aussi journal ou Log) utilisée par ADOPTED permet de conserver la trace de toutes les opérations déjà exécutées au niveau d'un site. Cette structure permet de garder les informations utiles pour garantir la transformation correcte des opérations. Cependant, la transposition en avant d'une opération par rapport aux opérations du site distant et qui lui sont concurrentes n'est pas triviale en considérant que la collaboration est réalisée sur un réseau P2P et que le nombre d'utilisateurs et d'opérations est très élevé, les états « S » et « S' » sont dans la plus part du temps différents **[4]**. Il peut arriver, dans des situations de concurrence partielle par exemple, que des opérations concurrentes à une opération « Op » soient confondues avec des opérations qui précèdent causalement « Op ».

En 2006, Abdessamad Imine a proposé un nouvel algorithme de transformation afin de garantir la convergence pour les objets linéaires dans des situations de conflit similaires à celle décrite sur le scénario de la Figure10 c'est-à-dire lors de la présence des opérations d'insertion et de suppression en même temps. Cet algorithme est basé sur la trace des positions stockées durant le processus de transformation. Par ailleurs, cette solution exige le ré-ordonnancement de l'histoire des opérations si ces dernières ne sont pas identiques pour obtenir une séquence d'opérations d'insertions avant celle des suppressions. Malheureusement, l'obtention d'un ordre entre l'insertion et l'effacement et d'un coût très élevé et nécessite la conception de tout un nouvel algorithme d'intégration. **[4, 8]**

Les autres contraintes imposées par l'algorithme ADOPTED telles que le maintient de l'histoire des opérations, le graphe multidimensionnel, le vecteur d'état renvoyé à chaque interaction et la recherche d'un chemin de transformation pour chaque requête posent des problèmes de gestion et d'occupation mémoire, de bande passante ainsi que des difficultés pour passer à l'échelle lorsque le nombre de collaborateurs augmente. **[8]**

A vrai dire, lorsqu'il s'agit de traiter une forte activité d'un grand nombre d'utilisateurs l'algorithme ne permet pas de remplir toutes les transformations possibles car plus les requêtes distantes attendent leur traitement longtemps, plus les requêtes concurrentes vont s'accumuler sur le Log et plus ça va prendre du temps pour les transformées. **[17]**

IV.5. Conclusion :

Dans ce chapitre, nous avons essayé de présenter l'algorithme ADOPTED qui est utilisé dans le cadre d'une collaboration décentralisée, d'expliquer son fonctionnement et de décrire les structures qu'il utilise pour garantir l'intégration des différentes opérations du réseau.

Nous avons vu que l'algorithme ADOPTED est toujours considéré comme incompatible avec les caractéristiques des réseaux P2P en raison de son incapacité de passer à l'échelle. D'une autre part, les procédures de cet algorithme ne prennent pas en considération le cas de la concurrence partielle et ne satisfassent pas la propriété TP2 dans toutes les situations. Dans ce cas, nous ne pouvons pas embarquer la fonction de transformation décrite précédemment dans ADOPTED puisqu'il nécessite la satisfaction de TP2.

Dans la partie qui suit, nous allons proposer un nouveau modèle pour le mécanisme de la transformation qui peut servir pour atteindre la cohérence de l'objet collaboratif quelque soit l'ordre d'exécution des opérations sur les différents sites et quelque soit la taille du réseau sur lequel l'approche des transformées opérationnelles sera appliquée.

V. Chapitre IV : Implémentation et interfaces

V.1. Introduction :

Dans ce chapitre, nous abordons le rôle de notre application. Le but est de proposer un nouveau type de données capable d'assurer la convergence des données dans un environnement d'édition collaborative P2P indépendamment de l'ordre d'exécution des opérations d'édition sur les différents sites.

Afin de gérer ce nouveau type de données, des nouvelles fonctions de transformation conformes à cette structure de données doivent être développées et optimisées de sorte que la transformation s'effectue le plus rapidement possible.

Puisqu'un très grand nombre d'opérations peut être traité dans un intervalle de temps rétréci, la solution proposée ne permet pas seulement d'assurer que le système passe facilement à l'échelle mais aussi d'éviter le regroupement de plusieurs opérations simultanées et par conséquent d'améliorer les performances et d'assurer la convergence des données.

V.1.1. Motivation:

Le contexte général de notre travail porte sur les applications collaboratives dans les réseaux P2P, plus particulièrement sur l'édition collaborative des documents à structure linéaire dans ces environnements de communication. Une application distribuée doit permettre aux utilisateurs de travailler sur les mêmes documents d'une façon collaborative synchrone ou asynchrone. Pour cela, et afin d'améliorer la disponibilité des données dans ces systèmes qui sont très dynamiques, les applications P2P suivent un mode de réplication optimiste. Les répliques d'un même document peuvent être modifiées par plusieurs utilisateurs à la fois. C'est ainsi que les données peuvent être récupérées à partir d'un site détenant la réplique en cas d'indisponibilité d'un autre.

Cependant, la cohérence à terme des répliques n'est pas forcement garanti après leurs mises à jour. Comme nous l'avons vu, les modifications concurrentes effectuées sur les différentes répliques peuvent créer des divergences de copies. D'autre part, quand certains pairs rejoignent le réseau après l'avoir quitté (mouvement dynamique), cela pose le problème de la mise à jour de leur réplique. Ce non garanti de la cohérence des données provoque un état non cohérent du système.

Par ailleurs, nous avons présenté au 2ème chapitre le modèle TO comme étant la solution la plus répandue pour mettre en place un mécanisme de réplication optimiste dans un contexte distribué. Ce modèle est basé sur un ensemble de fonctions de transformation pour assurer la cohérence des données. Nous avons vu aussi que l'algorithme « ADOPTED » permet l'intégration des opérations de mise-à-jour mais ne permet pas en contrepartie de passer à l'échelle en terme de nombre d'utilisateurs. En effet, à mesure que le nombre d'utilisateurs augmente, les données répliquées deviennent volumineuses. Il va falloir donc plus de temps pour les modifier et ainsi les opérations de mises-à-jour vont s'accumuler à nouveau sur le

Log. Les états des répliques deviennent critiques et les risques d'incohérences peuvent augmenter involontairement. Encore, nous ne pouvons pas embarquer la fonction de transformation décrite précédemment dans ADOPTED puisqu'elle est incorrecte et nécessite la satisfaction de TP2.
Si le système parvient à traiter rapidement et correctement les opérations concurrentes, on peut dire que les situations de divergences vont se produire rarement, ou presque jamais et le système passe ainsi vers un état stable (cohérence des répliques).

V.1.2. Contribution:

Dans le reste de ce chapitre, nous proposons une solution originale aux problèmes rencontrés dans les solutions précédentes, tout en donnant une nouvelle technique de propagation des mises-à-jour des données dans un réseau P2P.

Notre solution s'appuie sur le modèle des transformées opérationnelles TO pour régler le problème de la cohérence des répliques. Nous allons suivre le même principe pour respecter les critères du modèle CCI mais à la différence des fonctions de transformations « TI » utilisées dans l'approche TO qui utilisent un ordre de priorité pour préserver les intentions des utilisateurs, notre solution introduit un nouveau paramètre permettant d'ordonner les éléments des répliques suivant les intentions des pairs.

Comme nous l'avons vu dans le 1er chapitre, les systèmes d'édition collaborative peuvent manipuler des documents partagés de différents types. Notre travail s'intéresse au traitement des objets collaboratifs qui admettent une structure linéaire. Ces objets peuvent être assimilés à une liste finie d'éléments d'un type de données particulier où chaque élément peut être considéré comme un caractère, un paragraphe, une page, un nœud XML… etc.

