Cohérence des données dans les environnements d'édition collaborative

Pascal Molli

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

Cohérence des données dans les environnements d'édition collaborative

MÉMOIRE

présenté et soutenu publiquement le 26 Avril 2007

pour l'obtention de l'

Habilitation de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

Pascal Molli

Composition du jury

Président :	Jean Ferrié, Professeur à l'Université de Montpellier
Rapporteurs :	Philippe Pucheral, Professeur à l'Université de Versailles Jacky Estublier, Directeur de Recherche CNRS, Grenoble
Examinateurs :	Claude Godart, Professeur à l'Université Henri Poincaré Michaël Rusinowitch, Directeur de recherche INRIA, LORIA

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

Mis en page avec la classe thloria.

Table des matières

Table des figures

1 Avant-propos

Le document est organisé comme suit :

- La première partie détaille mon curriculum vitae. Je détaille succintement les traits marquants de mon parcours.
- Le seconde partie présente mon activité de recherche. J'insiste plus particulièrement sur mes résultats récents.
- La troisième partie décrit les logiciels auxquels j'ai participé et mon rôle dans chacun d'eux.
- La dernière partie présente mon activité d'enseignement.

Première partie

Curriculum Vitae

Pascal MOLLI
né le 15 octobre 1967 à Villerupt (France)
Marié, 2 enfants
Campus Scientifique
Bâtiment LORIA-Inria Lorraine, BP239
54506 Vandoeuvre-lès-Nancy.
03.83.59.30.77 (bureau)
03.83.41.30.79 (fax)
Pascal.Molli@loria.fr
http://www.loria.fr/~molli

Domaine de recherche Édition collaborative, Réplication optimiste, Transformées opérationnelles, CSCW.

Domaine d'enseignement Programmation orienté objet, Conception orienté objet, Systèmes distribués.

2 Situation Actuelle

- Maître de Conférences en Informatique à l'Université Henri Poincaré, Nancy 1, nommé au 1er septembre 1997.
- Chercheur au LORIA dans le projet ECOO Environnement et COOpération[1] sous la direction de Claude Godart, professeur à l'Université Henri Poincaré, Nancy 1, ESSTIN [2].

3 Titres Universitaires

1996 - Docteur en Informatique de l'Université Henri Poincaré , Nancy 1. Thèse soutenue le 11 juin 1996 à l'ESSTIN et préparée sous la direction de Claude Godart, professeur à l'Université Henri Poincaré, Nancy 1, ESSTIN.
Titre : *"Environnements de développement coopératifs"*
Mention : *Très honorable avec félicitations du jury.*
Jury :

Président	M. Jean-Marie Pierrel, professeur à l'Université Henri Poincaré, Nancy 1.
Rapporteurs	M. Jean Ferrié, professeur à l'Université de Montpellier.
	M. Jacky Estublier, Directeur de recherche à l'IMAG Grenoble.
Examinateurs	M. Claude Godart, professeur à l'Université Henri Poincaré, Nancy 1, ESSTIN.
	M. Rachid Guerraoui, École Polytechnique Fédérale de Lausanne.
	M. Gérôme Canals, Maître de Conférences à l'Université Nancy 2.

1991 - Diplôme d'Études Approfondies en Informatique - CRIN, Université Henri Poincaré, Nancy 1. Septembre 1991
Titre : *"Transactions Longues"*

[1] `http://www.inria.fr/recherche/equipes/ecoo.fr.html`
[2] École Supérieure des Sciences et Technologies de l'Ingénieur de Nancy

Responsable : Claude Godart.
Mention : Assez Bien.

1990 - Maîtrise en Informatique - Université Henri Poincaré, Nancy 1. Juin 1990.
Mention : Assez Bien.

1989 - Licence en Informatique - Université Henri Poincaré, Nancy 1. Juin 1989
Mention : Assez Bien.

