Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

Toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

➢ Contact SCD Nancy 1 : theses.sciences@scd.uhp-nancy.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Concurrence et Conscience de Groupe dans l'Édition Collaborative sur Réseaux Pair-à-Pair

THÈSE

présentée et soutenue publiquement le 13 Novembre 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Sawsan Alshattnawi

Composition du jury

Rapporteurs :	Laurence Nigay	Professeur , Université Joseph Fourier, Grenoble
	Bruno Defude	Professeur, TELECOM & Management Sud Paris
Examinateurs :	Frédéric Alexandre	Directeur de Recherche, INRIA Nancy-Grand Est
	Eric Lecolinet	Maître de Conférences, TELECOM ParisTech
	Pascal Molli	Maître de Conférences HDR, Université Henri Poicaré, Nancy 1
	Gérôme Canals	Maître de Conférences, Université de Nancy 2

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Je tiens d'abord à remercier Gérôme Canals d'avoir encadré mon travail depuis le D.E.A et pendant ces années de thèse et pour ses conseils. Merci beaucoup Gérôme.

Je souhaite exprimer toute ma gratitude à Pascal Molli, mon directeur de thèse, pour ses conseils et pour m'avoir proposé un sujet aussi intéressant.

Je tiens à remercier Claude Godart, responsable du projet ECOO, pour l'opportunité qu'il m'a offerte en m'accueillant au sein de son équipe.

Je souhaite présenter mes remerciements aux membres du jury qui ont accepté d'évaluer mon travail de thèse.

Je remercie toute l'équipe ECOO pour son accueil chaleureux durant ces années. En particulier, je remercie Ronel Rivas, stagiaire dans l'équipe ECOO et étudiant de Licence Professionnelle, pour sa contribution aux réalisations logiciels décrits dans cette thèse.

Je remercie aussi tous mes amis avec lesquels j'ai participé à des moments inoubliables. En particulier, ma meilleure amie et ma soeur Rokia Bendaoud et Shadi Al shehabi pour leurs conseils depuis mon inscription au D.E.A. Je remercie Thomas Meilender d'avoir lu et corrigé la dernière version de cette thèse. Je remercie aussi Antoine Falcone des moyens informatiques du Loria.

J'exprime ma reconnaissance et mon respect envers Dr. Mashhour Elrfaai de l'Université de Yarmouk de m'avoir encouragé, et je remercie cette Université pour la bourse de thèse dont j'ai bénéficié.

Je voudrais remercier du fond du coeur mes parents, mes frères et mes soeurs de m'avoir supporté et aidé.

Je ne pourrais pas finir sans inclure dans ces remerciements les personnes les plus chères qui ont supporté mon indisponibilité : mon mari, merci pour ton éternel soutien et ton amour, merci pour tout ce que tu as fait pour moi pendant notre séjour en France, mes enfants : Naseraldeen, Raheek, et Fatimah qui est née pendant la réalisation de cette thèse.

Je dédie cette thèse
à mes parents,
à ma vie : Mohammad,
à mon coeur : Naseraldeen
et à mes yeux :Raheek et Fatimeh.

Table des matières

Table des figures **ix**

Introduction **1**

1 Organisation du document 2

1 Problématique et cadre de travail **5**

1.1 Travail collaboratif asynchrone et Wikis pair à pair 7

1.1.1 Contexte général du travail : le Travail Collaboratif Assisté par Ordinateur 7

1.1.2 Édition collaborative asynchrone et Wikis 10

1.1.3 Vers des wikis pair-à-pairs 15

1.2 Le Problème de la concurrence dans un Wiki P2P 19

1.2.1 Opérations concurrentes dans un Wiki P2P 19

1.2.2 Rendre les utilisateurs conscients de la concurrence 22

1.3 Concurrence et conscience de groupe dans les systèmes collaboratifs 23

1.3.1 Conscience de groupe 23

1.3.2 Concurrence et conscience de groupe dans les systèmes synchrones . 24

1.3.3 Cas des wikis 25

1.3.4 Conscience de groupe pour le travail coopératif asynchrone 27

1.3.5 Systèmes de gestion de versions 28

1.4 Objectifs de la thèse 31

1.4.1 Objectifs 31

1.4.2 Contraintes liées au contexte 33

1.4.3 Plan de la thèse 33

2 Wooki : un wiki P2P **35**

2.1 Les principes 36

2.2 Les algorithmes 37
2.2.1 L'algorithme Woot 37
2.2.2 Algorithmes de diffusion des patchs 40
2.3 Architecture et mise en oeuvre 41
2.3.1 Conclusion 45

3 Un mécanisme de conscience des modifications concurrentes 47
3.1 Introduction 48
3.1.1 Rappel des objectifs et contraintes 48
3.1.2 Résultats attendus 50
3.1.3 Les principes généraux de notre approche 51
3.2 Algorithmes pour la collecte et le calcul des informations 53
3.2.1 La détection de la concurrence 53
3.2.2 Le calcul des histoires concurrentes 59
3.3 Mécanismes pour la conscience de la concurrence 68
3.3.1 Réception d'un patch distant 69
3.3.2 Demande de visualisation d'une page wiki 70
3.3.3 Demande d'édition d'une page wiki 73
3.3.4 Demande de sauvegarde d'une page modifiée 75
3.3.5 Changement de statut 76
3.4 La Représentation de l'Historique d'une page Wiki 78
3.5 conclusion 82

4 Mise en Oeuvre 83
4.1 La visualisation d'opérations concurrentes 84
4.2 La visualisation de l'histoire d'une page 85
4.2.1 Visualisation du log 85
4.2.2 Visualisation de l'historique des modifications d'une page 86
4.2.3 Visualisation du graphe des modifications d'une page 87
4.3 Conclusion 90

5 Bilans et Perspectives 93
5.1 Bilan 94
5.2 Comparaison aux travaux existants du domaine 95
5.3 Perspectives 100

5.3.1 Perspectives concernant l'évaluation de l'approche 100
5.3.2 Perspectives concernant les fonctionnalités du système 102

Bibliographie **105**

Table des figures

1.1 Le modèle du trèfle [4] 8
1.2 La matrice d'Ellis [15] 9
1.3 Visualisation d'une page wiki (extrait de Wikipedia en français) 12
1.4 Édition d'une page wiki (extrait de Wikipedia en frainçais) 13
1.5 Historique d'une page wiki (extrait de Wikipedia,fr) 14
1.6 Vers des wikis pair-à-pairs 17
1.7 Opérations concurrentes dans un wiki P2P 21
1.8 Fusion d'éditions concurrentes dans Mediawiki 26
1.9 Fusion d'éditions concurrentes dans Mediawiki 26

2.1 Graphe d'opérations dans Woot 39
2.2 Visualisation d'une page Wooki 42
2.3 Visualisation de l'historique d'une page dans Wooki 43
2.4 Architecture du serveur Wooki 44
2.5 Algorithmes de traitement des requêtes dans Wooki 44

3.1 Le résultat attendu 50
3.2 processus de visualisation de page dans un wiki 51
3.3 Notre approche pour la conscience de la concurrence 52
3.4 R-Vecteur avec R=2 58
3.5 Exemple : état initial de la page 60
3.6 Exemple : les modifications introduites sur la page 60
3.7 Exemple : échange des patchs 61
3.8 Exemple : état initial de la page 71
3.9 Exemple : visualisation de l'état final du scénario 72
3.10 Exemple : Édition de la page wiki 74
3.11 Exemple : fin d'édition de la page wiki 75
3.12 Exemple : filtrage des contenus au moment de la sauvegarde 77

3.13 Exemple : transition vers le statut *éditée* 78
3.14 Exemple : 2 patchs de correction concurrents 79
3.15 Exemple : histoire non linéaires des versions sur les sites 1, 2, 3 80
3.16 Représentation simplifiée de l'historique concurrent 81

4.1 Wooki : architecture intégrant la conscience de la concurrence 84
4.2 Exemple : visualisation d'une page *fusionnée* 86
4.3 visualisation le log de page spécifique 87
4.4 Visualisation de détails sur un patch 88
4.5 Visualisation de l'historique des modifications d'une page 89
4.6 Visualisation du patch et accès au log pour une ligne de l'historique 90
4.7 Graphe des patchs 91
4.8 Détails des patchs sur le graphe 91

5.1 Widgets collaboratifs pour l'édition synchrone (tiré de [31]) 96
5.2 Conscience de l'action des autres dans l'outil Gobby 97
5.3 Conscience des changements dans un outil de DAO collaboratif 98
5.4 History Flow : visualisation d'une séquence de versions ([75]) 99
5.5 State treemap ([44]) 99

Introduction

Les wikis sont devenus dans les années récentes les outils d'édition collaborative les plus populaires. En particulier, ils ont rendu possible l'édition collaborative massive et la création collaborative d'objets de forte valeur. Le plus célèbre site Wiki est l'encyclopédie Wikipedia [78]. Créée en 2000, Wikipédia contient plus de 9,000,000 articles dans plus de 250 langues. Wikipedia reçoit plus de 13 millions de requêtes par jour et 200,000 changements sont effectués chaque jour [79].

Avec ce succès sans cesse renouvelé et la croissance continue en taille et complexité des données et en nombre d'utilisateurs, l'architecture client/serveur traditionnelle du web sur laquelle sont basés les wikis actuels atteint ses limites. Ce constat est renforcé avec l'arrivée des wikis sémantiques, qui en plus des données textuelles, manipulent des données formelles sur lesquelles on peut écrire des requêtes. On peut également imaginer l'arrivée de wiki manipulant des données de plus en plus complexes : données géographiques ou géo-référencées, données multimédia, réseaux sociaux imbriqués, données liées à des mondes virtuels 3D etc...

Pour faire face à cette forte montée en charge annoncée, plusieurs travaux récents ont proposé de remplacer l'architecture centralisée traditionnelle par une architecture complètement décentralisée sur un réseau pair-à-pair [51, 46, 76]. L'ambition est de faire bénéficier l'édition collaborative des qualités des applications P2P :

- tolérance aux pannes améliorée grâce à la réplication des données sur les noeuds du réseau,
- capacité à supporter une utilisation massive à grande échelle, grâce à la multiplication des points d'accès au service et à la réplication des données,
- infrastructure simple à déployer et à partager entre plusieurs organisations,
- meilleur support pour le travail nomade et/ou déconnecté, grâce à la capacité à gérer des groupes dynamiques et des noeuds entrant/sortant à tout moment du réseau.

Le travail décrit dans ce mémoire a pour contexte la construction d'un wiki fonctionnant de manière entièrement décentralisée sur un réseau P2P. Ce projet en cours dans l'équipe ECOO débouche sur la construction d'un système, nommé Wooki, qui est un

Wiki P2P basé sur l'algorithme Woot [51].

Notre vision d'un wiki P2P est la suivante. Un wiki P2P est un ensemble de serveurs wiki interconnectés au sein d'un réseau P2P. Les pages wiki sont répliquées sur l'ensemble du réseau. Chaque serveur dispose ainsi des données et de la logique applicative pour rendre le service de manière autonome. Ceci permet d'une part à un utilisateur de s'adresser à un serveur quelconque, et d'autre part à un serveur d'être déconnecté du réseau de manière temporaire tout en continuant à être utilisé sans restriction.

Le cœur d'un tel outil est un mécanisme de réplication optimiste chargé de diffuser les modifications apportées sur un site à l'ensemble des sites du réseau, et d'intégrer les modifications reçues des sites distants sur la copie locale des pages. Ce mécanisme de réplication, pour traiter de manière correcte et non bloquante les modifications émises en concurrence, dispose d'un algorithme de fusion des modifications concurrentes. Ces fusions sont réalisées par chaque serveur au fur et à mesure de l'arrivée d'opérations d'édition distantes.

Contrairement à un wiki classique et à de nombreux outils d'édition collaborative asynchrone où les fusions sont réalisées manuellement ou de manière semi-automatique sous le contrôle d'un utilisateur, les fusions dans un wiki P2P de ce type sont réalisées de manière automatique par les serveurs, en dehors du contrôle des utilisateurs.

La conséquence est que certaines pages du wiki, produites par fusion, sont visibles sur le serveur sans avoir été contrôlées par leurs auteurs. Elles peuvent de plus contenir des incohérences dues aux décisions prises par l'outil de fusion.

Le travail présenté ici a pour objectif de doter les wiki P2P d'un mécanisme apportant une réponse à ce problème nouveau. L'approche choisie consiste à construire un mécanisme de conscience des modifications concurrentes, qui va permettre aux visiteurs et aux auteurs d'une page wiki d'avoir conscience de son statut vis-à-vis de la concurrence : cette page résulte-t-elle d'une fusion par le serveur ou d'une édition par un auteur ? Dans le cas où elle résulte d'une fusion, quelles sont les parties de la page produites par cette fusion et qui nécessitent donc d'être contrôlées ?

Le mécanisme proposé ici doit bien entendu être adapté aux contraintes d'utilisation dans un réseau P2P. En particulier, il doit être entièrement décentralisé, être capable de passer à une grande échelle et ne doit pas remettre en cause la correction du mécanisme de réplication utilisé.

1 Organisation du document

Notre document est organisé de la manière suivante.

Le chapitre 1 est consacré à une présentation du contexte général de l'édition collaborative et des outils d'édition asynchrone actuels. Le cas de l'édition asynchrone sur un wiki pair-à-pair est abordé en détail. En nous basant sur un examen de l'état de l'art dans le domaine de la conscience de groupe asynchrone, nous dégageons une problématique, des contraintes et des objectifs précis pour notre travail.

Le chapitre 2 présente ensuite le contexte particulier de notre étude. Le système Wooki, qui a servi de cadre à nos réflexions et réalisations est présenté, en particulier d'un point de vue technique.

Ensuite, le chapitre 3 décrit notre contribution principale, la conception d'un mécanisme de conscience des modifications concurrentes permettant aux visiteurs d'un wiki et aux auteurs des pages de ce wiki d'avoir une compréhension des actions de fusion entreprises par les serveurs du réseau. Nous présenterons dans ce chapitre les algorithmes que nous avons mis en oeuvre, notamment pour réaliser des calculs de concurrence, et les choix réalisés pour visualiser les effets de la concurrence.

Le chapitre 4 présente les réalisations et les développements associés à notre travail. Il constitue une validation et une preuve de faisabilité pour notre approche, sur le plan technique.

Enfin, ce document se termine par un bilan des contributions et une réflexion sur des perspectives ouvertes par notre travail dans le chapitre 5.

Chapitre 1

Problématique et cadre de travail

Sommaire

1.1 Travail collaboratif asynchrone et Wikis pair à pair 7

1.1.1 Contexte général du travail : le Travail Collaboratif Assisté par Ordinateur 7

1.1.2 Édition collaborative asynchrone et Wikis 10

1.1.3 Vers des wikis pair-à-pairs 15

1.2 Le Problème de la concurrence dans un Wiki P2P 19

1.2.1 Opérations concurrentes dans un Wiki P2P 19

1.2.2 Rendre les utilisateurs conscients de la concurrence 22

1.3 Concurrence et conscience de groupe dans les systèmes collaboratifs 23

1.3.1 Conscience de groupe 23

1.3.2 Concurrence et conscience de groupe dans les systèmes synchrones 24

1.3.3 Cas des wikis 25

1.3.4 Conscience de groupe pour le travail coopératif asynchrone . . 27

1.3.5 Systèmes de gestion de versions 28

1.4 Objectifs de la thèse 31

1.4.1 Objectifs 31

1.4.2 Contraintes liées au contexte 33

1.4.3 Plan de la thèse 33

Cette thèse a pour cadre général le Travail Coopératif Assisté par Ordinateur (TCAO). Plus précisément, nous nous intéressons à l'ingénierie des systèmes collaboratifs distribués. Notre problématique de recherche découle de l'émergence de systèmes pour l'édition collaborative de documents fonctionnant sur des architectures décentralisées de grande taille, et en particulier sur des réseaux pair-à-pair. Ces approches, fondées sur la réplication des données partagées, permettent d'envisager des infrastructures de travail collaboratif robustes et capables de supporter une forte montée en charge tout en restant simples à partager entre plusieurs organisations. Un certain nombre de résultats et prototypes sont récemment apparus et concernent notamment les wikis pair-à-pair. Cette évolution des outils collaboratifs sera détaillée de manière plus approfondie dans la première partie (1.1) de ce chapitre.

Si cette approche nouvelle est prometteuse, elle soulève néanmoins de nombreux problèmes. Parmi eux, le problème le plus important est celui lié au maintien de la cohérence des copies lorsque des modifications concurrentes (ou simultanées) ont lieu sur différentes copies d'une même donnée partagée par les participants d'une session collaborative. Un grand nombre de résultats, propositions et travaux existent sur ce thème. Même si, pour l'instant, aucune solution générale satisfaisant toutes les contraintes posées ne s'est réellement imposée, ces résultats permettent de construire des outils offrant des garanties du point de vue de la concurrence. Associé à ce problème de maintien de cohérence, un autre problème concernant les modifications concurrentes existe. Il est lié à la nature interactive des applications collaboratives. Dans un système coopératif, les utilisateurs doivent avoir conscience des actions des différents participants afin de situer leur propre action dans la perspective du groupe. Le système offre des mécanismes spécifiquement dédiés à cela, désignés par le terme génériques de *mécanismes de conscience de groupe*. Dans la partie 1.2 nous montrerons que notre contexte de travail décentralisé sur réseau P2P amène des problèmes nouveaux et spécifiques concernant la conscience de groupe, et en particulier la conscience des modifications concurrentes. Cette problématique fait l'objet de cette thèse.

Ensuite, dans la section 1.3 de ce chapitre, nous étudierons comment ce problème de gestion d'opérations concurrentes et de conscience de groupe associée est traité dans différents systèmes de support au travail coopératif. Nous aborderons les systèmes d'édition collaborative synchrone, les wikis actuels, les approches pour la conscience de groupe en milieu asynchrone et les systèmes de gestion de versions.

Finalement, nous terminerons ce chapitre consacré à la problématique de notre travail en énonçant de manière détaillée les objectifs visés par cette thèse.

1.1 Travail collaboratif asynchrone et Wikis pair à pair

1.1.1 Contexte général du travail : le Travail Collaboratif Assisté par Ordinateur

Les outils de travail collaboratifs connaissent un succès grandissant. Ils permettent à des groupes de personnes distribués dans l'espace et le temps d'interagir pour la réalisation d'objectifs communs. Ces outils sont très divers : certains s'adressent à de petits groupes de professionnels centrés autour d'un objectif cible, alors que d'autres sont dédiés à de larges communautés partageant des intérêts communs. Ces outils concernent maintenant tous les domaines d'application classiques de l'informatique : de la gestion des organisations (systèmes de workflow) à l'interaction sociale (réseaux sociaux), en passant par l'ingénierie concourante (forges logicielles).

Internet et le web jouent bien sur un rôle central dans le fort développement des technologies de la collaboration. Ses capacités de communication et ses protocoles standardisés en font une plate-forme de développement privilégiée pour toutes les applications dédiés aux groupes. Le système BSCW (Basic Support for Cooperative Work) [3] est l'un des précurseurs des outils de collaboration fondés sur le web. Il a notamment popularisé la notion d'espace de travail partagé en ligne regroupant différentes sortes d'informations (documents partagés, annotations, liens, discussions etc...) que l'on retrouve dans de nombreux outils et infrastructures en ligne actuellement.

Plusieurs cadres d'analyse des outils de travail collaboratifs ont été proposés. En particulier, le trèfle GT Scoop [4] permet une analyse dans l'espace des fonctions offertes, et la matrice d'Ellis [15] permet de classifier les outils selon la situation de collaboration qu'ils prennent en compte.

Le modèle du trèfle GT-Scoop, présenté en figure 1.1 organise l'espace fonctionnel des outils collaboratifs en 3 dimensions principales :

- les fonctions de production : ensemble des fonctionnalités dédiées à la création et modification des données partagées au sein d'un outil. Cela inclut des fonctionnalités de stockage et de contrôle de la concurrence de façon à préserver l'intégrité et la cohérence de ces données, ainsi que la gestion des conflits éventuels.
- les fonctions de communication : ensemble des fonctionnalités permettant aux participants d'une activité de collaboration d'interagir de manière directe. On considère donc ici les communications personnes-personnes.
- les fonctions de coordination : ensemble des fonctionnalités permettant aux participants de coordonner leurs actions entre eux. Cette coordination peut être explicite,

c'est-à-dire soumise à des règles/polices explicitement exprimées et manipulées par l'outil, ou implicite, grâce à des mécanismes de conscience du groupe et des actions des autres.

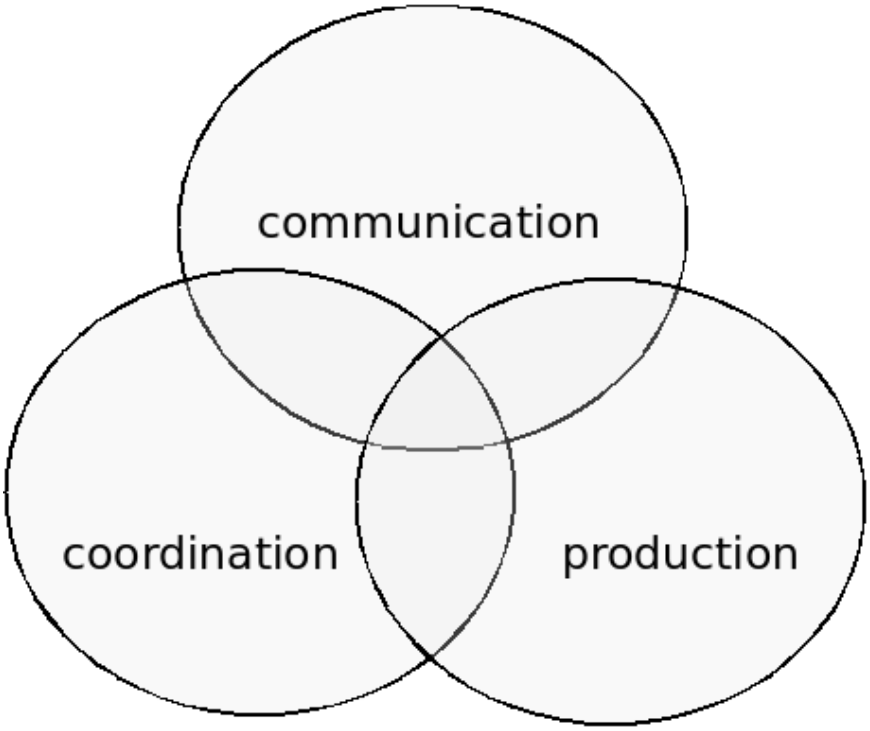

FIG. 1.1 – Le modèle du trèfle [4]

Ce modèle du trèfle permet d'analyser les fonctionnalités offertes par un outil collaboratif et de comparer différents outils. D'autre part, les outils ne proposent pas tous des fonctionnalités selon ces 3 dimensions.

Ellis, dans [15], propose une classification des outils collaboratifs reposant sur deux caractéristiques : l'espace et le temps. L'axe de l'espace considère la distance spatiale entre les utilisateurs, alors que l'axe du temps considère la distance temporelle. L'axe espace se divise comme suit :

1. même lieu : correspond au cas où les membres du groupe sont colocalisés au moment de l'utilisation du système collaboratif,
2. lieux différents : correspond au cas où les membres du groupe sont distribués géographiquement dans l'espace.

L'axe temps est divisé comme suit :

1. même moment, ou *synchrone* : correspond au cas où les membres du groupe utilisent simultanément le système collaboratif.

2. moments différents, ou *asynchrone* : correspond au cas où les membres du groupe utilisent le système collaboratif à des moments différents.

Cette division donne naissance à une matrice 2x2 dite "matrice d'Ellis [15]" présentée dans le Tableau 1.2. Cette matrice permet de classifier les outils de collaboration selon les modes d'interaction qu'il proposent : colocalisés ou à distance, synchrone ou asynchrone.

Quatre régions distinctes sont représentées dans la matrice d'Ellis.

Tout d'abord celle issue du croisement « même moment » et « même lieu », dans laquelle se trouvent les systèmes informatiques qui supportent le travail en mode face-à-face (aussi appelé « meeting facilities »), tels que RoomWare[56]. L'idée est d'équiper l'environnement d'une salle de réunion (murs, tables) d'outils interactifs favorisant le travail de groupe.

Dans la deuxième région, « même lieu » et « moments différents », apparaissent les interactions asynchrones sur un même lieu. Laurillau [38] illustre cette catégorie avec la prise de parole au cours d'une présentation, car dans ce cas, l'enchaînement des actions est planifié et elles se déroulent dans un même lieu. Un exemple de ce cas est donné par Miller et al. [43] qui proposent un outil pour le contrôle interactif des applications dans une salle équipée de plusieurs ordinateurs branchés sur des projecteurs. L'idée est de permettre aux participants dans une telle salle de contrôler, à tour de rôle, les applications qui sont projetées dans la salle.

	Même moment	**Moments différents**
Même lieu	Interaction face-à-face *RoomWare, tableau blanc interactif*	Interaction asynchrone *team rooms panneaux interactifs publics*
Lieux différents	Interaction synchrone et distribuée *mediaspaces, messageries instantannées éditeurs multi-utilisateurs*	Interaction asynchrone et distribuée *email, forum, blogs BSCW, forges logicielles wiki, réseaux sociaux, workflow*

FIG. 1.2 – La matrice d'Ellis [15]

Les autres régions sont caractérisées par une distribution géographique, les utilisateurs se trouvant dans des lieux différents (dans des bureaux distincts, dans différents lieux d'une même ville, ou même dans des villes, pays ou continents différents). La troisième région correspond aux systèmes informatiques qui supportent les interactions synchrones distribuées. Dans ce type de système, les utilisateurs sont connectés simultanément (au même moment) [38], l'interaction a lieu en temps réel mais les utilisateurs sont dispersés. Les Mediaspaces [12, 6] et les messageries instantanées sont des systèmes relevant

typiquement de cette catégorie. De nombreux outils d'édition de documents fonctionnant en mode synchrone (ou "temps réel") ont également été proposés, comme Grove [15], rIbis[57], DistEdit [36], GroupKit [60].

La dernière région concerne les systèmes qui supportent des interactions asynchrones et distribuées, dans lesquels les utilisateurs n'ont pas besoin d'être connectés simultanément pour interagir. L'interaction peut se produire en temps différé entre des utilisateurs distribués géographiquement. Le courrier électronique classique et les forums en ligne relèvent de cette catégorie. De même, de nombreux outils pour la coordination, workflow, agendas et calendriers partagés sont des outils asynchrones. Les outils de gestion de révisions et de versions, initialement conçus pour la gestion de code source, permettent l'édition asynchrone de textes. De nombreux outils d'édition asynchrone, dont les wikis, s'inspirent de cette approche. Enfin, une grande proportion des outils fonctionnant sur le web sont également des outils de ce type ; c'est notamment le cas des espaces de travail en ligne, type BSCW, ou des forges logicielles (sourceforge, Gforge, LibreSource). C'est aussi le cas des outils du web social : marque-pages partagés, géo-référencement, wikis et blogs.

1.1.2 Édition collaborative asynchrone et Wikis

L'édition collaborative est l'activité de groupe consistant à créer et modifier un ou plusieurs documents écrits de manière collective. Ces documents peuvent être de toutes sortes : textes éventuellement formatés, documents applicatifs, figures et dessins, programmes etc ...Lorsqu'elle est supportée par une infrastructure informatique, cette activité se base sur des outils logiciels qui peuvent être des outils génériques (échange par courrier électronique ou *via* un serveur de fichier) ou des outils plus dédiés (gestion de versions, gestion de contenu, wikis).

De manière générale on désigne sous le terme d'*éditeur collaboratif* un outil explicitement conçu pour supporter une activité collaborative de création de document, et incluant toutes les fonctionnalités nécessaires, de l'interface utilisateur au stockage des documents. Il faut noter que ceci peut résulter de l'assemblage d'un éditeur non collaboratif avec un outil conçu pour le partage collaboratif de documents. L'intérêt de tels outils, par rapport à une approche à base d'échange de fichiers, est qu'ils intègrent des fonctionnalités spécifiques de la collaboration : mécanismes de coordination, de gestion de la concurrence, conscience de groupe, communication, gestion des conflits ...

Il existe aujourd'hui un grand nombre d'outils d'édition collaborative, pour le travail synchrone ou asynchrone. Un important travail de recherche dans le domaine a été mené

par la communauté CSCW depuis le milieu des années 80 et a vu l'apparition de nombreux outils issus des laboratoires comme Quilt [18], PREP [48], Grove [15], rIbis[57], DistEdit [36], GroupKit [60] ou Alliance/grif [9]. Les outils actuels, largement diffusés et accessibles par tous comme Gobby [23], SynchroEdit [70], mobWrite [19] ou googleDocs [24] se sont largement inspirés de ces travaux et en reprennent l'essentiel des principes.

Il s'agit pour la plupart d'outils de travail synchrone : un groupe d'utilisateurs réalise des tâches d'édition de manière simultanée, les opérations d'un participant se répercutant de manière quasi-instantanée sur l'interface des autres. Ils sont en général basés sur la notion de session : un participant crée une session de travail que les autres rejoignent et qui se termine lorsque le créateur de l'application quitte la session. Ils sont bien adaptés au travail de petits groupes pour de courtes durées.

Parallèlement se sont développés des outils pour le travail asynchrone. Entre autre, la communauté du génie logiciel a vu se développer et se populariser des outils de gestion des révisions d'un fichier de code source, comme RCS [72] puis de gestion de version et de configuration comme CVS [5] et ses successeurs. De par leur nature asynchrone, ils sont mieux adaptés au travail à long terme puisqu'il n'est pas nécessaire de maintenir une session connectée ; ils sont également mieux adaptés au travail en grands groupes : il n'est pas nécessaire d'être tous connectés simultanément ; enfin, ils sont mieux adaptés aux applications réclamant de pouvoir travailler par périodes isolées dans un environnement stable - tester un code source, compiler un document latex...

Ces outils, comme beaucoup d'autres maintenant, fonctionnent sur un principe commun. Chaque participant dispose d'un espace de travail privé créé par copie de l'espace public. Dans cet espace, il est seul à agir et il travaille donc en isolation au sein d'un environnement stable. Périodiquement, il publie ses modifications au sein de l'espace public afin de les mettre à disposition des autres participants. Ceux-ci peuvent alors y accéder et les intégrer à leur espace privé. La gestion de l'espace public est organisé de manière à pouvoir tracer les modifications apportées successivement par chaque utilisateur. De plus, ces outils proposent très souvent des fonctionnalités d'assistance à l'intégration de modifications publiques dans l'espace privé, principalement sous la forme de mécanismes de fusion de données.

Parallèlement, et en se basant sur une approche très voisine de ces derniers outils, se sont développés les Wikis. Le premier wiki, appelé Wikiwikiweb [80] a été créé en 1995 par Ward Cunningham pour gérer simplement un site web, le Portland Pattern Repository [54]. Depuis, les wikis sont devenus l'outil d'édition collaborative le plus populaire et de très nombreux moteurs de wiki sont disponibles (voir par exemple [77] pour une liste très complète). En particulier, les wikis ont rendu possible l'édition collaborative *de*

masse, dont l'encyclopédie ouverte Wikipedia [78], l'un des sites web les plus fréquentés au monde, est un des résultats les plus probants.

Un moteur wiki est essentiellement un éditeur collaboratif asynchrone sur le web. Les documents créés et manipulés sont des hypertextes visualisables selon les principes et les protocoles du web. Un wiki est donc un site web dont le contenu est éditable par ses visiteurs. Un wiki offre 3 fonctionnalités essentielles à ses visiteurs : la visualisation des pages créées (appelées aussi *pages wiki*), l'édition et la création de pages wiki, la gestion d'un historique des modifications apportées successivement à chaque page du wiki.



FIG. 1.3 – Visualisation d'une page wiki (extrait de Wikipedia en français)

La visualisation du wiki repose sur les principes classiques de navigation sur le web (voir figure 1.3). Tous les internautes disposant des droits d'accès au site peuvent visualiser et parcourir les pages du wiki en utilisant un navigateur web.

L'édition de page wiki est accessible à tous les visiteurs en cliquant sur un bouton dédié ("modifier" ou "éditer" selon les wikis). L'édition et la mise en page se fait en utilisant une syntaxe simplifiée dite "syntaxe wiki" qui rend non nécessaire la maîtrise d'html par le visiteur, comme illustré dans la figure 1.4. Lorsque l'édition est terminée, le visiteur peut enregistrer ses modifications. La sauvegarde d'une édition rend le nouveau contenu immédiatement disponible pour tous les visiteurs. Cet enregistrement n'écrase pas l'ancien contenu de la page éditée, mais crée une nouvelle version de la page. l'historique

complet de chaque page est ainsi conservé par le moteur wiki, mais lors de la navigation sur le site seules les versions les plus récentes de chaque page sont parcourues.

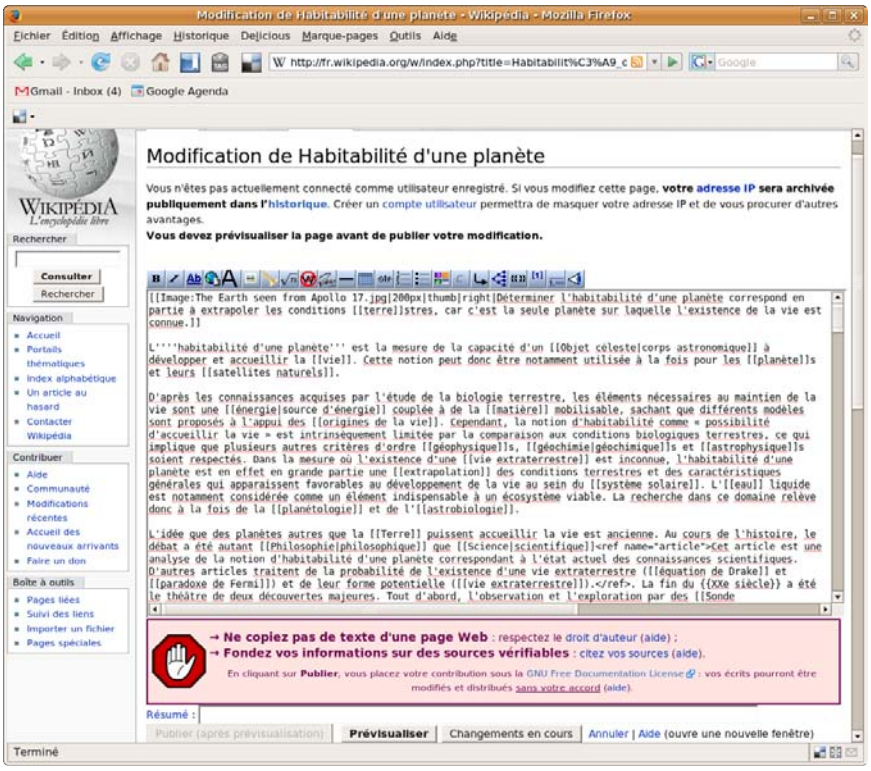

FIG. 1.4 – Édition d'une page wiki (extrait de Wikipedia en fraînçais)

L'historique des modifications est visualisable par les visiteurs, comme dans la figure 1.5. Il offre aux visiteurs la possibilité de visualiser une trace complète des versions successives de la page ainsi que le contenu individuel de chacune des versions. Ceci leur permet comprendre l'évolution du contenu d'une page dans le temps, et éventuellement, de revenir à une version antérieure. En général, il est également possible de visualiser les différences entre deux versions quelconques de l'historique.

D'autres fonctionnalités sont très souvent disponibles dans les wikis. En particulier, la gestion des utilisateurs et des contrôles d'accès est une fonctionnalité présente dans quasiment tous les moteurs de wiki. Elle permet de restreindre l'accès à certaines pages, notamment en édition, aux seuls visiteurs identifiés et autorisés. Des fonctions de recherche et de listage de l'ensemble de pages sont également très courantes. Enfin, de nombreux moteurs de wiki sont extensibles grâce à un mécanisme de "plugin".