Ce nouveau paramètre que nous allons définir représente un poids pour chaque élément. Dans ce cas, on peut définir un document partagé par une séquence de pairs (*élément, poids*) dont les éléments seront ordonnés suivant leurs poids uniques.

V.2. Modèle de Coordination :

Dans notre modèle, nous définissons la requête q comme un quadruplet (u, k, v, o) où u est l'identifiant du site collaborateur (ou l'utilisateur) émetteur de la requête, k est le numéro séquentiel des requêtes générées localement, v le vecteur d'état du site, et enfin o est l'opération à être exécutée sur l'état partagé. Deux types d'opérations sont possibles :

1) *Ins (i, (c,ω))* : permet d'insérer le caractère c avec le poids ω a la position i.

2) *Del(i)* : permet de supprimer un élément a la position i.

V.2.1. Le poids de position unique :

Soit « U » l'ensemble des identifiants des utilisateurs. On suppose par ailleurs que les identifiants des utilisateurs sont uniques, dans ce cas, deux utilisateurs distincts n'ont pas les mêmes identifiants.

On définit le poids de position par une paire d'éléments où le 1er représente la valeur de la position et le second représente l'identifiant de l'utilisateur qui a créé ce poids :

$$\mathcal{W} = \{(p, u) : p \in \mathbb{N}^*, u \in U\}$$

Pour un état de document $S = \omega_1\omega_2 ... \omega_L$ on utilise les notations suivantes :

- $|S|$ signifie la longueur de S.
- ω_i représente le poids de l'élément à la position « i » tel que : $i \in \mathbb{N}$ $(1 \leq i \leq L)$.

V.2.2. Principe d'ordonnancement des poids de position :

On peut dénoter le premier composant du poids (la position) par $l(\omega)$ et le deuxième (l' identifiant du site) par $r(\omega)$. La relation d'ordre $\omega_1 < \omega_2$ est vérifiée dans l'une des situations suivantes:

- Si : $l(\omega_1) < l(\omega_2)$
- Si : $l(\omega_1) = l(\omega_2)$ et $r(\omega_1) < r(\omega_2)$

On peut dire alors que chaque élément créé au niveau d'un site aura un poids unique car le poids minimal que peut générer un utilisateur « u » est $\omega_0^u = (1, u)$, par ailleurs, pour deux poids ω_1 et ω_2 générés par un même utilisateur « u », le composant droit $r(\omega_1) = r(\omega_2) = u$, de plus, pour n'importe quel poids ω son successeur immédiat $\omega' = s(\omega)$ tel que : $s(\omega) = (l(\omega)+1 , r(\omega_2))$.

Donc pour un document $S = \omega_1\omega_2 ... \omega_L$ nous avons : $\omega_1 < \omega_2 < ... < \omega_i < ... < \omega_L$

V.2.3. Utilisation d'un poids de position unique :

Les utilisateurs peuvent modifier le document partagé en effectuant l'une des opérations suivantes :

V.2.3.1. L'insertion d'un nouvel élément :

Lors d'une insertion d'un nouvel élément dans un document, un nouveau poids doit être généré pour celui-ci afin que le document partagé résultant reste ordonné.
L'opération Ins (i, ω) permet d'insérer un poids « ω » à la position « i » sachant que tous les poids à partir de cette position seront incrémentés et décalés d'une position vers la droite.

Pour un document $S = \omega_1\omega_2 ... \omega_L$ deux cas d'insertion sont possibles :

- *L'insertion au début de « S »* : Cela implique que l'utilisateur « u » doit insérer son élément à la position « i = 1 » avec un poids minimal ω_0^u= (1, u). Le reste des éléments se trouvant sur la droite de la position d'insertion seront automatiquement décalés d'une position pour obtenir un nouvel S = $\boldsymbol{\omega_0^u}\ \omega_1'\ \omega_2'\ \ldots\ \omega_L'$ avec : ω_1'= s(ω_1), ω_2'= s(ω_2) … et ω_L'= s(ω_L).

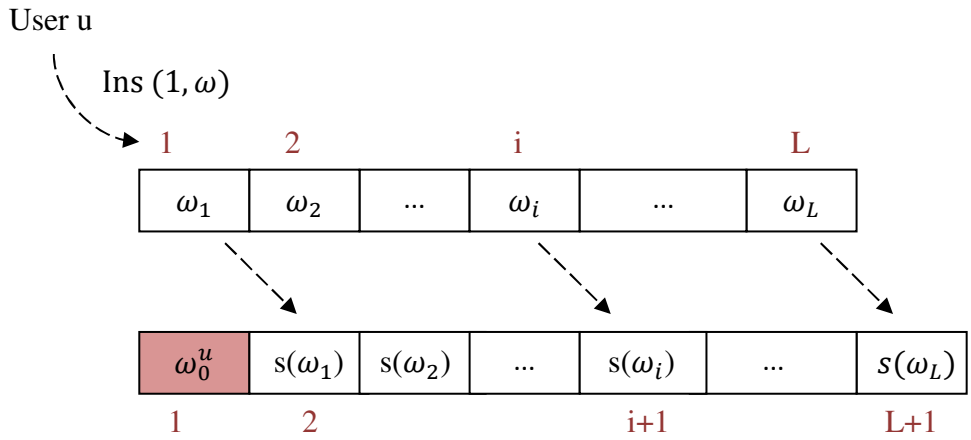


Figure 28 : Insertion d'un élément au début d'un document.

- *L'insertion au milieu ou à la fin de S :* L'utilisateur « u » doit inserer son élément à la position « i » tel que 1 ≤ i ≤ L+1. Le nouveau poids ω de la position « i » sera calculé par rapport à son prédécesseur immédiat ω_{i-1} tel que : $\omega = (l(\ \omega_{i-1}) + 1, u)$.

De la même façon que le cas d'une insertion au début, les poids correspondants aux indices supérieurs à « i » seront décalés vers la droite. On obtient ainsi un nouveau document S = $\omega_1 \ldots \omega_{i-1}\boldsymbol{\omega}\ \omega_i' \ldots \omega_L'$ avec ω_i'= s(ω_i) et ω_L'= s(ω_L).

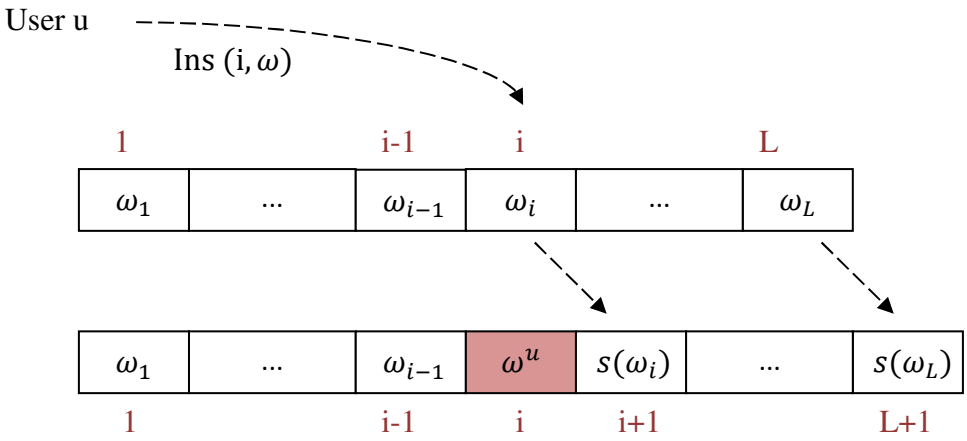


Figure 29: Insertion d'un élément au milieu ou à la fin d'un document.

V.2.3.2. La suppression d'un élément :

Lorsqu'un utilisateur souhaite effacer un élément de son document, il doit utiliser l'opération Del(i) qui permet de supprimer directement le poids se trouvant à la position i.