1988 - DEUG A - Université Henri Poincaré, Nancy 1. Juin 1988

4 Principaux résultats de recherche

Coo-Serialisabilité la COO-sérialisabilité est un critère de correction pour l'édition collaborative des données. La COO-sérialisabilité étend la traditionnelle sérialisabilité en cassant l'isolation tout en préservant la cohérence. J'ai formalisé la COO-sérialisabilité ainsi que les modèles transactionnels qui en découlent en ACTA, un environnement formel dédié à l'expression de modèles transactionnels. Ce travail a fait l'objet d'une publication majeure[4][Information Sciences].

Le protocole permettant d'évaluer de manière incrémentale ces critères a été publié à [18][ATMA'96] et à [38][SCM7].

Le protocole COO est implanté au sein de l'environnement COO[19][ICSE96] développé par l'équipe ECOO du LORIA. Cet environnement permet de travailler de manière coopérative dans le cadre d'un atelier de génie logiciel.

Edition collaborative multi-synchrone J'ai montré que les outils d'édition collaborative en temps réels, sont tout à fait adaptés à l'édition collaborative asynchrone. Il est donc possible d'écrire un outil qui supporte à la fois l'édition collaborative temps réel et distribuée dans le temps. Ces outils sont alors des éditeurs collaboratifs multi-synchrone [25].

Synchroniseurs sûrs et génériques La réconciliation de données divergentes est un problème récurrent dans le domaine de l'ingénierie concurrente, de l'informatique mobile et dans la gestion de configuration. De nombreux outils de synchronisation ou de fusion sont actuellement disponibles. Cependant, ils ne définissent pas ce qu'est une synchronisation correcte. Nous utilisons le modèle des transformées opérationnelles pour raisonner sur la correction des synchronisations. Nous avons proposés un algorithme ainsi que les fonctions de transformation adéquates pour synchroniser un système de fichiers. A la différence des synchroniseurs classiques, notre synchroniseur garantit les propriétés de convergence, de causalité et de respect de l'intention. Il permet en outre de synchroniser de nouveaux types de données en définissant de nouvelles fonctions de transformation [49]. L'environnement de preuve VOTE[46] nous a permis d'écrire et de prouver la correction de fonction de transformation pour les fichiers textes, le système de fichiers lui-même ainsi que pour les fichier XML.

Preuve automatique de fonctions de transformation Les algorithmes de transformées opérationnelles s'appuient sur la définition de fonctions de transformation. Si ces fonctions ne sont pas correctes, alors ces algorithmes ne peuvent pas garantir la cohérence des données partagées.

La preuve de ces fonctions de transformation, même sur des types simples comme une chaîne de caractères, est une tâche difficile. Si nous voulons développer des fonctions de transformation pour des types complexes, la preuve est impossible sans l'aide d'un ordinateur. Ces problèmes de preuve sont un obstacle majeur pour toute l'approche des transformées opérationnelles.

Notre approche consiste à utiliser le prouveur automatique de théorème SPIKE [10] pour automatiser la preuve des fonctions de transformation. SPIKE est un prouveur de théorèmes

[10] Adel Bouhoula. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science*, 170(1-2) :245–276, 1996.

basé sur la définition de règles de réécriture et sur l'induction [32][45]. Il a été développé dans le cadre du projet INRIA CASSIS avec lequel nous menons une collaboration étroite depuis 2002.

L'environnement VOTE permet de spécifier facilement les fonctions de transformation. Cet environnement génère ensuite la spécification SPIKE correspondante et SPIKE peut alors déterminer rapidement la correction de la spécification. Si la spécification est incorrecte, SPIKE fournit les contre-exemples. Il est alors facile de localiser les erreurs dans la spécification des fonctions de transformations.

Cette approche nous a permis de trouver des contre-exemples pour toutes les fonctions de transformation existantes [26] écrites depuis 1989. Enfin cette année, nous avons proposé de nouvelles fonctions de transformation sur-lesquelles nous avons pu prouver automatiquement la correction [30, 10].

5 Logiciels diffusés

Libresource (Diffusé depuis Juin 2005) Libresource [3] est une plate-forme de travail collaboratif diffusée sous licence libre en QPL et commercialisée par la société Artenum.

Libresource a reçu le prix spécial du jury 2006, de la conférence ObjectWebCon. J'ai été récompensé par la médaille du LORIA 2005 pour cette réalisation.