D'un point de vue technique, les moteurs wiki actuels sont basés sur l'architecture web classique. Un serveur détient la logique applicative et l'ensemble des données (les pages

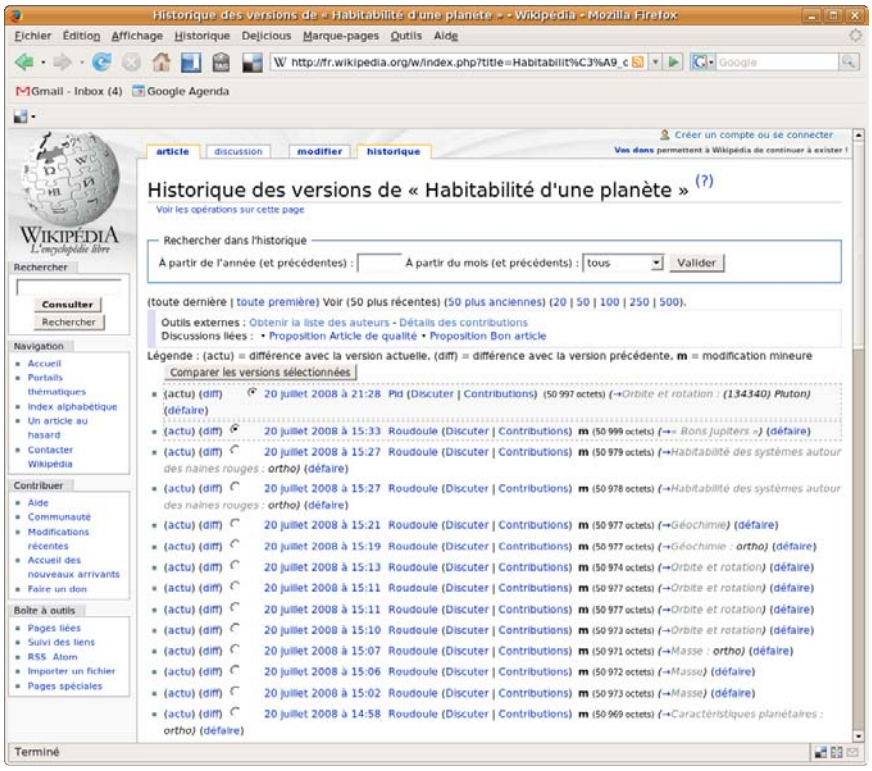

FIG. 1.5 – Historique d'une page wiki (extrait de Wikipedia,fr)

wiki). Les visiteurs interagissent avec ce serveur en utilisant un client web. Le protocole d'échange client/serveur est le couple HTTP/HTML.

Lorsqu'un client demande à visualiser une page, le serveur extrait le contenu de la page de son système de stockage (un SGBD ou un système de fichier). Ce contenu, un texte en syntaxe wiki, est ensuite transmis à un composant de rendu qui va générer le code HTML correspondant. Le résultat est renvoyé au client.

Lorqu'un client demande l'édition d'une page, le contenu de la page est extrait du système de stockage et envoyé directement au client. Ce contenu est affiché par le client dans une zone d'édition html ("HTML AREA") pour permettre sa modification par l'utilisateur. Aucune interaction entre le client et le serveur n'a lieu durant cette période de modification par l'utilisateur.

Lorsqu'un client demande la sauvegarde d'une page modifiée, le nouveau contenu est transféré du client vers le serveur. Une nouvelle version de la page est créée à partir de ce contenu et insérée dans le système de stockage du serveur.

Ce mode de fonctionnement est asynchrone et pas très éloigné des outils décrits précédemment. Pour éditer une page, l'utilisateur dispose d'une copie stockée dans son navigateur. Cette copie est stable durant toute la session d'édition puisqu'aucune interaction n'a

lieu entre la demande d'édition et la demande de sauvegarde. Lorsqu'il le décide, l'utilisateur rend public ses modifications en transférant sa copie privée vers le serveur qui sert de dépôt commun. Les différents contributeurs du wiki peuvent travailler en temps différé sans interagir autrement que *via* le stockage des pages sur le serveur. Dans certains cas, des éditions simultanées d'une même page peuvent se produire. La gestion de ces éditions concurrentes sera discutée dans la section 1.3.

Cette architecture client/serveur traditionnelle des applications web atteint aujourd'hui ses limites. Le succès grandissant des wikis de communautés s'accompagne d'une croissance forte de la taille et de la complexité des données gérées, du nombre d'utilisateurs et donc du nombre de modifications apportées à chaque page. Ainsi, des problèmes de résistance à la montée en charge, de passage à la grande échelle et de tolérance aux pannes se posent pour les serveurs. Ce problème est renforcé par l'arrivée de wikis dits "sémantiques", c'est-à-dire permettant l'annotation des contenus (pages,liens) avec des étiquettes sémantiques formelles permettant le raisonnement et l'écriture de requêtes intelligentes.

Les techniques classiques du web, à base de grappes de serveurs et d'équilibrage de charge, ne résolvent pas les problèmes de fiabilité et restent onéreuses et assez difficiles à mettre en oeuvre. Surtout, elles sont difficiles à partager entre plusieurs organisations. Par exemple, Wikipedia est hébergé sur les machines de la Wikipedia foundation qui doit faire face, seule, à la forte croissance de l'encyclopédie.

Par ailleurs, la nécessité de pouvoir interagir avec le serveur pour réaliser les tâches d'édition rend très problématique l'utilisation des wikis dans un contexte mobile ou nomade dans lequel les déconnexions sont courantes. Impossible, par exemple, d'éditer un document wiki de manière sûre au cours d'un déplacement en dehors d'une connexion internet.

1.1.3 Vers des wikis pair-à-pairs

Afin de dépasser les limites de l'architecture centralisée traditionnelle des applications web, l'idée de construire des wikis fonctionnant en mode pair-à-pair est apparue récemment [46, 76], notamment dans le projet ECOO du LORIA au travers du système Wooki (qui sera décrit dans le chapitre 2). L'objectif est de construire un wiki fonctionnant de manière entièrement décentralisée sur un réseau de type P2P et répliquant les pages wiki sur les différents noeuds du réseau.

On peut envisager différentes approches pour construire un tel système. L'approche choisie dans notre équipe, illustrée en figure 1.6 est la suivante. Un wiki P2P est constitué d'un ensemble de serveurs wikis interconnectés dans lequel chaque serveur joue un rôle

identique. Les pages wiki sont dupliquées totalement sur chaque serveur appartenant au réseau. Ainsi, chaque serveur dispose des données et de la logique pour rendre l'ensemble du service wiki (consultation, édition, historique) de façon autonome. Bien entendu, tous les serveurs peuvent être utilisés, de façon indifférenciée, pour consulter ou éditer les pages : une modification apportée à une page sur un serveur est propagée aux autres serveurs grâce à un mécanisme de réplication. Cette caractéristique permet d'utiliser un serveur même dans le cas où il est déconnecté temporairement du réseau. La composition du réseau n'est pas connue à priori et est dynamique : un noeud peut joindre ou quitter le réseau de pairs à tout moment.

Un tel système pose évidemment des problèmes importants de conception, notamment liés au maintien de la cohérence des données, mais offre des avantages par rapport à l'architecture centralisée :

Robustesse et tolérance aux pannes : grâce à la réplication massive des données sur un ensemble de serveurs, le système est beaucoup plus robuste. Un serveur en panne, ou un problème réseau empêchant l'accès à un serveur n'empêche pas les autres serveurs de fonctionner et d'être utilisés par les clients.

Montée en charge, passage à l'échelle : en multipliant les serveurs, le système peut supporter simplement de fortes montées en charge lui permettant de passer à une grande échelle en nombre d'utilisateurs. L'aspect dynamique du système permet d'introduire de nouveaux noeuds à tout moment.

Infrastructure partagée : le système étant constitué d'un ensemble de serveurs interconnectés, l'infrastructure est simple à partager entre plusieurs sites ou organisations, en terme d'hébergement et d'administration. On peut envisager, pour un wiki de grande échelle, que plusieurs organisations coopèrent en fournissant chacune un ou plusieurs serveurs qu'elles administrent.

Confiance accrue : une infrastructure partagée et répliquant les données sur de multiples serveurs évite aux utilisateurs de confier leurs données à une organisation unique et donc d'éviter les risques de censure ou d'appropriation des données déposées ou d'arrêt du service rendant les données inaccessibles.

Support du travail nomade : la duplication des données sur différents serveurs et la possibilité d'utiliser un serveur même s'il est déconnecté temporairement du réseau permet d'envisager des usages nomades. Un utilisateur peut créer une copie (éventuellement partielle) du wiki sur un dispositif personnel (un ordinateur portable, voire un PDA), et consulter/éditer le wiki en étant déconnecté, puisque les données existent localement.

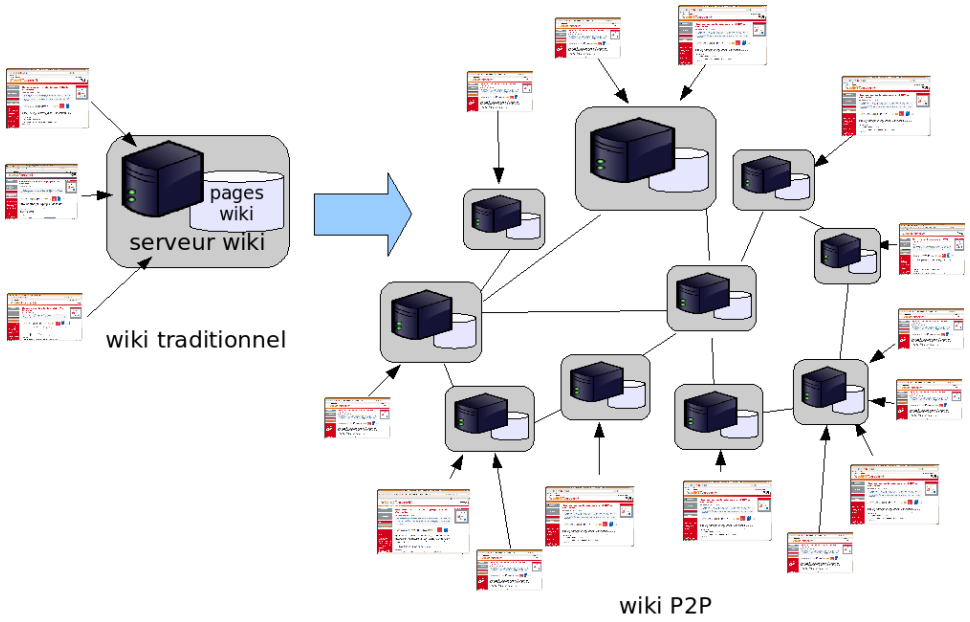

FIG. 1.6 – Vers des wikis pair-à-pairs

Pour construire un wiki pair-à-pair de ce type, le premier problème à résoudre est celui du maintien de la cohérence des copies des pages. Le critère classique utilisé dans les SGBDs répartis ou certaines applications distribuées est connu sous le nom de cohérence forte (ou atomique). Selon ce critère, le système est correct si les copies sont toutes identiques après chaque opération de modification. Ce critère est trop difficile à atteindre dans un système P2P de grande taille et dynamique. Le critère utilisé se base une version plus faible de la cohérence : la cohérence à terme. Un wiki P2P est dit correct si :

1. lorsque le système est au repos, c'est-à-dire lorsque plus aucune modification n'est apportée, toutes les copies de toutes les pages convergent vers un état identique dans un temps fini [61, 41] ;
2. les intentions d'édition des utilisateurs sont respectées dans l'état de convergence [68].

Le premier point est relativement classique. Il implique que les copies d'une page peuvent être divergentes de manière temporaire à condition que le système garantisse leur convergence à terme lorsque plus aucune modification n'est introduite. Le second point doit être précisé. Il est issu de la communauté de l'édition collaborative et fait débat car il est difficile à formaliser. Il est introduit pour imposer que l'état final de convergence ne soit pas quelconque, mais un état dans lequel les différents utilisateurs retrouvent leurs opérations d'édition avec un résultat correspondant à leur intention initiale.

Du point de vue de la mise en oeuvre, un tel critère peut être maintenu par un protocole de réplication optimiste. Le principe de fonctionnement d'un tel protocole est le suivant :

- Lorsqu'une opération d'édition locale est reçue par un serveur, elle est appliquée immédiatement sur la copie locale, puis diffusée aux autres serveurs du réseau. Les copies sont alors temporairement divergentes.
- Lorsqu'une opération d'édition distante est reçue par un serveur, elle est intégrée à la copie locale en la fusionnant si nécessaire aux modifications concurrentes locales ou distantes déjà reçues.

Ce mode de fonctionnement fait apparaître deux composants pour assurer la réplication : un algorithme de diffusion des opérations d'édition vers le réseau de pairs et un algorithme de fusion pour l'intégration des modifications concurrentes. Pour assurer le critère de cohérence présenté ci-dessus, ces composants doivent posséder les propriétés suivantes :

- le mécanisme de diffusion des opérations d'édition doit garantir la livraison à tous les sites du réseau, y compris les sites temporairement déconnectés. Il est clair qu'un site n'ayant pas reçu toutes les opérations d'édition ne peut pas converger vers un état correct.
- le mécanisme de fusion doit être déterministe, et si possible commutatif et associatif.
- le mécanisme de fusion doit garantir le respect des intentions des utilisateurs [68] dans l'état produit.

Il faut noter que l'état de convergence est construit par le mécanisme de fusion/intégration. C'est donc sur lui que reposent les plus fortes contraintes. Le déterminisme est nécessaire pour assurer la convergence. En effet, deux opérations d'édition concurrentes émises sur deux sites différents seront fusionnées sur chacun de ces deux sites, et également sur l'ensemble des sites du réseau. Pour garantir la convergence, le résultat de la fusion doit impérativement être identique sur chacun des sites. Il est très important de noter au sujet du déterminisme que cette contrainte rend caduques toutes les approches de fusion dans lesquelles l'utilisateur est associé, notamment pour effectuer des choix ou résoudre des conflits que l'algorithme ne sait pas traiter. En effet, Il serait impossible dans une telle approche de garantir que différents utilisateurs sur différents sites prendront des décisions identiques et ceci rendrait la convergence très hypothétique. La commutativité et l'associativité sont utiles pour prendre en compte le fait que l'ordre de livraison des opérations d'édition n'est pas forcément le même sur tous les sites. En l'absence de cette propriété, la contrainte est reportée sur l'algorithme de diffusion.

Plus de détails sur la construction d'un tel système seront donnés dans le chapitre 2 au travers de la description d'un système réel : le moteur de wiki Wooki, un prototype de moteur P2P pour des wikis sémantiques construit dans le projet ECOO.

1.2 Le Problème de la concurrence dans un Wiki P2P

1.2.1 Opérations concurrentes dans un Wiki P2P

Comme pour toutes les applications multi-utilisateurs, les éditeurs collaboratifs doivent traiter le problème de la concurrence dans les opérations de lecture/écriture du document partagé. Deux opérations sont concurrentes si elles sont émises simultanément d'un point de vue logique, c'est-à-dire aucune des deux ne précède l'autre.

Les problèmes dus à la concurrence d'opérations dans l'édition collaborative sont essentiellement liés à des pertes de mises à jour : l'effet d'une opération d'édition est perdu suite à l'exécution d'une opération concurrente qui vient le remplacer (l'écraser). Ce problème est très fréquent : si plusieurs utilisateurs modifient simultanément le même document sans aucune coordination, le risque est très grand que les modifications apportées par l'un soient perdues lors d'une sauvegarde effectuée par un autre. Ce problème se pose dans l'édition synchrone ou asynchrone.

Plusieurs techniques ont été proposées pour apporter des solutions à ce problème, que l'on peut classer en deux grandes catégories :

les approches exclusives : dont le principe est de garantir un accès exclusif au document à un seul utilisateur à la fois. On trouve ici des techniques dites de "floor control" ou "turn taking" [25] qui consistent à coordonner les accès dans un tour de rôle, souvent géré à l'aide d'un jeton : seul le détenteur du jeton peut entreprendre des opérations de modification du document. On trouve également des techniques à base de verrouillage associées très souvent à des techniques de partitionnement des objets partagés, comme par exemple dans ShrEdit [40] ou DistView [55]. Un utilisateur souhaitant éditer une partie d'un document demande la pose d'un verrou sur la partition correspondante. Si un verrou est déjà présent, il est mis en attente, sinon il obtient l'accès.

les approches permissives : dont le principe est d'autoriser les modifications simultanées et de les fusionner à l'aide d'un algorithme dédié [58, 66, 67]. Cette fusion peut se faire en continu, c'est-à-dire au fur et à mesure de la production d'opérations dans un système synchrone [13], ou de manière différée dans un système asynchrone comme, par exemple, un outil de gestion de versions.

Dans un wiki P2P basé sur un protocole de réplication optimiste, c'est le deuxième type d'approche qui est utilisé, comme nous l'avons déjà rapidement introduit dans la section 1.1.3. Ainsi, dans un wiki P2P, la gestion des opérations concurrentes repose sur le mécanisme de réplication optimiste en charge de diffuser une opération introduite sur

un site vers les autres sites, puis d'intégrer les opérations distantes en les fusionnant si nécessaire avec les opérations concurrentes déjà intégrées.

Il est important de souligner une caractéristique essentielle des wiki P2P du point de vue de la gestion de la concurrence. Dans ce type de système, la fusion d'opérations concurrentes se produit :

- sur tous les sites participants au réseau P2P, et non pas seulement sur les sites producteurs des opérations,
- à chaque réception d'une opération, au moment où elle est reçue par un serveur,
- elle est donc déclenchée et exécutée par le serveur, en dehors du contrôle des utilisateurs,
- elle produit son résultat sur le serveur et ce résultat est public dans le sens où il est immédiatement visible par les visiteurs du wiki.

Ce comportement est assez différent de celui des systèmes asynchrones classiques. Dans ces systèmes, les fusions sont déclenchées à l'initiative des utilisateurs, afin de synchroniser leur espace privé avec de nouvelles valeurs produites dans l'espace public. Cette fusion est réalisée sous leur contrôle, et produisent des résultats dans l'espace privé des utilisateurs.

Ce comportement particulier a des conséquences majeures :

1. au moment de l'enregistrement d'une modification, un utilisateur n'a aucun moyen de connaître l'état final réel du document ; la connaissance d'opérations concurrentes distantes étant soumise à un délai de propagation, il est possible que des fusions soient réalisées ultérieurement sans qu'il en ait connaissance,

2. ces fusions sont réalisées automatiquement par les serveurs, sans contrôle de la part d'utilisateurs humains. L'état du document après ces fusions n'a donc pas été validé par un auteur, mais résulte d'un calcul. Les algorithmes de fusion, malgré leur efficacité et les propriétés qu'ils peuvent garantir, sont très loin de pouvoir prendre en compte des aspects sémantiques dans tous les cas,

3. ce document dont le contenu résulte d'une fusion automatique est cependant public : il est accessible par tous les visiteurs autorisés du site et pas seulement à ses auteurs.

Ces conséquences sont illustrées dans un petit scénario présenté dans la figure 1.7. Dans ce scénario, une page contenant un texte est répliquée sur deux sites (éventuellement plus). Le contenu de cette page est modifié par 2 utilisateurs opérant chacun sur l'un des 2 sites.

On suppose que les deux utilisateurs agissent sans coordination, chacun réalisant la modification qui lui semble appropriée. L'un remplace donc le mot *serpent* par *merle*, alors que le second, de manière concurrente, remplace le mot *oiseau* par *reptile*. Chacun enregistre son changement, qui est appliqué localement sur chacun des sites immédiate-

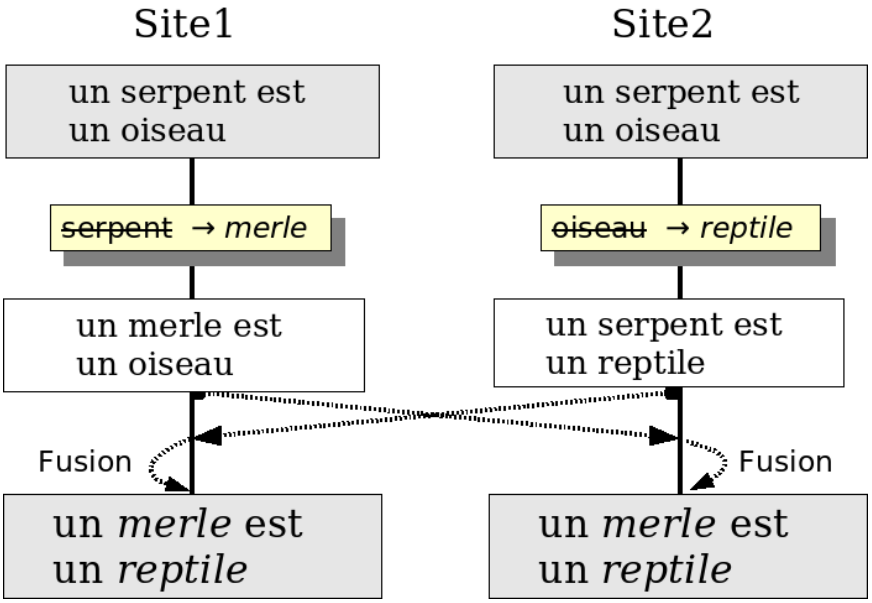

FIG. 1.7 – Opérations concurrentes dans un wiki P2P

ment. Chacun des deux utilisateurs observe à ce moment là un contenu qui lui semble cohérent et correspondant à ses intentions, mais n'a aucune conscience du changement réalisé par l'autre utilisateur.

Ensuite, le moteur de réplication fait son travail : les deux changements sont diffusés sur les réseaux, reçus par les différents sites, dont les sites émetteurs, et fusionnés pour produire un nouveau contenu tenant compte des deux modifications. Rappelons que cette fusion est opérée sur tous les sites, de manière automatique et sans contrôle de la part des utilisateurs. Cette fusion produit une nouvelle valeur : un texte dans lequel *serpent* est remplacé par *merle* **et** *oiseau* par *reptile*. Il faut remarquer que ce contenu respecte parfaitement les conditions de convergence (le même état est obtenu sur les différents sites) et de préservation des intentions des utilisateurs (les deux modifications sont intégrées). Ce contenu ne satisferait probablement aucun des deux auteurs, et pourtant c'est le contenu accessible par tous les visiteurs accédant au wiki sur l'un des sites du réseau. Remarquons que ce problème se produit sur tous les sites du réseau, qu'ils soient émetteurs d'une modification ou non. De plus, les auteurs des modifications ne sont avertis du problème que dans le cas où, après avoir sauvegardé une modification, ils font un nouvel accès à la page et en examine le contenu avec attention. Enfin, soulignons que le problème n'est pas dû à l'algorithme de fusion : de nombreux algorithmes fusionneront les modifications de l'exemple sans détecter de conflits (les modifications ont lieu sur des lignes différentes). Le problème est lié au fait que la fusion est réalisée dans l'espace public et en dehors du contrôle des utilisateurs.

1.2.2 Rendre les utilisateurs conscients de la concurrence

Pour remédier à ce problème, nous proposons une approche basée sur la *conscience de groupe*.

Notre proposition consiste à étendre le moteur de wiki avec un mécanisme rendant les visiteurs et les auteurs du wiki *conscients* du statut d'une page vis-à-vis de la concurrence. Nous appellerons un tel mécanisme un *mécanisme de conscience de la concurrence*.

Ce mécanisme doit permettre d'une part aux visiteurs d'un wiki d'être informés du fait qu'une page a été produite par un algorithme de fusion sans avoir été relue par son ou ses auteurs. D'autre part, il doit aider les auteurs d'une page à résoudre les éventuels problèmes de consistance de son contenu qui seraient dus à la concurrence. Cela passe par la compréhension de l'état courant de la page ainsi que de son historique.

Plus concrètement, ce mécanisme doit permettre aux visiteurs et auteurs d'un wiki de répondre au questions suivantes :

1. La page visualisée a-t-elle été produite par un auteur humain, ou bien a-t-elle été produite par l'algorithme de fusion déclenché par le serveur ? En d'autres termes, le contenu visualisé a-il-été validé par un auteur humain ?

2. Dans le cas où la page visualisée résulte d'une fusion, quelles régions de la page sont concernées par cette fusion ? Quel est l'impact de cette fusion sur le contenu observé ?

3. Dans l'historique d'une page, quelles versions sont le résultat d'une fusion et quelles versions sont le résultat d'une édition par un auteur ? Quelles modifications concurrentes ont produit une version fusionnée ?

Par ailleurs, il est important que le mécanisme que nous allons proposer fonctionne dans le contexte prévu pour un wiki P2P et ne remette pas en cause ses propriétés. Cela signifie concrètement que ce mécanisme doit fonctionner dans un cadre P2P totalement décentralisé, éventuellement à grande échelle avec des noeuds pouvant joindre et quitter le réseau à tout moment. De plus, le mécanisme de conscience de concurrence doit être sans effet sur la correction du système : la convergence à terme et le respect des intentions doivent être conservés.

Nous reviendrons plus en détail sur les objectifs de notre travail dans la section 1.4. Dans la section suivante, nous présentons rapidement les approches existantes en matière de concurrence et de conscience de la concurrence dans les systèmes existants.

1.3 Concurrence et conscience de groupe dans les systèmes collaboratifs

Nous présentons dans cette section les principales approches concernant les mécanismes de conscience à la concurrence, ou des mécanismes voisins, dans les systèmes existants. Avant d'aborder différentes classes de système, nous faisons une rapide présentation de la notion de *conscience de groupe*.

1.3.1 Conscience de groupe

La conscience de groupe (en anglais, *group awareness*) est l'un des thèmes phares des travaux sur les systèmes collaboratifs et a émergé très tôt dans le domaine du TCAO. Cette notion désigne le phénomène suivant. Lorsque des personnes travaillent ensemble de manière simultanée, le groupe crée un espace social d'information permettant l'émergence d'une conscience du groupe et la coordination des activités individuelles, notamment les activités de production et de communication. Lorsque la collaboration est médiatisée au travers d'un système logiciel, cette information doit être fournie par le système à ses utilisateurs. On désigne les mécanismes en charge de cette information sous le terme de *mécanismes de conscience de groupe*. Plusieurs définitions ont été proposées pour la notion de conscience de groupe, la plus reconnue étant celle donnée par P. Dourish et V. Bellotti dans un papier à CSCW'92 [11] : *la conscience de groupe permet la compréhension de l'activité des autres, ce qui fournit un contexte pour sa propre activité. Ce contexte permet d'assurer que les actions individuelles sont situées dans l'activité globale du groupe [...] Cette information permet au groupe de gérer le processus de collaboration* [1]. Ces mécanismes de conscience de groupe améliorent la synergie de groupe dans une collaboration médiatisée[29] et permettent la coordination implicite (sans règles explicitement gérées par le système) entre les acteurs de la coopération.

Cette problématique de conception de mécanismes de conscience de groupe, tant du point de vue système que du point de vue interaction homme-machine, était présente dès les premiers travaux et systèmes relevant du travail collaboratif, par exemple dans QUILT [18], PREP [47], ou GROVE [14]. Elle est devenue une des problématiques principales du domaine du TCAO et un grand nombre d'approches ont été explorées. On peut souligner notamment, parmi beaucoup d'autres, les travaux autour des *mediaspaces* [12],

[1] Awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole [...] The information, then, allows the group to manage the process of collaborative work.([11])

les travaux concernant des *widgets collaboratifs* [28], ou des travaux relatifs à des modèles [59].

Selon le type d'information considéré, les différentes approches pour la conscience de groupe sont habituellement classées selon différentes dimensions :

- Conscience des personnes et de la présence (*Presence Awareness* [20]) : permet d'avoir connaissance des membres du groupe et de leur présence à un moment donné. Ces mécanismes ont été fortement popularisés par les systèmes de messagerie instantanée à travers la notion de *buddy list*.
- Conscience des activités (*Activity Awareness* [49]) : permet d'avoir conscience des activités en cours dans l'espace de collaboration,
- Conscience sociale (*Social awareness* [73]) : permet d'avoir conscience de la structure sociale du groupe et des relations entre ses membres,
- Conscience de l'espace de travail (*Workspace Awareness* [30]) : permet d'avoir conscience de l'état de l'espace de travail partagé et de son évolution.

Ces catégories sont bien entendues complémentaires et peuvent être vues comme différentes facettes de la conscience de groupe. Notre proposition d'un mécanisme de conscience de la concurrence se place dans cette liste comme une sous-classe de la conscience de l'espace de travail, visant à faire comprendre l'état de l'espace de travail partagé lorsqu'il résulte d'actions concurrentes.

1.3.2 Concurrence et conscience de groupe dans les systèmes synchrones

Les systèmes synchrones, et notamment les éditeurs ont, dès le début, intégré des mécanismes de gestion de la concurrence et de conscience de groupe.

En particulier, la concurrence a d'abord été traitée par des approches exclusives, dont l'effet est en réalité de la prévenir en n'autorisant aucune modification simultanée d'une même région d'un document. Les mécanismes associés de conscience de l'espace de travail était alors dédiés principalement à identifier les auteurs des différentes modifications.

Dans les éditeurs plus récents autorisant des modifications simultanées, la concurrence est gérée selon plusieurs stratégies. Les éditeurs basés sur une architecture client-serveur gèrent la concurrence en sérialisant les requêtes sur le serveur [27]. Parallèlement, de nombreux travaux ont concerné les éditeurs de groupe synchrones basés sur une architecture décentralisée utilisant la réplication. Autour de ce problème, on a vu apparaître une famille d'approches basée sur la transformation d'opérations (OT), qui fait encore aujourd'hui l'objet de nombreux travaux [13, 68, 69, 50]. L'idée centrale est que chaque

site applique immédiatement ses opérations locales puis les diffuse au groupe. A la réception, une opération distante subit une transformation (d'où le nom de l'approche) visant à tenir compte de l'état local de la copie du document.

Du point de vue de la conscience de groupe, un grand nombre de travaux ont été réalisés dans ce contexte, relevant essentiellement de la conscience de l'espace de travail. On peut citer par exemple les télé-pointeurs, les vues radar ou les barres de défilement multi-utilisateurs [26, 34, 33]. Ces approches ont pour objet essentiel les actions des différents utilisateurs : qui agit sur quelle partie du document à un moment donné ?

Par rapport à notre problématique, ces approches sont assez différentes, principalement en raison du contexte synchrone. En effet, un éditeur synchrone est basé sur l'idée de *wysiwis* [64] : What You See Is What I See. En d'autres termes, tous les participants de la session collaborative observent l'état du document en temps réel et de manière immédiate. La fusion des modifications est réalisée au fur et à mesure de la diffusion des opérations et le résultat de cette fusion est immédiatement visible. Les utilisateurs étant présents simultanément, un problème lié à des actions concurrentes est donc immédiatement perçus par les auteurs et peut être corrigé en temps réel. Ainsi, dans un scénario tel que celui présenté dans la section 1.2, le problème de la modification simultanée de deux parties sémantiquement liées du document serait immédiatement détecté et corrigé.

1.3.3 Cas des wikis

Dans les wikis actuels, la concurrence est gérée au niveau de la page avec deux approches différentes :

- Certains moteurs de wiki, comme par exemple XWiki [81], gèrent le problème de la concurrence avec un mécanisme de verrouillage. La concurrence est détectée et prévenue au moment où un utilisateur demande l'*édition* d'une page.
- D'autres moteurs, et notamment MediaWiki, le moteur de Wikipedia, ont une approche plus optimiste. La concurrence est détectée lorsqu'un utilisateur demande la sauvegarde d'une modification. La concurrence est détectée en comparant les versions de la page éditée : si la version courante est différente de la version éditée, alors une modification concurrente a eu lieu et a été sauvegardée.

Du point de vue de la conscience de la concurrence, les wikis utilisant un verrouillage des pages en cours d'édition se contentent d'afficher un message lorsqu'une page verrouillée est demandée pour être éditée. L'utilisateur demandeur est ainsi averti qu'un autre utilisateur est en train de modifier la page.

Dans Mediawiki, ainsi que les wikis analogues du point de vue la concurrence, lorsqu'une édition concurrente est détectée lors de la sauvegarde d'une page, la fusion des 2 versions - la version courante stockée sur le serveur et la nouvelle version en cours de sauvegarde par l'utilisateur - est demandée à l'utilisateur. Cependant, dans Mediawiki,

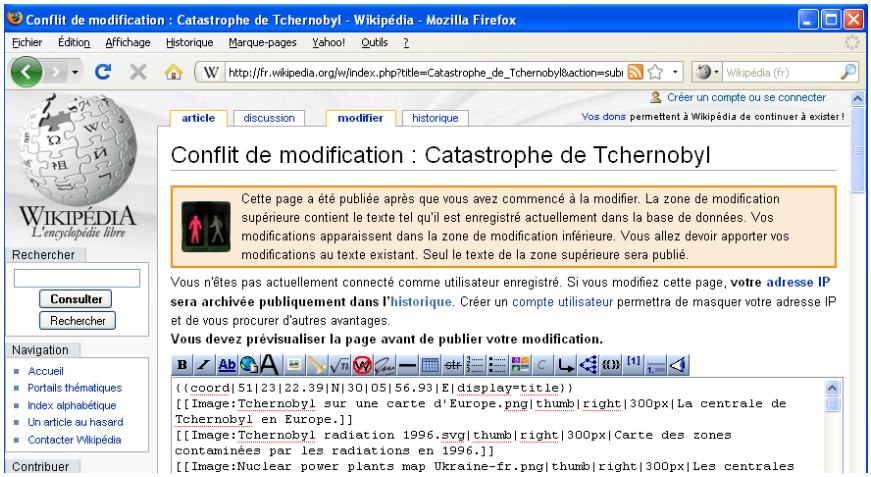

FIG. 1.8 – Fusion d'éditions concurrentes dans Mediawiki

c'est à l'utilisateur que revient la charge de réaliser cette fusion. Le système se contente d'afficher les deux contenus et propose une fenêtre d'édition pour construire le contenu fusionné, comme illustré dans la figure 1.8. La fusion est donc réalisée au cours de l'opération de sauvegarde, sous le contrôle entier de l'utilisateur. Les informations de conscience de la concurrence sont très limités : l'utilisateur peut visualiser les deux contenus et la différence entre les deux versions est repérée comme dans figure 1.9.

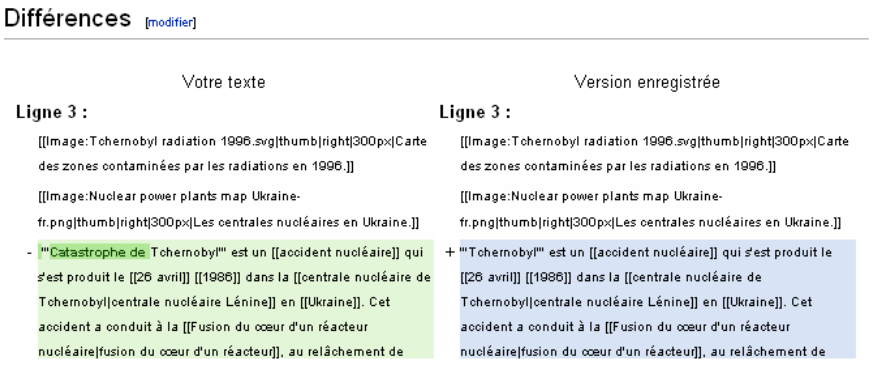

FIG. 1.9 – Fusion d'éditions concurrentes dans Mediawiki

L'historique des modifications apportées à une page fournit également des informations permettant aux utilisateurs de comprendre le contexte collaboratif. En effet, l'historique fournit des informations sur le chemin parcouru pour obtenir l'état courant de la page

visualisée, et précise quels sont les utilisateurs responsables des différentes modifications apportées. En cela, ce mécanisme relève de la *conscience du passé*. Cependant, dans les wikis traditionnels, cet historique est toujours linéaire : il montre toujours une séquence linéaire de versions. Un utilisateur visualisant cet historique n'a aucun moyen de comprendre que certaines versions résultent de la fusion de modifications concurrentes. En effet, ces versions devraient avoir au moins deux prédécesseurs : la version éditée, et la nouvelle version sauvegardée par ailleurs au cours de l'édition. L'aspect purement linéaire de l'historique dans un wiki ne permet pas de retranscrire cette information.

1.3.4 Conscience de groupe pour le travail coopératif asynchrone

Plusieurs efforts concernant la conscience de groupe en milieu asynchrone sont à noter.

La notion de *conscience des changements* (change awareness) a été formalisée par Tam et Greenberg dans [71]. Ils définissent cette notion comme *la capacité des individus à suivre les changements faits par d'autres participants, au cours du temps, sur un objet partagé de manière asynchrone*[2]. Ils proposent un cadre pour décrire et comparer des mécanismes réalisant la conscience des changements.

Ces mécanismes collectent des informations pour répondre aux questions classiques de la conscience de groupe [7] appliquées aux changements. Ceci conduit à structurer la conscience des changements selon six points :

1. *Où* les changements ont-ils été faits ?
2. *Qui* a fait ces changements ?
3. *Quels* changements ont été faits ?
4. *Comment* les choses ont-elles été changées ?
5. *Quand* les changements ont-ils été faits ?
6. *Pourquoi* les changements ont-ils été faits ?