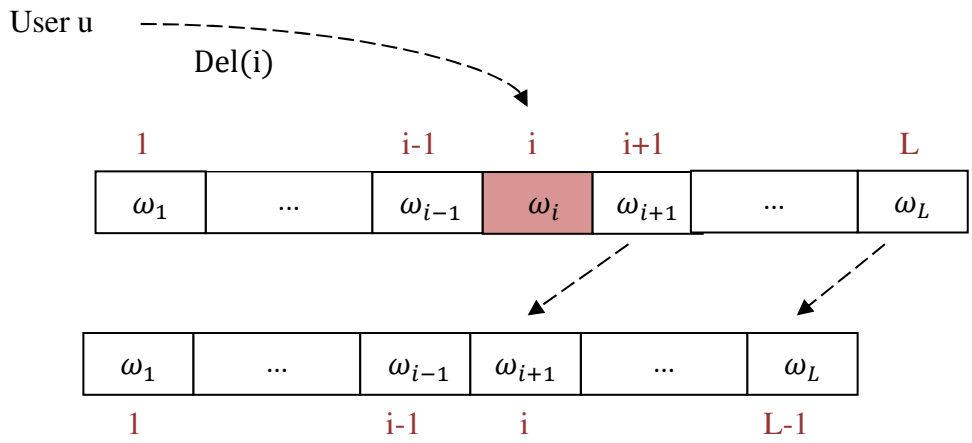


Figure : Suppression d'un élément.

V.3. Architecture de notre système de collaboration

L'application que nous allons élaborer est basée sur le modèle de réplication optimiste TO dans un environnement distribué où chaque utilisateur dispose d'une copie du document partagé. Cette application est destinée aux milieux collaboratifs comme un outils favorisant la communication entre les utilisateurs sans perte d'informations.

Nous devons permettre dans un premier temps à chaque utilisateur de modifier volontairement et à n'importe quel moment la réplique qu'il détient sans imposer la moindre contrainte. Nous devons assurer en parallèle que la communication entre les pairs se fait directement (c'est-à-dire sans passer par une entité intermédiaire ni d'un site central).

En second lieu, nous devons garantir la mise-à-jour des répliques distantes suite aux modifications générées par un site du réseau. Chaque modification effectuée sur un site est traitée

comme étant une requête locale. Une fois propagée dans le réseau, cette requête sera considérée comme étant distante par rapport aux autres sites. L'état d'une réplique est défini par l'ensemble des requêtes générées et reçues au niveau d'un site et par rapport à l'ordre selon lequel ces requêtes ont été exécutées sur ce site.

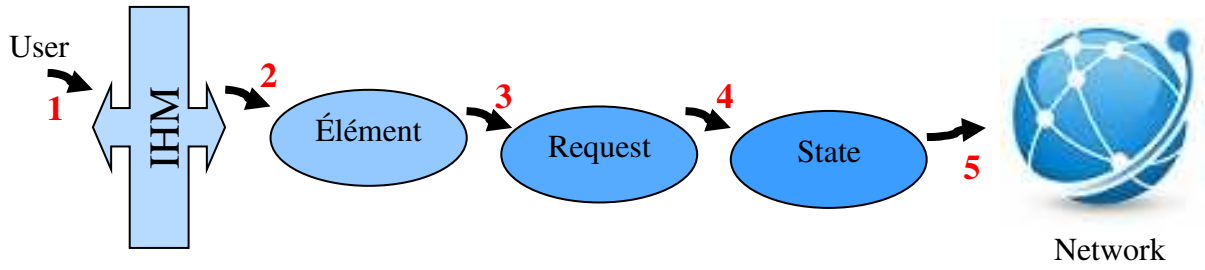


Figure 31 : Génération d'une requête locale.

La Figure 31 présente le principe de la génération d'une requête locale :

1- **Génération d'une opération :** du point de vue de l'utilisateur, le document ne contient qu'une séquence de caractères visibles. Les collaborateurs auront la possibilité de modifier leurs documents soit en générant des opérations d'insertion ou de suppression de caractères.

2- **Création d'une requête :** lorsque l'interface utilisateur (IHM) génère une opération d'insertion, cette opération ne contient que la valeur alphabétique du caractère à insérer et sa position dans la chaîne visible. Le système sera donc chargé de récupérer l'élément (caractère dans notre cas) qui vient d'être inséré ainsi que sa position d'insertion. De même, lors de la génération d'une opération de suppression, nous devons récupérer la position de l'effacement et retrouver le caractère qui doit être supprimé.

3- **Génération d'un nouveau poids** : une requête d'insertion entraine la création d'un nouveau poids ω pour l'élément à inserer. Ce poids sera calculé à partir de la position d'insertion « i » récupérée dans l'étape précédante de la manière décrite sur les Figures (28 et 29) .

4- **Exécution d'une requête :** dès qu'un nouveau poids est généré pour un élément, le système applique la requête localement. Sachant que le type d'une requête est le même que celui de l'opération qui a provoquée la création de cette requête (insertion ou suppression) , l'état d'une réplique change immédiatement suite à la moindre modification. Le résultat « S(O) » obtenu après l'exécution d'une opération O sur l'état S = $\omega_1\omega_2...\omega_L$ de la réplique peut être résumé comme suit :

$$S(O)\begin{cases} \omega_1 ... \omega_{i-1}\ \boldsymbol{\omega^u}\ s(\omega_i) ... s(\omega_L) & \text{Si } O = Ins(i, \boldsymbol{\omega^u}),\ i \le L+1 \text{ et } \omega_{i-1} \le \boldsymbol{\omega^u} \le succ(\omega_i) \\ \omega_1 ... \omega_{i-1}\ \omega_{i+1} ... \omega_L & \text{Si } O = Del(i) \text{ et } i \le L \\ \text{N'est pas définie} & \text{Sinon} \end{cases}$$

5- **Envoi de la requête locale :** une fois une requête exécutée, le système envoi le nouvel état vers le reste des utilisateurs du réseau.

Lorsqu'une requête émise sera reçue par un collaborateur, le site distant sera chargé de trouver une position d'insertion qui permet de garder la cohérence du document partagé. La Figure 32 présente le mécanisme de la réception d'une requête distante :

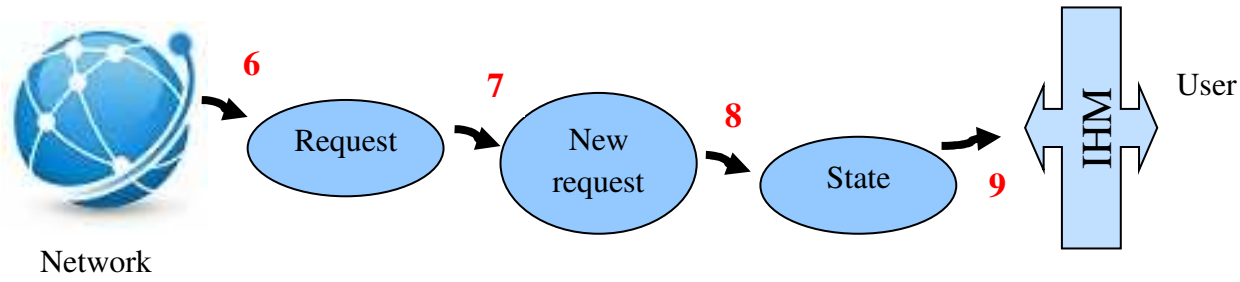


Figure 32: Réception d'une requête distante.