Libresource est un transfert industriel réussi.

jXYDiff (Diffusé depuis mars 2006) jXYDiff est une librairie java pour calculer les différences entre deux fichiers XML. Elle est diffusée sous licence libre QPL. jXYDiff a été réalisé en coopération avec Eric Cobena sur la base de ses travaux de thèse dirigés par S. Abiteboul.

JlibDiff (Diffusé depuis 1999) jLibDiff est librairie java pour calculer les différences entre deux fichiers textes. Elle est diffusée sous licence LGPL et disponible sur Sourceforge.

6 Recherche contractuelle

COCAO (1998-2001) .

L'objet de cette étude était de mieux comprendre les principes de la coopération pour la mettre en oeuvre de façon simple et acceptable par les utilisateurs. Les résultats obtenus sont une taxonomie d'usages coopératifs de base et des protocoles de coopération mettant en oeuvre ces usages par assemblage.

Le travail s'est fait en coopération avec une équipe du CNET à Lannion et une équipe du CRAI à Nancy (Centre de Recherche en Architecture et Ingénierie).

[3] `www.libresource.org`

[32] Sorin Stratulat. A general framework to build contextual cover set. *Journal of Symbolic Computation*, 32(4) :403–445, 2001.

[26] G ?rald Oster. *R ?plication optimiste et coh ?rence des donn ?es dans les environnements collaboratifs r ?partis.* PhD thesis, Universit ? Henri Poincar ?, Nancy1, November 2005.

Le résultat de ce projet est le prototype MOTU [39] qui a servi de base à la réalisation du film de démonstration des Motus.

CPER QSL : Action VXP (2001-2003) Le thème "Qualité et Sûreté des Logiciels (QSL)[4]" est l'un des thèmes du pôle "Intelligence Logicielle" du Contrat Plan État Région (CPER) 2000–2006.

Le projet "Virtual eXtreme Programming" (VXP) doit permettre à une équipe virtuelle de travailler selon la méthodologie eXtreme Programming (XP). Cependant, la méthodologie XP impose que l'équipe de développement soit dans une même pièce et travaille en même temps. Par définition, une équipe virtuelle est un groupe de personne distribué dans l'espace, le temps et à travers les organisations. Nous pensons que les éditeurs collaboratifs multi-synchrones sont tout à fait adaptés pour résoudre ce paradoxe. Dans le cadre de VXP, nous spécialisons un éditeur collaboratif multi-synchrone pour prendre en compte la méthodologie XP.

Dans le cadre du CPER, j'ai organisé une journée d'initiation :"gestion les architectures logicielles" et un séminaire d'initiation à la gestion de configuration. Mes interventions lors de ces journées sont basées sur les cours détaillés en sections 1.3 et 1.4.

Le résultat concrêt de cette action est le prototype SAMS [25]

COOPERA : RIAMM (2002-2004) .

Le projet Coopera a été financé de février 2002 à janvier 2004 par le Ministère de la culture, via le réseau RIAMM [5]

L'objectif du projet Coopera a consisté à concevoir et réaliser un environnement technique et pédagogique pour la mise en œuvre de démarches pédagogiques innovantes, incluant des échanges d'informations et des productions de documents communs.

Ce projet a permis de mener des analyses d'usage de la plate-forme collaborative Toxic.

Le résultat de ce projet est le prototype TOXIC [28, 9]

CPER QSL : Action Mobi5 (2004-2006) Le travail mobile a démocratisé l'utilisation de synchroniseurs de données, comme ActiveSync de Microsoft, HotSync de 3COM ou I-Sync d'Apple. Ces synchroniseurs souffrent cependant de nombreux défauts : la correction de la synchronisation n'est pas définie (d'où des risques de perte de données), la synchronisation reste souvent limitée à deux copies, la résolution de conflits de synchronisation est seulement semi-automatique et l'ajout de nouveaux types de données n'est pas géré. Le modèle des transformées opérationnelles, bien que développé à l'origine pour les systèmes de travail de groupe temps réel, peut être utilisé pour construire un synchroniseur de données d'un type radicalement nouveau.