Les auteurs proposent également différentes perspectives permettant de visualiser ces différents axes d'information : perspective des objets partagés, perspective des participants, perspective espace de travail. Ces perspectives différentes permettent différentes vues sur les changements, selon le point d'intérêt des utilisateurs.

L'approche de la conscience des changements est bien entendu très pertinente pour le travail asynchrone. Cependant, on remarque que la notion de changement concurrent est absente de cette analyse. Rien n'est prévu pour mettre en relation des changements apportés de manière concurrente à un document. Dans une telle approche, la fusion de deux

[2]Change awareness is the ability of individuals to track the asynchronous changes made to a collaborative document or graphical workspace by other participants over time.

modifications (ou plus) concurrentes apparaîtrait comme un changement unique apporté au document. Dans le cas d'un wiki P2P, l'auteur de ce changement serait le serveur. Par contre, un tel mécanisme ne pourrait pas distinguer, à l'intérieur d'un changement correspondant à une fusion, les différentes contributions concurrentes.

D'autres approches ont porté leurs efforts sur la mesure de la quantité de changements apportés lors d'une synchronisation et leur localisation dans les documents. Ces calculs sont réalisés lors de la mise à jour d'une copie privée avec l'espace public ("update"). Dans [45], des métriques sont proposées pour mesurer la divergence entre des fichiers partagés en cours d'édition. Cette information est utile pour prévenir les problèmes de fusions d'opérations concurrentes, mais n'est d'aucune aide pour visualiser cette fusion lorsqu'elle a eu lieu et détecter les éventuels problèmes qu'elle introduit. Dans [52, 53], les auteurs se concentrent en particulier sur la prise en compte de la structure hiérarchique des documents, et définissent des mesures cumulatives sur cette structure. L'approche permet d'offrir des visualisations adaptées aux différents niveaux de la structure d'un document, afin de localiser les zones d'un document modifiées lors d'une synchronisation. Cette approche permet en outre, lors de la visualisation du contenu d'un document, d'afficher les zones touchées par l'incorporation d'un changement distant au cours de la synchronisation. Cette approche fournit donc une conscience de la concurrence très intéressante. Son application au cas d'un wiki P2P est cependant limité. En effet, cette approche considère uniquement deux changements concurrents : le changement apporté à l'espace local, et le changement apporté à l'espace de synchronisation. Ce changement est vu comme un tout même s'il résulte de plusieurs contributions successives, et l'outil proposé se focalise sur la visualisation de ce changement, faisant l'hypothèse que l'utilisateur connaît les modifications qu'il a lui-même apportées. Dans un wiki P2P par contre, le nombre de changements concurrents n'est pas limités en nombre ou dans le temps, et il est nécessaire de visualiser tous les changements concurrents puisque les visiteurs potentiels peuvent parfaitement n'être l'auteur d'aucun d'entre eux.

1.3.5 Systèmes de gestion de versions

Les systèmes de gestion de versions ont pour objectif principal de tracer les modifications successives apportées lors du développement d'un produit, principalement un produit logiciel. Les systèmes suffisamment récents permettent également d'organiser et contrôler le développement parallèle. C'est le cas notamment des systèmes libres les plus populaires, CVS [5] et SVN [65], ou du système propriétaire le plus répandu, ClearCase [1]. On peut les qualifier en cela de systèmes pour l'édition collaborative. Ils fonctionnent en mode ex-

clusivement asynchrone en se basant sur le paradigme dit de "copier-modifier-fusionner" (*copy-modify-merge*).

Le principe de ce mode de fonctionnement est le suivant. Un utilisateur désireux de participer à un projet de développement coopératif crée d'abord une *copie* des fichiers à partir d'un dépôt contenant les versions stables de ces fichiers. Il obtient, par cette copie, un espace de travail privé dans lequel il peut apporter librement les *modifications* qu'il souhaite. Lorsqu'il veut rendre publiques ces modifications, il doit auparavant synchroniser son espace privé avec le dépôt public qui contient éventuellement de nouvelles versions des fichiers partagés produites par des utilisateurs travaillant en parallèle. Le processus de synchronisation consiste à *fusionner* la version courante du dépôt avec la version de l'espace privé. Une fois la fusion réalisée, la publication peut avoir lieu et conduit à la création de nouvelles versions courantes dans le dépôt public.

Dans ces systèmes, les fusions se produisent au moment où l'utilisateur réalise la synchronisation de son espace privé avec l'espace public. Lors de ce processus, le système indique quels fichiers sont mis à jour en précisant si une fusion a été nécessaire. A l'intérieurs des fichiers, des indications sur cette fusion ne sont disponibles que dans les cas de conflits, c'est-à-dire dans les cas où l'algorithme ne parvient pas à prendre une décision seul. Dans ces cas-là, l'algorithme insère dans le texte fusionné un *bloc de conflit* semblable à celui-ci :

```
<<<<<<< driver.c
   exit(nerr == 0 ? SUCCESS : FAILURE);
=======
   exit(!!nerr);
>>>>>>> 1.6
```

Ce bloc indique que la ligne de texte spécifiée a été modifiée concurremment, et que les changements se recouvrent. L'algorithme ne choisit pas entre les deux possibilités et demande à l'utilisateur de le faire. Dans tous les autres cas, c'est-à-dire dans tous les cas de fusion sans conflit, aucune indication n'est fournie à l'utilisateur. En particulier, le scénario de la section 1.2 ne donnerait lieu à aucune indication de la part de l'algorithme, les changements concurrents se produisant sur deux lignes distinctes et donc ne conduisant pas à un conflit.

Dans ces systèmes, et en particulier dans CVS [5], il est interdit de publier des fichiers contenant des blocs de conflits : les utilisateurs doivent impérativement résoudre ces conflits avant de créer de nouvelles versions dans le dépôt public. Il est clair que cet impératif rend l'approche inappropriée dans le cas de la réalisation d'un Wiki P2P. En

effet, nous avons expliqué plus haut la nécessité d'avoir une fusion déterministe qui ne doit donc pas faire appel à l'utilisateur. De plus, une telle approche conduirait les serveurs à attendre l'intervention d'un utilisateur avant de continuer à intégrer des modifications. Différents serveurs pourraient ainsi être bloqués sur différents conflits avec le risque de conduire le système à un blocage généralisé.

Des efforts ont été réalisés pour améliorer la conscience de la concurrence dans ces systèmes de gestion de version. C'est notamment le cas de deux approches assez voisines, les State Treemap [44] et le système Palantír [62]. Ces deux systèmes ont pour principe de permettre à l'utilisateur de visualiser l'état des fichiers de son espace privé vis-à-vis de l'espace public (localement modifié, nouvelle version, modification concurrente en cours...) et d'y ajouter une mesure de l'écart entre la version locale et la version publique. Cette mesure indique la "sévérité" de la concurrence, et permet à l'utilisateur d'anticiper des difficultés de fusion. Dans les deux cas, la granularité reste le fichier : aucune information n'est fournie par le système sur les parties d'un même document modifiées en concurrence.

Ces dernières années, des systèmes de gestion de versions distribuées sont apparus. Ils permettent la collaboration asynchrone sans serveur central gérant une copie de référence. Dans ce sens, ils sont assez proches d'un wiki P2P. La différence réside dans le fait que dans un tel système, il n'y a pas réellement d'espace public visible par tous les utilisateurs. Le système est organisé en un réseau d'espaces de travail, chacun était géré par un utilisateur. Chaque espace contient l'ensemble des versions des documents appartenant à cet espace. Les modifications introduites par un utilisateur dans son espace privé sont publiées vers un ou plusieurs autres espaces privés. Les utilisateurs responsables de ces espaces peuvent alors décider de les intégrer. Les synchronisations entre espaces privés sont donc là aussi à l'initiative et sous le contrôle des utilisateurs. Les fusions se produisent lors de ces synchronisations, dans les espaces privés. Les notifications de changements concurrents sont limitées à l'insertion de blocs de conflits dans les fichiers fusionnés contenant des modifications concurrentes non résolues. Par contre, dans certains de ces systèmes de gestion de versions distribuées, et notamment dans GIT [22], il est possible de publier des fichiers contenant des blocs de conflits. Le système fait l'hypothèse que l'utilisateurs intégrant une version contenant des blocs de conflits se chargera de résoudre ces conflits. Cette hypothèse n'est pas valide dans un wiki P2P, où la publication se fait dans un espace public où toutes les modifications sont intégrées. Il pourrait alors arriver qu'un bloc de conflit créé par un serveur pour représenter une fusion d'opérations concurrentes soit lui-même créé de façon concurrente à d'autres modifications. Cette situation conduirait à la création d'un nouveau bloc de conflit sur chaque serveur de manière concurrente. Les blocs de conflits risquent alors de croître sans fin, entraînant le système dans une boucle

lui interdisant de se stabiliser.

Malgré toutes ces considérations concernant la conscience de la concurrence très limitée dans les systèmes de gestion de version centralisés ou distribués, ces systèmes sont très utilisés et connaissent un grand succès : le manque d'informations relatives aux fusions de changements concurrents ne semble pas être un problème. Cela est dû d'une part au fait que des fusions sont réalisées sur des copies privées des documents, et d'autre part au contexte particulier, le développement de logiciel, dans lequel ils sont utilisés. Lorsqu'un utilisateur synchronise son espace avec un espace distant (espace public, ou autre espace privé), le résultat de cette synchronisation n'est visible que par lui-même. Si cette synchronisation contient des incohérences dues à des changements concurrents, il est le seul à pouvoir accéder à ces documents incohérents. D'autre part, dans le cas du développement de logiciel, cet utilisateur va ensuite réaliser un certain nombre de tâches en utilisant des outils (compilateur, analyseur de code, environnement de tests) qui eux, détecteront de manière sure les incohérences de toutes natures pouvant résider dans les documents.

La situation est bien sûr différente dans un wiki P2P. Tout d'abord, nous l'avons déjà dit, le résultat d'un fusion est immédiatement disponible dans un espace public. Les incohérences éventuelles sont donc visibles par l'ensemble des visiteurs de cet espace public. Par ailleurs, les pages d'un wiki sont des documents quelconques, il n'existe pas d'outil capable de détecter les incohérences d'un document dues à la concurrence, hormis les auteurs du documents eux-mêmes, qu'il est donc nécessaire d'aider en leur montrant de la manière la plus explicite possible les effets de cette concurrence dans le document.

1.4 Objectifs de la thèse

1.4.1 Objectifs

L'objectif de ce travail est de concevoir et réaliser un mécanisme de conscience des opérations concurrentes pour un wiki réparti sur un réseau P2P. Ce mécanisme doit permettre de faire face au problème des fusions de modifications concurrentes réalisées automatiquement par les différents serveurs du réseau. Un wiki P2P utilisant ce mécanisme doit pouvoir indiquer à tout visiteur d'une page wiki quelconque accédant à un serveur quelconque si cette page résulte d'une fusion de modifications concurrentes, et si c'est le cas, indiquer dans cette page les régions sur lesquelles la fusion a eu un impact.

Ce mécanisme permettra :

– aux visiteurs, d'avoir conscience des incohérences éventuelles contenues dans le do-

cument et dues à une fusion,
- aux auteurs, d'avoir une assistance dans la correction des éventuelles incohérences dues à la fusion.

Plus précisément, ce mécanisme devra permettre de répondre aux questions suivantes :

1. La page visualisée a-t-elle été produite par un auteur humain, ou bien a-t-elle été produite par l'algorithme de fusion déclenché par le serveur ? En d'autres termes, le contenu visualisé a-il-été validé par un auteur humain ? Cette information doit être disponible quelque soit le visiteur et le moment de visualisation. Bien entendu, cette information se rapporte à l'état de la page tel qu'il est disponible au moment de la requête et sur le serveur auquel est adressée la requête.

2. Dans le cas où la page visualisée résulte d'une fusion, quelles régions de la page sont concernées par cette fusion ? Quel est l'impact de cette fusion sur le contenu observé ? Quel est l'impact de chaque modification individuelle ? Là encore, l'information offerte se réfère à l'état de la page tel qu'il est disponible au moment de la requête et sur le serveur utilisé. Il doit être possible d'identifier les différents changements individuellement, et cela doit concerner l'ensemble des changements émis en concurrence dans l'histoire de la page.

3. Dans l'historique d'une page, quelles versions sont le résultat d'une fusion et quelles versions sont le résultat d'une édition par un auteur ? Quelles modifications concurrentes ont produit une version fusionnée ? La visualisation de l'historique d'une page doit explicitement rendre compte des changements opérés en concurrence. La visualisation de l'historique ne peut donc pas se baser sur une séquence linéaire de modifications/versions.

Le mécanisme proposé se basera sur l'exploitation d'un historique des changements et devra inclure :

- des algorithmes pour le calcul et la collecte d'informations relatives à la concurrence, notamment le calcul de l'état d'une page au moment de sa visualisation, et le calcul de l'ensemble de modifications entrant en concurrence,
- des techniques de visualisation de ces informations adaptées au cas d'un wiki, et respectant l'esprit de ce type d'outils.

Un certain nombre de contraintes sont à prendre en compte, détaillées dans la section suivante.

1.4.2 Contraintes liées au contexte

Notre mécanisme doit fonctionner dans le cadre d'un wiki P2P. Ce cadre nous impose un certain nombre de contraintes :

décentralisation : le mécanisme de conscience de la concurrence doit fonctionner de manière décentralisée, comme le wiki pour lequel il est conçu. En particulier, il ne doit pas utiliser de serveur centralisé pour stocker ou diffuser des informations. Le calcul et la collecte des informations doit être fait localement sur chaque site du réseau.

respect de la convergence à terme : le mécanisme de conscience de la concurrence ne doit avoir aucun impact sur la correction générale du système et ne doit pas remettre en cause la convergence à terme des copies résidant sur les sites du réseau. En particulier, il n'est pas souhaitable d'ajouter des informations de conscience de groupe dans le contenu des pages comme cela est fait dans les systèmes de gestion de version.

cohérences des informations de conscience de la concurrence : lorsque le système est au repos et que les copies des pages ont toutes convergées vers un état commun, l'information de conscience de concurrence délivrée par le système doit être identique sur tous les sites. En d'autres termes, si une page est dans un état identique sur différents sites, la même information sera calculée et visualisée sur chacun de ces sites. Lorsque la page est dans un état différent sur différents sites, l'information transmise doit être cohérente avec l'état stocké sur le site utilisé.

passage à l'échelle, dynamicité : dans un wiki P2P, le nombre de sites participants, et le nombre de sites introduisant des changements est potentiellement grand, non borné et dynamique. Le mécanisme de collecte et de calcul des informations de conscience de groupe doit donc être capable de gérer ce passage à une grande échelle. La contrainte est moins forte sur la visualisation, en effet, on peut supposer que le nombre de modifications concurrentes au moment d'un accès à une page est très inférieur aux nombres de sites produisant des changements.

1.4.3 Plan de la thèse

La suite de ce manuscrit est organisé de la façon suivante. Le prochain chapitre (chapitre 2) précise de manière plus concrète le contexte de travail de cette thèse et décrit le wiki P2P qui a été utilisé. Ce système, Wooki, est un prototype en cours de construction dans notre équipe.

Ensuite, le chapitre 3 est consacré à notre proposition d'un mécanisme de conscience de la concurrence pour le wiki P2P Wooki. Nous détaillerons les algorithmes de calcul et de collectes des informations ainsi que les modes de visualisation de cette information.

Nous présenterons ensuite, dans le chapitre 4 une réalisation de nos propositions dans le prototype Wooki, avant de conclure dans le chapitre 5 en rappelant nos contributions dans cette thèse et en discutant des perspectives du travail.

Chapitre 2

Wooki : un wiki P2P

Sommaire

2.1 Les principes 36
2.2 Les algorithmes 37
2.2.1 L'algorithme Woot 37
2.2.2 Algorithmes de diffusion des patchs 40
2.3 Architecture et mise en oeuvre 41
2.3.1 Conclusion 45

Wooki [76] est un prototype de Wiki P2P construit dans le projet ECOO du Loria. Ce prototype sert de cadre d'expérimentation pour les différents travaux autour de l'édition collaborative menés dans l'équipe. C'est notamment le cas pour les propositions et travaux réalisés dans le cadre de cette thèse. Avant de présenter en détail ces propositions dans le chapitre 3, nous présentons donc notre contexte particulier de travail dans ce chapitre.

2.1 Les principes

Wooki est un wiki fonctionnant de manière totalement décentralisée sur un réseau pair-à-pair. Un réseau Wooki est un réseau de serveurs Wiki interconnectés dans lequel aucun serveur ne joue un rôle particulier. Les données dans Wooki sont des pages wiki classiques, construites en utilisant une syntaxe wiki usuelle. Ces pages sont dupliquées sur l'ensemble des serveurs : chaque serveur détient une copie de l'ensemble des pages. Cette réplication massive des données permet à chaque serveur de rendre le service d'accès et d'édition des pages de façon totalement autonome.

Cette caractéristique rend le système très robuste, capable de monter en charge, et capable de supporter des déconnexions temporaires de certains serveurs, ainsi que des partitions dans le réseau. Wooki a été construit dans l'objectif d'obtenir un outil d'édition collaborative de masse, fiable et tolérant aux pannes et supportant le travail déconnecté sans restriction.

Le coeur du système est un mécanisme de réplication optimiste qui garantit la convergence à terme et le respect des intentions des utilisateurs. Ainsi, dans Wooki, lorsqu'un utilisateur réalise une opération d'édition, le changement qu'il produit au moment de la sauvegarde est appliqué immédiatement sur la copie locale de la page éditée, puis diffusé sur le réseau. Lorsqu'un site reçoit un changement distant, il l'intègre à sa copie locale en le fusionnant aux éventuels changements concurrents déjà intégrés. Cette fusion est réalisée par l'algorithme Woot [51], un algorithme dédié à l'édition collaborative P2P et proposé par l'équipe ECOO.

Les changements produits sont diffusés sur le réseau sous forme de *patchs*. Un patch représente un changement sous la forme d'une séquence d'opérations élémentaires d'édition (insertion, suppression). Il est calculé au moment de la sauvegarde d'une modification par un utilisateur, et représente donc les opérations permettant de passer d'une version de la page éditée à la version suivante.

Un serveur Wooki est donc un serveur wiki embarquant trois composants spécifiques :

– un composant de calcul des patchs, basé sur un algorithme de *Diff*. Ce composant est appelé lors de la sauvegarde d'une modification par un utilisateur.

- un composant d'intégration et fusion de patchs sur la copie locale des pages, utilisé pour appliquer une modification disponible sous la forme d'un patch, qu'elle soit locale ou reçue d'un serveur distant.
- un composant de diffusion des pacths sur le réseau, utilisé par un serveur pour transmettre un patch créé localement vers les serveurs distants.

Nous détaillons dans la suite les algorithmes à la base du système, puis son architecture et sa mise en oeuvre.

2.2 Les algorithmes

Nous présentons ici les algorithmes utilisés pour la réplication optimiste dans Wooki. Ces algorithmes ont notamment la charge d'assurer la correction du système, c'est-à-dire la convergence à terme et le respect des intentions.

2.2.1 L'algorithme Woot

L'algorithme Woot [51] est le coeur du système : il assure l'intégration des patchs dans une page wooki de façon déterministe et sans contrainte sur l'ordre de réception de ces patchs. Il offre les garanties sur la convergence des copies et le respect des intentions. Il a été explicitement conçu pour supporter les contraintes des réseaux P2P, en particulier la grande échelle, les partitions et la composition dynamique du réseau. Woot est capable de manipuler et fusionner des structures linéaires quelconques, c'est-à-dire toutes les structures sur lesquelles il est possible de définir un ordre total. Une chaîne de caractères, comme le contenu textuel d'une page wiki, est une structure linéaire typique. Woot peut ainsi manipuler une page wiki comme une séquence de caractères, comme une séquence de mots, ou comme séquence de lignes. Dans Wooki, l'algorithme est utilisé au niveau des lignes.

Les caractéristiques principales de l'algorithme sont les suivantes.

Modèle de stockage : l'algorithme se base sur une structure de données qui lui est propre pour le stockage des pages wikis. Dans ce modèle, chaque élément de la structure éditée (les lignes) possède un identifiant unique et non mutable dans le système. De plus, chaque élément de cette structure est décoré d'un booléen indiquant si l'élément est visible ou non. Ainsi, le texte d'une page wooki est composé d'une séquence de lignes représentées par des triplets : $\langle L_{id}, valeur, Visibilit\rangle$ où :

- L_{id} est l'identifiant unique de la ligne, représentée comme la paire (S_{id}, op_{id}), où S_{id} est l'identifiant du site ayant créé la ligne et op_{id} un numéro de séquence local à ce site.
- *valeur* représente le contenu textuel de la ligne wiki,
- *Visibilité* est un booléen indiquant si la ligne est visible ou non.

Lorsqu'une opération du wiki nécessite l'accès au contenu d'une page, cette page doit être extraite du système de stockage. Cette extraction consiste simplement à parcourir séquentiellement les triplets et à ne retenir que la *valeur* de ceux marqués comme *visibles*.

Opérations d'édition : l'algorithme utilise uniquement deux opérations d'éditions : une opération d'insertion d'un élément dans le modèle de stockage, et une opération de suppression d'un élément, à l'exclusion de toute autre opération. Ainsi, une modification d'un élément existant doit être représentée comme la suppression de l'ancienne valeur, suivie de l'insertion de la nouvelle valeur.

La suppression : l'opération de suppression, notée $DEL(e_{id})$, qui permet de supprimer l'élément ayant pour identifiant e_{id} consiste en fait à rendre cet élément invisible en modifiant le drapeau de visibilité. L'élément, et son identifiant, est cependant conservé dans la structure de données. Ainsi, cette structure stocke l'ensemble de l'histoire d'une page.

L'insertion : l'opération d'insertion est aussi particulière dans Woot. Son profil est différent de celui de l'opération d'insertion utilisée dans la plupart des outils d'édition. Cette opération a pour profil : $INS(e, L_{av}, L_{ap})$ que l'on note également $INS(L_{av} < e < L_{ap})$ et permet de spécifier l'insertion de l'élément e *entre* les éléments L_{av} et L_{ap}. Cette opération a pour effet d'insérer un nouvel élément dans la structure de données : elle calcule son point d'insertion, génère son identifiant s'il n'existe pas, et positionne le booléen de visibilité à VRAI.

Cette manière de spécifier le point d'insertion du nouvel élément, non pas par sa position dans le texte, mais de manière relative aux éléments déjà présents permet à l'algorithme de choisir un lieu d'insertion pour cet élément tout en prenant en compte plus explicitement les intentions d'édition des utilisateurs. En garantissant que cette contrainte sur le point d'insertion est satisfaite dans tous les cas, Woot respecte les intentions des utilisateurs. Notons que si l'opération a déjà été exécutée, l'élément e est déjà présent dans le stockage de données et l'opération peut être abandonnée. Notons également que les éléments L_{av} et L_{ap} doivent exister dans le stockage. Si ce n'est pas le cas, l'opération

est mise en attente de la réception des opérations créant l'élément manquant. Enfin, il faut remarquer que le fait de supprimer un élément utilisé pour spécifier l'insertion d'un autre ne pose pas de problème à l'algorithme puisque l'élément supprimé est conservé mais seulement marqué comme non visible, et il peut donc calculer le lieu d'insertion. Ainsi, dans l'opération $INS(''un\ merle\ est'', L_0, L_2)$, le fait que les lignes L_0 ou L_2 (ou les 2) soient supprimées n'empêche pas de calculer le point d'insertion du texte.

Définis sous cette forme, les couples d'opérations (DEL, DEL) et (DEL, INS) commutent grâce notamment au fait que les paramètres des opérations sont des identifiant uniques d'éléments, et non pas des positions dans un document. Ainsi, la séquence $DEL(L_1); DEL(L_2)$ donne le même résultat que la séquence $DEL(L_2); DEL(L_1)$: les 2 lignes sont marquées comme non visibles. De même, les séquences $DEL(L_i); INS(L_j < e < L_k)$ et $INS(L_j < e < L_k); DEL(L_i)$ donneront le même résultat : une structure dans laquelle L_i est invisible et e inséré entre L_j et L_k. Par contre, le couple (INS, INS) ne commute pas dans tous les cas, notamment lorsque les deux opérations spécifient le même point d'insertion. Dans ce cas là, l'algorithme doit choisir entre deux séquences possibles d'insertion. Il réalise ce choix en utilisant l'identifiant unique attribué aux sites.

Le principe de fonctionnement de l'algorithme est le suivant. Dans le modèle Woot, un ensemble quelconque d'opérations d'édition peut se représenter sous la forme d'un graphe traduisant l'ordre partiel induit par les précédences exprimées dans les opérations d'insertion. Un exemple est donné en figure 2.1.

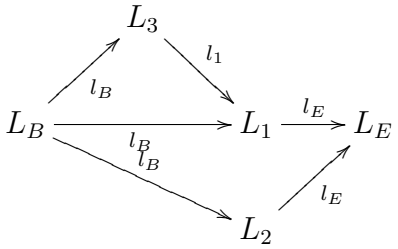

FIG. 2.1 – Graphe d'opérations dans Woot

Ce graphe correspond aux opérations $INS(L_1, l_B, l_E); INS(L_2, l_B, l_E); INS(L_3, l_B, l_1)$ où les deux lignes L_1 et L_2 sont insérées en concurrence dans une page vide (L_B et L_E représentent les marqueurs de début et fin du texte), et L_3 est insérée après L_1 dans le temps (car elle utilise l_1 dans ses paramètres) et entre le début du texte et la ligne L_1 du point de vue de la position. Notons que ce graphe sera construit de manière identique quelque soit l'ordre d'arrivée des opérations.

Chaque site disposant d'une copie des données partagées maintient cette copie en utilisant le modèle de stockage décrit plus haut. Au fur et à mesure de l'arrivée d'opé-

rations à intégrer, qu'elles soient locales ou distantes, l'algorithme construit le graphe de dépendance et le linéarise de manière à déterminer la position d'insertion des éléments à intégrer. L'algorithme assure que :

1. le résultat de la linéarisation est indépendant de l'ordre d'intégration des opérations dans le graphe, et donc de l'ordre d'arrivée des opérations,
2. le résultat de la linéarisation est identique sur tous les sites réalisant cette linéarisation.

Ces garanties rendent l'algorithme déterministe et indépendant de l'ordre d'arrivée des opérations sur un site. Tous les sites ayant reçus le même ensemble d'opérations atteindront le même état : l'état d'une page Wooki dépend uniquement de l'ensemble d'opérations reçues et non pas de l'ordre de réception de ces opérations.

L'algorithme Woot n'impose donc aucune contrainte d'ordre sur le mécanisme de diffusion des patchs, hormis le fait qu'il doit garantir la livraison de ces patchs.

2.2.2 Algorithmes de diffusion des patchs

Les algorithmes de diffusion des patchs ont en charge la propagation des changements émis par un site vers les autres sites. Ainsi que précisé ci-dessus, l'algorithme Woot n'impose aucune contrainte sur l'ordre de livraison des patchs. La seule contrainte reposant sur le mécanisme de diffusion est d'assurer que tous les patchs diffusés seront livrés à tous les sites. Ce mécanisme doit permettre des déconnexions temporaires : un site déconnecté doit pouvoir diffuser les patchs qu'il produit et recevoir les patchs distants lorsqu'il rejoint de nouveau le réseau.

Ces faibles contraintes d'ordonnancement permettent d'éviter les algorithmes de type Broadcast ordonnés classiques des systèmes distribués pour mettre en oeuvre des algorithmes conçus pour les systèmes pair-à-pairs. L'approche choisie dans Wooki est la suivante.

La diffusion aux sites connectés est réalisée grâce à un algorithme de diffusion épidémique probabiliste [17, 10]. Le principe est le suivant. Chaque site du réseau dispose d'une table de ses voisins connus. Pour diffuser un message, il sélectionne un sous-ensemble de cette table de manière probabiliste et diffuse le message à ce sous-ensemble, en indiquant un nombre de rediffusion maximum. A la réception, chaque site récepteur effectue le même processus, en décrémentant le nombre de rediffusion. Le message est ainsi acheminé de proche en proche à l'ensemble des sites connectés. En choisissant finement les paramètres de l'algorithmes (taille des sous-ensembles sélectionnés, nombre maximum de rediffusion)

l'algorithme offre des garanties de livraison à l'ensemble des sites. Concrètement, Wooki utilise l'algorithme LPBCAST [16], qui fonctionne sur ce principe et y ajoute la gestion dynamique de la composition du réseau. Cette gestion des membres est basée sur la diffusion et la découverte dynamique des tables de voisinages. Lors de la diffusion d'un message, le site émetteur y insère un extrait de sa table. Le récepteur peut ainsi examiner cette table et ajouter dans sa propre table les voisins dont il n'a pas connaissance. Bien entendu, cet algorithme doit être complété pour traiter le cas des déconnexions.

La gestion des déconnexions est assurée par un mécanisme complémentaire réalisant un protocole d'anti-entropie [10]. Ce protocole fonctionne de la manière suivante. Chaque site maintient un historique de l'ensemble des patchs qu'il a reçus et appliqués. Lorsqu'un site rejoint le réseau après une période de déconnexion, il transmet son historique à l'un de ses voisins. Ce voisin calcule la différence entre l'historique reçu et son propre historique et détermine ainsi la liste de patchs non reçus par le demandeur au cours de sa période déconnectée, ainsi que la liste des patchs produits par ce demandeur durant cette même période. Les 2 sites procèdent ensuite à l'échange des patchs. Ce protocole d'anti-entropie est déclenché automatiquement lorsqu'un site rejoint le réseau pour la première fois ou après une déconnexion, mais aussi de manière périodique pour réparer les effets d'éventuelles partitions du réseau qui n'auraient pas été détectées.

2.3 Architecture et mise en oeuvre

Le serveur Wooki est une application Web, fonctionnant adossé à un serveur Web et traitant des requêtes HTTP. Il propose une interface wiki classique, permettant de visualiser, éditer ou accéder à l'historique des différentes pages.

La visualisation et l'édition de pages dans Wooki est classique, et illustrée en figure 2.2.

Par contre, la visualisation de l'historique, illustrée en figure 2.3 est différente des visualisations classiques. L'ensemble des lignes insérées et détruites est affiché, et les lignes détruites et non visibles dans la version courante sont grisées et surlignées d'un trait. De cette visualisation, il est possible d'accéder, pour chaque ligne, à des informations concernant l'ensemble des patchs ayant touchés cette ligne.

L'interface du système propose également un certain nombre de services pour gérer l'interaction avec le réseau de pairs : ajout de voisins, visualisation de la table de voisins, connexion/déconnexion au réseau, déclenchement d'une anti-entropie avec un voisin.

Du point de vue de l'architecture, le serveur wiki est composé de plusieurs modules, illustrés en figure 2.4.

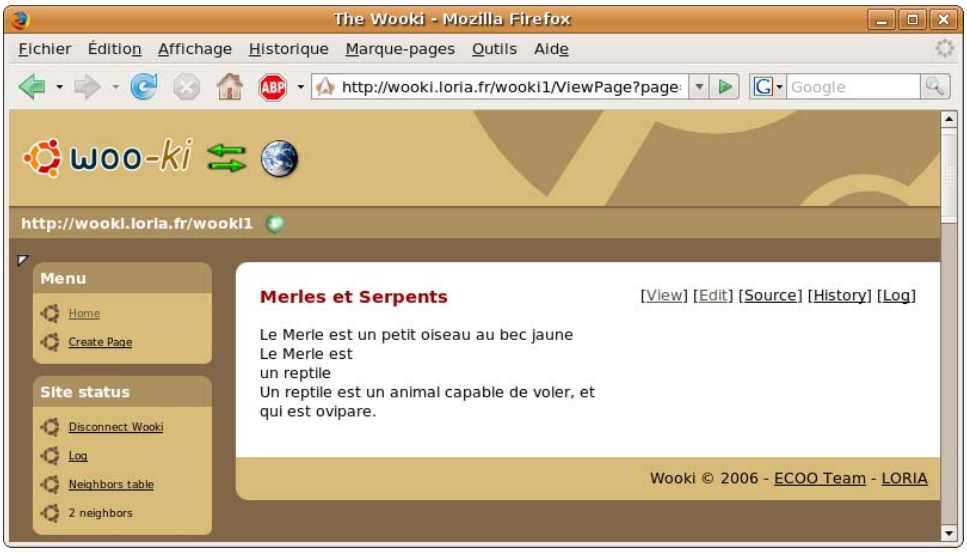

FIG. 2.2 – Visualisation d'une page Wooki

WootEngine : est le composant réalisant l'algorithme Woot. Il gère donc le stockage des pages Wooki, et assure l'intégration des patchs dans cette structure. Il permet également l'extraction de pages de cette structure. Il utilise le composant **Clock** pour générer des identifiants uniques lors de l'insertion de nouvelles lignes dans la structure de données.

AntiEntropy, LPBCAST : sont les composants pour l'interaction avec le réseau de pairs. Le composant AntiAntropy réalise le protocole d'anti-entropie et gère pour cela un log de patchs. LPBCAST réalise l'algorithme de diffusion épidémique et assure donc la gestion des tables de routage.

Radeox : est le composant de rendu HTML, c'est-à-dire chargé de convertir un document textuel en syntaxe wiki en un document HTML visualisable dans un navigateur Web.

WookiApp : est le composant central réalisant l'application Web. Il est chargé de recevoir les requêtes en provenance des utilisateurs, de coordonner leur exécution et de répondre à ces requêtes.