6- **Intégration d'une requête :** notre application suit un mode de collaboration synchrone, de ce fait le système doit rester toujours en écoute pour collecter les opérations propagées sur le réseau. Dès qu'une opération est détectée, nous devons garantir son intégration dans le système. Comme l'ordre de réception des opérations peut être différent de leur ordre de génération le système doit s'en charger d'analyser les différentes opérations reçues et de décider laquelle d'entre elles peut être exécutée à un moment donné (respect de la causalité).

7- **Transformation d'une requête :** Les opérations qui sont reçues en concurrence necéssitent une transformation afin de préserver les intentions des utilisateurs et de garder l'ordonnacement des élément de la réplique. Le paramètre de la position unique pour une opération « O » est défini par rapport à l'état du document S. Le rôle de ce paramètre est de garder l'ordonnancement total des éléments sur les copies partagées entre les pairs.

Pour l'intégration d'une opération de suppression Del(i), nous supprimant simplement le caractère de la position « i » quelque soit le poids qu'il possède (Figure 30).

L'intégration d'une opération Ins (i, ω^u) dans un document reste un peu plus compliquée. Nous devons placer l'élément à insérer dans la position « i » qui se trouve parmi les caractères situés entre «i-1» et «i+1» (voir Figure 29). Ces caractères peuvent être des caractères qui ont été supprimés précédemment ou des caractères qui ont été insérés en concurrence. La procédure de transformation doit réaliser un ordonnancement en se basant sur la position « i » ainsi que le paramètre de position unique $\boldsymbol{\omega^u}$ pour obtenir un nouveau poids $\boldsymbol{\omega^u}$ vérifiant la relation d'ordre : $\omega_{i-1} \leq \boldsymbol{\omega'} \leq \text{succ}(\omega_i)$.

8- **Exécution d'une requête :** la procédure d'exécution d'une requête distante reste la même que celle utilisée pour l'exécution des requêtes locales du fait que le nouveau poids calculé permet de vérifier les conditions d'exécution.

9- **Synchronisation de l'état :** le système doit visualiser le nouvel état obtenu après l'exécution des requêtes distantes à l'utilisateur. Ceci nécessite d'actualiser l'état de la réplique. L'avantage incontestable de ce rafraîchissement d'état est de fournir à l'utilisateur la dernière version du document partagé.

Nous pouvons schématiser l'architecture générale de notre application comme suit :

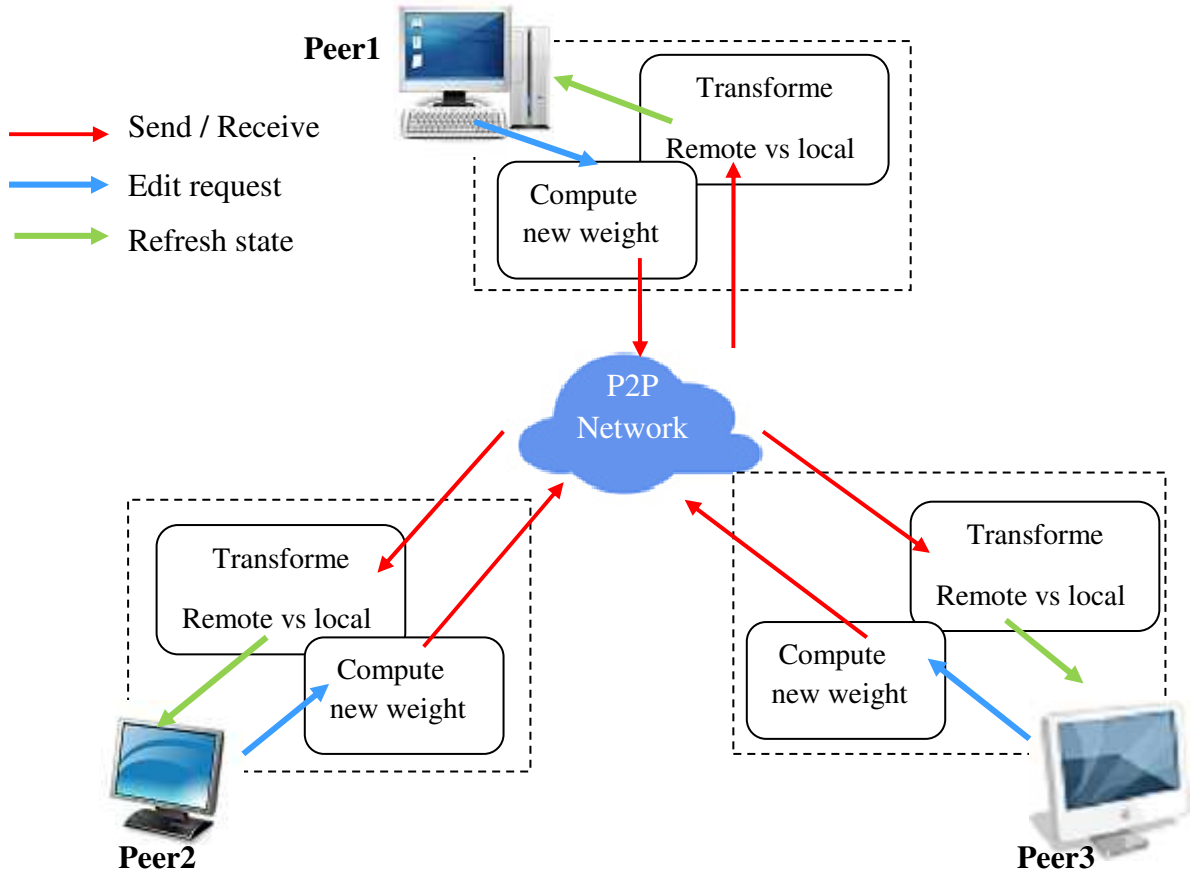


Figure 33 : Architecture générale de l'application

V.3.1. Exemple illustratif :

Pour mettre en évidence ce système de poids unique nous présentons sur la Figure 34 un exemple (*IMINE et MECHAOUI*) où chaque flèche indique l'intégration d'une opération au niveau d'un site.

Les trois utilisateurs lancent une session collaborative à partir du même état initial qui est dans cet exemple un document vide. Le site 2 génère une première opération O1 = Ins (1, ('B', (1,2))). Etant donné que le document est vide, le caractère « B » sera inséré à la première position accompagné par son poids minimal ω_0^2 = (1,2) sur le document partagé. Cette opération sera envoyée vers les autres sites (1 et 3) pour garantir la mise à jour de leurs répliques. Après l'exécution de l'opération O1 sur tous les sites, les trois sites 1, 2 et 3 génèrent simultanément trois opérations O2, O3 et O4 telles que : O2 = Ins (1,('A',(1,1))), O3 = Del(1) et O4 = Ins (2,('C',(2,3))). Une fois le système au repos (mises-à jour terminées) on remarque que les trois sites convergents vers la même valeur « AC ».

Pour mieux comprendre ce mécanisme, nous allons essayer d'analyser l'intégration des opérations au niveau de chaque site.