Un Synchroniseur sûr Un algorithme de synchronisation basé sur les Transformées Opérationnelles garantit les propriétés de convergence (les différentes versions des réplicats doivent être identiques après synchronisation), de causalité (préservation de l'ordre des opérations) et de respect de l'intention dans tous les cas.

un Synchroniseur générique Dans le modèle des Transformées Opérationnelles, l'algorithme d'intégration est indépendant des types de données. C'est donc le même

[4] `http://qsl.loria.fr`

[5] Recherche et Innovation en Audiovisuel et Multimédia, www.cnc.fr/riam

algorithme garantissant les mêmes propriétés qui va réconcilier un système de fichiers, le contenu d'un fichier XML, un calendrier partagé, une base de données etc.

L'objectif du projet est de construire un synchroniseur sûr et générique basé sur le modèle des Transformées Opérationnelles. Un tel synchroniseur doit permettre à un utilisateur ou à un groupe d'utilisateurs de synchroniser des données en toute sécurité, i.e. en respectant les propriétés de convergence, de causalité et de préservation de l'intention.

Le résultat concrêt de cette action sont les fonctions de transformation utilisées dans Libresource [27].

Projet RNTL LibreSource (2002/2004) L'objectif de Libresource[6] est de développer une plate-forme modulaire intégrant les briques indispensables à la mise en place d'applications collaboratives. L'identification de ces modules génériques, nécessite une analyse fine des besoins et contraintes liés au travail collaboratif. La structure modulaire de Libresource permettra la conception de déclinaisons métiers intégrant les outils spécifiques à certaines communautés.

ARC Recall (2006-2008) L'objectif de l'ARC est de développer des algorithmes de réplication optimiste adaptés à l'édition collaborative massive. Ces algorithmes doivent permettre le déploiement des applications collaboratives classiques sur des réseaux P2P. Ces applications pourront alors passer à l'échelle et tolérer les pannes sans nécessiter des solutions matérielles coûteuses. Cette ARC montrera le potentiel des réseaux P2P non seulement pour diffuser du contenu mais aussi pour créer et éditer du contenu.

Qualipso (2006-2010) Le projet européen Qualipso a pour objectif d'améliorer la qualité des logiciels Open Source. J'interviens dans ce projet en tant qu'expert des plate-formes collaboratives de développement. L'expérience acquise dans Libresource doit permettre de fixer les spécifications d'une plate-forme collaborative de développement de nouvelle génération.

7 Animation scientifique

Gdr-PRC I3 (2003-2004) Gdr-PRC I3[7] (Information-Interaction-Intelligence), Axe 4 "Interaction et Coopération" GT 4.2 "mobilité et ubiquité", action synchronisation.

La multiplication des ordinateurs de poche et des assistants personnels, la généralisation de l'utilisation des cartes à puces et la multiplication des systèmes embarqués dans des objets d'usage courant (automobile, télévision...) sont des témoins de l'essor actuel de l'informatique mobile. L'explosion de la téléphonie mobile constitue un autre exemple significatif de la formidable extension de ce domaine. Dans ce contexte en pleine effervescence, la mise en place de méthode de conception ergonomique de l'interaction avec un support mobile, de modèles d'architecture logicielle (plate-forme logicielle) pour la conception et le développement d'applications, notamment pour l'accès à des bases de données, sur des supports mobiles et/ou à capacité réduite est cruciale [6, 7].

Dans ce cadre, l'action synchronisation doit fournir un support pour supporter les alternances de travail connecté et déconnecté. J'ai fait une présentation le 27/05/2003 sur le

[6] `http://www.libresource.org`
[7] `http://sis.univ-tln.fr/gdri3/`

thème de la synchronisation de données divergentes[8].

Les publications [6, 7] ont concrétisées les réflexions menées dans ce GDR.

FIDJI 2002,2003 Je suis membre du comité de programme de FIDJI 2002 et 2003 [17,18].

FIDJI est un forum international pour des chercheurs et des praticiens intéressés par les avancées et les applications du génie logiciel pour le développement distribué d'application. Pour ce qui concerne les technologies, l'atelier se concentre sur des technologies reliées "par Java". C'est une occasion de présenter et observer les dernières recherches, résultats, et idées dans ces secteurs.