Le composant WookiApp gère principalement 3 types de requêtes en provenance des utilisateurs plus l'arrivée d'un patch distant. Les algorithmes liés au traitement de ces requêtes sont donnés en figure 2.5 :

get(PageId) : demande de visualisation d'une page. Le traitement de cette requête consiste à demander au composant WookiEngine l'extraction de la page depuis le modèle de stockage, puis demander le rendu HTML de cette page au composant Radeox pour, finalement, envoyer ce contenu HTML au navigateur dont est issue la

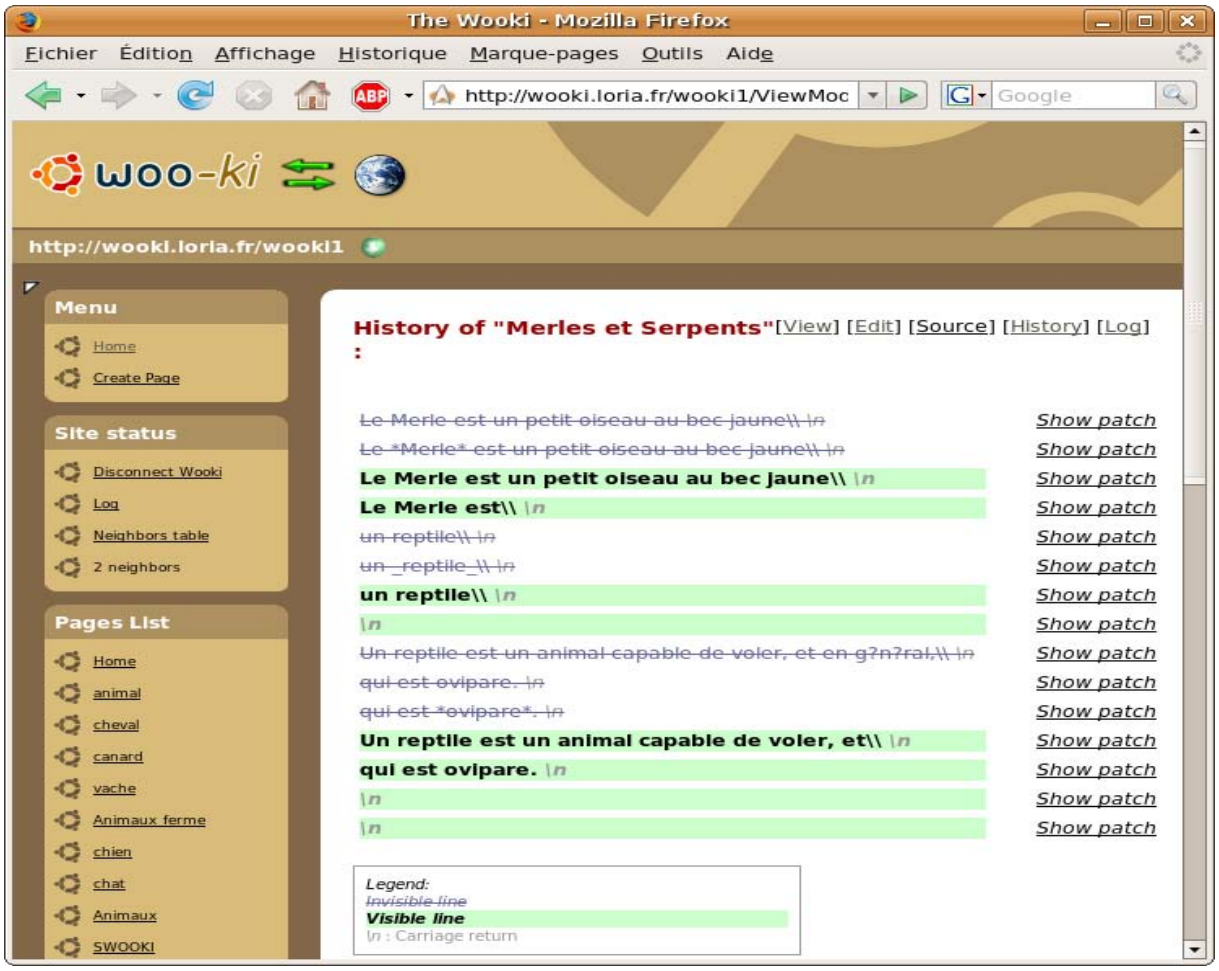

FIG. 2.3 – Visualisation de l'historique d'une page dans Wooki

requête.

edit(PageId) : demande d'édition d'une page. Le traitement consiste à demander l'extraction du contenu de cette page du modèle de stockage au composant WootEngine. Ce contenu textuel est alors stocké dans un espace temporaire, et renvoyé tel quel au navigateur dont est issue la requête. Ce navigateur l'insère dans une zone HTML d'édition pour permettre sa modification.

save(PageId, contenu) : demande de sauvegarde de modification d'une page. Le nouveau contenu est transmis au serveur. Le traitement consiste à calculer dans un premier temps le patch correspondant à la modification. Ce patch est construit en calculant la différence entre la nouvelle version transmise par le client, et l'ancienne version stockée dans l'espace temporaire par l'opération de demande d'édition. Une fois le patch construit, il est transmis au composant WootEngine pour être intégré à la copie locale de la page. Cette intégration modifie le patch en y ajoutant les iden-

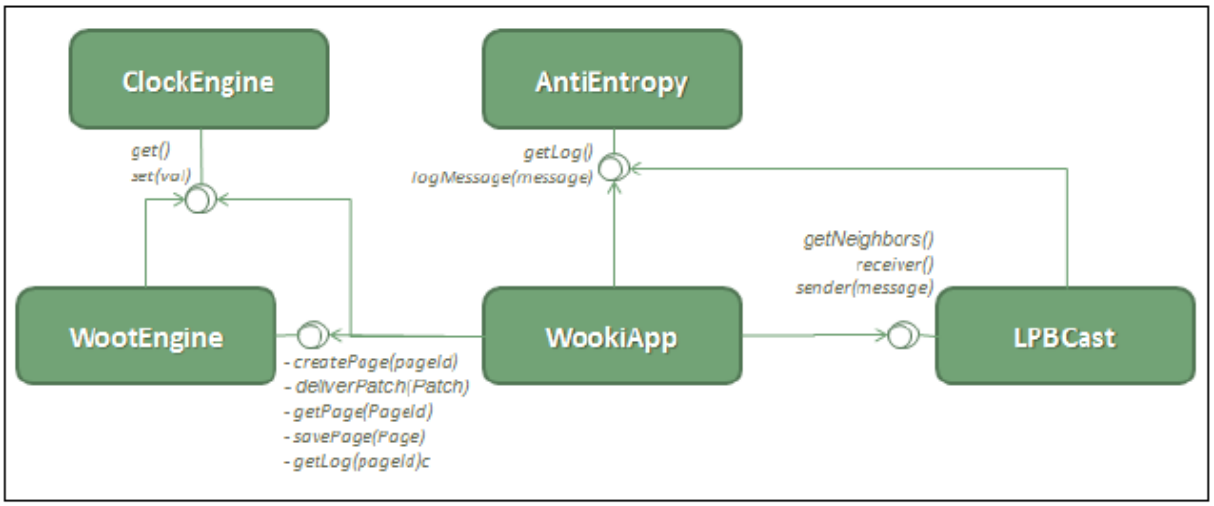

FIG. 2.4 – Architecture du serveur Wooki

tifiants des lignes insérées. Ce patch est ensuite transmis au composant LPBCAST pour diffusion sur le réseau, et au composant AntiEntropy pour stockage dans son log.

receive(Patch) : demande d'application d'un patch distant. Cette requête est émise par le composant de gestion de la diffusion ou le composant d'anti-entropie. Lorsqu'un patch est reçu par un de ces 2 composants, il est transmis au composant WookiApp qui se charge de demander son intégration immédiate au composant WookiEngine. Une fois qu'il est intégré, le patch est transmis au composant AntiEntropy pour stockage dans le log.

```
on GET(PageId) :
  page := wookiEngine.extract(PageId) ;
  htmlText := HTMLRenderer.render( page ) ;
  return htmlText ;
end;
```

```
on SAVE (PageId, pageContent) :
  oldCOntent := Temp.load( PageId );
  Patch := Diff ( oldContent, pageContent );
  Patch.setPage(PageId) ;
  wookiEngine.applyPatch ( PageId, Patch );
  AT.log(PageId, Path) ;
  LPBCAT.send(Patch) ;
end;
```

```
on EDIT (PageId) :
  page := wookiEngine.extract(PageId) ;
  Temp.save(PageId, page);
  return page ;
end;
```

```
on RECEIVE (Patch) :
  pageId := Patch.getPage()
  wookiEngine.applyPatch ( PageId, Patch );
  AT.log(PageId, Path) ;
end;
```

FIG. 2.5 – Algorithmes de traitement des requêtes dans Wooki

2.3.1 Conclusion

Dans sa version initiale, le prototype Wooki ne contenait aucun mécanisme de conscience de la concurrence. L'utilisateur visualisant une page n'avait aucun moyen de savoir que le contenu de la page résultat d'une intégration de patchs concurrents.

Chapitre 3

Un mécanisme de conscience des modifications concurrentes

Sommaire

3.1 Introduction 48

3.1.1 Rappel des objectifs et contraintes 48

3.1.2 Résultats attendus 50

3.1.3 Les principes généraux de notre approche 51

3.2 Algorithmes pour la collecte et le calcul des informations . . 53

3.2.1 La détection de la concurrence 53

3.2.2 Le calcul des histoires concurrentes 59

3.3 Mécanismes pour la conscience de la concurrence 68

3.3.1 Réception d'un patch distant 69

3.3.2 Demande de visualisation d'une page wiki 70

3.3.3 Demande d'édition d'une page wiki 73

3.3.4 Demande de sauvegarde d'une page modifiée 75

3.3.5 Changement de statut 76

3.4 La Représentation de l'Historique d'une page Wiki 78

3.5 conclusion 82

Ce chapitre présente le détail de notre proposition pour la construction d'un mécanisme de conscience des modifications concurrentes adapté au contexte d'un wiki P2P. Rappelons que dans un tel système, certaines versions des pages visibles peuvent avoir été produites automatiquement par le serveur au moment de l'intégration de modifications concurrentes. Dans ce cas, le contenu accessible par les visiteurs n'a pas été produit et validé par les auteurs de la page et risque donc de contenir des incohérences non souhaitées et parfois difficiles à détecter.

Notre objectif est de concevoir et réaliser un mécanisme qui détecte ces cas et rend explicite dans une page wiki les zones sur lesquelles la fusion a eu un impact, et donc sujette à caution. Un tel mécanisme permettra aux visiteurs d'être conscients des risques d'incohérence dans la page visualisée, et aux auteurs de la page d'intervenir ultérieurement de manière assistée pour corriger ces éventuelles incohérences.

Dans une première partie introductive (3.1), nous rappelons les contraintes à prendre en compte pour notre mécanisme, issues de de notre contexte de travail. Ensuite, nous présentons le résultat attendu pour la visualisation des informations de conscience de groupe, avant de présenter les principes généraux de notre approche.

Dans une seconde partie (3.2) , nous aborderons l'aspect algorithmique de notre travail, en détaillant les principes de collecte et de calcul des informations pour la conscience de la concurrence. En particulier, nous discuterons de la détection de la concurrence et du calcul des parties concurrentes dans une histoire d'opérations.

Ensuite, nous aborderons la partie visualisation (3.3) des informations de conscience de concurrence dans le cadre d'un wiki P2P. Nous montrerons notamment comment sont traitées les requêtes des utilisateurs et comment sont affichées les informations. Nous poursuivrons sur la visualisation de l'historique d'une page dans la partie (3.4).

3.1 Introduction

3.1.1 Rappel des objectifs et contraintes

Ainsi que nous l'avons introduit dans le chapitre 1, pour faire face au problème des fusions automatiques réalisées dans l'espace public par les différents serveurs d'un wiki P2P, nous proposons un mécanisme de *conscience de la concurrence*. Le rôle de ce mécanisme est le suivant :

1. détecter et afficher le *statut* d'une page vis-à-vis de la concurrence, au moment de son accès ; s'agit-il d'une page *fusionnée*, c'est-à-dire produite par une fusion déclenchée par le serveur, ou s'agit-il d'une page *éditée*, c'est-à-dire produite par une édition

réalisée par un utilisateur ?

2. dans le cas d'une page *fusionnée*, rendre explicite dans la page les zones résultant de la fusion de changements concurrents, en permettant de distinguer les modifications apportées par chaque changement. Cette information doit être transmise aux visiteurs de la page ainsi qu'aux auteurs.
3. rendre explicite dans l'historique d'une page le statut de chacune des versions de la page, ainsi que, pour les versions *fusionnées*, les modifications concurrentes dont elles résultent.

Ce mécanisme doit être adapté au contexte des wiki P2P. Pour ce faire, il doit satisfaire aux contraintes suivantes :

- décentralisation : la collecte et le calcul des informations nécessaires à la conscience de la concurrence doit se faire sans serveur central et de façon purement localisée sur chaque serveur. Aucun échange avec les autres serveurs du réseau, hormis les échanges liés au mécanisme de réplication optimiste, ne doit être nécessaire.
- respect de la convergence à terme : le mécanisme de conscience de la concurrence ne doit pas remettre en cause la correction du système. En particulier, il est souhaitable d'éviter une approche qui modifierait le contenu stocké des pages wiki, pour y insérer par exemple des informations du type *blocs de conflits*, ou plus généralement toute approche qui conduirait à la création et diffusion de patchs spécifiques à la conscience de groupe qui pourraient entrer en concurrence entre eux ou avec des patchs représentant des modifications de contenu.
- correction et cohérence : Le mécanisme doit être correct, dans le sens où, lorsqu'il détecte une page ayant le statut *fusionnée*, cette page résulte réellement d'une fusion. Les informations présentées doivent être cohérentes sur les différents serveurs : pour une page dans un état identique sur différents serveurs, les informations de conscience de concurrence produites et visualisées sur ces différents serveurs sont identiques.
- contexte P2P : le mécanisme doit être utilisable dans un contexte dynamique et de grande échelle en terme de serveurs participants au réseau. Le nombre de serveurs participants est dynamique et souvent non connu ; il peut être très grand. Ceci impacte principalement les algorithmes de collecte et de calcul des informations. Par contre, on fait l'hypothèse que le nombre de modifications concurrentes considérées dans *une* fusion, et donc à visualiser au sein d'une page, reste faible comparé au nombre de sites potentiellement producteurs de changements.

Ces contraintes sont assez fortes et rendent caduques les approches de conscience de groupe asynchrone proposées jusqu'ici.

3.1.2 Résultats attendus

Avant de présenter les principes généraux de notre approche, nous souhaitons concrétiser notre objectif en présentant de manière visuelle le résultat attendu pour notre mécanisme.

Nous revenons pour cela au petit scénario utilisé pour illustrer notre problématique, dans la figure 1.7 du paragraphe 1.2. Dans ce scénario, deux lignes d'un même texte sont modifiées en concurrence par deux auteurs. Chaque sauvegarde est fait en parallèle, et le résultat final n'est pas celui espéré par chacun des auteurs.

Les principes de visualisation des informations de conscience de concurrence sont illustrés en figure 3.1. Ces principes sont les suivants :

1. le fait qu'une page est dans un état *fusionnée* doit être explicite en perceptible par l'utilisateur sans parcourir le contenu de la page ;
2. les zones résultant de la fusion apparaissent avec une couleur de fond qui permet de les distinguer ; la couleur est spécifique à chaque changement concurrent ;
3. les lignes supprimées apparaissent malgré tout, avec une indication explicite, par exemple une fonte *barrée* ou *grisée faible* ;
4. les lignes ajoutées apparaissent normalement sur le fond coloré ;
5. le reste du texte apparaît sans altération.

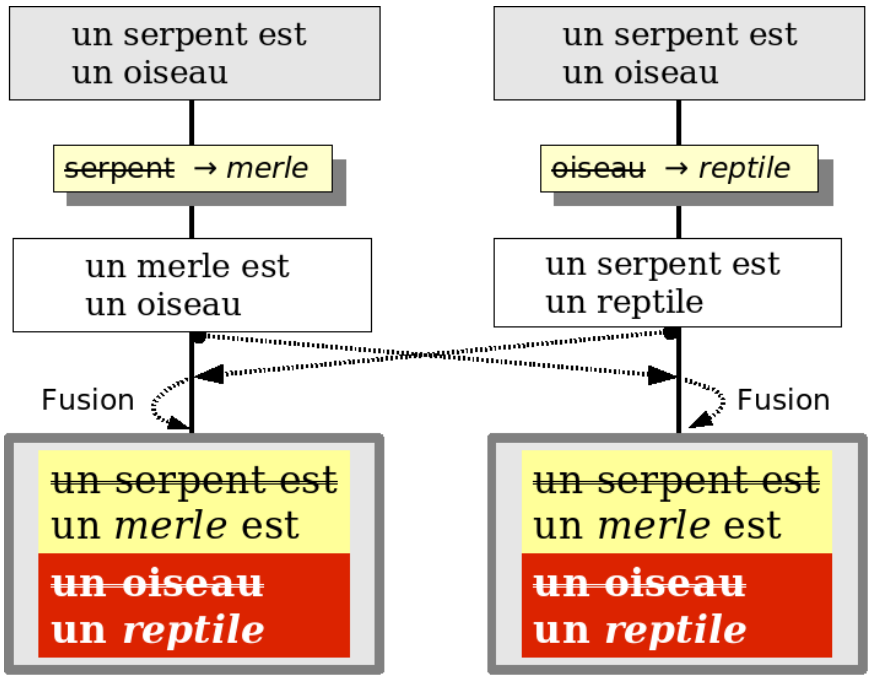

FIG. 3.1 – Le résultat attendu

3.1.3 Les principes généraux de notre approche

Avant d'aborder les principes de notre approche, rappelons brièvement le processus de visualisation d'une page dans un wiki. Ce processus est illustré en figue 3.2.

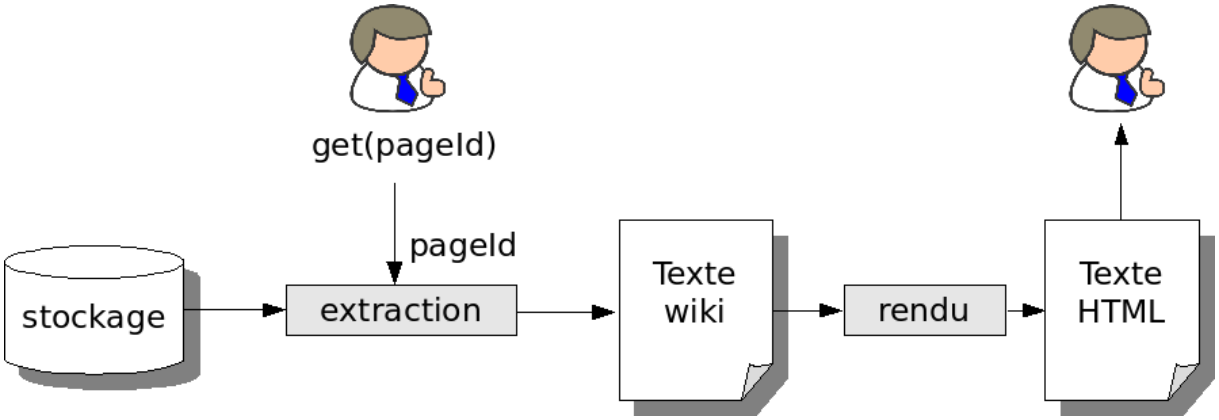

FIG. 3.2 – processus de visualisation de page dans un wiki

Le processus est le suivant :

1. un utilisateur demande la visualisation d'une page en transmettant l'identifiant de cette page (en général, le nom de la page) dans sa requête.
2. cet identifiant de page est utilisé pour extraire le contenu de la page du système de stockage. Cette extraction produit un texte en *syntaxe wiki*.
3. ce texte en syntaxe wiki est transmis au moteur de rendu HTML qui transforme la syntaxe wiki en balise html. Le résultat est un texte html.
4. ce texte html et renvoyé au navigateur de l'utilisateur.

Le principe général de notre mécanisme est illustré en figure 3.3. Il repose sur les points suivants :

1. Chaque serveur maintient un historique (log) des patchs qui ont été appliqués à la copie locale de chaque page, qu'ils s'agissent de patchs locaux ou distants.
2. Lorsqu'un accès à une page est demandé, ce log est utilisé pour calculer le statut de la page (*fusionnée* ou *éditée*) et pour extraire du log l'histoire concurrente de la page. Le calcul de cette histoire concurrente est le point le plus complexe, d'un point de vue algorithmique, de notre travail. Il requiert tout d'abord l'utilisation d'un mécanisme de détection de la concurrence entre les patchs, qui sera discuté dans la section 3.2.1. D'autre part, le calcul de cette histoire concurrente consiste à trouver les patchs concurrents dont résulte l'état courant de la page, puis à calculer l'état ancêtre de cet ensemble de patchs, c'est-à-dire l'état de la page ayant été

observé par tous les sites producteurs d'un patch concurrent. L'histoire concurrente de la page est alors formée de tous les patchs postérieurs à cet état. Ce calcul sera détaillé dans le paragraphe 3.2.2.

3. Le processus d'extraction de la page est adapté de façon à extraire toutes les lignes visibles ainsi que les lignes invisibles supprimées par une opération apparaissant dans l'histoire concurrente.
4. Toutes les lignes extraites ayant été créées ou supprimées par une opération de l'histoire concurrente sont marquées avec quelques informations : type de l'opération (INS ou DEL), date de création du patch, date d'intégration du patch, identifiant de patch, origine (site producteur du patch). Le résultat est un texte en syntaxe wiki, décoré de ces marques contenant des informations de conscience de concurrence.
5. Le processus de rendu HTML est également adapté pour tenir compte de ces marques et les traduire en HTML. Toutes les lignes marquées reçoivent ainsi un fond coloré. Les lignes détruites sont barrées. Des informations concernant le patch créateur sont ajoutées (dates, site créateur, liens vers l'historique).

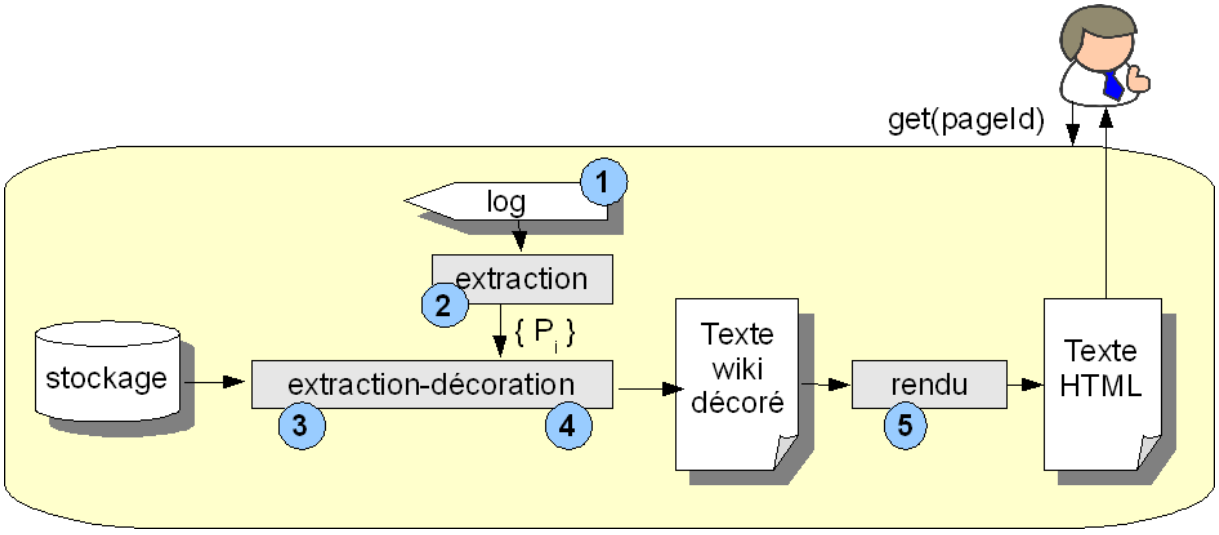

FIG. 3.3 – Notre approche pour la conscience de la concurrence

Quelques commentaires sont nécessaires sur cette approche par rapport aux objectifs fixés et aux contraintes existantes :

- A condition que le calcul de l'histoire concurrente soit possible et correcte, l'approche répond *à priori* aux objectifs : les pages *fusionnées* sont détectées et les zones impactées par la fusion sont repérées dans les pages.
- Le calcul est purement local, car exclusivement basé sur le log de patchs. L'exigence de décentralisation est donc respectée. Par ailleurs, ce calcul repose uniquement sur

ce log, c'est-à-dire l'ensemble des patchs reçus par un site. Pour peu que ce calcul soit déterministe (ce qui est le cas puisqu'il s'agit algorithmes), le principe de cohérence entre les différents sites est respecté.

- Les informations de conscience de groupe servent uniquement à décorer le texte au moment de sa visualisation, et ne sont jamais sauvegardées dans le contenu d'une page. Ces informations, au moment de leur affichage, pourraient d'ailleurs être utilisées de manière dissociée au texte de la page. De plus, mises à part d'éventuelles méta-données ajoutées dans les patchs, l'approche ne modifie pas le contenu des patchs et ne modifie pas le processus d'intégration de ces patchs. Le mécanisme de réplication optimiste n'est pas modifié et sa correction n'est pas remise en cause.
- En ce qui concerne les problèmes d'échelle et de dynamicité, ils sont liés principalement à la détection de la concurrence. Cette problématique sera discutée sans la section suivante.

3.2 Algorithmes pour la collecte et le calcul des informations

Nous présentons dans cette section des définitions et algorithmes concernant la collecte et le calcul des informations pour la conscience de groupe. Nous avons vu, dans le paragraphe 3.1, que ceci consiste essentiellement à extraire l'histoire concurrente d'un log des patchs appliqués sur une page. Nous présentons ce mécanisme dans la partie 3.2.2. Ce mécanisme repose sur un détecteur de concurrence entre patchs que nous présentons maintenant.

3.2.1 La détection de la concurrence

La détection de la concurrence est un problème classique et très étudié dans le domaine des systèmes distribués. Dans ce contexte, on dit que deux opérations sont concurrentes si elle ne peuvent pas être comparées d'un point de vue causal. La détection de la concurrence repose donc sur un mécanisme capable de capturer les dépendances causales entre opérations. Le principe est d'étiqueter les événements du système à l'aide de valeurs produites par une horloge, de telle sorte que ces étiquettes soient comparables selon un ordre reflétant exactement la causalité.

Malheureusement, ce principe est remis en cause dans notre contexte par un résultat important dû à Charron-Bost. Dans [8], Charron-Bost démontre que la causalité ne peut être capturée complètement qu'avec un mécanisme utilisant des étiquettes d'une taille

en $O(N)$, où N est le nombre de sites dans le système. Dans notre contexte, le nombre de sites est non borné, et donc le résultat de Charron-Bost nous indique que capturer complètement la causalité dans un wiki P2P nécessite de marquer les patchs échangés entre les serveurs avec des étiquettes de taille non limitée. Clairement, une telle approche ne permet pas un passage à grande échelle en nombre de sites. Le résultat de Charron-Bost nous indique également que le passage à l'échelle ne pourra être possible qu'à condition d'accepter un détecteur de concurrence imparfait. Ainsi, un mécanisme de détection de patchs concurrents dans un Wiki P2P résultera d'un compromis entre précision de la détection de concurrence et passage à la grande échelle.

Nous présentons d'abord rapidement deux mécanismes classiques de détection de la concurrence, avant de présenter l'approche que nous avons choisie, basée sur la notion d'horloge plausible.

Quelques définitions

La causalité est traditionnellement définie par la relation *happens-before* introduite par Lamport en 1978 [37]. Cette relation est définie de la manière suivante :

1. Soient a et b, deux événements dans le même processus, tels que a se produit avant b, alors $a \rightarrow b$ est vrai,

2. Soit a l'événement correspondant à l'envoi d'un message par un processus et b l'événement de réception de ce message par un autre processus, alors $a \rightarrow b$ est vrai.

Cette relation est transitive : si $a \rightarrow b$ et $b \rightarrow c$, alors $a \rightarrow c$. D'autre part, on dit que deux événements a et b sont concurrents si $a \rightarrow b$ n'est pas vrai et $b \rightarrow a$ n'est pas vrai non plus. En d'autres termes, aucun des deux ne précède l'autre.

Lamport a également défini une horloge logique, c'est-à-dire un étiquetage des événements, reflétant la relation causale. Cette horloge possède la propriété suivante : si $a \rightarrow b$, alors $H(a) < H(b)$. Une telle horloge est très simple à implanter, mais elle a peu d'intérêt pratique car on ne peut rien en déduire concernant l'ordonnancement réel des événements. En effet, $H(a) < H(b)$ n'implique pas forcément que $a \rightarrow b$.

Pour être utilisable en pratique, une horloge logique doit capturer complètement la relation causale, c'est-à-dire posséder en plus la propriété : si $H(a) < H(b)$ alors $a \rightarrow b$. Une telle horloge permet d'observer avec exactitude la précédence causale des événements dans un système, et donc de détecter les événements concurrents.

Vecteurs d'horloge

Les vecteurs d'horloge sont un mécanisme capturant la causalité introduit par Mattern [39]. Ces vecteurs sont construits de la manière suivante. Chaque site S_i maintient un vecteur VC_i avec lequel il étiquette les événements qu'il observe et qui peuvent être locaux ou reçus. Ce vecteur d'étiquetage possède les propriétés suivantes :

1. $VC_i[i]$ est le nombre d'événements produits par S_i,
2. $VC_i[j] = k$ signifie que S_i a reçu k événements en provenance de S_j.

Ces propriétés sont maintenues de la façon suivante :

- Avant d'exécuter une opération locale, un site S_i exécute : $VC_i[i] = VC_i[i] + 1$
- Lorsqu'un site S_i diffuse un message m, il étiquette ce message avec la valeur $e(m) = VC_i$ au moment de l'envoi.
- Lors de la réception d'un message m, le site S_j met à jour son vecteur en exécutant : $VC_j[k] = max(VC_j[k], e(m)[k])$ pour tout k.

A partir de ces définitions, la comparaison de deux vecteurs s'effectue de la manière suivante :

$V_i < V_j \equiv V_i \neq V_j \; and \; \forall k, V_i[k] < V_j[k]$

Cette comparaison capture exactement la causalité : $a \rightarrow b \equiv VC(a) < VC(b)$ et permet donc de réaliser un test de concurrence entre deux événements.

Cette technique d'étiquetage des événements peut être utilisée dans notre contexte pour étiqueter les patchs diffusés sur le réseau. Elle permet également d'étiqueter les versions successives d'une page, en attribuant comme étiquette de version la valeur du vecteur local après l'opération ayant conduit à la création de la version. Un tel mécanisme permet alors de comparer des versions :

- $E(V_1) = E(V_2)$: les versions V_1 et V_2 sont identiques,
- $E(V_1) < E(V_2)$: la version V_1 est antérieure (dans le graphe de version) à V_2, et donc il existe un ensemble de patchs permettant d'obtenir V_2 à partir de V_1,
- $E(V_1) \parallel E(V_2)$: les versions sont concurrentes, c'est-à-dire produites par des patchs concurrents non fusionnés.

Notons cependant que dans le contexte d'un wiki il faudra utiliser un vecteur *par page wiki* d'une part, et d'autre part ces vecteurs auront une entrée par site du réseau si aucune concurrence n'est possible sur un site. Si par contre on souhaite permettre la concurrence sur un site, par exemple pour permettre à plusieurs utilisateurs d'utiliser le même serveur, alors il faudra utiliser des vecteurs comportant une entrée par utilisateur et par site, voire une entrée par session d'édition et par site dans le cas où un utilisateur peut ouvrir plusieurs sessions simultanées.

Le principal obstacle à l'utilisation de ce mécanisme dans notre contexte de wiki P2P est lié à la taille des vecteurs utilisés pour l'étiquetage des messages, et donc échangés avec ces messages. En effet, ces vecteurs ont une taille égale au nombre de sites dans le réseau. Ce nombre de sites étant non borné et variable, la taille des messages est donc impossible à fixer statiquement et également non bornée.

On peut proposer une implantation efficace consistant à considérer que les entrées absentes d'un vecteurs sont équivalentes à une entrée ayant une valeur 0. De cette manière, on permet la dynamicité, et on réduit la taille des vecteur au nombre de sites ayant introduit des modifications. Cependant, cette taille reste non bornée et donc passe difficilement à grande échelle en chargeant de manière très importante les communications sur le réseau.

Historique à base de hachage

Une autre approche, qui ne dépend pas du nombre de sites et donc sensée améliorer le passage à l'échelle, a été proposée dans [35].

Elle consiste à identifier chaque version d'un document avec sa signature digitale, calculée à l'aide d'une fonction de hachage. Chaque version est alors étiquetée avec son histoire, c'est-à-dire l'ensemble des versions dont elle dérive et leurs relations de précédence, dans laquelle les versions sont représentées par leur signature.

Lorsqu'un patch est diffusé, il est étiqueté avec son contexte de production, c'est-à-dire l'étiquette de la version sur laquelle il a été créé. A la réception d'un patch, on peut alors simplement décider en comparant l'étiquette du patch reçu et l'étiquette de la version locale s'il s'agit d'un patch concurrent (la version locale n'appartient pas au contexte de création du patch), ou non (la version locale appartient au contexte de création du patch).

Ainsi, on obtient un étiquetage dont la taille est proportionnelle au nombre de modifications et non plus au nombre de sites. De plus, on peut envisager dans certains cas de couper les histoires associées à chaque version en retirant de ces histoires des versions très anciennes. Le risque est alors de détecter comme concurrents des patchs qui ne le sont pas. Cette taille non proportionnelle au nombre de sites est l'argument principal avancé par les auteurs pour justifier leur approche par rapport aux vecteurs de versions. En réalité, cet argument est très discutable. En effet, le nombre de modifications est lui aussi non borné, et d'autre part, il est forcément supérieur ou égal au nombre de sites introduisant des modifications. Or nous avons vu précédemment qu'il n'est pas compliqué de construire des vecteurs de versions ayant une taille proportionnelle au nombre de sites produisant des changements.

Vecteur d'horloge à taille fixe et Horloges Plausibles

Notre proposition consiste à utiliser des R-Vecteurs, c'est-à-dire des *Vecteurs d'horloge à taille fixe*, comme ceux proposés dans [74]. L'idée est simple, elle consiste à utiliser un vecteur d'horloge exactement comme décrit plus haut, c'est-à-dire avec les mêmes règles de gestion et de comparaison, mais de limiter sa taille à une valeur prédéfinie R. Si le système contient plus de R sites, alors certains sites partagent la même entrée.

On obtient un mécanisme qui possède les propriétés suivantes. Lorsque le nombre de sites N est inférieur à R, le mécanisme se comporte comme un vecteur d'horloge classique, et capture parfaitement la causalité.

Lorsque $N > R$, il est montré dans [74] que le mécanisme des $R-Vecteurs$ est une horloge *plausible*, c'est-à-dire vérifiant les propriétés suivantes :

1. toutes les dépendances causales sont capturées par l'horloge : si $a \rightarrow b$ alors $RV(a) < RV(B)$,
2. l'horloge ne détecte pas de fausses concurrences : si $RV(a) \parallel RV(b)$, alors $a \parallel b$,
3. par contre, il peut arriver que deux événements concurrents dans le système soient rapportés comme causalement liés par l'horloge.

Évidemment, lorsque $N > R$, le R-Vecteur ne capture la causalité que de manière imparfaite, et on obtient un détecteur de concurrence imparfait également. La précision de la détection est cependant difficile à établir et s'évalue expérimentalement. Les auteurs ont rapportés un taux d'erreur stable de l'ordre de 0.20 dans un contexte client/serveur, avec 100 sites et un vecteur de taille 3. Cependant, d'autres paramètres ont une influence sur la précision, dont en particulier les histoires et le type d'échange. Nous ne connaissons pas de résultat établi dans un contexte P2P.

Du point de vue de la réalisation d'un tel mécanisme, lorsque le système contient plus de R sites produisant des patchs, plusieurs sites partagent la même entrée et il faut disposer d'une fonction d'affectation des sites aux entrées qui soit déterministe et donne le même résultat sur tous les sites. Le plus simple est d'utiliser une fonction *modulo*. Ainsi, le site S_i se voit attribuer l'entrée e où $e = i \ mod \ R$. Cependant il est proposé dans [21] d'autres fonctions d'affectation plus efficaces, et on trouve aussi des fonctions réalisant une affectation dynamique et adaptative améliorant la précision.

Un exemple est présenté en Figure 3.4, avec un vecteur de taille ($R = 2$) sur quatre sites. Chaque événement est étiqueté par un vecteur à deux entrées. Dans chaque vecteur, l'entrée 0 est partagée par les sites 0 et 2, et l'entrée 1 est partagée par les sites 1 et 3. Les flèches montrent l'envoi et la réception des messages. Sur le site 3 l'événement $e_{3,1}$ a pour vecteur $< 0, 1 >$; sur le site 0, l'événement $e_{0,2}$ a pour vecteur $< 2, 0 >$. Ces événements sont concurrents puisque qu'il n'existe aucun chemin de causalité les reliant.

Ils sont détectés comme concurrents par le mécanisme car $< 0,1 > \| < 2,0 >$. Par contre, le mécanisme détecte que $e_{3,1}$ sur le site 3 précède $e_{1,2}$ sur site 1 car $< 0,1 > \rightarrow < 0,2 >$ alors que ces 2 événements sont également concurrents.

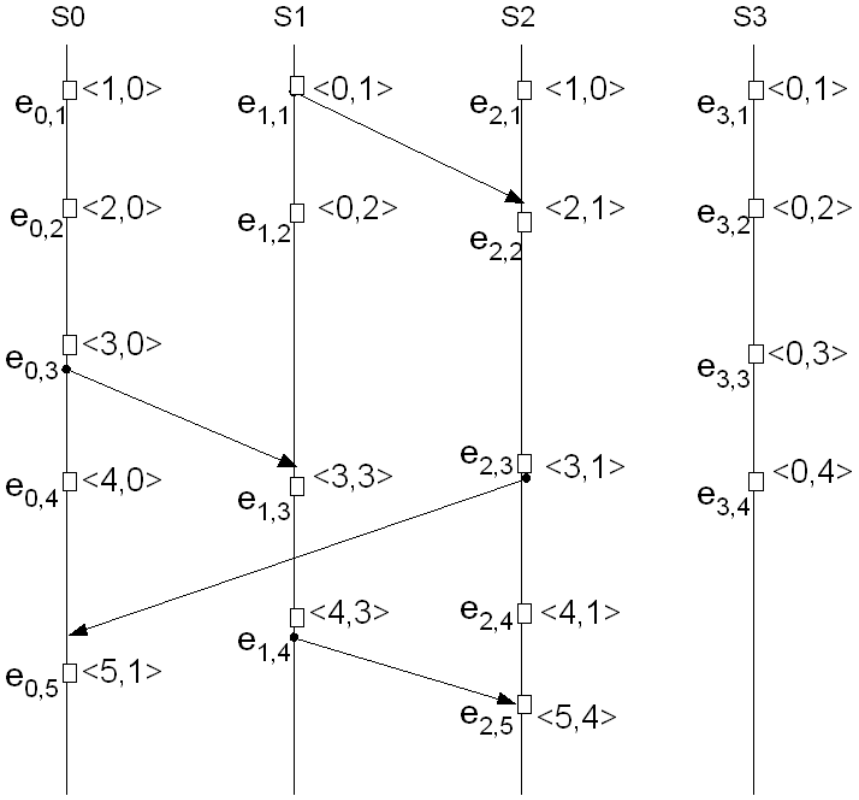

FIG. 3.4 – R-Vecteur avec R=2

Appliqué dans notre contexte, le mécanisme des R-vecteurs nous conduit à un compromis entre précision de la détection et capacité de passage à l'échelle. Même si de nombreux paramètres influent sur la précision, il est cependant clair que dans un contexte donné, plus la taille du vecteur s'approche du nombre de sites et plus la précision est grande. Inversement lorsque la taille du vecteur diminue, la précision diminue également. Aux extrémités, lorsque R=1, le mécanisme se comporte comme une horloge de Lamport et quand R=N le mécanisme se comporte comme un vecteur d'horloges.