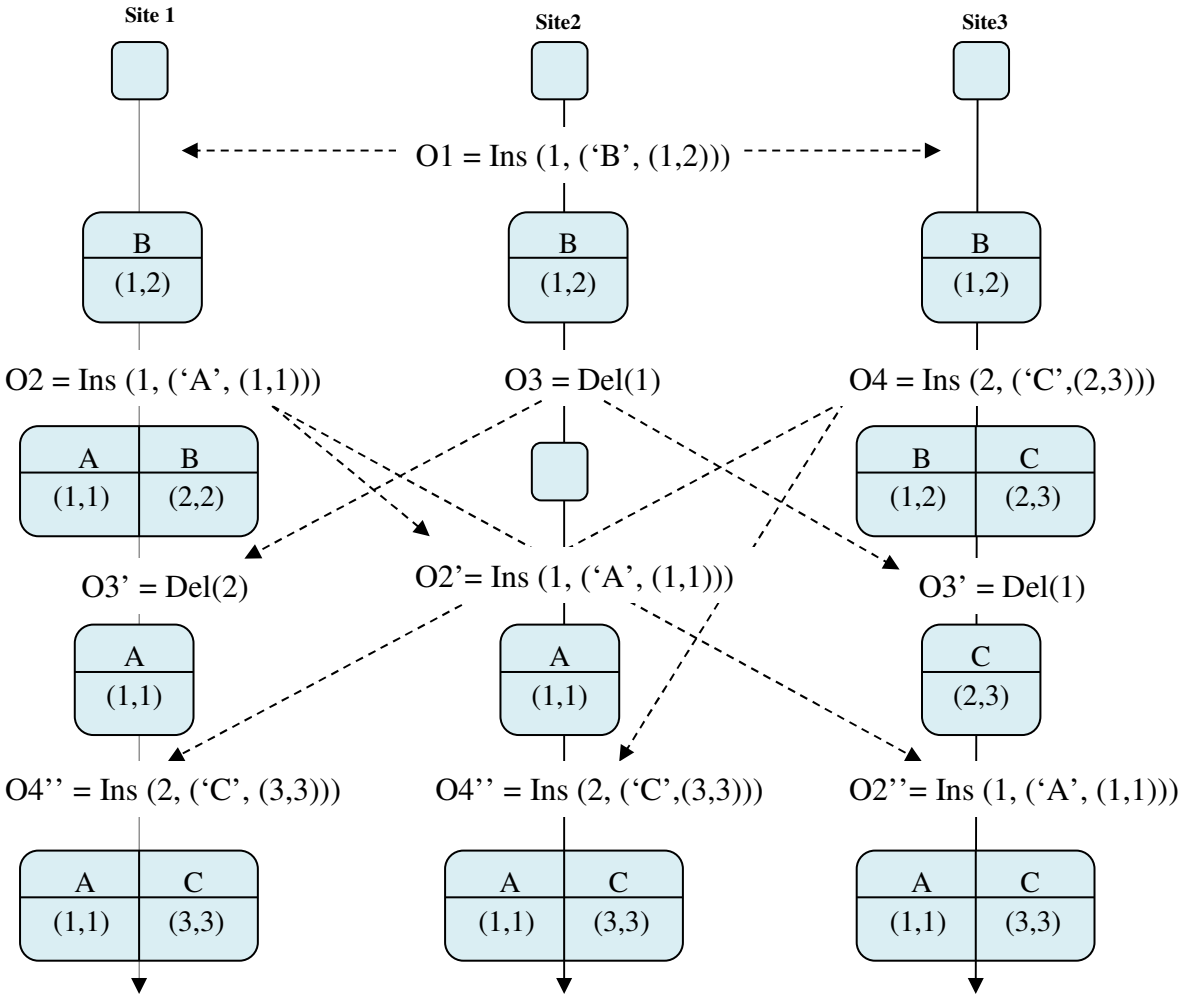


Figure 34: édition concurrente basée sur l'ordre des poids des pairs

- **Site1 :** après l'intégration de l'opération O1 on obtient l'état « B ». L'opération O2 permet ensuite d'insérer le caractère « A » à la 1ère position. Le poids généré pour cette opération sera naturellement le poids minimal : ω_0^1= = (1,1). A la phase de l'exécution, le poids du caractère « B » sera décalé d'une position vers la droite et on obtient la chaine

« AB ». Lorsque O3 atteint ce site, elle sera transformée par rapport à son opération concurrente O2 pour obtenir une nouvelle opération O3' = IT (O3, O2) =Del(2). Cette opération permet d'éliminer le caractère « B » sans imposer une modification du poids. On obtient ainsi l'état « A » sur lequel sera intégrée l'opération O4.

Sachant que O4, O3 et O2 sont concurrente, l'opération O4 devra être transformée par rapport à la séquence [O2, O3'] pour avoir enfin l'état « AC ».

$$\textbf{IT (O4, O2) = IT (} \overbrace{\textbf{Ins (2,('C',(2,3)))}}^{O4} \textbf{ ,} \overbrace{\textbf{Ins (1,('A',(1,1)))}}^{O2} \textbf{) =} \overbrace{\textbf{Ins (3,('C',(3,3)))}}^{O4'}$$

$$\textbf{IT (O4', O3') = IT (} \overbrace{\textbf{Ins (3,('C',(3,3)))}}^{O4'} \textbf{ ,} \overbrace{\textbf{Del(2)}}^{O3'} \textbf{) =} \overbrace{\textbf{Ins (2,('C',(3,3)))}}^{O4''}$$

- **Site2 :** Comme dans le site1, l'opération O1 modifie l'état de la réplique à « B ». Une autre opération O3 est générée sur ce site pour éliminer le premier caractère du document : O3= Del(1) et l'état de la réplique redevient de nouveau vide. Lors de l'intégration d'O2, celle-ci sera transformée par rapport à O3 :

$$\textbf{IT (O2, O3) = IT (}\overbrace{\textbf{Ins (1,('A',(1,1)))}}^{O2} \textbf{ ,} \overbrace{\textbf{Del(1)}}^{O3} \textbf{) =} \overbrace{\textbf{Ins (1,('A',(1,1)))}}^{O2'}$$

L'exécution de O2' permet d'avoir l'état « A » sur lequel sera intégrée l'opération O4. Celle-ci sera transformer par rapport à la séquence [O3, O2']. Le site2 finit donc par avoir la chaine « AC ».

$$\textbf{IT (O4, O3) = IT (} \overbrace{\textbf{Ins (2,('C',(2,3)))}}^{O4} \textbf{ ,} \overbrace{\textbf{Del(1)}}^{O3} \textbf{) =} \overbrace{\textbf{Ins (1,('C',(2,3)))}}^{O4'}$$

$$\textbf{IT (O4', O2') = IT (}\overbrace{\textbf{Ins (2,('C',(2,3)))}}^{O4'} \textbf{ ,} \overbrace{\textbf{Ins (1,('A',(1,1)))}}^{O2'} \textbf{) =} \overbrace{\textbf{Ins (3,('C',(3,3)))}}^{O4''}$$

- **Site3 :** Après avoir intégrer l'opération O1, une autre opération d'insertion notée O4=Ins (2, ('C',(2,3))) est générée sur l'état « B » et qui exige d'ajouter le caractère « C » juste après le « B » . Une fois O3 arrivée sur ce site, elle sera transformée par rapport à O4 : IT(O3, O4) = Del(1) = O3', le caractère « B » sera donc éliminé du document partagé sans

imposer la moindre modification sur le poids suivant et l'état du site sera égal à « C ». Quant à l'opération O2, elle doit être transformée par raport à la séquence [O4 , O3'] comme suit :

$$\mathbf{IT\ (O2, O4) = IT\ (\ \overbrace{Ins\ (1,(`A',(1,1)))}^{O2},\ \overbrace{Ins\ (2,(`C',(2,3)))}^{O4}\ ) = \overbrace{Ins\ (1,(`A',(1,1)))}^{O2'}}$$

$$\mathbf{IT\ (O2', O3') = IT\ (\ \overbrace{Ins\ (1,(`A',(1,1)))}^{O2'}\ ,\ \overbrace{Del(1)}^{O3'}\ ) = \overbrace{Ins\ (1,(`A',(1,1)))}^{O2''}}$$

V.4. Implémentation du prototype :

Dans cette section, nous allons présenter les outils utilisés pour implémenter un prototype d'application collaborative basée sur une version modifié de l'algorithme ADOPTED. Dans cette nouvelle version, nous avons utilisé la structure de donnée et la nouvelle fonction de transformation proposé par IMINE (Univ Nancy & LORIA) et MECHAOUI (Univ Mostaganem) afin d'éviter le problème de divergence de l'algorithme ADOPTED causé par la concurrence partielle.