ICEIS 2003,2004 Je suis membre du comité de programme de ICEIS 2003 et 2004 [1,2]

Les thèmes ICEIS sont les suivants :

- Bases de données et systèmes d'information
- Intelligence artificielle et Systèmes d'aide à la décision
- Spécification et analyse des systèmes d'information
- Agents logiciels et "Internet Computing".

Collaborative Editing 2006 Je suis membre du comité de programme du 8ième workshop sur l'édition de groupe [3]. C'est le workshop le plus pointu de mon domaine.

Relecture pour des conférences et journaux Je relis régulièrement les articles de recherche concernant l'édition collaborative pour les conférences CSCW,ECSCW,GROUP et pour le journal IEEE Transaction on Parallel and Distributed System.

8 Coopérations nationales et internationales

8.1 LIRMM : Jean Ferrié, Michelle Cart

Nous travaillons de manière étroite avec Jean Ferrié et Michelle Cart du LIRMM Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier[9]. Ils sont à la pointe du domaine des transformées opérationnelles avec les algorithmes SOCT2,3,4,5.

Nous pensons que le principe des transformées opérationnelles peut servir de modèle général pour la conception et la réalisation de Synchroniseurs génériques, adaptables à tous types d'objets et indépendants de leur structure, moyennant l'écriture de fonctions de transformation appropriées (cf section 2.1.2)

[8] `http://www.loria.fr/~molli/rech/talk/synchro.ppt`

[9] `http://www.lirmm.fr`

[17] Nicolas Guelfi, Egidio Astesiano, and Gianna Reggio, editors. *Scientific Engineering for Distributed Java Applications, International Workshop, FIDJI 2002, Luxembourg-Kirchberg, Luxembourg, November 28-29, 2002, Revised Papers*, volume 2604 of *Lecture Notes in Computer Science*. Springer, 2003.

[18] Nicolas Guelfi, Egidio Astesiano, and Gianna Reggio, editors. *Scientific Engineering of Distributed Java Applications, Third International Workshop, FIDJI 2003, Luxenbourg-Kirchberg, Luxenbourg, November 27-28, 2003, Revised Papers*, volume 2952 of *Lecture Notes in Computer Science*. Springer, 2004.

[1] *Proceedings of the 5th International Conference on Enterprise Information Systems*, Angers, FRANCE, Avril 2003.

[2] *Proceedings of the 6th International Conference on Enterprise Information Systems*, Porto, Portugal, Avril 2004.

[3] *8th International Workshop on Collaborative Editing Systems*, Banff, Canada, November 2006.

La coopération avec le LIRMM est concrétisée pour l'instant de la manière suivante :

- J'ai participé en tant qu'examinateur au jury de thèse de Nicolas Vidot qui a eu lieu le 20 septembre 2002.
- Gérald Oster, dont j'ai co-encadré la thèse, a passé une semaine au LIRMM dans le cadre de l'action AS Mobilité, pour travailler sur la définition et la résolution automatique de conflits dintégration de copies divergentes.
- Nicolas Vidot a passé une semaine au LORIA pour travailler sur les condition nécessaire à l'annulation de groupe dans les éditeurs collaboratifs.

8.2 CASSIS-LORIA : Michaël Rusinowitch

L'approche des transformées opérationnelles (OT) permet de construire des éditeurs collaboratifs. Dans le cadre de cette approche, les algorithmes comme aDOPTed, GOTO, SOCT 2,3,4 font l'hypothèse qu'il existe des fonctions de transformations T qui vérifient les conditions suivantes :

- $C1 : op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$.
  $T(op_2, op_1)$ signifie que op_2 est transformée selon op_1.
- $C2 : T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$

La preuve de ces conditions est complexe et fastidieuse. Nous coopérons avec l'équipe CASSIS à la preuve automatique de ces propriétés en utilisant le prouveur automatique SPIKE. Nous avons développé un jeu prouvé de transformées pour les chaînes de caractère [45], XML [26, 46], Système de fichier ... Nous continuons notre coopération à la preuve automatique des conditions $C3$ et $C4$ développées par le LIRMM.