Il faut aussi remarquer que lorsque $N > R$, il s'agit d'une horloge plausible et donc la conséquence est d'observer comme causalement reliés des événements concurrents en réalité. La conséquence sur notre mécanisme de conscience de concurrence sera que certains cas de concurrence seront non détectés :

- des pages *fusionnées* rapportées comme *éditées*,
- dans certaines pages *fusionnées*, des patchs oubliés dans l'histoire concurrente et non soulignés dans la visualisation de la page.

Ainsi, le système souffrira de *défaut d'information* transmise à l'utilisateur. Notons qu'en utilisant un mécanisme d'historique à base de hachage utilisant de l'élagage d'histoires pour passer à l'échelle, on obtiendrait l'effet exactement inverse, puisque dans ce cas là des dépendances causales sont rapportées comme concurrentes. Le mécanisme de conscience de groupe rapporterait donc des cas de concurrences non réels.

Notre position est qu'un défaut d'information est préférable et moins perturbant pour l'utilisateur, à condition de rester dans un taux de concurrence non détectée acceptable. Ce taux est difficile à établir, il doit faire l'objet d'une étude expérimentale avec des utilisateurs. En particulier, il serait intéressant d'établir des seuils d'acceptabilité du système et éventuellement déterminer une limite d'intérêt du système, c'est-à-dire un taux à partir duquel le système est pire qu'un système sans conscience de groupe. A l'inverse, il nous semble que surcharger l'utilisateur avec des informations erronées est particulièrement néfaste pour l'acceptabilité du système et la confiance que ses utilisateurs peuvent avoir en lui. Cela reste cependant à établir et à comparer avec le cas inverse de manière expérimentale.

Dans wikipédia la page la plus fréquentée a un nombre de modifications égal à 30000 opérations. Si on considre la taille de vecteur selon le nombre de modifications alors que un vecteur de taille égal 300 entrées peut produire un taux d'erreur 0.20.

Nous pensons donc que ce mécanisme des R-Vecteurs est porteur d'un compromis acceptable entre précision et taille des informations échangées, *dans notre contexte de conscience de la concurrence pour un wiki P2P.*

Dans la suite, nous considérons que c'est ce mécanisme qui est utilisé pour établir la concurrence entre deux patchs, même si pour plus de simplicité nous utilisons parfois des vecteurs d'horloges.

3.2.2 Le calcul des histoires concurrentes

Cette section est dédiée à la notion *d'histoire concurrente* et aux algorithmes associés. Cette notion est à la base de notre mécanisme de conscience de la concurrence, dont le rôle est de visualiser l'impact de cette histoire concurrente dans le contenu d'une page wiki au moment où elle est accédée.

Nous nous appuyons sur un court exemple, pour présenter d'abord un ensemble de définitions avant d'aborder les algorithmes.

Un exemple

L'exemple que nous proposons illustre une histoire concurrente dans laquelle 3 sites sont inter-connectés. Chaque site dispose d'une copie des pages. Le scénario démarre avec un état d'une page wiki partagé par ces trois sites. Cet état initial est présenté en figure 3.5.

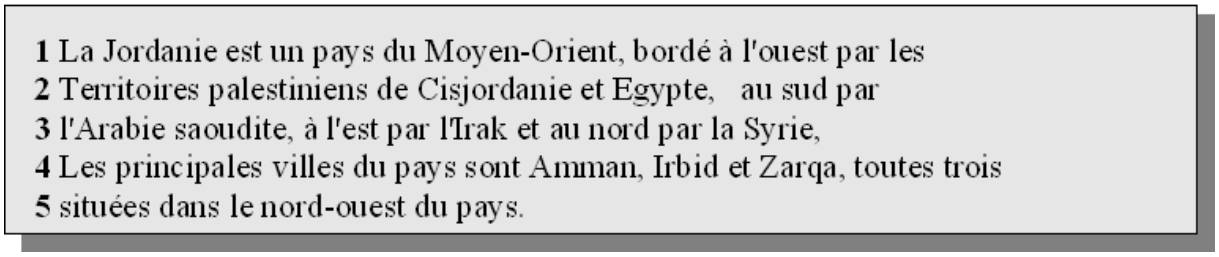

FIG. 3.5 – Exemple : état initial de la page

A partir de cet état initial, trois modifications, correspondant à trois patchs, sont introduites par des utilisateurs de la manière suivante. Un utilisateur connecté au site 1 insère deux lignes entre la ligne 3 et la ligne 4 et sauvegarde. Cette modification correspond au patch p_{n1} dans notre exemple. En même temps, un utilisateur connecté au *site* 2 modifie la ligne 2 et sauvegarde, en produisant le patch p_{n2}. Plus tard, le même utilisateur sur le même site supprime des lignes 4 et 5 et sauvegarde, en produisant le patch p_{n3}. Ces modifications sont présentées dans la figure 3.6.

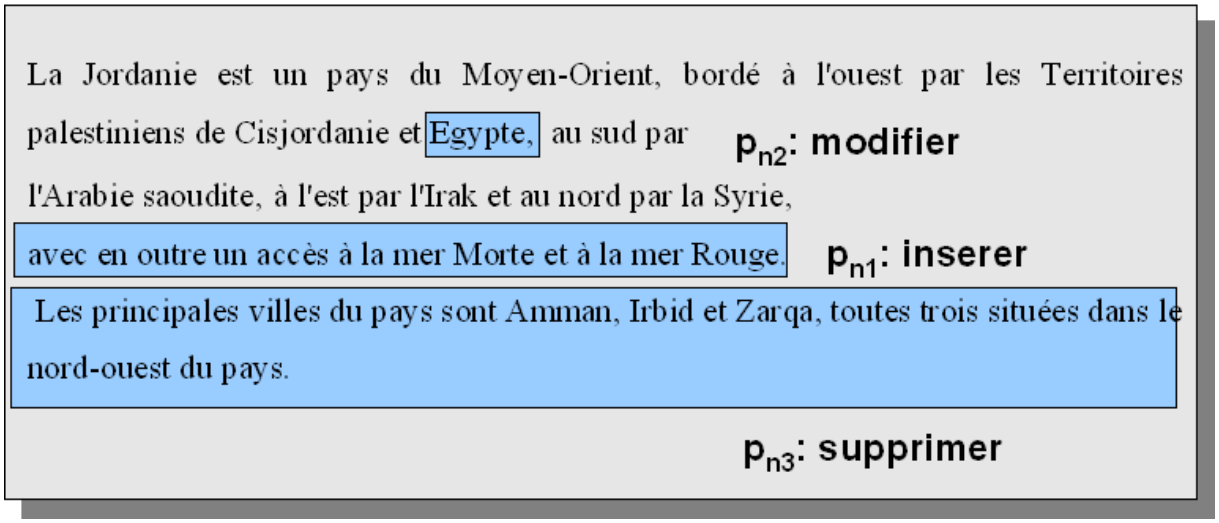

FIG. 3.6 – Exemple : les modifications introduites sur la page

On suppose que l'échange des patchs se produit selon le scénario illustré en figure 3.7. Au début du scénario, on suppose que l'état initial de la page, noté S_n dans la figure ne

contient pas de concurrence, et donc que la page possède le statut *éditée* sur les 3 sites. Sur site 1, le patch P_{n1} est produit, c'est un patch non concurrent sur ce site et l'état obtenu S_{n1} est étiqueté *éditée*. Le patch est ensuite propagé sur le réseau. Au même moment, le site 2 produit le patch P_{n2}, ce qui le conduit à l'état *éditée* S_{n2}, puis il propage P_{n2} sur le réseau. Ensuite, ce même site 2 produit le patch P_{n3} et le propage. L'état obtenu sur site 2 est S_{n23}, étiqueté *éditée*.

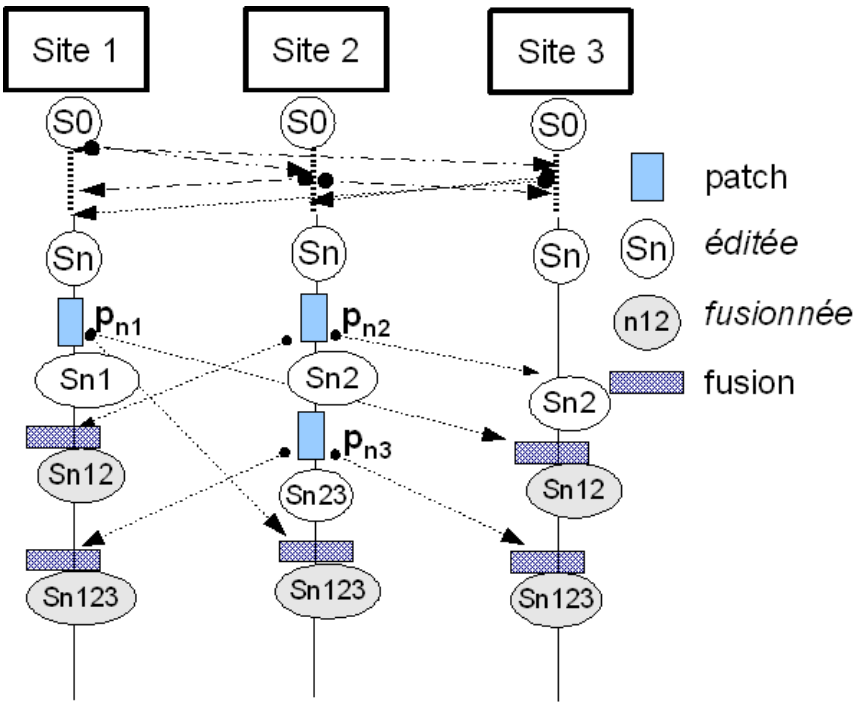

FIG. 3.7 – Exemple : échange des patchs

Sur le *site* 1, lorsque P_{n2} est reçu, son intégration nécessite, évidemment, une fusion avec P_{n1}, et l'état obtenu de cette fusion S_{n12} a le statut *fusionné*, représenté en gris sur la figure 3.7.

La même situation se produit lors de la réception de P_{n3} sur le *site* 1. L'état obtenu S_{n123} a le statut *fusionné*.

Définitions

Nous présentons maintenant quelques définitions sur la concurrence des états et patchs. Certaines de ces définitions sont inspirées de [69].

Definition 1 *État d'une page*

L'état d'une page P, noté PS, est défini par l'ensemble de patchs appliqués à cette page :

1. L'état initial de la page est $PS = \{\}$.
2. Un patch P appliqué à une page transforme son état PS en un état $PS' = PS \bigcup \{P\}$.

Dans notre exemple, l'état de la page S_0 est $\{\}$, et l'état de la page S_n est l'ensemble de patchs appliqués pour obtenir l'état S_n. Ainsi, $S_n = \{p_0, \ldots p_n\}$. De même, les états apparaissant suite à la production et l'échange des patchs p_{n1}, p_{n2}, p_{n3} sont :

- $S_{n1} = S_n \bigcup \{p_n 1\} = \{p_0, \ldots p_n, p_{n1}\}$,
- $S_{n2} = S_n \bigcup \{p_{n2}\} = \{p_0, \ldots p_n, p_{n2}\}$,
- $S_{n12} = \{p_0, \ldots p_n, p_{n1}, p_{n2}\}$,
- $S_{n23} = \{p_0, \ldots p_n, p_{n2}, p_{n3}\}$,
- $S_{n123} = \{p_0, \ldots p_n, p_{n1}, p_{n2}, p_{n3}\}$

Basé sur cette définition de l'état d'une page, nous pouvons définir le contexte de génération d'un patch, qui capture l'état sur lequel le patch a été produit et l'état actuel.

Definition 2 *Contexte de génération d'un patch*

Pour un patch P, son contexte de génération $GC(P)$ est $GC(P) = PS \bigcup \{P\}$, où PS est l'état de la page sur lequel P a été produit, c'est-à-dire l'état de la page au moment où l'utilisateur a demandé son édition.

Ainsi, dans notre exemple, cette définition nous donne :

- $GC(p_{n1}) = \{p_0, \ldots, p_n, p_{n1}\}$,
- $GC(p_{n2}) = \{p_0, \ldots, p_n, p_{n2}\}$,
- $GC(p_{n3}) = \{p_0, \ldots, p_n, p_{n2}, p_{n3}\}$,

La notion du contexte de génération d'un patch permet l'introduction de la relation de précédence entre les patchs qui sont liés par la relation causale. Cette relation de précédence traduit la causalité.

Definition 3 *Précédence des patchs,* $\rightarrow$

Soient deux patchs P_i et P_j, on dit que P_i précède P_j, au sens causal, et on note $P_i \rightarrow P_j$: *si et seulement si* $P_i \neq P_j$ et $GC(P_i) \subset GC(P_j)$.

Revenons à notre exemple, on a $GC(p_{n2}) \subset GC(p_{n3})$, d'où on conclut que $p_{n2} \rightarrow p_{n3}$.

Nous pouvons maintenant définir la notion de concurrence entre deux patchs. De manière usuelle, on dit que deux patchs sont en concurrence (ou conflictuels) s'ils ne sont pas liés par une relation de précédence causale.

Definition 4 *Concurrence de Patchs,* $\|$

Soient deux patchs P_i et P_j, P_i est concurrent avec P_j, $P_i \parallel P_j, ssi$ ni $P_i \rightarrow P_j$ ni $P_j \rightarrow P_i$.

Dans notre exemple, Nous avons $p_{n1} \parallel p_{n2}$ et $p_{n1} \parallel p_{n3}$.

Pour des raisons pratiques, nous étendons cette définition de la concurrence de patchs à la notion de concurrence patch/état.

Definition 5 *Concurrence état/patch* $\parallel$

Un patch est concurrent avec un état d'une page si cet état n'est pas inclus dans le contexte de génération du patch :

Soient un P_i et une page dans l'état PS. On dit que P_i est concurrent avec PS, *ssi* $PS \not\subset GC(P_i)$

Par contre, un patch est non concurrent avec l'état de la page si l'état est inclus dans le contexte de génération de ce patch. Cela capture l'idée que cet état a été vu quand le patch est produit. De cette définition de la concurrence état/patch, nous pouvons maintenant donner une définition précise du statut d'une page du point de vue de la concurrence.

Definition 6 *Page* ***fusionnée***

On dit qu'une page a le statut *fusionnée* dans le cas où le dernier patch appliqué était un patch concurrent. Plus précisément, la page est *fusionnée ssi* son état actuel PS a été obtenu par l'application d'un patch P sur l'état précédent PS' et $PS' \parallel P$.

Definition 7 *Page* ***éditée***

On dit qu'une page a le statut *éditée* dans le cas où le dernier patch appliqué était un patch *non* concurrent. Plus précisément, la page est *éditée ssi* son état actuel PS obtenu de l'application d'un patch P sur l'état précédent PS' et $PS' \nparallel P$.

Revenons à notre exemple pour examiner la situation sur les différents sites.

Sur le *site* 1, après intégration du patch P_{n1}, on obtient l'état S_{n1} étiqueté comme *édité*. En effet, l'état S_n fait évidemment partie du contexte du patch P_{n1} et donc ce patch n'est pas concurrent. L'application d'un patch local conduit *toujours* à un état *édité*. A l'arrivée du patch P_{n2}, celui-ci est détecté comme concurrent. En effet, $p_{n2} \parallel n_1$ car $n_1 \not\subset GC(p_{n1})$. L'état produit est étiqueté comme *fusionnée*. De même, le patch P_{n3} est détecté comme concurrent à son arrivée et l'état final S_{n123} est étiqueté *fusionnée*.

Sur le *site* 2, P_{n2} et P_{n3} sont produits localement et en séquence. Ils ne sont donc pas concurrents, et l'état S_{n23} doit être étiqueté comme *édité*. Notre définition est cohérente avec cette observation, puisque $n_2 \nparallel p_{n3}$. En effet, $GC(p_{n3}) = \{p_0, \ldots, p_n, p_{n2}, p_{n3}\}$,

et $n_2 = \{p_0, \ldots, p_n, p_{n2}\}$, d'où $n_2 \subset GC(p_{n3})$. Par contre, lorsque P_{n1} arrive, il est détecté comme concurrent, puisque $GC(p_{n1}) = \{p_0, \ldots p_n, p_{n1}\}$ alors que $S_{n23} = \{p_0, \ldots p_n, p_{n2}, p_{n3}\}$. Ainsi, l'état S_{n123} est également étiqueté comme *fusionnée* sur le site 2.

La situation est un peu différente sur *site* 3 puisqu'il n'a produit aucun patch et ne fait que recevoir et intégrer ceux produits par *site* 1 *et* 2. Quand P_{n2} arrive, il est considéré comme non concurrent et l'état obtenu après l'intégration S_{n2} est *édité*. En effet , $S_n = \{p_0, \ldots, p_n\}$, $GC(p_{n2}) = \{p_0, \ldots, p_n, p_{n2}\}$, et $Sn \subset GC(p_{n2})$. Quand P_{n1} est reçu, il est considéré comme concurrent, et l'état obtenu S_{n12} comme *fusionnée*. Cela vient de $S_{n2} = \{p_0, \ldots, p_n, p_{n2}\}$ alors que $GC(p_{n1}) = \{p_0, \ldots, p_n, p_{n1}\}$. Ici, on a $p_{n1} \parallel p_{n2}$, ainsi que $p1 \parallel n_2$. Lorsque P_{n3} est reçu, il est aussi détecté comme concurrent puisque $S_{n12} = \{p_0, \ldots p_n, p_{n1}, p_{n2}\}$, alors que $GC(p_{n3}) = \{p_0, \ldots, p_n, p_{n2}, p_{n3}\}$. L'état qui en résulte, S_{n123} est, sur ce site aussi, étiqueté comme *fusionnée*.

Les définitions proposées pour spécifier le statut d'une page, *fusionnée* ou *éditée* donnent un résultat indépendant de l'ordre de réception des patchs, et donc du site sur lequel le statut est calculé. En effet, l'état S_{n123} est étiqueté avec le statut *fusionnée* qu'il soit produit par la séquence $p_{n1}; p_{n2}; p_{n3}$ (sur le site 1), la séquence $p_{n2}; p_{n3}; p_{n1}$ (sur le site 2) ou la séquence $p_{n2}; p_{n1}; p_{n3}$.

Discutons maintenant de la définition de la notion d'histoire concurrente. Cette notion représente l'ensemble des patchs dont les effets devront être marqués par le mécanisme de conscience de concurrence lors d'un accès à une page. Lorsque la page possède un statut *éditée*, cet ensemble doit être vide. Lorsque la page possède un statut *fusionnée*, cet ensemble est un sous-ensemble de l'état de la page, contenant au moins le dernier patch intégré (celui qui a conduit à une fusion) ainsi qu'un patch qui lui est concurrent.

Une première vision intuitive de la notion d'histoire concurrente d'un état consiste à la définir comme étant formée :

- du patch produisant cet état
- et de l'ensemble des patchs de cet état qui lui sont concurrents.

Ainsi, cette définition utilisée dans le cas de l'état S_{n2} :

- sur le site 1, lorsque P_{n2} arrive et est détecté comme concurrent, on a $p_{n1} \parallel p_{n2}$. L'ensemble $\{p_{n1}, p_{n2}\}$ est l'histoire concurrente de l'état S_{n2} et l'effet de ces deux patchs sera marqué. Notons que ces deux patchs ont un ancêtre commun, l'état S_n, puisqu'il sont tous les deux produits sur cet état.
- sur le site 2, la situation est analogue. Lorsque P_{n1} arrive, il est détecté comme concurrent, on a $p_{n1} \parallel p_{n2}$. L'ensemble $\{p_{n1}, p_{n2}\}$ est l'histoire concurrente de l'état S_{n2}.

Cependant, une telle définition n'est pas satisfaisante dans des situations plus complexes. Par exemple, dans le cas de l'état S_{n123} :

- sur le site 1, le dernier patch intégré est p_{n3} et nous avons juste $p_{n3} \parallel p_{n1}$, et donc l'histoire concurrente serait $\{p_{n3}, p_{n1}\}$, P_{n2} en étant exclu. Cependant, il semble clair que l'état S_{n123} résulte de la concurrence entre p_{n1} et la séquence $(p_{n2}\ ;\ p_{n3})$. Le patch p_{n2} devrait appartenir à cette histoire concurrente.
- sur le site 3, la situation est exactement identique à celle du site 1. La définition ci-dessus conduit à exclure P_{n2} de l'histoire concurrente.
- sur le site 2, le dernier patch intégré est P_{n1} et nous avons $p_{n3} \parallel p_{n1}$, et $p_{n2} \parallel p_{n1}$. Donc, notre définition conduit à l'histoire concurrente formée de $\{p_{n3}, p_{n2}, p_{n1}\}$.

Ainsi, la définition intuitive conduit à des résultats incomplets par rapport aux objectifs, et de plus des résultats différents selon les sites et l'ordre d'arrivée des patchs.

Notre approche est la suivante. Lorsqu'un patch P est appliqué à l'état PS, pour obtenir l'état PS', nous extrayons de l'historique de PS' tous les patchs postérieurs à l'état qui a été vu par P et tous les patchs concurrents en PS. Cet état est l' ancêtre commun, c'est-à-dire l'état duquel P et tous ses patchs concurrents proviennent.

Cet état est défini par l'intersection du contexte de génération de P avec le contexte de génération de ses patchs concurrents. Nous pouvons maintenant définir la notion d'histoire concurrente.

Definition 8 *Histoire Concurrente d'un état PS*

L'Histoire Concurrente d'un état PS, produit par application d'un patch P sur un état S, noté $HC(PS)$, est défini comme $HC(PS) = PS - \{GC(P) \bigcap GC(p_i)\}$, où $p_i \in S$ et $p_i \parallel P$.

Il faut noter que si $S \nparallel P$, c'est-à-dire si l'état PS possède le statut *édité*, alors $HC(PS) = \{\}$. L'historique concurrent d'un état est non vide uniquement si cet état possède le statut *fusionnée*.

Dans notre exemple, nous avons maintenant :

- sur le site 1, lorsque P_{n3} est intégré, nous avons $HC(S_{n123}) = \{p_{n1}, p_{n2}, p_{n3}\}$. En effet, $p_{n3} \parallel p_{n1}$, et $GC(p_{n3}) = \{p_0, \ldots p_n, p_{n2}, p_{n3}\}, GC(p_{n1}) = \{p_0, \ldots p_n, p_{n1}\}$, donc l'état ancêtre commun de $\{p_{n1}, p_{n3}\}$ est $GC(p_{n3}) \bigcap GC(p_{n1}) = \{p_0, \ldots p_n\}$. Ainsi, tous les patchs reçus après l'état initial font partie de l'histoire concurrente.
- sur le site 3, une situation analogue se produit,
- sur le site 2, lorsque P_{n1} est intégré, nous avons également $HC(S_{n123}) = \{p_{n1}, p_{n2}, p_{n3}\}$, puisque $p_{n3} \parallel p_{n1}$, et $p_{n2} \parallel p_{n1}$,. L'ancêtre commun à ces trois patchs est de manière évidente l'état S_n.

Il est important de noter que l'histoire concurrente d'un état dépend seulement des patchs constituant cet état, mais est indépendant du site et de l'ordre de réception ou d'intégration des patchs. Dans la situation finale de notre exemple, les trois sites atteignent le même état et l'histoire concurrente calculée sur chacun d'eux est identique.

Algorithmes

D'un point de vue algorithmique, on peut envisager une implantation directement inspirée des définitions précédentes qui consisterait à étiqueter chaque patch émis sur le réseau avec son contexte de génération et à utiliser des opérations ensemblistes pour réaliser les tests de concurrence et le calcul de l'histoire concurrente. Une telle approche est en réalité très voisine de l'approche des histoire de hachage et ne supporte pas un passage à l'échelle : les contextes de génération des patchs sont non bornés et grandiraient sans limite.

Comme indiqué dans la section 3.2.1, nous adoptons une approche basée sur les R-Vecteurs. Dans cette approche, chaque site maintient un R-Vecteur par page pour caractériser l'état courant de la page. Les patchs sont également étiquetés avec un R-Vecteur. Cette étiquette est posée par le site sur lequel le patch est créé, après application locale du patch. Elle a pour valeur le vecteur caractérisant la page après cette application. On suppose également que le patch porte avec lui l'identifiant de la page sur laquelle il s'applique.

Pour calculer l'historique, deux opérations sont nécessaires :

1. *computeConcStatus*(*Patch*) est appelée chaque fois qu'un patch, local ou distant, est intégré à une page. Cette opération calcule l'état concurrent de la page.

   computeConcStatus(P)
   PageId = P.getPageId()
   Page = Storage.getPageById(PageId)
   if isConcurrent(Page, P) **then**
   return (**fusionnée**)
   else
   return (**éditée**)
   end if

2. *getConcurrentHistory*(*PageId*) est appelée lorsqu'une page ayant le statut *fusionnée* est demandée. Cette opération extrait l'histoire concurrente du log de patchs de la page.

   getConcurrentHistory(PageId)

```
Log = Storage.getLog( PageId)
Patch lp = Log.getLastPatch()
ResultSet = {lp}
for all patch P_i ∈ Log do
  if isConcurrent(lp,P_i) then
    ResultSet = ResultSet ⋃{P_i}
  end if
end for
Ac = getCommonAncestor(ResultSet)
for all patchs P_j ∈ Log, Ac ← P_j do
  resultSet = ⋃{P_j}
end for
return( ResultSet )
```

L'histoire concurrente est extraite par l'algorithme *getConcurrentHistory*. Cet algorithme extrait du log le dernier patch qui est appliqué sur la page, puis teste la concurrence entre ce patch et tous les patchs dans le log. Chaque patch concurrent est alors ajouté à l'ensemble *ResultSet*. Ensuite, L'algorithme calcule l'ancêtre commun pour les patchs dans *ResultSet*. Pour terminer, l'algorithme ajoute tous les patchs postérieurs à cet état dans l'ensemble *ResultSet*.

Les algorithmes présentés ci-dessus utilisent deux fonctions dépendantes du mécanisme de détection de concurrence utilisée : *isConcurrent(O1, O2)* qui teste la concurrence entre deux objets, page ou patch et *getCommonAncestor(E)* qui calcule l'ancêtre commun d'un ensemble de patchs. Notre implantation étant basée sur des R-Vecteurs, les tests de concurrence sont les tests classiques des vecteurs d'horloge.

isConcurrent(O1, O2) : Boolean : Teste la concurrence en comparant les deux vecteurs portés par les objets O1 et O2. On suppose que ces objets proposent une interface permettant d'obtenir le vecteur porté et l'identifiant du site d'origine. Le fait de connaître le site d'origine des événements comparés permet de réaliser un test de concurrence très simple :

```
isConcurrent(O1, O2) :
V1 = O1.getVector() ; S1 = O1.getSiteId() ;
V2 = O2.getVector() ; S2 = O2.getSiteId() ;
if (V1.get(S1) > V2.get(S1) AND V2.get(S2) > V1.get(S2) ) OR (V1 == V2) then
  return true
else
```

return false
end if

Deux remarques sont à faire sur cet algorithme. D'une part, afin d'atténuer (un peu) les risques de non détection de concurrence du mécanisme des R-Vecteurs, deux objets différents porteur d'un vecteur identique sont considérés comme concurrents. D'autre part, le gestion de la dynamicité est assurée par la méthode d'accès dans les vecteurs : lorsqu'un accès à la valeur pour une entrée non présente dans le vecteur est demandé, la méthode retourne la valeur 0 et ajoute cette entrée dans le vecteur.

getCommonAncestor({P_i}) : Vector ; calcule l'ancêtre commun d'un ensemble de patchs. Ce calcul est simple, il consiste à retenir, pour chaque entrée la valeur minimum trouvée dans l'ensemble. La prise en compte de la dynamicité conduit à rechercher cette valeur minimum pour toutes les entrées de tous les vecteurs de l'ensemble.

getCommonAncestor(ResultSet) :
allEntries = $\bigcup$ $p_i.getEntries()$, *where* $p_i \in ResultSet$
for All $k \in allEntrie$ **do**
  for All $P_j \in ResultSet$ **do**
    Common.setValue(k) = MIN(P_j.get(k), Common.get(k))
  end for
end for
return Common

3.3 Mécanismes pour la conscience de la concurrence

Nous présentons maintenant plus en détail le fonctionnement général de notre mécanisme de conscience de groupe. Ce mécanisme affecte les traitements principaux réalisés par l'application de gestion du wiki, c'est-à-dire :

1. la réception d'un patch distant,
2. la requête de visualisation d'une page,
3. la requête d'édition d'une page,
4. la requête de sauvegarde d'une page éditée.

Rappelons brièvement les principes de fonctionnement de ce mécanisme.

1. le mécanisme gère des étiquettes posées sur les pages et les patchs pour réaliser des calculs de détection de concurrence. Ces étiquettes sont des R-Vecteurs,

2. le mécanisme s'appuie sur un log des patchs intégrés, stocké sur chaque serveur,
3. à chaque réception d'un patch distant, ce patch est intégré à la page concernée, et le statut de la page (*fusionnée, éditée*) est mis à jour,
4. à chaque accès à une page, si la page possède le statut *fusionnée*, l'histoire concurrente de la page est calculée. Puis, lors de l'extraction du contenu de cette page, les lignes concernées par cette histoire concurrente sont décorées à l'aide de marques dédiées,
5. la visualisation de la page est réalisée par un moteur de rendu html adapté pour prendre en compte les marques dédiées à la conscience de la concurrence.

Nous examinons maintenant les différents traitements, en commençant par le cas de la réception d'un patch et en poursuivant par le cas des requêtes en provenance de l'utilisateur.

3.3.1 Réception d'un patch distant

La réception d'un patch distant consiste à intégrer le patch dans l'état courant de la page, et à effectuer quelques calculs liés à la conscience de la concurrence, à savoir :

1. détermination du statut de la page, ceci est réalisé en appelant la fonction décrite plus haut,
2. calcul de la valeur de l'étiquette associée au nouvel état de la page. En utilisant des R-Vecteurs, il s'agit là du processus classique de réception d'un événement distant dans les vecteurs d'horloge, qui consiste à prendre le MAX de chaque entrée entre le vecteur local porté par la page et le vecteur distant porté par le patch reçu.
3. insertion du patch reçu dans le log pour calcul ultérieur des histoires concurrentes.

L'algorithme est donc le suivant :

on **RECEIVE(Patch P)** :
PageId = P.getPageId()
Page = Storage.getPageById(PageId)
Page.setStatus(computeConcStatus(P))
Storage.applyPatch(PageId, P)
Storage.appendToConcurrencyLog(P)
VLOC = Page.getVector() ; VDIST = Page.getVector()
for all k $\in VLOC.getEntries() \bigcup VDIST.getEntries()$ **do**
  VLOC.setValue(k) = MAX(VLOC.get(k), VDIST.get(k))
end for

Page.setVector(VLOC)

3.3.2 Demande de visualisation d'une page wiki

Lorsque le serveur reçoit la demande $GET(pageId)$ de l'utilisateur, le contenu de la page correspondante est extrait du système de stockage. Lorsque le statut de la page possède la valeur *fusionnée*, l'histoire concurrente de la page est calculée et utilisée pour décorer le contenu de la page à l'aide de marques dédiées. Ce contenu est ensuite passé au moteur de rendu HTML qui transforme la syntaxe wiki, ainsi que les marques de conscience de concurrence en balises html. Le texte html est alors retourné au navigateur dont est issue la demande.

L'algorithme correspondant à ce traitement est le suivant :

on **GET(PageId)** :
Page = Storage.getPageById(PageId)
if Page.getStatus() == **fusionnée then**
  HC = Storage.getConcurrentHistory(Page)
  pageContent = Page.extractContentAndDecorate(HC)
else
  pageContent = Page.extractContent()
end if
return(Renderer.render(pageContent))

Notre mécanisme de visualisation présente les informations de conscience de la concurrence à l'utilisateur en marquant les effets de la partie concurrente de l'historique dans la page qu'elle s'affiche quand la page demandée est *fusionnée*. Le fait qu'une page possède le statut *fusionnée* est indiqué par un marqueur placé à côté de son titre.

L'approche que nous adoptons pour marquer les effets des patchs concurrents est la suivante :

1. les lignes non affectées par l'histoire concurrente apparaissent normalement, sans aucune modification visuelle,
2. les lignes affectées par l'histoire concurrente apparaissent avec une couleur de fond ; on choisit une couleur de fond par patch dans l'histoire concurrente,
3. les lignes supprimées par une opération de l'histoire concurrente sont gardées visibles dans le document, mais sont barrées par une ligne fine,
4. les lignes insérées par une opération de l'histoire concurrente sont en gras,
5. les lignes mises à jour par des opérations de l'histoire concurrente apparaissent avec

l'ancienne valeur comme une ligne supprimée et la nouvelle valeur comme une ligne insérée.

Pour illustrer ces principes de visualisation des informations de conscience de la concurrence, nous utilisons notre exemple présenté dans la section 3.2.2. L'état initial de la page est présenté en figure 3.8. La page est dans l'état *éditée* est la visualisation est celle d'un wiki classique.



FIG. 3.8 – Exemple : état initial de la page

L'état final de la page, après intégration des 3 patchs de l'exemple, est présenté en figure 3.9. La page possède le statut *fusionnée* : cette indication correspond au marqueur placé à côté du titre. Trois patchs concurrents sont visualisables sur cette illustration, correspondant chacun a une couleur de fond différente.

La visualisation des lignes de la page impactées par l'histoire concurrente est réalisée grâce à l'insertion de marques dédiées dans la page wiki au moment de son extraction. Ces marques sont complémentaires à la syntaxe wiki et interprétées par le moteur de rendu pour être traduites en html. Ces marques encadrent chaque ligne insérée ou supprimée par une opération de l'histoire concurrente, et possèdent un certain nombre d'attributs

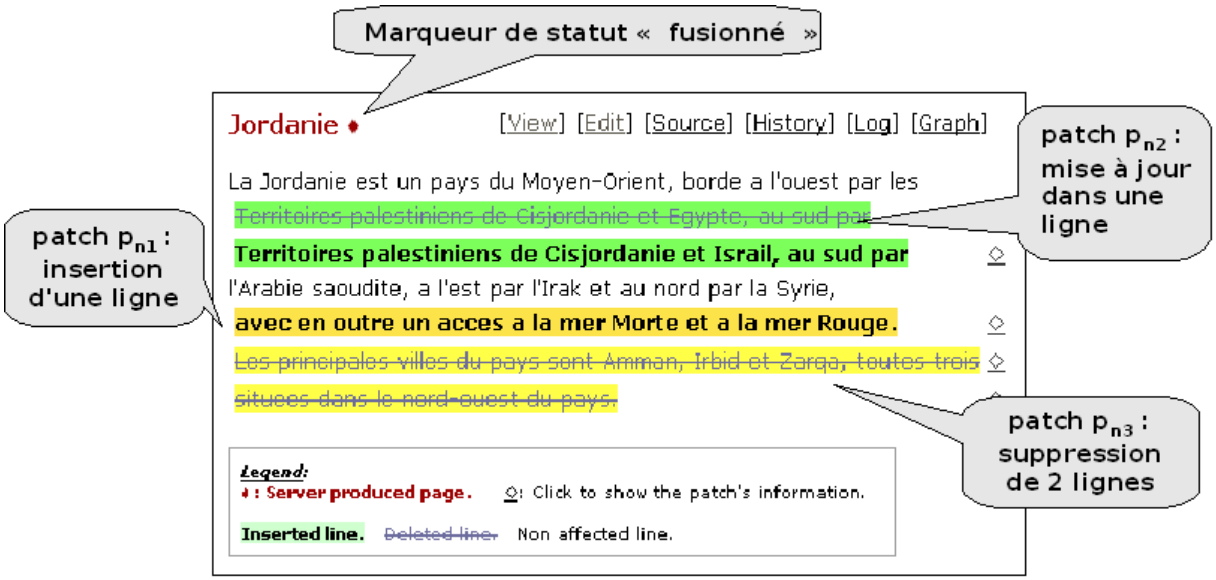

FIG. 3.9 – Exemple : visualisation de l'état final du scénario

permettant d'ajouter des informations supplémentaires : identifiant du patch, identifiant du site producteur du patch, dates de création et d'intégration du patch. Deux paires de marques sont utilisées (la syntaxe est peu significative ; non utilisons ici la syntaxe choisie pour le prototype) :

- la paire **{inserted *atts*}**, ...**{/inserted}**, encadrant une ligne insérée,
- la paire **{deleted *atts*, ...}**, encadrant une ligne supprimée.