V.4.1. Les outils utilisés:

L'IDE NetBeans : C'est un environnement de développement intégré basé sur Java (IDE) qui est également désigné comme une plate-forme de composants utilisés pour développer des applications Java.

L'IDE est conçu pour limiter les erreurs de codage et faciliter la correction d'erreur avec des outils tels que les NetBeans FindBugs ou le Debugger pour gérer du code complexe, les points d'arrêt et le suivi de l'exécution. Bien que le NetBeans est conçu spécifiquement pour les développeurs Java, il peut supporter également d'autres langages comme C, C ++, PHP ,XML, HTML en plus de JavaScript ... Cet outil inclue un éditeur de texte riche en fonctionnalités avec des outils de Refactoring (ré-usinage de code) et des modèles de code, un glisser-déposer pour permettre la conception de l'interface graphique... L'IDE NetBeans peut fonctionner sur tout système d'exploitation qui prend en charge une JVM compatible, y compris Linux et Windows.

La plate-forme prend en charge la création de nouvelles applications et le développement des applications existantes en utilisant des composants logiciels modulaires. Cette plateforme est en elle-même extensible et peut être étendue pour supporter de nouveaux langages. Il a été convertit à l'open source par Sun Microsystems en juin 2000.

Cet IDE est donc le choix idéal pour notre application. L'installation de l'IDE comprend l'installation du logiciel NetBeans lui-même et l'installation du JDK - ou Java Development Toolkit.

Le JDK: Java Development Toolkit : C'est un kit de développement java qui contient d'une part le JRE (c'est à dire les composants nécessaires au lancement d'applications Java ainsi qu'une machine virtuelle Java), et d'autre part un ensemble d'outils pour compiler et débugger le code écrit en langage de développement java.

L'intergiciel de communication JGroups : JGroups est une boîte à outils pour la messagerie fiable. Il peut être utilisé pour créer des clusters dont les nœuds peuvent envoyer des messages les uns aux autres. Les principales fonctionnalités se résument dans la création et la suppression des clusters. Les nœuds d'un cluster peuvent être répartis sur le réseau et peuvent facilement joindre ou quitter le groupe de communication.

Cet API (Application Programming Interface) permet l'envoi et la réception de messages de nœud à cluster (point à multipoint s) ou l'envoi et la réception de messages de nœud à nœud (point à point). La caractéristique la plus puissante de JGroups est sa pile de protocole flexible, qui permet aux développeurs de l'adapter pour le correspondre exactement à leurs exigences d'application et les caractéristiques du réseau.

JGroups est livré avec plusieurs protocoles : protocoles de transport: UDP ou TCP, protocoles de commande : FIFO ... Il permet également la fragmentation des messages volumineux et la retransmission de messages perdus. D'une autre part JGroups permet aux développeurs de créer des applications de messagerie fiable (one-to- one ou one -to-many). Il permet aux développeurs d'applications d'économiser beaucoup de temps, et permet le déploiement d'applications dans des environnements différents sans avoir à modifier le code.

V.4.2. Présentation de notre prototype :

Notre prototype est une application d'édition collaborative de texte encourageant la communication pair-à-pair entre un ensemble de collaborateurs en utilisant JGroups. Elle offre aux participants une interface d'interaction permettant d'introduire un ensemble d'opérations d'insertions et de suppressions des caractères de texte. Ces opérations peuvent être facilement échangées entre les membres du réseau sans aucune perte d'information tout en maintenant la cohérence du document partagé entre les utilisateurs.

V.4.2.1. Interface de l'application:

Cette Figure ci-dessous montre la première interface de l'application.

Figure 35: Interface d'accueil de l'application

L'avantage de cette interface se trouve dans la simplicité de son fonctionnement, car même un utilisateur non expérimenté peut manipuler facilement cet outil. Dès que l'interface se lance, l'utilisateur aura la possibilité de joindre le groupe de collaboration juste en introduisant un nom d'utilisateur.

Le bouton « Join now » permet de créer un canal de communication avec les personnes déjà connectées sur ce groupe. Ceci garantit l'envoie et la réception des informations qui circulent entre les membres du groupe.

Une fois la personne connectée, une deuxième interface se lance.

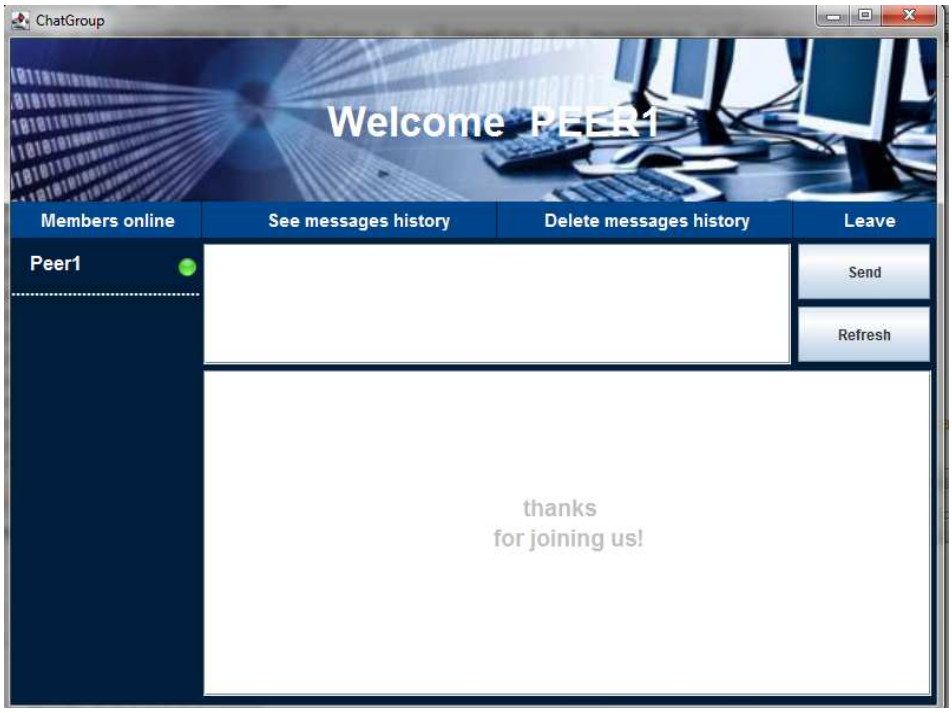


Figure 36 : Interface principale de l'application

Cette interface contient :

1- La liste des membres connectés : Cette liste contient les noms d'utilisateurs de toutes les personnes formants le groupe de communication.

Figure 37: Liste des membres connectés

2- La zone d'édition de texte : Cette zone permet à l'utilisateur d'introduire la séquence de caractère qu'il souhaite partager avec les autres membres connectés.

Figure 38 : La zone d'édition de texte de l'application

3- Le bouton « Send » : Ce bouton permet d'envoyer le texte écrit sur la zone d'édition aux autres participants.

Figure 39 : Bouton d'envoi des modifications

4- Le bouton « Refresh » : Ce bouton permet d'actualiser (synchroniser) l'état du site en affichant les modifications effectuées par les autres membres sur la zone d'édition.

Figure 40 : Bouton de synchronisation de l'état

5- La zone contenant les requêtes des utilisateurs: Les interactions des utilisateurs (les écritures et les effacements) sont considérées comme étant des requêtes. Nous essayons d'afficher ces requêtes dans le but de suivre les modifications effectuées par les collaborateurs (génération ou réception d'une requête).