8.3 University of New South Wales, Australia : Pradeep Ray

On trouve aujourd'hui sur internet des services WEB de gestion d'équipes virtuelles. Des fournisseurs comme Sourceforge, Savannah, Groove, BSCW... permettent l'hébergement d'équipes distribuées. Si ces plates-formes fournissent des fonctionnalités, elles doivent en plus fournir un niveau de qualité de service (QoS) suffisant. Nous travaillons avec Pradeep Ray[10] sur la définition et l'expérimentation de QoS sur des services WEB d'hébergement d'équipes virtuelles [24]. La plate-forme TOXIC[11] sert de base a cette expérimentation [44, 9].

8.4 University of Sydney, Australia : Olivera Marjanovic

Nous travaillons sur l'adaption des plate-formes collaborative aux besoins de l'E-learning [31, 34, 11]. En effet, les plate-formes actuelles sont orientées vers la diffusion connaissances. Nous voulons faire une plate-forme permettant la construction collaborative d'un savoir commun.

8.5 Université de Tripoli : Hala Naja

RORAX est programme de recherche Franco-Libanais financé sur la période 2006-2008. Nous travaillons avec Hala Naja de l'université de Tripoli sur la réplication optimiste et réparation

[10] `http://www.sistm.unsw.edu.au/people/pradeep/`
[11] `http://woinville.loria.fr`

automatique de documents XML.

9 Encadrements

Thèses

- J'ai co-encadré avec Claude Godart la thèse de Gérald Oster intitulée " Réplication optimiste et cohérence des données dans les environnements collaboratifs répartis" et soutenue le 3 novembre 2005. Gérald Oster est parti en postdoc à l'ETH zurich dans l'équipe Globis de novembre 2005 à septembre 2006. Gérald Oster est maintenant Maître de Conférence à l'ESIAL.
- Je suis rapporteur de la thèse de Claudia Ignat, actuellement doctorante dans l'équipe Globis à l'ETH Zurich sous la direction de Moira Norrie. Claudia Ignat a soutenu le 31 juillet 2006 une thèse intitulée "Consistency Maintenance of Hierarchical Documents". Elle est acceptée en stage post-doctoral dans l'équipe ECOO à partir de septembre 2006.
- Je suis examinateur dans les thèses suivantes :
  - Christophe Bobineau, "Gestion de transactions en environnement mobile", Décembre 2002.
  - Nicolas Vidot, "Convergence des Copies dans les Environnements Collaboratifs Répartis", Septembre 2002.
  - Lydialle Chateigner, "Intergiciel extensible à base de composants adaptables pour l'informatique mobile : Réplication optimiste et réconciliation", Juillet 2006
- J'ai co-encadré avec Claude Godart, la thèse d'Abdelmajid Bouazza sur la période 1998-2002 portant sur le travail multi-synchrone. Si le travail a débouché sur un résultat majeur [25], malheureusement, Abdelmajid Bouazza n'a pu soutenir sa thèse pour des raisons personnelles.

DEA et Master Recherche

J'ai encadré les stages de DEA suivants :

- Abdelmajid Bouazza (1996-1997). Modélisation d'espaces de travail.
- Gérald Oster (2000-2001). Visualisation de la divergence dans les équipes virtuelles.
- Sébastien Jourdain (2001-2002). SAMS : Environnements coopératifs Synchrones, Asynchrone, Multi-Synchrone pour les équipes virtuelle. Sébastien Jourdain est aujourd'hui ingénieur dans la compagnie Artenum partenaire du projet RNTL.
- Xin Chen (2004-2005). Réplication optimiste de données XML dans un environnement collaboratif répartis.
- Stéphane Weiss (2005-2006). Annulation de groupe dans les éditeurs collaboratifs. Stéphane Weiss continue en thèse sur le sujet : "édition collaborative massive". Ce sujet est en rapport avec l'ARC RECALL.