Il faut noter que deux marques distinctes sont utilisées pour délimiter une ligne insérée, alors qu'une ligne supprimée apparaît comme un attribut d'une seule marque. Nous reviendrons sur ce choix dans la section consacrée au traitement de la sauvegarde. Il est dû au fait qu'une ligne insérée est effectivement dans le contenu de la page : les marques ne font que la décorer, alors qu'une ligne supprimée ne fait pas partie du contenu du page, mais fait partie de la décoration. Nous sommes ainsi cohérents avec le souhait de simplement ajouter des annotations au contenu de la page sans le modifier réellement.

L'algorithme d'extraction et décoration du contenu de la page fonctionne de la manière suivante. Il parcourt le stockage Woot de la page ligne par ligne. Pour chaque ligne, il regarde s'il existe une opération de l'histoire concurrente qui modifie cette ligne. Selon les cas, il réalise :

1. ligne invisible, non concernée par l'histoire concurrente : rien,
2. ligne visible, non concernée par l'histoire concurrente : ajout de la ligne dans le texte résultat,
3. ligne invisible, supprimée par une opération de l'histoire concurrente : ajout du

marqueur ouverture de suppression, ajout de la ligne, ajout du marqueur fermeture de suppression,

4. ligne visible, insérée par une opération de l'histoire concurrente : ajout du marqueur ouverture d'insertion, ajout de la ligne, ajout du marqueur fermeture d'insertion.

3.3.3 Demande d'édition d'une page wiki

Les pages ayant le statut *fusionnée* peuvent évidemment être éditées comme n'importe quelle page régulière. Dans de nombreux cas, la page *fusionnée* est éditée pour corriger les incohérences dues à la fusion. Pour aider les utilisateurs dans cette tâche, les marques de conscience de concurrence sont également insérées dans le texte wiki édité.

Lorsque le serveur reçoit la requête $EDIT(pageId)$, il extrait le contenu de la page et y insère les marques comme il le fait dans le cas d'une requête de type GET. Ce contenu n'est cependant pas transmis au moteur de rendu HTML, mais directement renvoyé au client pour être édité en syntaxe wiki. De plus, ce contenu est stocké dans un espace temporaire. Cette valeur stockée servira, lors de la sauvegarde après édition, à calculer le patch. De même, le vecteur associé à la page est sauvegardé, de manière à pouvoir calculer le vecteur associé au patch lors de la sauvegarde.

L'algorithme de traitement de la requête EDIT est le suivant :

```
on EDIT( PageId ) :
Page = Storage.getPageById(PageId)
if Page.getStatus() == fusionnée then
  HC = Storage.getConcurrentHistory(Page)
  pageContent = Page.extractContentAndDecorate(HC)
else
  pageContent = Page.extractContent()
end if
Storage.save(PageId, pageContent, page.getVector())
return( pageContent )
```

La figure 3.10 illustre la zone d'édition de texte pour un utilisateur ayant demandé la modification de la page wiki de notre exemple, dans l'état final où les trois patchs sont intégrés.

Le texte en syntaxe wiki est accompagné des marques de conscience de la concurrence. Les zones encadrées par ces marques correspondent aux lignes touchées par la fusion. Ces marques incluent des informations complémentaires : identifiant du patch, date de création du patch, site de création du patch.

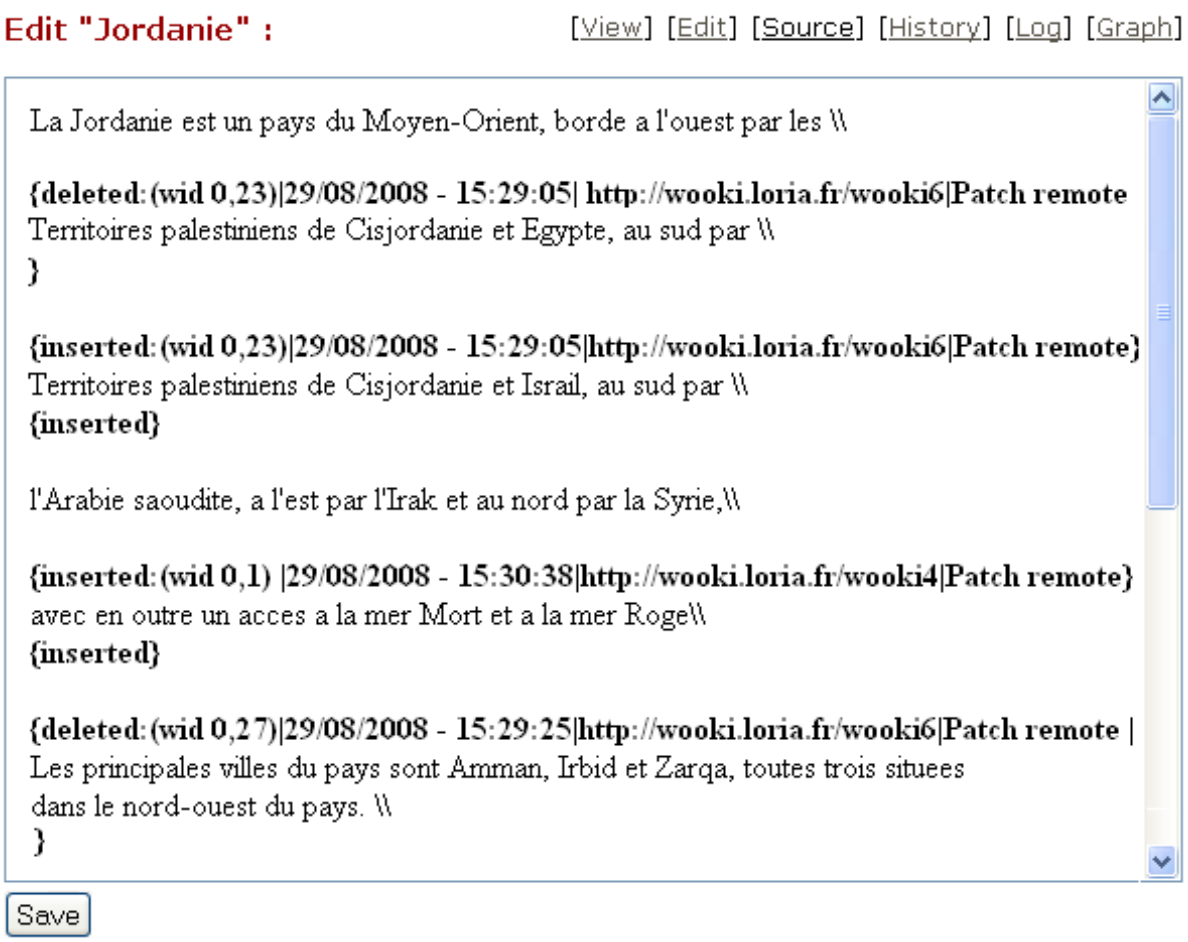

FIG. 3.10 – Exemple : Édition de la page wiki

Pour modifier le contenu de la page, l'utilisateur édite le texte en syntaxe wiki. Il peut bien sûr apporter des modifications sur les zones fusionnées et encadrées par des marques. Le principe d'édition des zones marquées est que les marques seront toutes retirées complètement du texte au moment de la sauvegarde. Une ligne encadrée par des marques d'insertion apparaîtra donc comme une ligne classique non décorée, alors qu'une ligne incluse dans une marque de suppression sera retirée avec la marque.

Ainsi, s'il veut supprimer une ligne marquée par une balise d'insertion, il lui suffit de couper la ligne de texte. Il peut également couper, ou conserver, les marques d'insertion.

S'il veut rétablir une ligne marquée par une suppression, il lui suffit de supprimer les marques de suppression encadrant cette ligne. Cette ligne peut alors être modifiée. Un exemple de résultat d'édition, avant sauvegarde, est donné en figure 3.11. Il correspond au cas où un utilisateur éditant la page dans son état final (trois patchs intégrés) fait une modification destinée à ré-insérer les deux dernières lignes supprimées auparavant.

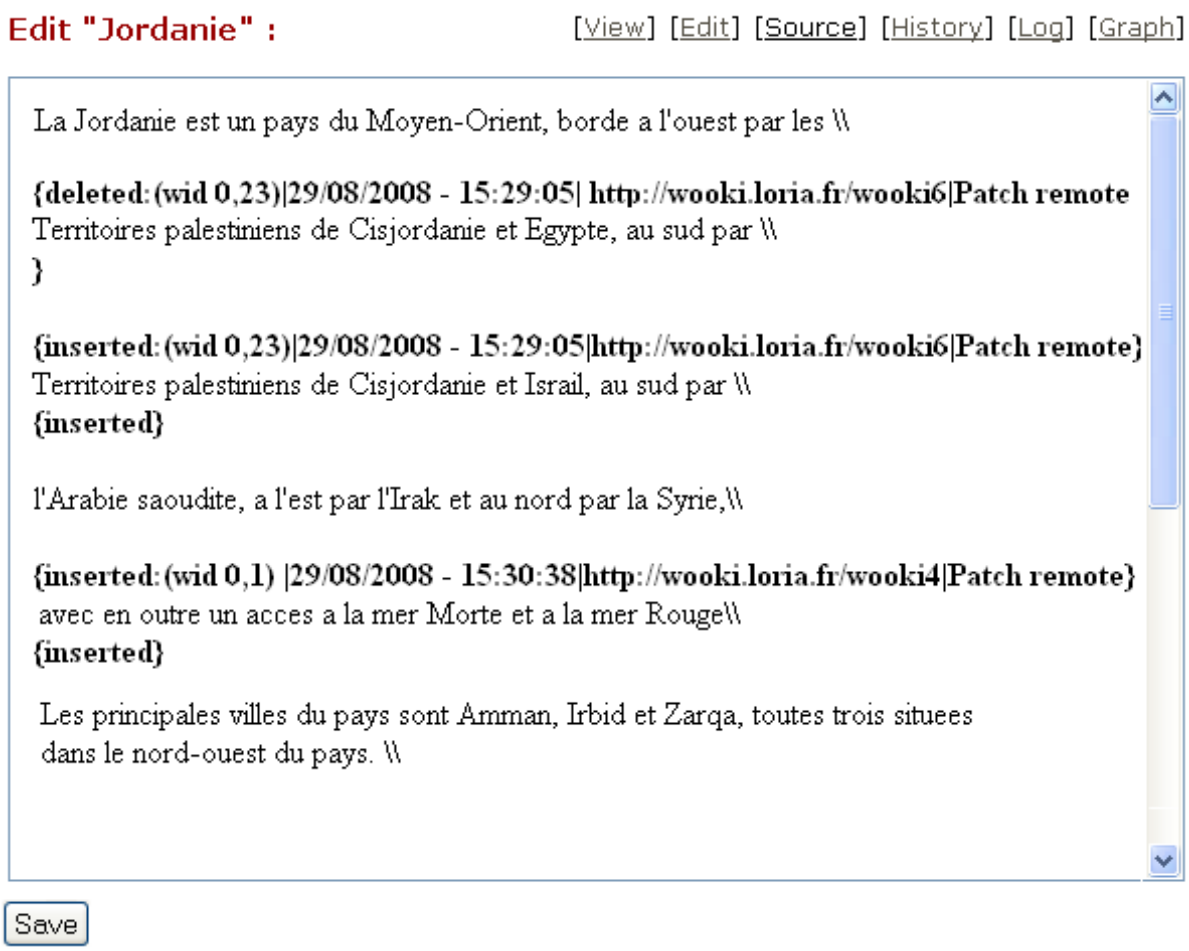

FIG. 3.11 – Exemple : fin d'édition de la page wiki

3.3.4 Demande de sauvegarde d'une page modifiée

La sauvegarde d'une page éditée consiste à transmettre un nouveau contenu pour la page depuis le client d'édition vers un serveur wiki. Ce serveur se charge alors de calculer le patch correspondant à la modification, puis d'intégrer localement le patch avant de la diffuser sur le réseau de pairs.

Dans notre mécanisme, ce processus est modifié pour ajouter une opération de filtrage destinée à retirer des textes toutes les éventuelles marques d'insertion ou suppression concurrente de lignes. Ce filtrage est nécessaire pour pouvoir calculer le patch sans perturbation de la part du mécanisme de conscience de la concurrence. Ce filtrage est appliqué à la fois au nouveau contenu transmis par le client et à l'ancien contenu stocké temporairement sur le serveur. D'autre part, lors de la création du patch, il est nécessaire de calculer la valeur du vecteur associé. Ceci est fait en incrémentant l'entrée correspondant au site local dans le vecteur sauvegardé de manière temporaire. Enfin, l'intégration et la diffusion

du patch est réalisé en appelant la requête RECEIVE. L'algorithme de traitement d'une demande de sauvegarde est le suivant :

```
on SAVE( PageId, pageContent ) :
oldContent = Filter(Storage.loadContent(PageId))
oldVector = Storage.loadVector(PageId)
newContent = Filter( pageContent )
patch.setListop( Diff( oldContent, pageContent ) )
patch.setPageId(PageId)
newVector = oldVector
newVector.setValue(SiteId, oldVector.getEntry(SiteId)+1)
patch.setVector(newVector)
RECEIVE( patch )
```

La figure 3.12 illustre le processus de filtrage, appliqué à l'exemple. La version sauvegardée avant édition et la version nouvelle sont toutes deux filtrées afin de retirer toutes les marques de conscience de concurrence. Le calcul de différence destiné à construire le patch correspondant est ainsi non perturbé par le mécanisme de conscience de groupe.

3.3.5 Changement de statut

Nous examinons maintenant la façon dont se produisent les changements de statut *fusionnée-éditée*.

Nous avons déjà expliqué que la transition *éditée-fusionnée* se produit sur chaque site lors de la réception d'un patch détecté comme concurrent. La transition inverse doit également être détectée et marquée correctement de façon à ce que, lorsqu'un utilisateur corrige des incohérences dans une page *fusionnée*, cette correction conduite à une page *éditée* sur tous les sites.

Le principe est le suivant. Lorsqu'un utilisateur édite et sauvegarde une page au statut *fusionnée*, le patch correspondant, s'il n'entre en concurrence avec aucun autre patch, conduit systématiquement la page au statut *éditée*. En d'autres termes, on suppose qu'un utilisateur réalisant une édition résout systématiquement toutes les éventuelles incohérences présentes dans la pages. Ce patch est diffusé au réseau et, sur chaque site, la même transition se produit à condition qu'aucun patch concurrent ne soit créé. La figure 3.13 illustre ce processus dans le cas de notre exemple : dans l'état final, un utilisateur édite la page pour y apporter des corrections, la conduisant sur le site 3 du statut *fusionnée* au statut *éditée*. Ce patch créé est diffusé aux autres sites où la même transition se produit. Finalement, la page revient au statut *éditée* sur l'ensemble des sites.

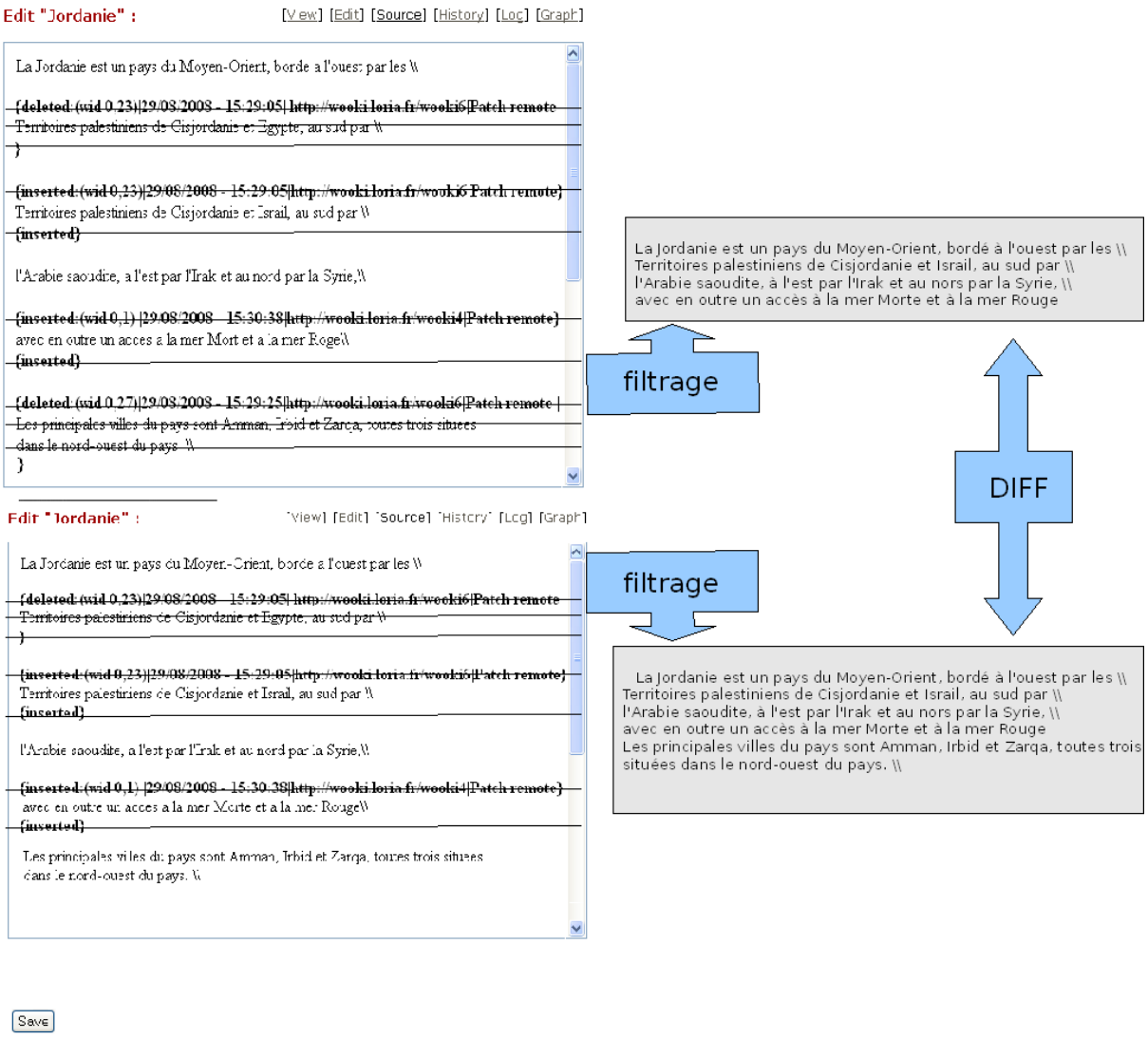

FIG. 3.12 – Exemple : filtrage des contenus au moment de la sauvegarde

Bien entendu, il peut se produire des cas où un patch de correction est produit en concurrence avec un autre patch. Ce dernier patch peut être lui-même un patch de correction produit sur le même état *fusionnée* ou un autre patch, produit sur un état différent. La figure 3.14 illustre un cas de ce type sur notre exemple : 2 utilisateurs éditent en concurrence le même état final, et chacun de son côté apporte les corrections qu'il juge nécessaire. Ces 2 modifications concurrentes sont fusionnées et ramènent la page à un statut *fusionnée*.

Enfin, dans certains cas un utilisateur peut souhaiter valider une page ayant le statut *fusionnée* pour la faire passer au statut *éditée sans lui apporter de modification*, et donc sans réaliser d'opération d'édition. Le cas se produit si le résultat de la fusion est cohérent

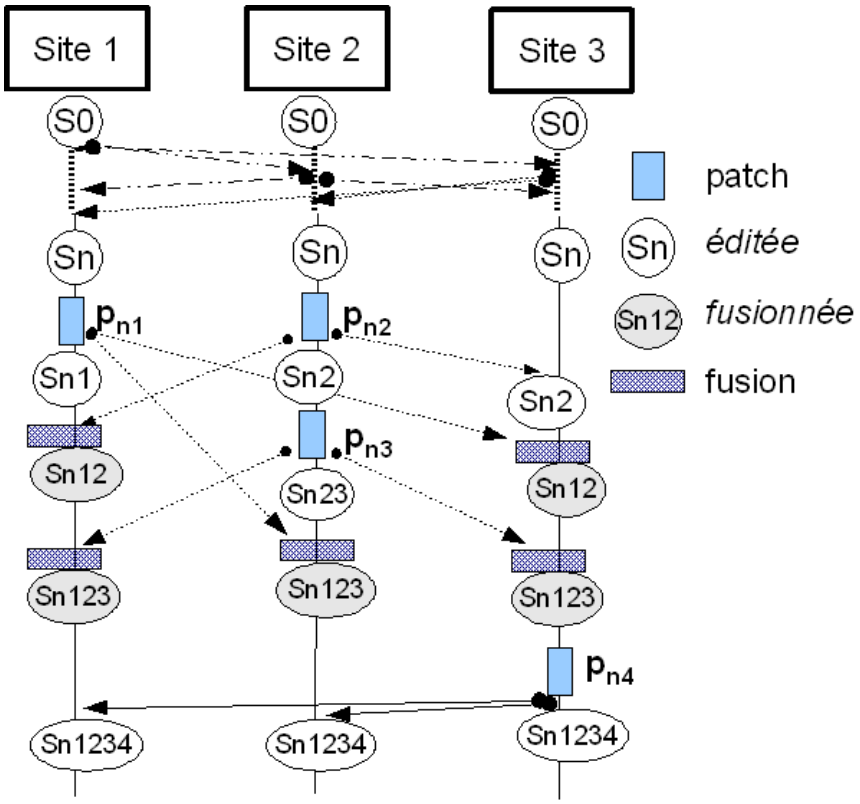

FIG. 3.13 – Exemple : transition vers le statut *éditée*

et convient aux auteurs de la page. Dans un tel cas, notre approche consiste à créer et diffuser un patch vide d'opération, dont le seul rôle est d'assurer la transition de statut.

3.4 La Représentation de l'Historique d'une page Wiki

Dans un wiki classique, l'historique des versions proposé aux utilisateurs est simple à construire et reflète la séquence réelle des révisions apportées à une page. Chaque nouvelle version sauvegardée par un utilisateur conduit à l'ajout d'une entrée dans l'historique. Toutes les versions ont le même statut.

Dans un wiki P2P, cette approche est remise en cause. D'une part, les versions n'ont pas toutes le même statut, certaines résultent d'une édition et d'autres d'une fusion. D'autre part, la présentation linéaire ne permet pas de rendre compte des modifications concurrentes ayant conduit à une version fusionnée. Enfin, l'ordre d'application des patchs, et donc de création des versions, peut différer d'un site à l'autre. L'historique linéaire serait alors différent sur chaque site.

Pour faire face à ces problèmes, notre approche vise à représenter l'historique des modifications de façon à ce que l'utilisateur puisse percevoir sa nature concurrente et non

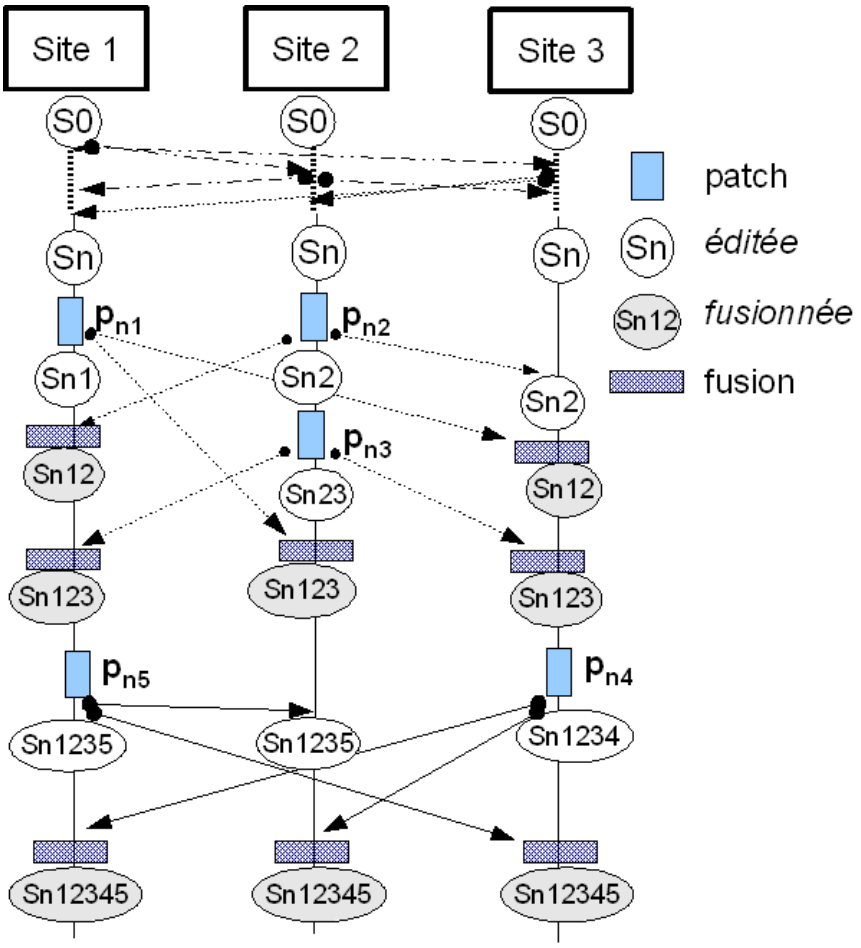

FIG. 3.14 – Exemple : 2 patchs de correction concurrents

linéaire. En particulier, il doit pouvoir distinguer les versions ayant un statut *édité* des versions ayant un statut *fusionnée*. Dans le cas des versions *fusionnées*, il doit pouvoir identifier les modifications concurrentes ayant conduit à la fusion.

Nous représentons l'histoire par un graphe directe acyclique dont les noeuds représentent les versions de page et les arcs représentent les patchs. La figure 3.15 illustre cette représentation graphique dans le cas de notre exemple, réduite aux versions et patchs produits dans notre scénario. On suppose *Sn* est une version *éditée*, elle est représentée par un nœud transparent.

La figure présente l'historique concurrent sur chaque site. Les noeuds portent les identifiants de versions, alors que les arcs sont identifiés par les identifiants de patchs auxquels ils correspondent. De plus, les arcs apparaissant en pointillés représentent l'évolution de la page sur le site local alors que les arcs en gras représentent les patchs et les états produits sur les sites distants.

Ainsi, sur le site 1, lorsque P_{n1} est généré, l'état est resté *édité* car ce patch n'est

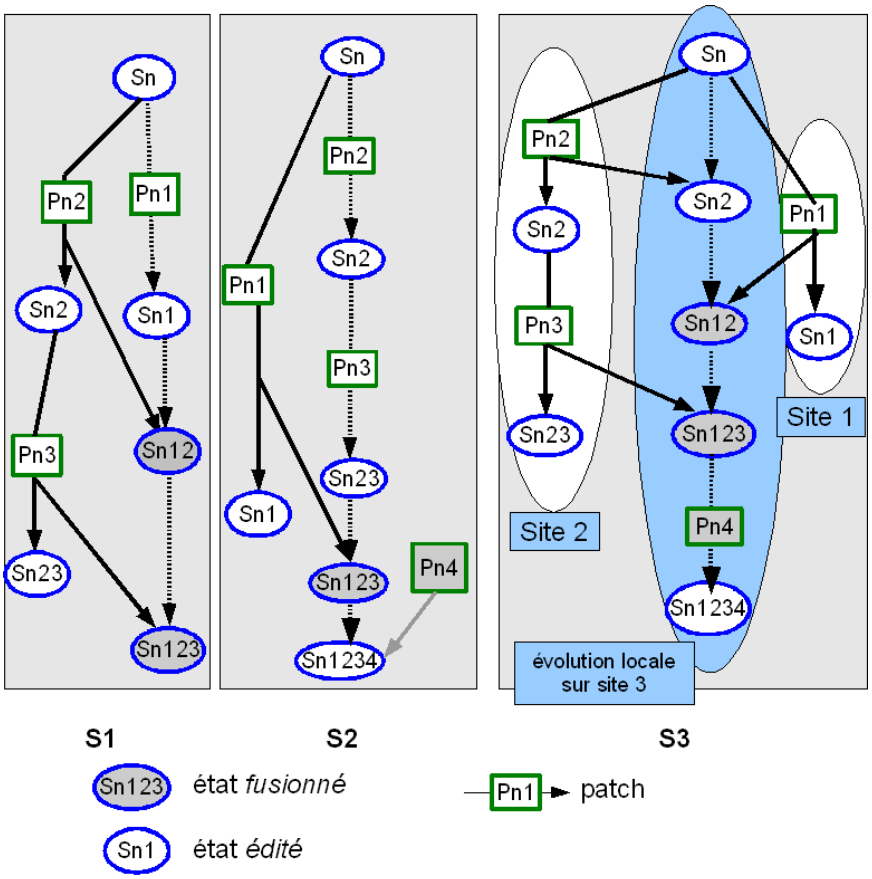

FIG. 3.15 – Exemple : histoire non linéaires des versions sur les sites 1, 2, 3

pas concurrent avec l'état de la page Sn. Lorsque P_{n2} est reçu, la concurrence est testée et détectée avec le dernier patch, et l'état de la page est changé en *fusionnée* ; le noeud correspondant est ombré. La représentation permet à l'utilisateur d'identifier :

- que la version S_{n12} est fusionnée, grâce au fond coloré du noeud,
- que cette version a été produite par intégration du patch distant P_{n2} (arc gras),
- que la version locale précédente S_{n1} avait été produite par le patch local P_{n1}, (arcs pointillés),
- que les patchs P_{n1} et P_{n2} sont concurrents, car situés sur des branches parallèles du graphe.

Sur ce site 1, deux branches seulement apparaissent : la branche locale, et une seule branche distante puisque tous les patchs distants sont reçus d'un seul et même site, S2.

Sur le site 3, trois branches parallèles apparaissent. La branche locale, la branche correspondant à S1 et la branche correspondant à S2.

Cette visualisation, si elle présente de manière synthétique l'ensemble des informations nécessaires pour comprendre l'histoire concurrente d'une page, est cependant complexe

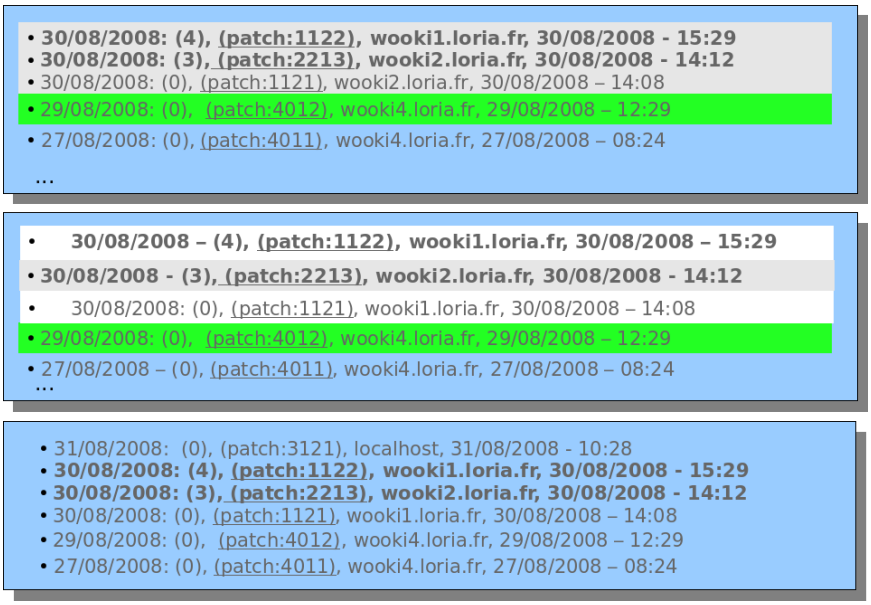

FIG. 3.16 – Représentation simplifiée de l'historique concurrent

et peut vite conduire à des graphes de taille importante difficiles à appréhender par les utilisateurs. Pour cette raison, nous proposons une visualisation alternative inspirée de l'historique des wikis traditionnels.

L'approche consiste à afficher les états successifs produits localement, en indiquant pour chacun d'eux le patch responsable. De plus, on distingue les états *fusionnés* des états *édités*. Lorsque l'état courant de la page possède le statut *fusionnée*, l'histoire concurrente est identifiée et l'ancêtre commun est marqué.

La figure 3.16 illustre cette représentation simplifiée.

La figure présente plusieurs histoires visualisées sur le site 3 à différents moments. La représentation se lit de la manière suivante :

1. Chaque ligne correspond à un état (ou une version) locale de la page. La première colonne (à gauche) indique la date de création de l'état. Les états sont classés du plus récents (l'état courant) au plus ancien (l'état initial),
2. les états *fusionnés* apparaissent en gras, les états *édités* apparaissent en fonte standard,
3. pour les états *fusionnés*, le nombre de lignes entrant en concurrence dans la page est précisé,
4. lorsque l'état courant est *fusionnée*, l'histoire concurrente est identifiée par une couleur de fond, l'état ancêtre commun est identifié également par une couleur de fond distincte. C'est le cas dans la 1ere histoire présentée.

5. pour chaque état, la ligne est complétée par une indication de l'identifiant du patch ayant conduit à cet état, l'url du site producteur de ce patch ainsi que la date de production de ce patch.
6. les identifiants de patchs sont des liens qui permettent de faire apparaître les contextes de génération du patch, c'est-à-dire l'ensemble des patchs présents dans l'état sur lequel a été généré le patch. Il s'agit donc des patchs précédant causalement le patch. Cela permet à l'utilisateur de percevoir les patchs concurrents, ainsi que les états ayant été vus par l'utilisateur lors de la création du patch. Cette indication est illustrée dans la deuxième histoire présentée dans la figure.
7. lorsque l'état courant est *édité*, l'histoire apparaît de manière très voisine à celle d'un wiki classique. C'est le cas par exemple dans la troisième histoire de notre figure, qui correspond au cas où un patch local a été produit sur le site 3.

3.5 conclusion

Nous avons présenté dans ce chapitre *la conscience de la concurrence*, un mécanisme pour wikis P2P. Ce mécanisme permet aux utilisateurs d'un wiki d'être conscients du statut des pages wiki vis-à-vis de la concurrence, et de déterminer exactement les zones dans les pages concernant cette concurrence. Nous avons aussi proposé une représentation pour l'évolution de versions en montrant les versions résultant d'une fusion de modifications concurrentes ou d'une édition.

Notre mécanisme est adapté au contexte des réseaux P2P :

1. le calcul des informations de la conscience se fait localement sur chaque nœud,
2. le mécanisme affiche les mêmes informations sur chaque nœud,
3. notre mécanisme n'a aucun impact sur le mécanisme de réplication,
4. il passe à l'échelle grâce au mécanisme de vecteur d'horloge à taille fixe.

Chapitre 4

Mise en Oeuvre

Sommaire

4.1 La visualisation d'opérations concurrentes 84

4.2 La visualisation de l'histoire d'une page 85

4.2.1 Visualisation du log 85

4.2.2 Visualisation de l'historique des modifications d'une page . . . 86

4.2.3 Visualisation du graphe des modifications d'une page 87

4.3 Conclusion 90

Le mécanisme de conscience de la concurrence présenté dans le chapitre précédent a été implanté dans le prototype Wooki. Ce prototype, que nous avons décrit au chapitre 2 est un wiki P2P opérationnel distribué en licence libre. Il intègre toutes les fonctionnalités classiques d'un wiki (à l'exception de la gestion des utilisateurs).

Deux types de fonctionnalités ont été conçues et implantées : des fonctionnalités de détection et visualisation des opérations concurrentes, et des fonctionnalités de visualisation de l'historique d'une page.