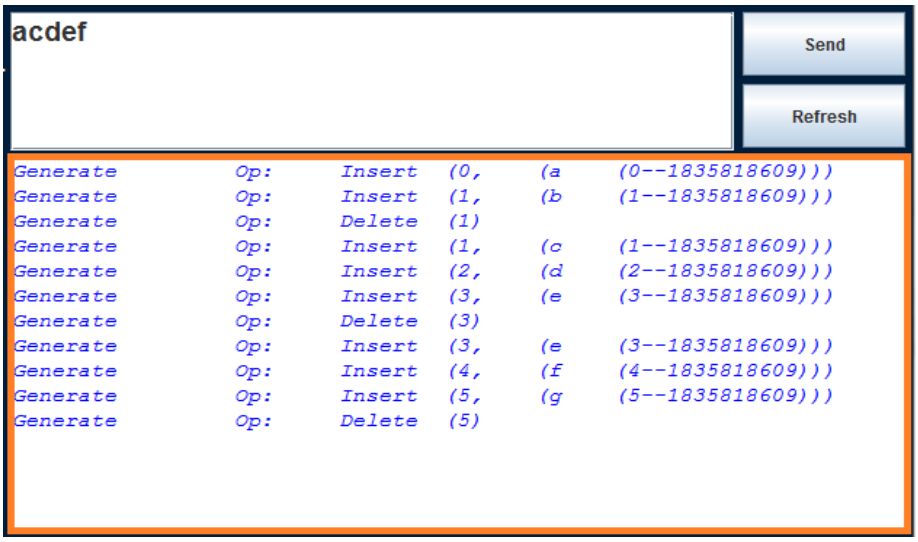


Figure 41: La partie contenant les requêtes des utilisateurs.

Pour chaque opération générée sur l'interface, le système construit une requête contenant les informations de l'action effectuée par l'utilisateur : le caractère inséré ou supprimé, la position de l'opération ainsi que le nouveau poids calculé pour chaque élément.

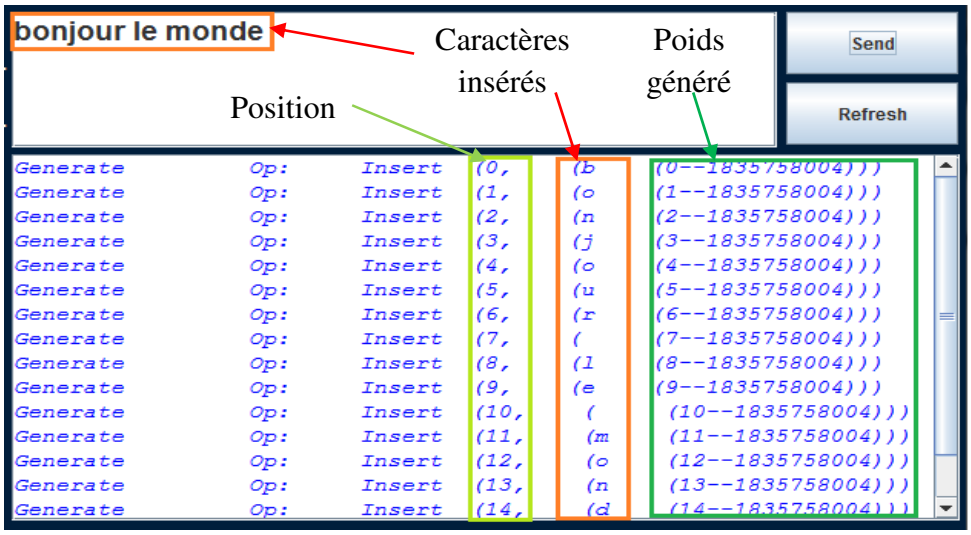


Figure 42 : Affichage des requêtes générées.

Les modifications des autres utilisateurs sont reçues comme étant des requêtes distantes. Ainsi chaque site possèdera de la liste de ses requêtes générées ainsi que celles reçues à partir du réseau (opérations distantes).

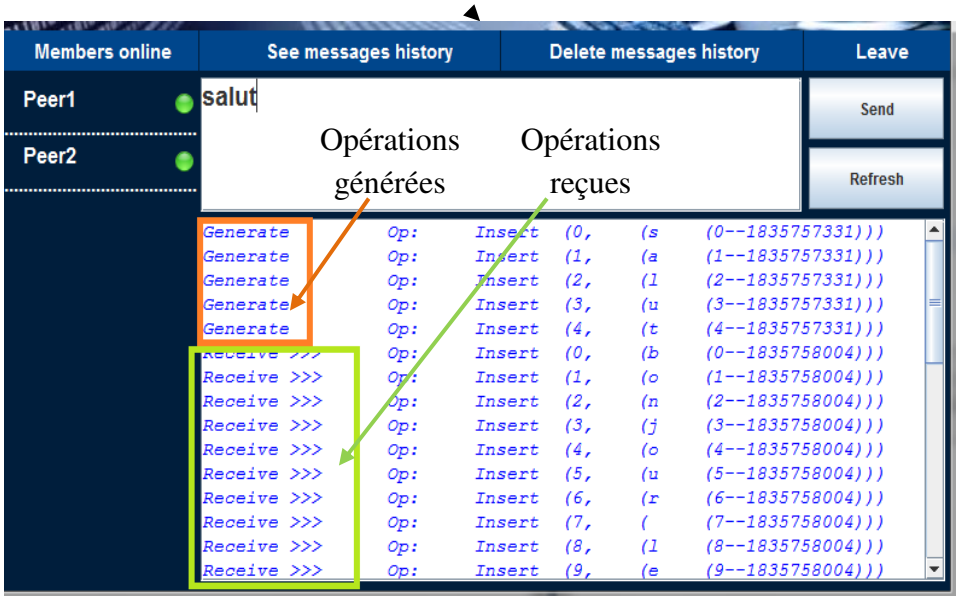


Figure 43 : Liste des requêtes générées et reçues sur un site

6- Consultation de l'historique : Ce bouton permet d'afficher la liste de toutes les requêtes générées par tous les utilisateurs formant le groupe. Ceci peut être utile lors de l'arrivée d'un nouveau membre pour voir les états par lesquels la copie partagée est passée.

Figure 44: Consultation de l'historique des modifications

7- Suppression de l'historique : ce bouton permet d'effacer la liste de toutes les requêtes exécutées précédemment.

Figure 45: Suppression de l'historique des requêtes.

8- Déconnexion d'un membre : le bouton « leave » permet à un membre de quitter le groupe. La personne peut rejoindre le réseau à n'importe quel moment et aura également la possibilité de consulter les modifications qu'elle a ratée et de travailler sur la version la plus récente du document.

Figure 46 : Déconnexion d'un membre.

V.5. Conclusion :

L'objectif principal de notre travail est de concevoir et d'implanter une solution de réplication optimiste basée sur l'algorithme ADOPTED et adaptée aux besoins d'un éditeur collaboratif fonctionnant sur un réseau P2P et supportant des utilisateurs en situation de mobilité. Pour cela nous avons modifié la version actuelle d'ADOPTED en utilisant une nouvelle structure de données et de nouvelles fonctions de transformations afin de pallier les problèmes causés par la concurrence partielle.

Dans ce chapitre, nous avons mis en avant les outils utilisés pour la conception et l'implémentation de notre application, nous avons aussi exposé son objectif, qui est de gérer la concurrence dans les systèmes d'éditions collaborative distribués en se basant sur le modèle TO. Nous avons également essayé de décrire et d'illustrer ses fonctionnalités à travers les différentes interfaces que nous avons présenté.