Ingénieurs

- Marc Patten (1999-2003), stagiaire puis ingénieur associé INRIA. Développement de MOTU, TOXIC puis COOPERA. Marc Patten est maintenant architecte chez Unilog (www.unilog.fr)

à Paris.
- Guillaume Bort (Libresource RNTL - 2003/2004), Développement de Libresource. Guillaume Bort travaille actuellement comme architecte chez Zenexity (http ://www.zenexity.fr/)
- Sébastien Jourdain (2002/2004), Ingénieur associé INRIA, Développement de Libresource. Sébastien Jourdain travaille désormais chez Artenum (http ://www.artenum.com/)
- Florent Jouille (2004/2006), Ingénieur associé, Développement de Libresource.

Deuxième partie

Activités de recherche

Introduction

Les outils d'édition collaborative permettent à un groupe de personnes distribuées dans le temps, dans l'espace et à travers les organisations, de travailler ensemble sur les mêmes documents.

Il est regrettable de constater aujourd'hui que la manière la plus répandue de travailler ensemble reste encore l'échange des fichiers par courrier électronique. Il faut alors prendre garde à ne pas générer de modifications concurrentes sous peine d'avoir à fusionner les différentes versions à la main. Cette manière de travailler est peu efficace et ne permet pas la parallélisation des tâches d'édition.

Un système d'édition collaborative efficace doit permettre à n'importe qui de modifier n'importe quel type de données à n'importe quel moment.

Les éditeurs Wikis permettent bien à n'importe qui d'éditer n'importe quand, mais seulement des pages Wikis. Ils offrent cependant une manière simple d'éditer des pages Web ce qui explique leur énorme popularité. Les éditeurs Wikis souffrent cependant d'un problème de fond, ils supportent mal l'édition concurrente d'une même page Wiki par 2 utilisateurs en parallèle. Dans ce cas, le dernier qui écrit va masquer les modifications du premier. C'est une forme de mise-à-jour perdue.

Un système comme CVS traite le problème différemment. CVS permet bien à n'importe qui d'éditer n'importe quand, mais seulement des fichiers textes. En cas d'édition concurrente conflictuelle, CVS génère un bloc de texte représentant le conflit directement dans le fichier incriminé.

Un système d'édition collaborative efficace doit donc permettre à n'importe qui d'éditer n'importe quel type de données n'importe quand en évitant les mises à jour perdues.

- La taille du groupe est une donnée importante. On ne coopère pas de la même manière à 2, à 10, ou à 10000. Un modèle de partage de données adapté pour un groupe fermé de 2 personnes peut être rédhibitoire pour un groupe ouvert de 1000 personnes. La communication et la coordination d'un groupe de 2 personnes n'a rien à voir avec la communication et la coordination d'un groupe de 100 personnes.
- Le type des documents partagés jouent aussi un rôle important. On ne partage pas un calendrier de réservation de salle, comme une page Wiki ou encore comme on partage un article de recherche. Chaque type de donnée a aujourd'hui son outil collaboratif dédié. On peut partager son calendrier avec "google calendar", ses pages Wikis avec "mediawiki", ou encore son article de recherche avec le suivi de documents de Microsoft Word.
- Le nombre de documents partagés amène un ensemble de contraintes. 10 personnes travaillant sur 1 document texte ne se gère par comme 10 architectes travaillant avec une armoire à plan contenant des milliers de documents.
- La provenance des participants est aussi déterminante : la coopération inter-organisationnelle pose des problèmes qui n'existent pas au sein d'une même organisation. Le niveau de confiance et de confidentialité n'est pas le même.

Je me suis principalement aux problèmes de cohérence des données partagées dans les systèmes d'édition collaborative. Aujourd'hui plusieurs approches existent pour gérer la cohérence des données partagées :

l'approche "turn taking". Cette approche évite les problèmes en n'autorisant qu'un seul utilisateur actif. Chacun modifie le document à son tour. Dans ce cas, il n'existe pas de modifications concurrentes, toutes les modifications sont naturellement sérialisées. Il n'y a donc pas besoin de fusionner deux versions du documents. De nombreux logiciels utilisent cette approche en la déclinant selon différentes variantes [16]. Cette approche ne permet pas de supporter plusieurs utilisateurs actifs. On ne peut donc pas paralléliser les tâches d'édition.