4.1 La visualisation d'opérations concurrentes

Un composant spécifique pour la détection de la concurrence et le calcul des histoires concurrentes a été réalisé et intégré à l'architecture de wooki. La nouvelle architecture est présentée en figure 4.1. Ce composant, nommé *AwarenessEngine* implante les fonctions suivantes :

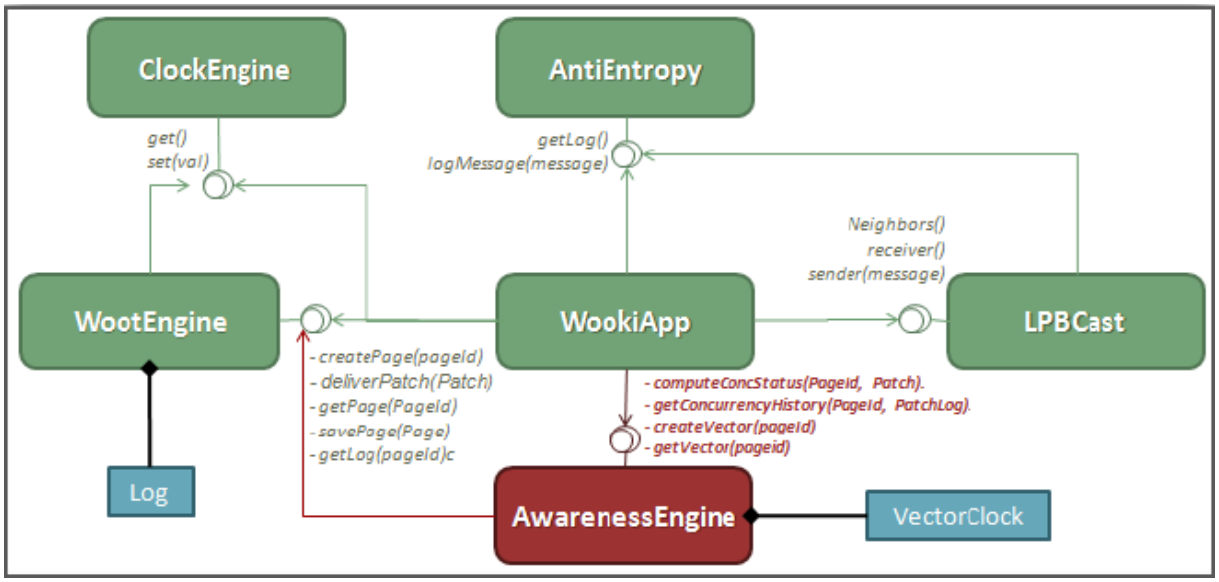

FIG. 4.1 – Wooki : architecture intégrant la conscience de la concurrence

Vecteur : création et gestion des vecteurs sur les pages et les patchs. Les vecteurs sont implantés dans une table de hachage. Une interface abstraite a été définie, ce qui permet de changer l'implantation de façon transparente. En particulier, il est possible de passer d'une implantation à base de vecteurs d'horloge classiques à une implantation utilisant des R-Vecteurs sans aucun impact sur le reste du mécanisme.

computeConcStatus(PageId, Patch) : est invoquée chaque fois que des patchs, locaux ou distants, sont intégrés à une page. Cette opération calcule le statut de la

page vis-à-vis de la concurrence, et le stocke.

getConcurrencyHistory(PageId) : est invoqué quand une page *fusionnée* est demandée. Cette opération retourne la partie concurrente de l'histoire des patchs.

Ces fonctions sont utilisées par le composant réalisant l'application Web. Le traitement des requêtes d'accès et de sauvegarde des pages est adapté afin de déclencher les tests de concurrence et les calculs sur l'histoire. Tous les traitements décrits dans la section 3.3 sont implantés dans ce composant.

La gestion du log de patchs, utilisé principalement par le composant de conscience de groupe, est confiée au composant WootEngine. C'est en effet lui qui réalise l'intégration des patchs sur les copies locales des pages. Chaque patch appliqué peut donc être stocké par ce composant. Ce composant réalise également l'extraction du contenu textuel des pages et assure l'insertion des marques de concurrence lorsque cela est nécessaire.

Enfin, le rendu final, c'est-à-dire la traduction de la syntaxe wiki et des marques de concurrences en html, est assuré par le composant Radeox. Radeox est un composant indépendant distribué en licence libre et écrit en java, utilisé dans de nombreux wikis. Un de ses intérêts est la possibilité d'y définir des macros. Nous avons utilisé ce mécanisme pour le traitement des marques de concurrence. En fait, ces marques sont des macros Radeox, interprétées directement par le moteur de rendu. Ceci permet d'agir sur le mode de visualisation de la concurrence en programmant ces macros, et donc de façon totalement indépendante du reste du mécanisme et de la syntaxe des marques ajoutées dans le texte. Pour rappel, un exemple de visualisation d'une page fusionnée est donné en figure 4.2

4.2 La visualisation de l'histoire d'une page

La visualisation de l'histoire d'une page est réalisée au travers de trois modules complémentaires permettant d'avoir différents points de vue sur cette histoire. Dans la version actuelle du prototype, notre proposition de visualisation présentée en 3.4 n'a pas pu être réalisée complètement pour des raisons de temps. La visualisation de l'historique implantée et opérationnelle actuellement est un peu différente.

4.2.1 Visualisation du log

Ce module permet de visualiser le log des patchs pour chaque page gérée par le serveur. Il est accessible au travers du lien [Log] présent sur l'interface. Un exemple est présenté en figure 4.3.

Les principes d'affichage sont les suivants :

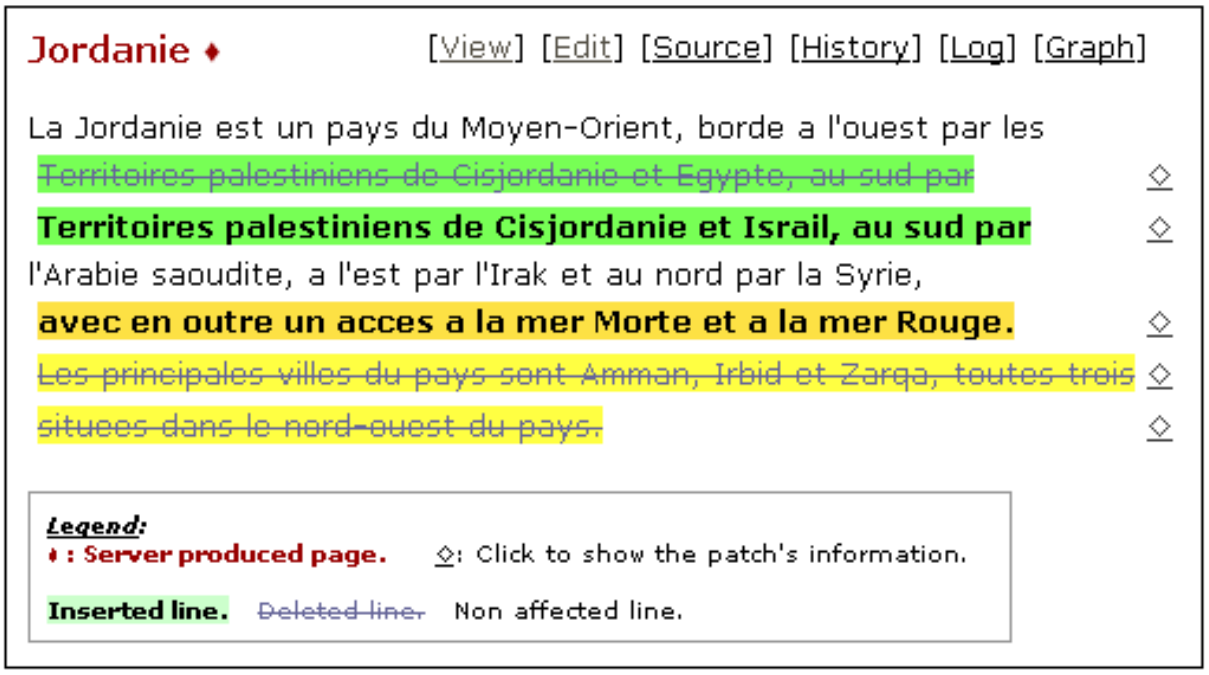

FIG. 4.2 – Exemple : visualisation d'une page *fusionnée*

- les patchs sont affichés dans l'ordre inverse de leur intégration locale : le plus récemment intégré localement apparaît en premier,
- les patchs distants apparaissent avec une couleur de fond, ce qui permet de les distinguer des patchs locaux.
- pour chaque patch, un certain nombre d'informations sont données : identifiant, url du site producteur, dates de création et d'intégration, liste des opérations contenues dans le patch.
- un lien permet d'obtenir des détails sur ce patch, notamment l'identifiant des lignes manipulées par les opérations ; ceci peut permettre, à des utilisateurs avertis, de comprendre en détail les fusions réalisées. Un exemple est présenté en figure 4.4.

4.2.2 Visualisation de l'historique des modifications d'une page

Ce module permet de visualiser l'historique des modifications d'une page Wooki au sein même de la page. Le principe est de visualiser l'ensemble des lignes ajoutées ou supprimées dans la page depuis sa création. Les lignes supprimées sont surlignées, et les lignes insérées sont décorées d'un fond coloré. Un exemple de visualisation de l'historique d'une page est donné en figure 4.5.

Pour chaque ligne, on peut obtenir des informations sur le patch ayant l'inséré ou supprimé. On peut de plus accéder à la visualisation du log centré sur le patch correspondant,

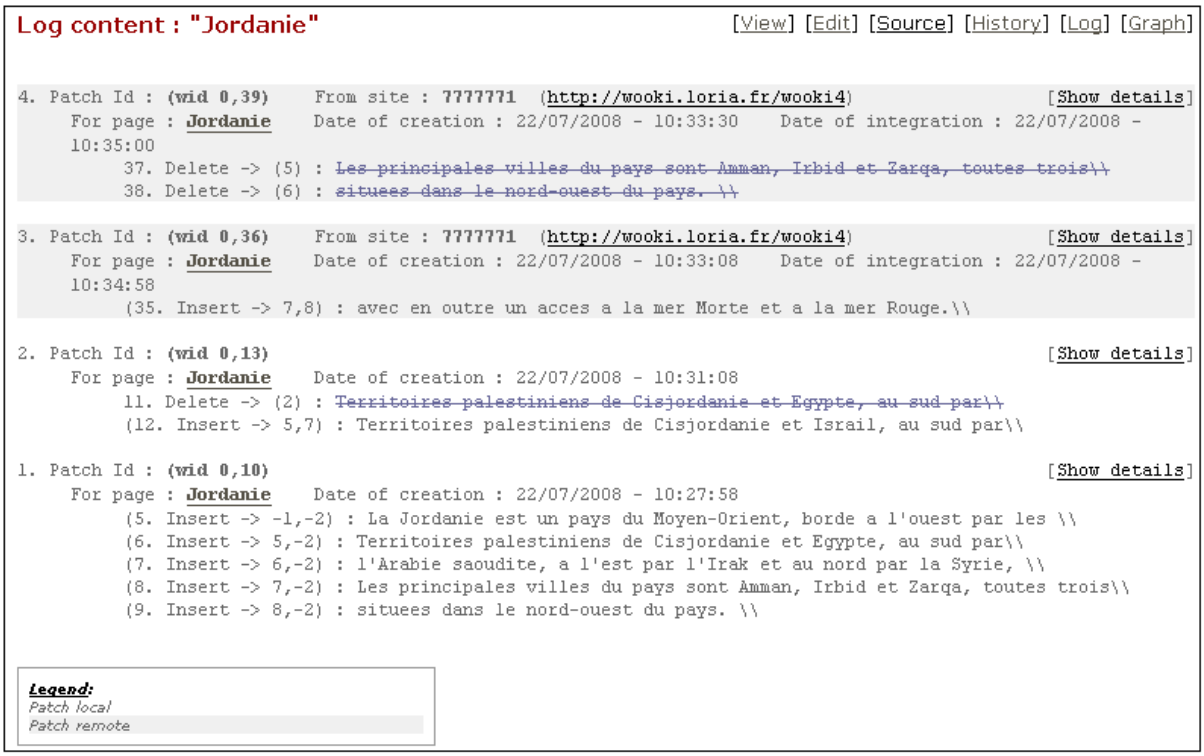

FIG. 4.3 – visualisation le log de page spécifique

comme illustré dans la figure 4.6.

Un aspect à souligner est que pour chaque ligne montrée dans ce qui est historique, on a l'option de voir l'information du patch qui l'a produite (inséré ou éliminé). Si on place la souris sur l'option [Show patch], il montrera une partie de l'information du patch producteur de la ligne. Ceci peut être observé dans l'illustration.

Cette visualisation de l'historique d'une page est en fait une vue complète sur le modèle Woot de la page. Rappelons que ce modèle de stockage conserve toutes les lignes ajoutées, les lignes supprimées étant seulement marquées comme invisibles. La création de cette visualisation consiste donc simplement à parcourir l'intégralité de ce stockage en affichant toutes les lignes, y compris les lignes invisibles.

4.2.3 Visualisation du graphe des modifications d'une page

Ce module offre une visualisation du log de patchs d'une page sous la forme d'un graphe orienté faisant explicitement apparaître les patchs concurrents sur des branches parallèles. Ce graphe contient :

– des nœuds représentants des états : ils sont étiquetés avec le vecteur correspondant à l'état. Un état *édité* est représenté par une couleur claire, et un état *fusionné* est

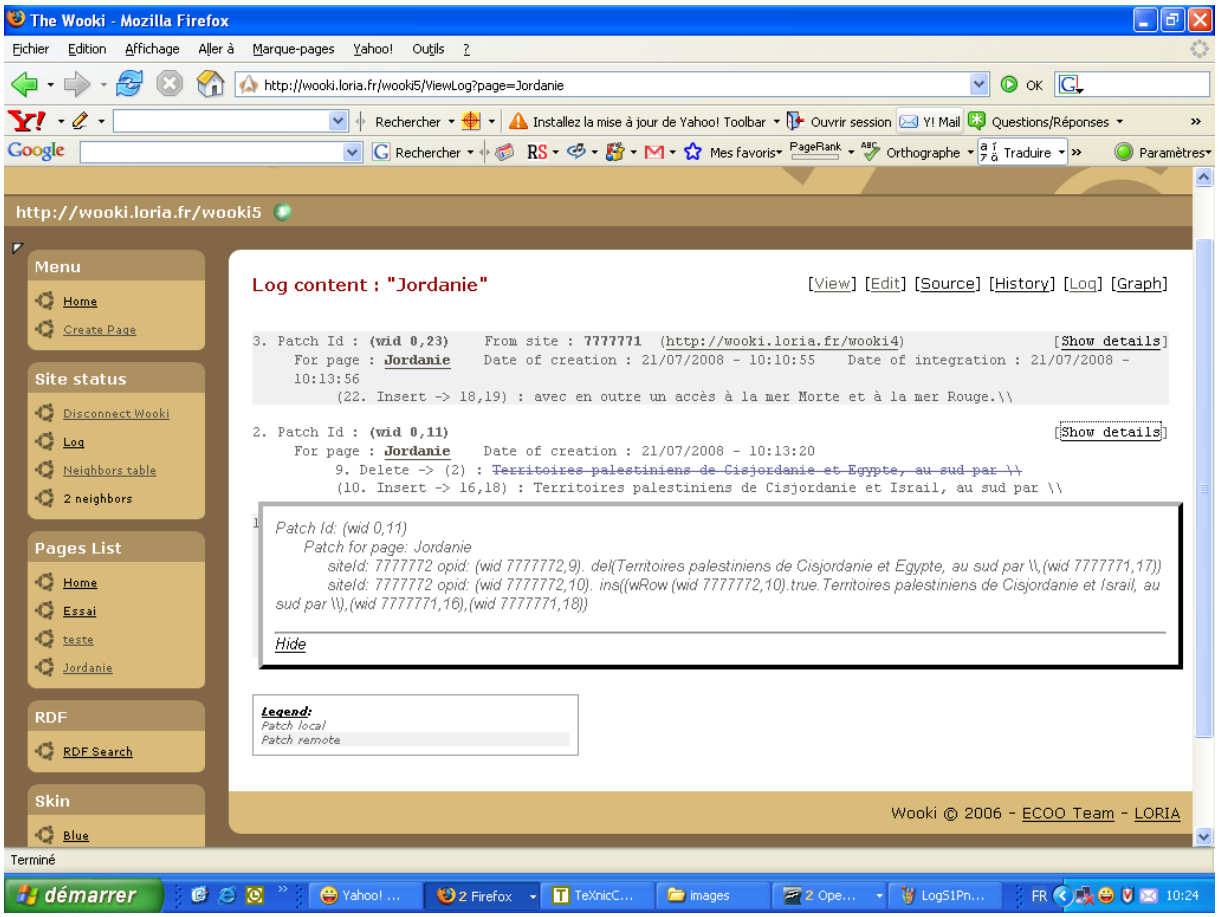

FIG. 4.4 – Visualisation de détails sur un patch

représenté par une couleur foncée,

- des noeuds représentants des patchs : ils sont étiquetés avec un identifiant de patch. Les patchs locaux sont en couleur claire, alors que les patchs distants sont en couleur foncée,
- des arcs état → patch : ils représentent le fait que la cible patch a été créée à partir de l'état source de l'arc,
- des arcs patch → état : ils représentent le fait que le patch source a été appliqué pour conduire à l'état cible.

La figure 4.7 illustre cette visualisation sous forme de graphe. Elle correspond au cas ou un premier patch local (PO_74) est produit et diffusé, et conduit à un état *édité*. Ensuite, deux patchs concurrents sont produits à partir de cet état, un patch local (PO_76) et un patch distant (PO_125) ; ils conduisent à l'état actuel de la page, ayant le statut *fusionné*.

L'utilisateur peut de plus obtenir des détails sur les différents patchs, comme illustré dans la figure 4.8. En particulier, il peut obtenir l'url du site producteur du patch ainsi

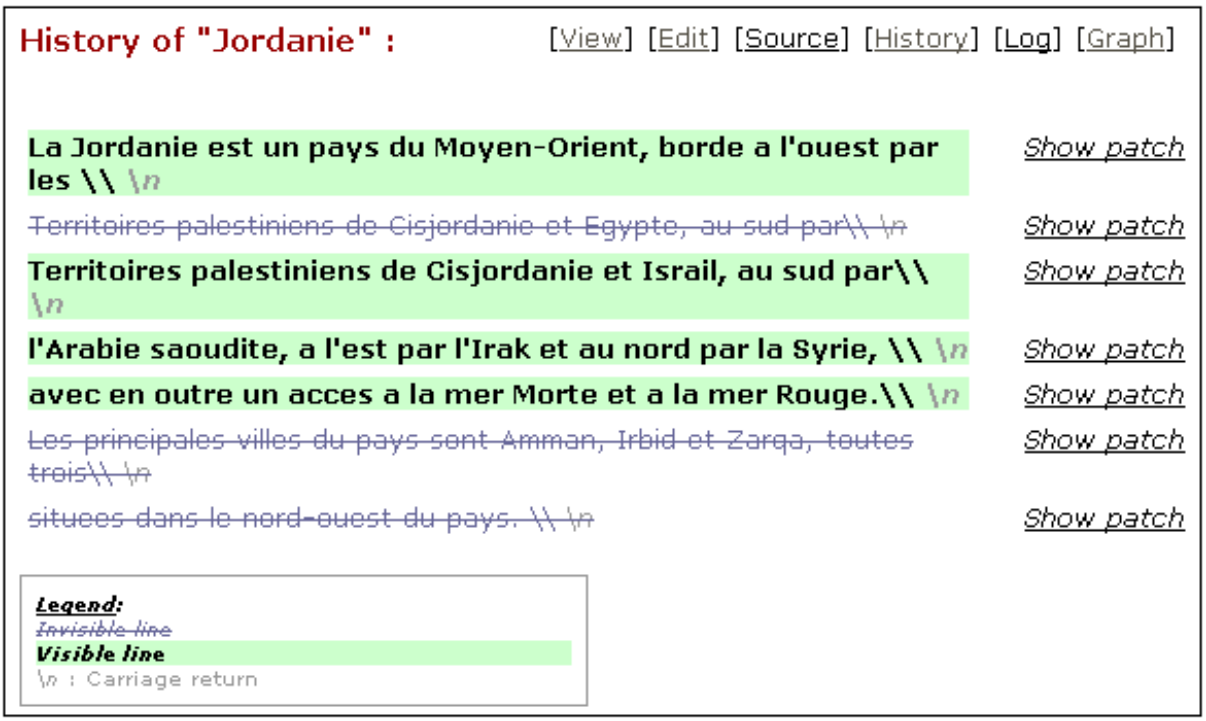

FIG. 4.5 – Visualisation de l'historique des modifications d'une page

que les dates de création et d'intégration locale du patch.

D'un point de vue technique, cette visualisation est réalisée par une applet construite en utilisant les librairies JGraphT pour construire le modèle de graphe, et JGraph pour réaliser le rendu (graph layout).

Elle se base sur un algorithme récursif parcourant le log de patchs. Chaque patch porte 2 vecteurs : le vecteur identifiant l'état dans lequel il a été créé, et le vecteur indiquant l'état résultant. L'algorithme démarre sur l'état courant de la page en créant un noeud pour le représenter, puis recherche dans le log tous les patchs dont il est le produit et les ajoute dans le graphe. Ensuite, pour chacun de ces patchs, il ajoute dans le graphe leur état source. Enfin, un appel récursif sur chaque état dans le graphe non encore traité est réalisé.

Deux remarques sont à faire sur cette représentation, qui est un peu différente de celle proposée dans la partie 3.4 :

1. dans cette représentation, tous les états n'apparaissent pas ; par exemple, dans la figure 4.7, l'état obtenu après l'application du patch PO_76 et avant l'intégration du patch PO_125 n'apparaît pas. Seuls apparaissent l'état final, et les états dans lesquels ont été produits des patchs. Cette représentation n'est donc pas fidèle à l'histoire locale réelle,

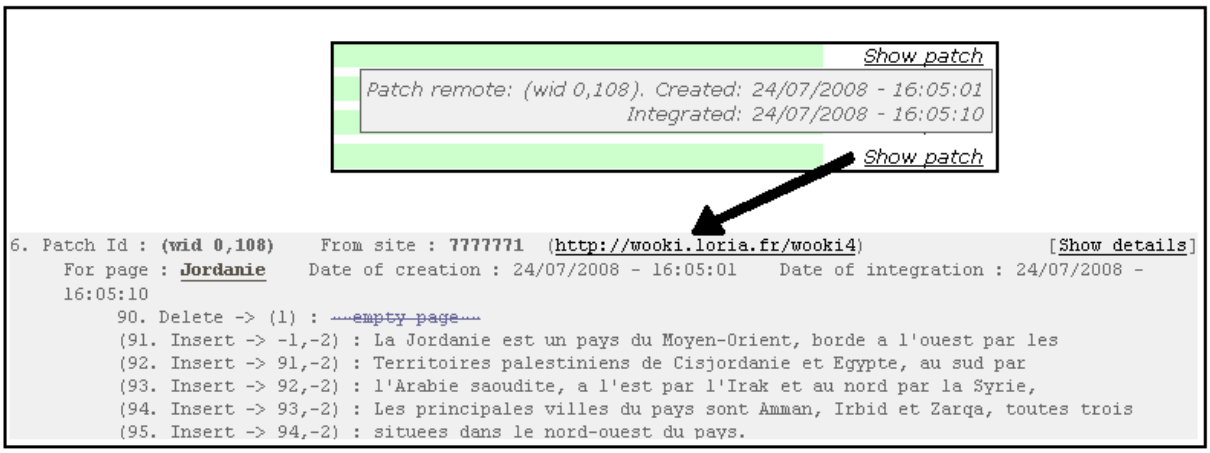

FIG. 4.6 – Visualisation du patch et accès au log pour une ligne de l'historique

2. par contre, cette représentation sera identique sur tous les sites, aux couleurs représentant les patchs locaux ou distants. On obtient donc une visualisation de la concurrence commune à tous les sites et donc partagée par tous les utilisateurs.

4.3 Conclusion

Notre mécanisme de conscience de la concurrence a été implanté dans le prototype Wooki. Ce prototype, diffusé en licence libre, est d'un bon niveau technique et peut être utilisé dans un cadre réel.

Il est maintenant doté d'un mécanisme permettant aux utilisateurs de repérer les pages construites par fusion automatique et, à l'intérieur de ces pages, de visualiser les zones de texte concernées par cette fusion.

Ce prototype a atteint un niveau de qualité tout à fait suffisant pour envisager des expérimentations en grandeur réelle. Ces expérimentations pourront servir de base à une évaluation des propositions que nous avons présentées.

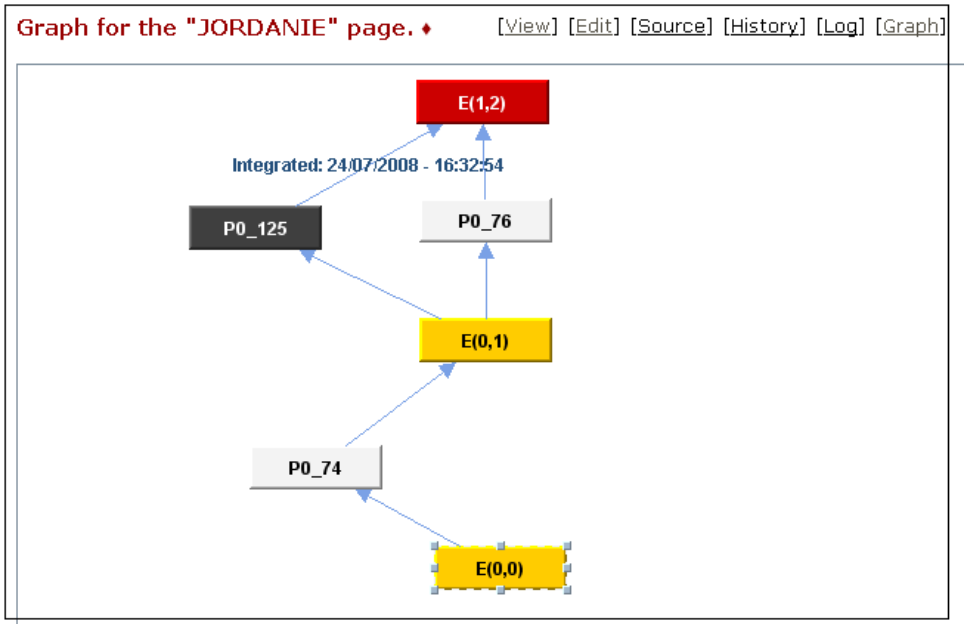

FIG. 4.7 – Graphe des patchs

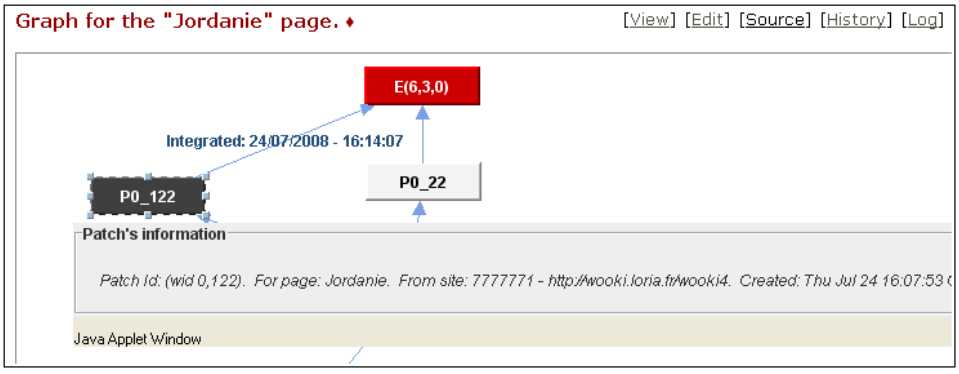

FIG. 4.8 – Détails des patchs sur le graphe

Chapitre 5

Bilans et Perspectives

Les wikis ont rendu l'édition collaborative très populaire et accessible à des pratiques massives. Le succès par exemple de Wikipédia, l'encyclopédie libre et collaborative, est l'image la plus frappante. Ces outils sont basés sur l'architecture client/serveur traditionnelle du web. Cette architecture atteint aujourd'hui ses limites. Afin de permettre une croissance encore forte du point de vue de la taille et de la complexité des données gérées ainsi que du nombre d'utilisateurs, l'idée de construire des wikis reposant sur une architecture décentralisée de type pair-à-pair est apparue. Notre approche pour construire un tel outil est de le fonder sur la réplication massive des données. Un wiki P2P est alors vu comme un ensemble de serveurs wikis interconnectés sur lesquels sont répliqués l'ensemble des pages wiki partagées.

Dans une telle approche, le premier problème à résoudre est celui de la réplication des pages. Une solution a été proposée par des membres de notre équipe, qui consiste à équiper chaque serveur d'un outil de fusion déterministe, commutatif et associatif (DCA) capable d'intégrer automatiquement les modifications réalisées sur les serveurs distants au fur et à mesure de leur arrivée. L'algorithme utilisé est l'algorithme d'intégration WOOT associé à des algorithmes classiques de dissémination des patchs de modification.

Cette approche soulève cependant un problème nouveau : dans un tel système certaines pages accessibles par les visiteurs sont produites automatiquement par le serveur exécutant une fusion. Ces pages sont visibles alors qu'elles n'ont pas été validées par leurs auteurs. De plus, la fusion automatique peut conduire à des résultats imprévisibles. En fait, la principale différence avec les wikis traditionnels ou les outils d'édition asynchrones classiques est que les fusions de données sont réalisées en dehors du contrôle des utilisateurs et dans l'espace public.

Notre proposition pour faire face à ce problème nouveau est d'équiper les serveurs wikis d'un mécanisme de conscience des modifications concurrentes. Son rôle est de permettre

aux visiteurs d'un wiki d'avoir conscience du statut d'une page vis-à-vis de la concurrence - s'agit-il d'une page *éditée* ou *fusionnée* ? - et dans le cas d'une page *fusionnée*, d'avoir une vision des zones de la page touchées par la fusion. Les travaux écrits dans ce manuscrit ont donc eu pour objet la construction d'un tel mécanisme.

5.1 Bilan

La première contribution de cette thèse est un mécanisme de conscience de la concurrence pour le wiki P2P Wooki [63]. Ce mécanisme détecte les pages produites par une fusion automatique et, dans les pages fusionnées, marque les lignes touchées par la fusion.

Ce mécanisme repose d'une part sur un détecteur de concurrence dont le principe consiste à étiqueter les patchs échangés entre les serveurs afin d'identifier l'état dans lequel ils ont été produits. Cette étiquette, par comparaison avec l'étiquette portée par l'état de réception, permet de déterminer s'il existe, ou non, une relation de précédence causale entre patchs ou entre patchs et états. Sur ce détecteur sont alors construits des algorithmes pour déterminer le statut d'une page et pour extraire la partie concurrente de l'histoire d'une page. Ces algorithmes fonctionnent en analysant le log de patchs reçus par un serveur. Enfin, le mécanisme de conscience de concurrence est réalisé au travers du traitement des requêtes des utilisateurs. Chaque fois qu'une page est accédée, son contenu est décoré au moment de son extraction par des marques dédiées encadrant les lignes concernées par la fusion. Ces marques sont ensuite interprétées par le moteur de rendu HTML.

Ce mécanisme a été conçu pour satisfaire les contraintes liées à son contexte d'utilisation :

- il est totalement décentralisé : le calcul des informations de conscience de la concurrence se fait sur chaque site en utilisant exclusivement des données locales,
- il n'a aucun impact sur la réplication et donc sur la correction du système puisqu'il vient seulement altérer la vue sur les pages mais pas leur contenu. Aucun patch ni aucun message supplémentaire n'est échangé entre les serveurs pour son compte,
- les informations offertes aux utilisateurs sont identiques sur tous les sites dans le même état, puisque les algorithmes dépendent uniquement de l'ensemble de patchs reçus sur un site et pas de leur ordre d'arrivée,
- le mécanisme passe à l'échelle, en particulier du point de vue de la détection de la concurrence, grâce à l'emploi de vecteurs d'horloge à taille fixe. Cette contrainte d'échelle et l'approche choisie nous ont conduit à un compromis entre précision de la détection et nombre de sites dans le réseau. Ce compromis semble acceptable mais

doit être évalué, nous en reparlerons dans la partie perspectives.

Ce mécanisme répond donc aux objectifs fixés, au moins dans sa conception. Une réalisation vient valider d'un point de vue technique notre approche. Une étude expérimentale doit maintenant être menée pour valider notre approche du point de vue de son utilisation.

La deuxième contribution de cette thèse concerne la visualisation d'un historique concurrent [2]. Notre proposition consiste à compléter la visualisation classique de l'historique d'un wiki de façon à marquer les états *édités* et *fusionnés* et à faire percevoir la concurrence. Un des problèmes à résoudre est que le graphe des versions et des patchs peut être complexe à comprendre dès que le nombre de modifications concurrentes devient significatif. La visualisation proposée se base sur la visualisation linéaire classique : chaque site affiche la séquence locale de versions dans l'ordre où elles sont apparues, et complète cet historique avec des informations permettant de repérer les états *édités* et *fusionnés* et les parties concurrentes dans l'histoire. Cette approche conduit à une visualisation dépendante du site, puisque s'appuyant sur l'ordre local d'application des patchs et d'apparition des versions.

Cette approche n'a pas été entièrement réalisée. Un mécanisme voisin a été implanté, mais ne permet pas dans son état actuel de valider notre proposition.

La troisième contribution est la réalisation d'un prototype intégré au système Wooki. Cette réalisation concerne essentiellement le mécanisme de conscience de concurrence. Le module de conscience de la concurrence fait maintenant partie de la distribution du système en licence libre et disponible sur sourceforge. Ce prototype constitue une première validation de nos propositions, du point de vue de la faisabilité technique. Il va permettre la réalisation d'une validation expérimentale de notre approche, du point de vue de son utilisation.

5.2 Comparaison aux travaux existants du domaine

Nous comparons dans ce paragraphe notre approche aux travaux existants dans le domaine.

Les outils d'édition synchrone proposent des mécanismes de conscience de groupe pas trés éloignés de nos propositions. Des travaux nombreux, menés principalement par Saul Greenberg et son équipe [34, 32, 26] , ont notamment portés sur des "widgets" collaboratifs visant à rendre perceptible l'action de chaque intervenant sur un document ou un espace partagé. L'idée est de faire comprendre à chaque participant d'une session de collaboration ce que font les autres participants, au moment où ils le font.

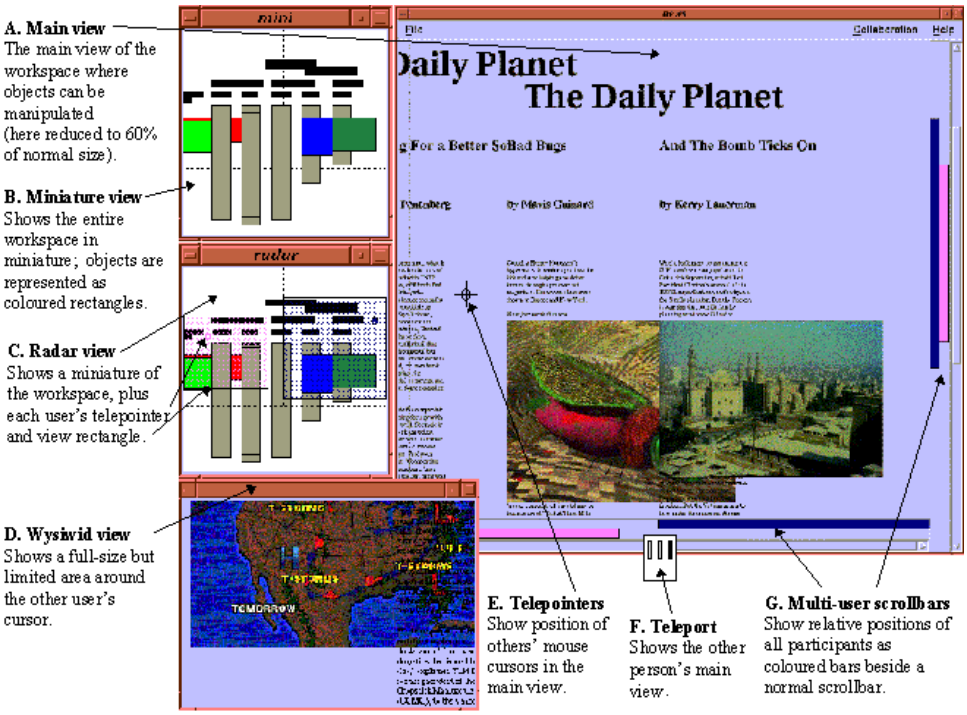

FIG. 5.1 – Widgets collaboratifs pour l'édition synchrone (tiré de [31])

La figure 5.1, tirée de [31], illustre les principaux widgets proposés et étudiés [29, 30] et notamment les télépointeurs, qui indiquent la position de la souris de chaque participant, les barres de défilement multi-utilisateurs, qui indiquent la position relative de chaque participant dans l'espace partagé, la vue radar qui représente, sur une vue miniature de l'espace complet, la fenêtre de visualisation de chaque participant.

Ces widgets permettent principalement à chacun des participants d'être au courant du lieu d'activité des autres participants et d'en déduire d'éventuelles interactions avec ses propres actions. De manière complémentaire, d'autres approches soulignent l'apport de chaque intervenant dans le document construit, comme dans la figure 5.2.