Conclusion générale

Les domaines de la communication et de l'édition collaborative sont émergeant de nos jours et évolue rapidement. Les applications distribuées et les outils encourageant la coopération dans les environnements P2P sont nombreux et différents. Notre objectif est de fournir des mécanismes permettant de déployer des systèmes d'édition collaborative de texte sur ces environnements. Les fondements théoriques proposés dans le cadre de ce mémoire ont permis de traiter plusieurs objectifs et d'ouvrir différentes perspectives que nous résumons ci-dessous :

L'objectif essentiel de ce travail été d'appliquer l'approche des transformées opérationnelles TO utilisée pour résoudre les conflits lors d'une édition concurrente d'un document partagé sur un réseau P2P tout en supportant les caractéristiques de cet environnement tels que le nombre massif de participant ainsi que leur mouvement dynamique sur les réseaux.

Le second objectif été de proposer une structure capable d'assurer la cohérence des données dans une application distribuée en s'appuient sur le modèle TO et permettant de supporter les coordinations des collaborateurs en particulier dans des contextes critiques. Afin de mettre en évidence cette nouvelle structure de données, nous avons modifié l'algorithme ADOPTED en remplaçant l'ancienne structure de données et ses fonctions de transformations par nos nouvelles fonctions qui sont conformes à notre structure de données proposée.

Nous étions chargés en dernier lieu de définir un support de travail collaboratif permettant d'utiliser cette nouvelle structure. L'outil que nous avons développé permet de faire de l'édition collaborative en pair-à-pair sur les documents textuels, cette édition se fait sans aucunes contraintes ni de mises-à-jour faites par l'utilisateur, les conflits sont réglés de façon automatique par le système avec une convergence des copies.

Cette solution s'avère intéressante à développer car il serait possible d'étudier plus profondément cette structure et d'essayer de l'adapter aux différents algorithmes d'édition collaborative décentralisée (par exemple l'algorithme SOCT2). Ce travail peut être facilement étendu pour devenir un vrai système d'édition collaborative et qui pourrait être testé dans des conditions réelles d'utilisation.

Comme perspectives, il serait possible d'enrichir cet outil pour affecter des droits de lecture ou d'écriture sur les différentes parties du document partagé. On pourrait imaginer plusieurs groupes ayant chacun un rôle précis et travaillant sur différentes parties du document. Chacune de ces parties pouvant être invisibles ou partiellement visibles pour les autres groupes. Il serait aussi intéressant de pouvoir jouer sur les droits d'accès pendant l'édition et de définir plus formellement une propriété de maintien pour la cohérence en se basant sur la structure proposée.

Bibliographie

[1] Stéphane Weiss. Edition collaborative massive sur les réseaux Pair-à-pair. Thèse de doctorat en informatique. Nancy : Université Henri Poincaré, 2010.

[2] Randolph Aurel Josias Oboubé. Convergence et sécurité d'accès dans les systèmes d'édition collaborative massivement répartis. Thèse pour l'obtention du diplôme de docteur en philosophie. Montréal, Ecole polytechnique de Montréal, département de génie informatique et génie logiciel, 2014.

[3] Gérald Oster. Réplication optimiste et cohérence des données dans les environnements collaboratifs répartis. Thèse de doctorat en informatique. Nancy : université Henri Poincaré, 2005.

[4] Abdessamad IMINE. Conception Formelle d'Algorithmes de Réplication Optimiste Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair. Thèse de doctorat en informatique. Nancy : université Henri Poincaré, 2006.

[5] Hafed Zarzour. Environnement collaboratif sur une infrastructure GRID pour la gestion électronique de documents. Thèse de doctorat en informatique. Annaba : Université Badji Mokhtar, 2013.

[6] Mounir Tlili. Infrastructure P2P pour la Réplication et la Réconciliation des Données. Thèse de doctorat en informatique spécialité bases de données. Nantes : école doctorale STIM, 2011.

[7] Olivier Dedieu, Réplication optimiste pour les applications collaboratives asynchrones. Thèse de doctorat en informatique. Marne la vallée : Université de Marne la vallée, 2000.

[8] Stéphane Martin, Lif. Édition collaborative des documents semi-structurés. Thèse de doctorat en informatique. Provence : université de Provence, laboratoire d'informatique fondamentale de Marseille, 2011.

[9] Applications collaboratives: exemples, architectures et service, https://tronche.com/biblio/cp-97/
Consulté le : 25/01/2016.

[10] Environnement Client/ Serveur
http://www.commentcamarche.net/contents/222-environnement-client-serveur#q=client+serveur&cur=2&url=%2F
Consulté le: 24-03-2016

[11] Chapitre 1 : Architectures des réseaux
http://sanarouissi.net/interne/ch1archi.htm
Consulté le: 24-03-2016

[12] Réseaux d'égale à égale
http://www.commentcamarche.net/contents/224-reseau-d-egal-a-egal
Consulté le : 02-05-2016.

[13] Hanaa Mazyad. Une Approche Multi-agents à Architecture P2P pour l'Apprentissage Collaboratif. Thèse de doctorat en informatique. Côte D'Opale : université du littoral Côte D'Opale, 2013.

[14] Réplication synchrone versus réplication asynchrone – evidian, http://www.evidian.com/fr/produits/haute-disponibilite-logiciel-clustering-application/replication-synchrone-versus-replication-asynchrone/,
Consulté le : 22/01/2016.

[15] Réplication de bases : entre continuité service et disponibilité - JDN
http://www.journaldunet.com/solutions/0301/030127_faq_replication.shtml,
Consulté le : 13/01/2016.

[16] Réplication de bases : entre continuité service et disponibilité - JDN
http://www.journaldunet.com/solutions/0301/030127_faq_replication.shtml
Consulté le 26-03-2016.

[17] Mathias Ressel, Doris Nitsche-Ruhland et Rul Gunzenhause. Approche d'intégration et de transformation orientée pour le contrôle de la concurrence et l'annulation dans les systèmes d'édition collaborative. Rapport de recherche. Allemagne : université de Stuttgart, 1996.

[18] Nicolas Vidot, Convergence des copies dans les environnements collaboratifs. Thèse de doctorat en informatique. Montpellier : Université Montpellier 2, 2002.

[19] Santosh Kumawat, Ajay Khunteta, A Survey on Operational Transformation Algorithms: Challenges, Issues and Achievements. Rapport de recherche. International Journal of Computer Applications. Inde, 2010.

[20] Pascal Molli, Gerald Oster, Hala Skaf-Molli, Abdessamad Imine. Safe Generic Data Synchronizer. Rapport de recherche. LORIA : France, 2003.

[21] Pascal Molli, Gerald Oster, Hala Skaf-Molli, Abdessamad Imine. Utilisation de l'approche de transformation pour construire un modèle sûr et générique pour la synchronisation des données divergentes. Rapport de recherche. LORIA : France, 2003. P.3.

[22] Pascal Molli, Pascal Urso, Abdessamad Imine, Gerald Oster. Fonction de transformation Pierre tombante pour assurer la cohérence dans les systèmes d'édition collaborative. Rapport de recherche. France, université de Nancy, 2006.

[23] Sun Chengzheng, Jia Xiaohua, Zhang Yanchun, Yang Yun, Chen David. Achieving convergence, causality-preservation and intention-preservation in Real-time cooperative editing systems. ACM Transactions on Computer-Human interaction 5(1) p.63-108, 1998.

[24] Abdessamad Imine, Pascal Molli, Gérald Oster, Michael Rusinowitch. Proving Correctness of Transformation Functions in Real-Time Groupware. Rapport de recherche. Finland, proceedings of the eighth European conference on computer-supported cooperative work, 14-18, 2003

[25] Hanifa Boucheneb, Abdessamad Imine. Experiments in model-checking optimistic replication algorithms. Rapport de recherche. INRIA, 2008.