De manière claire, ces approches permettent à un acteur de la collaboration de suivre les activités se déroulant en concurrence avec ses propres actions. Notre proposition s'inspire de cette approche mais la situe dans un cadre asynchrone. Plus concrètement, les mécanismes de consciences de groupe dans les systèmes synchrones sont réalisés en étiquetant chaque action dans l'espace de travail par son auteur. Les widgets fournissent différentes vues instantannées sur cette information. Dans notre approche, l'information nécessaire à la réalisation de la conscience de groupe doit être calculée et reconstruite à posteriori en analysant l'histoire des modifications apportées à une page.

Dans les wikis actuels, les mécanismes de conscience de groupe sont trés réduits et se limitent à l'historique et à d'éventuelles "watch-list". Du point de vue de la concurrence,

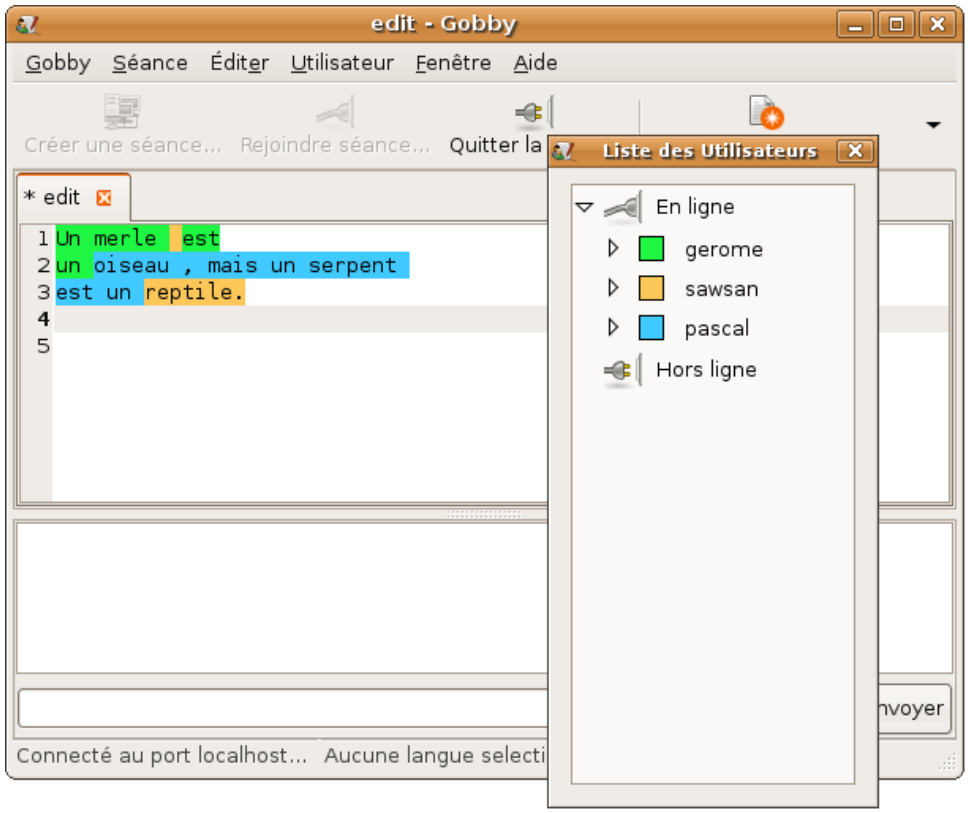

FIG. 5.2 – Conscience de l'action des autres dans l'outil Gobby

certains wiki (en particulier MediaWiki) offre une aide à la fusion de version en montrant les différences entre la version en cours de sauvegarde et la version enregistrée. Cette différence, si elle montre effectivement l'écart entre les 2 versions, ne montre cependant pas comment cet écart est obtenu, en particulier lorsque la version enregistrée est le fruit de plusieurs modifications successives. Notre mécanisme va plus loin dans la visualisation des opérations concurrentes, et surtout est adapté à un wiki décentralisé sur un réseau P2P. Il faut cependant noter que ce mécanisme serait applicable et utile dans un wiki centralisé utilisant une procédure de fusion automatique pour traiter les éditions concurrentes.

Dans le domaine des outils asynchrone, plusieurs approches ont notamment été proposées. La notion de conscience des changements a été formalisée par Tam et Greeberg dans [71]. La conscience des changements permet d'avoir conscience des évolutions successives d'un document depuis un moment dans le passé, selon différentes perspectives. Un exemple d'outil utilisant cette approche est proposé en figure 5.3. Dans cette figure, l'outil présente les modifications apportées à l'état du document depuis un point dans le passé. Ces modifications sont présentées de manière à montrer l'auteur de chacune des modifications, grace à un code de couleurs.

De même, l'approche des History Flow [75], qui permet de visualiser les écarts entre

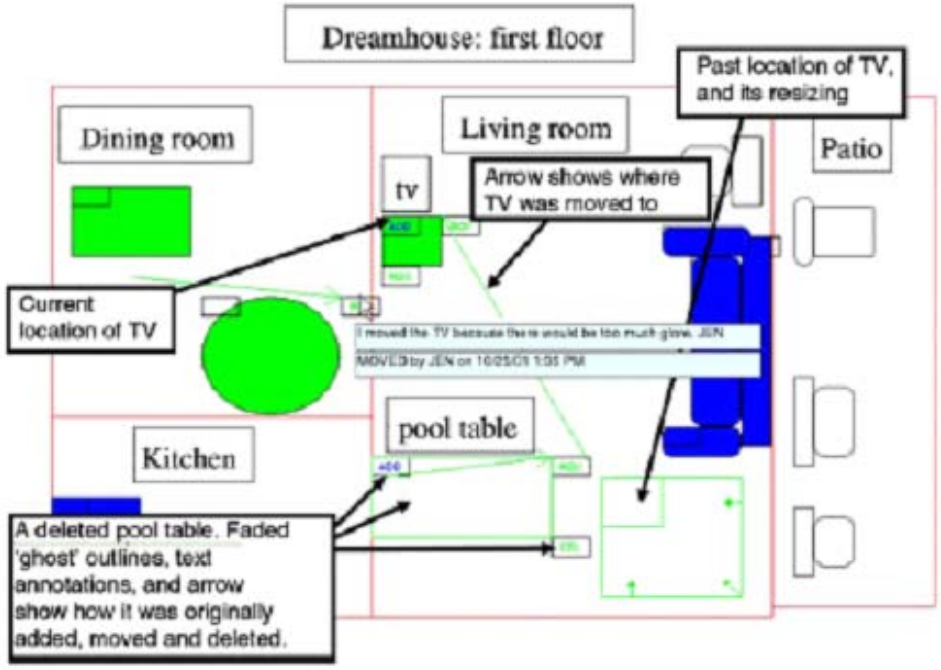

FIG. 5.3 – Conscience des changements dans un outil de DAO collaboratif

les versions successives d'un document, relève de cette conscience des changements. Les auteurs ont de plus le souci du passage à l'échelle et la volonté de représenter de manière synthétique les évolutions d'un document lorsque le nombre de versions devient trés grand. Une illustration de cette approche est donnée en figure ch :conclusion :fig :flow.

Cependant, la concurrence entre les modifications est rarement prise en compte dans ces approches. History flow par exemple est explicitement conçu pour des séquences linéaires de versions et on voit mal comment représenter la concurrence dans une telle visualisation. Notre mécanisme permet également d'avoir conscience de changements apportés à un document, mais selon une perspective particulière, celle de la concurrence. D'un point de vue plus concret, la visualisation des changements concurrents se base sur un calcul visant à extraire la partie concurrente dans une histoire, alors que la conscience des changements consiste à visualiser l'ensemble de l'histoire à partir d'un point désigné.

Certaines approches ont par contre explicitement été conçues pour donner une conscience des changements concurrents dans un cadre asynchrone. C'est notamment le cas de Palantir [62] et State treemap [44], deux systèmes assez voisins. Leur principe consiste à représenter l'état d'une arborescence de documents vis à vis de la concurrence au sein d'une visualisation de type treemap. Ces deux approches sont utilisées pour naviguer au sein d'un espace de travail privé et obtenir des informations de concurrence de chacun des documents par rapport à l'état d'un espace public. Un exemple de State Treemap est

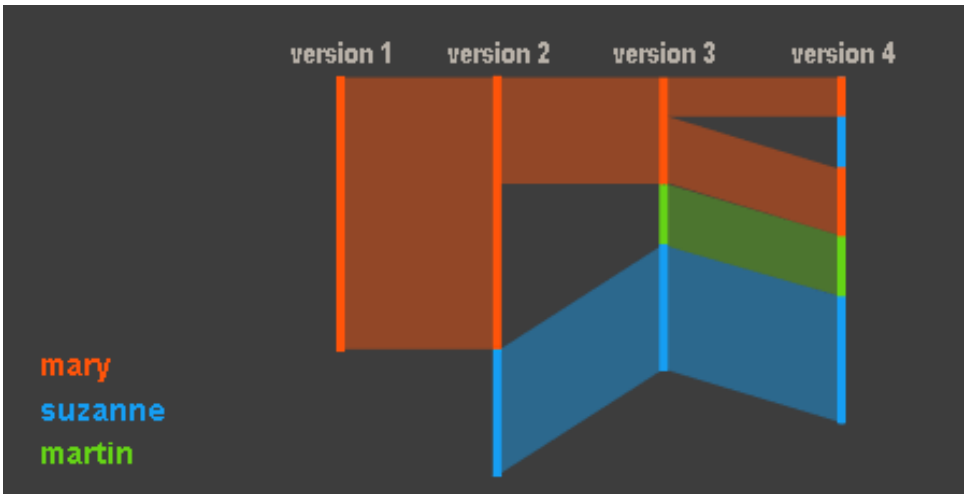

FIG. 5.4 – History Flow : visualisation d'une séquence de versions ([75])

donné en figure 5.5.

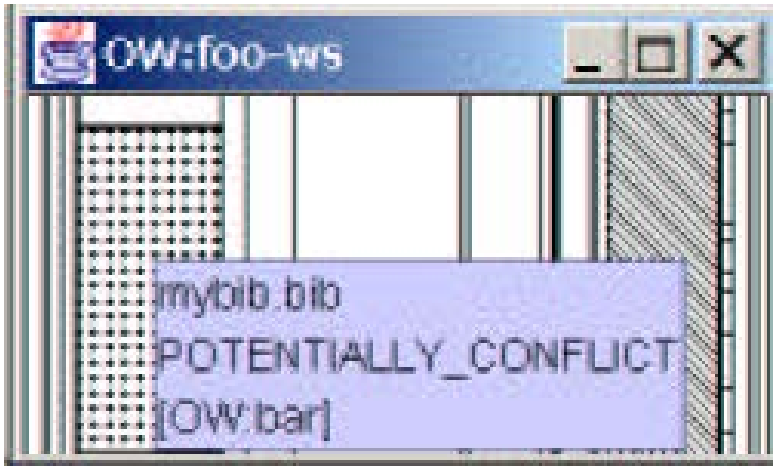

FIG. 5.5 – State treemap ([44])

De manière complémentaire, l'approche [45] propose des métriques pour mesurer la divergence entre deux versions d'un même document. On retrouve la même idée, mais étendue à des documents structurés hiérachiquement dans [52, 53], où des mesures cumulatives et des visualisations adaptées à chaque niveau de la hiérarchie sont proposées. Ces approches sont utiles pour prévenir les effets de la concurrence. Par contre, elles apportent peu d'aide pour résoudre les problèmes introduits par des changements concurrents au sein d'un document car leur granularité est celle du document. Dans ce sens, elles sont trés complémentaires de notre approche : on peut imaginer représenter l'état de l'ensemble des pages d'un wiki au sein d'un treemap, afin de repérer les zones du wiki sujettes à la concurrence, et combiner cela avec notre mécanisme pour représenter la concurrence au

sein de chaque page.

Dans les systèmes de gestion de versions, qu'ils soient centralisés comme SVN [65] ou distribués comme Git [22] et Mercurial [42], la conscience de la concurrence est fournie en deux temps : dans un premier temps, le système laisse une trace des fichiers ayant nécessités une fusion lors d'une synchronisation espace privé-espace public - ces fichiers sont ceux contenant des modifications concurrentes. Dans un deuxième temps, les modifications *en conflits* (et seulement celles-là) sont marquées grâce à des blocs de conflits insérés dans les documents fusionnés. La conscience de la concurrence est donc limitée à la conscience des conflits. D'autre part, seules deux versions sont considérées : la version locale et la version publique courante. Enfin, le mécanisme de conscience de concurrence a un impact sur l'état des documents partagés puisqu'il modifie ces documents en insérant des blocs de texte représentant les conflits. Notre approche est à la fois plus complète puisqu'elle offre des informations sur l'ensemble des modifications fusionnées et pas uniquement sur les modifications conflictuelles, et d'autre part elle offre des garanties du point de vue de la cohérence du système puisqu'elle n'a aucun impact sur l'état des documents partagées : les informations de conscience de groupe sont ajoutées dans la vue sur le document et non pas dans le document.

5.3 Perspectives

Les travaux présentés dans ce manuscrit, mêmes s'ils atteignent en grande partie leurs objectifs de départ, soulèvent néanmoins un certain nombre de questions et ouvrent des perspectives que nous comptons explorer à court et moyen terme.

Ces questions et perspectives sont de deux ordres : l'évaluation expérimentale de notre approche, qui reste à réaliser, d'une part, et des mécanismes et fonctionnalités complémentaires d'autre part.

5.3.1 Perspectives concernant l'évaluation de l'approche

A court terme, notre perspective principale concerne l'évaluation expérimentale de notre proposition. Le support de ces expériences sera le prototype qui a été développé et qui est suffisamment abouti pour cela. Plusieurs aspects doivent être évalués.

Évaluation de l'interface : comme pour tout système interactif, l'interface homme-système que nous proposons pour visualiser et naviguer dans les informations de conscience de la concurrence doit être évaluée. L'objectif est d'étudier dans quelle mesure les utili-

sateurs du système comprennent les informations que nous souhaitons leur transmettre. Certains point précis peuvent être évalués :

- reconnaissance des pages fusionnées,
- reconnaissance des zones fusionnées dans une page,
- compréhension de l'histoire concurrente d'une page.

Dans un premier temps, une étude classique d'utilisabilité basée sur l'observation des actions des utilisateurs face à l'interface peut être réalisée. Elle nécessite la mise au point d'un scénario d'expérience intégrant différents cas de fusion et des histoires plus ou moins complexes. Cette étude d'utilisabilité peut-être complétée par des questionnaires permettant de vérifier la compréhension de différentes situations caractéristiques par les utilisateurs.

Les résultats de cette évaluation permettront d'améliorer l'interface du système et de faire évoluer les modes de visualisation des informations liées à la concurrence.

Passage à l'échelle de l'interface : si le mécanisme de détection de la concurrence a été conçu explicitement pour passer à l'échelle, la question reste en suspens pour les aspects liés à l'interface. En particulier, on peut se poser les questions suivantes :

1. dans le cas où un grand nombre de modifications entre en concurrence, les visualisations choisies pour les pages et pour l'historique permettent-elles aux utilisateurs d'appréhender la concurrence ? Si elle existe, quelle est la limite d'utilisabilité de notre approche sur ce point ?
2. dans des cas réels d'utilisation, quel nombre de modifications concurrentes peut-on atteindre ? En d'autres termes, dans quelles limites le problème du passage à l'échelle de l'interface se pose-t-il réellement ?

La réponse à la deuxième question est très difficile à donner. Elle dépend entre autre des situations d'éditions et de la rapidité du mécanisme de réplication. Il semble difficile de réaliser une expérience réaliste compte-tenu du temps nécessaire. Une approche possible est de réaliser une étude sur des traces et des historiques de wikis existants, par exemple Wikipédia, afin d'évaluer le nombre de modifications concurrentes maximum qui a pu être atteint au cours de la vie d'un article. On pourra se concentrer sur certains articles typiques, par exemple les articles polémiques ayant suscité des"guerres d'édition" importantes.

La première question par contre peut-être abordée de manière expérimentale, à l'aide de scénarii préparés et joués par un automate. L'idée serait de faire exécuter un même scénario par un panel d'utilisateurs, en augmentant progressivement le nombre de concurrences visualisées, et en mesurant la qualité des corrections apportées par les utilisateurs

à chaque étape. Les différents scénarii pourraient correspondre à différentes situations classiques : guerre d'édition, vandalisme, spam.

Validité de la détection de la concurrence : le mécanisme de détection de concurrence passe à l'échelle au prix d'un compromis entre précision et nombre de sites participants. Rappelons qu'au delà d'une certaine limite, le mécanisme oublie certaines concurrences et les rapportent comme causalement dépendantes. Le taux d'erreur du mécanisme est très difficile à évaluer, il dépend fortement des scénarii d'utilisation et d'échange des patchs. On peut cependant créer des scénarii de test conduisant volontairement à des imprécisions. Une étude intéressante consiste à évaluer la limite d'intérêt du mécanisme, c'est-à-dire à partir de quel taux d'erreurs le mécanisme perd son intérêt et ne se comporte pas mieux qu'un wiki P2P sans conscience de la concurrence. On peut d'ailleurs se poser la question de l'existence d'une telle limite.

L'approche consiste ici à faire exécuter des scénarii d'édition à deux groupes d'utilisateurs : un groupe n'utilisant pas le mécanisme de conscience de concurrence, et un groupe utilisant ce mécanisme mais avec des imprécisions et des oublis dans la détection de la concurrence en nombre grandissant. Une comparaison qualitative des résultats de chaque groupe nous permettra d'apporter des éléments de réponse à la question.

Validité de l'approche : notre approche a pour objet de faire face au problème des fusions automatiques apparaissant dans les wikis P2P. Sans mécanisme de conscience de la concurrence, il nous semble qu'un wiki P2P est difficilement acceptable par ses utilisateurs. Une approche de validation générale de notre proposition serait de mener une expérience visant à montrer que notre système est aussi acceptable qu'un wiki classique. Ceci nous permettrait de conclure que notre système P2P bénéficie de tous les avantages liés à son architecture tout en restant aussi utilisable qu'un système reconnu et largement utilisé.

L'approche d'évaluation consisterait ici à mener une étude comparative entre deux groupes d'utilisateurs : un groupe utilisant un wiki classique (par exemple mediawiki) et un groupe utilisant notre wiki P2P. Chaque groupe se verrait proposer un certain nombre de scénarii incluant des situations de concurrence. Des questionnaires remplis par les utilisateurs et une analyse qualitative des résultats de chaque groupe nous apporterait des éléments de conclusion sur notre proposition.

5.3.2 Perspectives concernant les fonctionnalités du système

Un certain nombre de fonctionnalités complémentaires peuvent être envisagées à court ou moyen terme, en complément de notre mécanisme de conscience de la concurrence.

Coordination pour le règlement des conflits de concurrence : notre mécanisme, dans sa forme actuelle, se contente de souligner dans les pages les problèmes potentiels liés à la fusion d'opérations concurrentes. Il pourrait être intéressant d'y coupler un mécanisme pour notifier les auteurs d'une page de l'existence de ces problèmes, et pour coordonner les actions de corrections qu'ils pourraient entreprendre. Cela permettrait de minimiser la concurrence sur des modifications visant à corriger des fusions.

Vue synthétique sur un wiki : notre mécanisme fonctionne au niveau des pages uniquement. Il pourrait être utile de proposer une vue globale sur un wiki complet, c'est-à-dire un ensemble de pages, de façon à repérer rapidement les pages nécessitant une intervention corrective. Un wiki pouvant contenir un nombre important de pages, il est nécessaire de penser à un mode de visualisation adapté, par exemple un treemap ou un arbre hyperbolique.

Filtres de visualisation de la concurrence : actuellement, l'ensemble de l'histoire concurrente d'une page est soulignée dans sa visualisation. Si cette histoire est grande, cela peut rendre la compréhension globale de l'état de la page complexe. Une solution à explorer pour faire face à ce problème consiste à fournir à l'utilisateur la possibilité de filtrer les patchs inclus dans l'histoire concurrente selon différents critères : ancienneté, fenêtre temporelle, site producteur, auteur de la modification, type de modification... De tels filtres permettraient aux utilisateurs confrontés à une situation de concurrence complexe de l'aborder par parties afin de construire progressivement une compréhension globale de cette situation.

Conscience des changements : Tam et Greenberg ont défini dans [71] la capacité des individus à suivre les changements faits par d'autres participants, au cours du temps, sur un objet partagé de manière asynchrone. Un tel mécanisme n'est pas très éloigné du notre du point de vue de la réalisation : dans les deux cas il s'agit de visualiser un historique de modifications. Dans notre mécanisme, l'ensemble des patchs formant l'histoire concurrente est calculé par un algorithme réalisant des calculs de concurrence et déterminant un ancêtre commun. On peut envisager que cet ancêtre commun ne soit plus calculé en fonction de critères de concurrence, mais simplement choisi par un utilisateur. Notre mécanisme permettrait alors à cet utilisateur de visualiser l'ensemble des changements apportés à une page wiki depuis un point déterminé par lui, par exemple l'état de la page lors de sa dernière visite. On obtient ainsi un mécanisme de conscience des changements utile dans notre contexte P2P où des déconnexions sont possibles.

Bibliographie

[1] Larry Allen, Gary Fernandez, Kenneth Kane, David B. Leblang, Debra Minard, and John Posner. Clearcase multisite : Supporting geographically-distributed software development. In *Selected papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management*, pages 194–214, London, UK, 1995. Springer-Verlag.

[2] Sawsan Alshattnawi, Gérôme Canals, and Pascal Molli. A Non linear represenation of the page history in a P2P wiki system. In *8th Conference on e-Business, e-Services and e-Society, Towards Sustainable Society on ubiquitous Networks*, volume 286, pages 151–160. Boston : Springer, 2008.

[3] Wolfgang Appelt. Www based collaboration with the bscw system. In *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics (SOFSEM '99)*, pages 66–78, London, UK, 1999. Springer-Verlag.

[4] Michel Beaudouin-Lafon and Joëlle Coutaz. Gt scoop : rapport de recherche 1994-1995. Technical report, PRC CHM, GT SCOOP, 1995.

[5] Brian Berliner. CVS II : Parallelizing software development. *Proceedings of the USENIX Winter 1990 Technical Conference*, 341 :352, 1990.

[6] S.A. Bly, S.R. Harrison, and S. Irwin. Mediaspaces : Bringing people together in a video, audio and computing environment. . *Communications of the ACM*, 36(1) :28–47, January 1993.

[7] Gutwin Carl. *Workspace awareness in real Time groupware environments* . Ph.d. thesis, Departement of Computer Ccience, University of Calgary, Calgary, Canada, 1997.

[8] Bernadette Charron-Bost. Combinatorics and geometry of consistent cuts : Application to concurrency theroy. In *Proceedings of the Internationnal Workshop on Parallel and Distributed Algorithms*, pages 45–56, 1989.

[9] Dominique Decouchant, Vincent Quint, and Salcedo Manuel Romero. Structured cooperative editing and group awareness. In *Proceedings of the Sixth International Conference on Human-Computer Interaction*, volume I. Human and Future Computing of *I.12 Collaboration 3*, pages 403–408, 1995.

[10] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, (PODC '87)*, pages 1–12. ACM, 1987.

[11] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '92)*, pages 107–114, Toronto, Ontario, 1992. ACM Press.

[12] Paul Dourish and Sara Bly. Portholes : Supporting awareness in a distributed work group. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 541–547. ACM, 1992.

[13] Clarence Ellis and Simon Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2) :399–407, 1989.

[14] Clarence Ellis, Simon Gibbs, and Gail Rein. Design and Use of a Group Editor. In Cokton, editor, *Engineering for Human-Computer Interaction*. North-Holland, 1990.

[15] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware : some issues and experiences. *Commun. ACM*, 34(1) :39–58, 1991.

[16] P. TH. Eugster, R. Guerraoui, S. B. Handurkande, and A. M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4) :341–374, November 2003.

[17] Patrick TH. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, 5(37) :60–67, May 2004.

[18] Robert S. Fish, Robert E. Kraut, and Mary D. P. Leland. Quilt : a collaborative tool for cooperative writing. In *Proceedings of the ACM SIGOIS and IEEECS TC-OA 1988 conference on Office information systems*, pages 30–37, New York, NY, USA, 1988. ACM.

[19] Neil Fraser. mobwrite - real-time synchronization and collaboration service. http ://code.google.com/p/google-mobwrite/.

[20] Hans-W. Gellersen and Albrecht Schmidt. Look who's visiting : supporting visitor awareness in the web. *Int. J. Hum.-Comput. Stud.*, 56(1) :25–46, 2002.

[21] Anders Gidenstam and Marina Papatriantafilou. Adaptive plausible clocks. *icdcs*, 00 :86–93, 2004.

[22] Git - fast version control system. http ://git.or.cz.

[23] Gobby - a collaborative text editor. http ://gobby.0x539.de/.

[24] Googledocs and spreadsheets. http ://docs.google.com/.

[25] Saul Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems*, pages 227–237, New York, NY, USA, 1990. ACM.

[26] Saul Greenberg, Carl Gutwin, and Mark Roseman. Semantic telepointers for groupware. In *In Proceedings of the OzCHI Sixth Australian Conference on Computer-Human Interaction*, pages 54–61. Society Press, 1996.

[27] Saul Greenberg and David Marwood. Real time groupware as a distributed system : concurrency control and its effect on the interface. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW '94)*, pages 207–217, New York, NY, USA, 1994. ACM.

[28] C. Gutwin and R. Roseman. A usability study of workspace awareness widgets. In *ACM Conference on Human Factors in Computing System, Companion Proceedings - ACM (CHI'06)*, pages 214–215, April 13-17 1996. Also collected in Report 1995-580-32, December.

[29] Carl Gutwin and Saul Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, 6(3) :243–281, 1999.

[30] Carl Gutwin and Saul Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW '02)*, 11(3) :411–446, 2002.

[31] Carl Gutwin, Mark Roseman, and Saul Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *Proceedings of the ACM CSCW'96 Conference on Computer Supported Cooperative Work*, pages 258–267. ACM Press, 1996.

[32] S. Hayne, M. Pendergast, and S. Greenberg. Gesturing through cursors : implementing multiple pointers in groupsupport systems. *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, 4, 1993.

[33] Stephen Hayne, Mark Pendergast, and Saul Greenberg. Gesturing through cursors : implementing multiple pointers in group support systems. *System Sciences, 1993,*

Proceeding of the Twenty-Sixth Hawaii International Conference on, iv :4–12 vol.4, 5-8 Jan 1993.

[34] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. *Proceedings of the SIGCHI conference on Human factors in computing systems, (CHI '92)*, pages 3–9, 1992.

[35] Brent ByungHoon Kang, Robert Wilensky, and John Kubiatowicz. The hash history approach for reconciling mutual inconsistency. *In 23rd IEEE International Conference on Distributed Computing Systems, (ICDCS '03)*, pages 670–677, 2003.

[36] Michael J. Knister and Atul Prakash. Distedit : a distributed toolkit for supporting multiple group editors. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work, (CSCW '90)*, pages 343–355, New York, NY, USA, 1990. ACM.

[37] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[38] Yann Laurillau and Laurence Nigay. Clover architecture for groupware. In *CSCW '02 : Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 236–245, New York, NY, USA, 2002. ACM.

[39] Friedemann Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[40] Lola McGuffin and Gary Olson. Shredit : A shared electronic workspace. Technical report, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, 1992.

[41] Tom Mens. A state-of-the-art survey on software merging. *IEEE Trans. Softw. Eng.*, 28(5) :449–462, 2002.

[42] mercurial. http ://www.selenic.com/mercurial/wiki.

[43] James R. Miller, Serhan Yengulalp, and Patrick L. Sterner. A framework for collaborative control of applications. In *SAC '05 : Proceedings of the 2005 ACM symposium on Applied computing*, pages 1244–1249, New York, NY, USA, 2005. ACM.

[44] Pascal Molli, Hala Skaf-Molli, and Christophe Bouthier. State treemap : an awareness widget for multi-synchronous groupware. In *7th International Workshop on Groupware - CRIWG'2001*, pages 106–114, Darmstadt, Germany, September 2001.

[45] Pascal Molli, Hala Skaf-Molli, and Gérald Oster. Divergence awareness for virtual team through the web. In *Integrated Design and Process Technology, (IDPT 2002)*, pages 1–10, Pasadena, CA, USA, June 2002. Society for Desing and Process Science.

[46] Joseph C. Morris. DistriWiki : : a distributed peer-to-peer wiki network. *Proceedings of the 2007 international symposium on Wikis*, pages 69–74, 2007.

[47] Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok, and James H. Morris. Issues in the design of computer support for co-authoring and commenting. *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 183–195, 1990. PREP.

[48] Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok, and James H. Morris. Computer support for distributed collaborative writing : defining parameters of interaction. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW '94)*, pages 145–152, New York, NY, USA, 1994. ACM.

[49] Takahiko Nomura, Koichi Hayashi, Tan Hazama, and Stephan Gudmundson. Interlocus : workspace configuration mechanisms for activity awareness. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work (CSCW '98)*, pages 19–28, New York, NY, USA, 1998. ACM.

[50] Géald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *The Second International Conference on Collaborative Computing : Networking, Applications and Worksharing (CollaborateCom 2006)*, pages 1–10, Atlanta, Georgia, USA, November 2006. IEEE Press.

[51] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for p2p collaborative editing. In *CSCW '06 : Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268, New York, NY, USA, 2006. ACM.

[52] Stavroula Papadopoulou, Claudia Ignat, Gérald Oster, and Moira C. Norrie. Increasing Awareness in Collaborative Authoring through Edit Profiling. *Proceedings of the IEEE Conference on Collaborative Computing : Networking, Applications and Worksharing-CollaborateCom*, pages 1–10, 2006.

[53] Stavroula Papadopoulou and Moira C. Norrie. How a structured Document Model Can Support Awareness in Collaborative Authoring. In *3rd International IEEE Conference on Collaborative Computing : Networking, Applications and Worksharing*, pages 117–126, New York, USA, November 2007.

[54] Portland pattern repository. http ://c2.com/ppr.

[55] Atul Prakash and Hyong Sop Shim. Distview : support for building efficient collaborative applications using replicated objects. In *Proceedings of the 1994 ACM*

conference on Computer supported cooperative work, (CSCW '94), pages 153–164, New York, NY, USA, 1994. ACM.

[56] Th. Prante, N. A. Streitz, and P. Tandler. Roomware : Computers Disappear and Interaction Evolves. *IEEE Computer*, pages 47–54, December 2004.

[57] Gail L. Rein and Clarence A. Ellis. ribis : A real-time group hypertext system. *International Journal of Man-Machine Studies*, 34(3) :349–367, 1991.

[58] Matthias Ressel, Doris Nitsche−ruhl, and Rul Gunzenh Auser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *Proceedings of the ACM conference on Computer supported cooperative work*, pages 288–297, 1996.

[59] Tom Rodden. Populating the application : a model of awareness for cooperative applications. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work (CSCW '96)*, pages 87–96, New York, NY, USA, 1996. ACM.

[60] Roseman, Mark and Greenberg, Saul. Building real-time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1) :66–106, 1996.

[61] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys (CSUR '05)*, 37(1) :42–81, 2005.

[62] Anita Sarma, Zahra Noroozi, and André van der Hoek. Palantír : Raising awareness among configuration management workspaces. In *Proceedings of the 2003 IEEE International Conference on Software Engineering (ICSE'03)*, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[63] sawsan Alshattnawi, Gérôme Canals, and Pascal Molli. concurrency awareness in P2P wiki. In *The 2008 International Symposium on Collaborative Technologies and Systems (CTS 2008)*, pages 285–294. IEEE, 2008.

[64] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. Wysiwis revised : early experiences with multiuser interfaces. *ACM Trans. Inf. Syst.*, 5(2) :147–167, 1987.

[65] Subversion. Open Source Software Engineering Tools. http ://subversion.tigris.org/.

[66] Maher Suleiman, Michèle Cart, and Jean Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work, (GROUP '97)*, pages 435–445, New York, NY, USA, 1997. ACM.

[67] Maher Suleiman, Michèle Cart, and Jean Ferrié. Concurrent operations in a distributed and mobile collaborative environment. In *Proceedings of the Fourteenth International Conference on Data Engineering, (ICDE '98)*, pages 36–45, Washington, DC, USA, 1998. IEEE Computer Society.

[68] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1) :63–108, 1998.

[69] David Sun and Chengzheng Sun. Operation context and context-based operational transformation. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW '06)*, pages 279–288, New York, NY, USA, 2006. ACM.

[70] Synchroedit - synchronous editing for the web. http ://www.synchroedit.com/.

[71] James Tam and saul Greenberg. A framework for asynchronous change awareness in collaborative documents and workspaces. *International Journal of Human-Computer Studies*, 64(7) :583–598, 2006.

[72] Walter F. Tichy. RCS - A System for Version Control. *Software - Practice and Experience*, 15(7) :637–654, 1985.

[73] Konrad Tollmar, Ovidiu Sandor, and Anna Schömer. Supporting social awareness work design and experience. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work, (CSCW '96)*, pages 298–307, New York, NY, USA, 1996. ACM.

[74] Francisco J. Torres-Rojas and Mustaque Ahamad. Plausible clocks : constant size logical clocks for distributed systems. *Distrib. Comput.*, 12(4) :179–195, 1999.

[75] Fernanda Viegas, Martin Wattenberg, and Kushal Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the 2004 ACM Conference on Computer Human Interactions (CHI 2004)*, pages 575–582, Vienna, Austria, 2004. ACM Press.

[76] Stéphane Weiss, Pascal Urso, and Pascal Molli. Wooki : a p2p wiki-based collaborative writing tool. In *Web Information Systems Engineering*, Nancy, France, December 2007. Springer.

[77] Wikimatrix - compare them all. http ://www.wikimatrix.org/.

[78] Wikipedia. The Free Encyclopædia that Anyone Can Edit. http ://www.wikipedia.org/.

[79] Wikipedia Statistics. http ://stats.wikimedia.org/.

[80] Wikiwikiweb, also known as wards's wiki or just wiki. http ://c2.com/cgi/wiki ?WikiWikiWeb.

[81] XWIKI. Free your Knowledge. http ://www.xwiki.org/xwiki/bin/view/Main/WebHome.

Résumé : Récemment, les wikis sont devenus les outils d'édition collaborative les plus populaires. Ils doivent maintenant faire face à une forte augmentation en quantité et complexité des données gérées en nombre d'utilisateurs. Pour répondre à ce problème, le passage d'une architecture client/serveur vers une architecture décentralisée sur réseau pair-à-pair est une voie possible. Elle pose cependant des problèmes liés à la concurrence des mises à jour sur des sites distants. Ce document décrit deux contributions.La première propose un mécanisme totalement décentralisé pour la conscience de la concurrence dans une édition collaborative sur réseaux P2P. Son rôle est de permettre aux visiteurs d'un wiki d'avoir conscience du statut d'une page vis-à-vis de la concurrence - s'agit-il d'une page éditée ou fusionnée ? - et dans le cas d'une page fusionnée, d'avoir une vision des zones de la page touchées par la fusion. Ce mécanisme repose sur un détecteur de concurrence entre les patchs. La deuxième contribution porte sur la visualisation d'un historique concurrent. La visualisation proposée se base sur la visualisation linéaire classique : chaque site affiche la séquence locale de versions dans l'ordre où elles sont apparues, et complète cet historique avec des informations permettant de repérer les états édités et fusionnés et les parties concurrentes dans l'histoire.

Mots-clés : Édition collaborative, concurrence, conscience de groupe, Wiki, réseaux P2P.

Concurrence and group awareness in collaborative editing systems over Peer-to-Peer networks

Abstract : Currently, Wikis are the most popular form of collaborative editors.We anticipate large increasing of amount and complexity of data. To face this problem, some researches have been done to shift from centralized architecture to fully decentralized wikis relying on peer-to-peer networks. However, this approach leads to new problem related to concurrency and the way remote modifications are integrated at each site. To overcome this problem, this thesis introduces the idea of concurrency awareness and proposes two contributions. The first one is to build a concurrency awareness mechanism for a P2P wiki. This mechanism makes users aware of the status of the pages they access regarding concurrency : is it an edited page or a merged page ? In addition, in case of merged page, it indicates which region of the page has been merged. This mechanism depends over a concurrency detection mechanism. The second contribution deals with the representation of the concurrent history. Our visualisation is based over the classical history visualisation : the local versions are presented at the same order of their creation, and we added the information that present the status of these versions according to the concurrence.

Keywords : Collaborative edition, concurrence, group awareness, Wiki, P2P networks.