L'approche "verrouillage". Verrouiller les objets est une approche classique de contrôle de concurrence. Avant d'éditer, l'utilisateur verrouille l'objet à éditer pour obtenir un accès exclusif et peut donc ensuite modifier l'objet tout en étant protégé des accès concurrents. Si il n'y a qu'un seul document partagé, cette approche n'est pas très différente de l'approche "turn taking".

Si un document est considéré comme un objet composite, alors le verrouillage peut se faire au niveau, d'une ligne, d'un paragraphe ou d'une section. Dans ce cas, le travail en parallèle est possible, deux utilisateurs peuvent modifier deux parties disjointes du document en même temps contrairement à l'approche "turn-taking".

Malheureusement, cette approche n'est pas généralisable à toutes les situations. En effet, la durée du verrouillage n'est pas prédictible. Un utilisateur peut donc attendre un temps non déterminé avant de pouvoir faire ses modifications.

De nombreux systèmes d'édition collaborative utilise néanmoins cette approche [6].

L'approche Base de données. Dans cette approche, les données sont stockées dans une base de données et les utilisateurs interagissent avec les objets partagés à travers des transactions. Les protocoles transactionnels vont alors forcer des exécutions sérialisables. Cette approche assure la cohérence des données mais souffre de deux défauts majeurs :

- Les protocoles transactionnels comme le 2PC, utilisent des techniques de verrouillage et donc souffrent des mêmes problèmes que l'approche basée sur le verrouillage. Cela rejoint la problématique des transactions longues [4].
- Les exécutions générées par des activités d'édition collaboratives sont très souvent non sérialisables. C'est alors, la base même de la gestion des accès concurrents dans les bases de données qui est remise en cause.

L'approche multi-version, fusion d'état Les gestionnaires de configuration, tels que CVS [9], Subversion, ou encore ClearCase [12,5] permettent à des utilisateurs de partager des docu-

---

[16] S. Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of the ACM/IEEE Conference on Office Information Systems*, pages 227–237, Cambridge, Massachusets, 1990.

[6] Ronald Baecker, Geof Glass, Alex Mitchell, and Ilona Posner. Sasse : the collaborative editor. In Catherine Plaisant, editor, *CHI Conference Companion*, pages 459–462. ACM, 1994.

[4] D. Agrawal and A.El. Abbadi. Transaction Management in Database Systems. In A.K. Elmagarmid, editor, *Database transaction models for advanced applications*. Morgan Kauffman, 1990.

[9] Brian Berliner. CVS II : Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, California, United States, 1990. USENIX Association.

[12] Rational ClearCase. http ://www-306.ibm.com/software/awdtools/clearcase/. June 2005.

[5] Larry Allen, Gary Fernandez, Kenneth Kane, David Leblang, Debra Minard, and John Posner. ClearCase MultiSite : Supporting geographically-distributed software development. In Jacky Estublier, editor, *Software Configuration Management : Selected Papers of the ICSE SCM-4 and SCM-5 Workshops*, number 1005 in Lecture Notes in Computer Science, pages 194–214. Springer-Verlag, October 1995.

ments et de les éditer de manière collaborative. Ces environnements intègrent un système de gestion de versions qui repose sur un protocole optimiste de gestion des accès concurrents : le paradigme du *Copier-Modifier-Fusionner*.

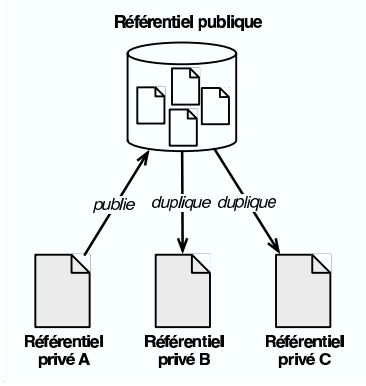


FIG. 1 – Référentiel/espaces de travail

On distingue deux types d'espaces (cf. figure 1) :

un référentiel. Cet espace maintient un ensemble multi-versionné des objets partagés. C'est à dire que pour chaque objet partagé, il conserve l'ensemble des versions de cet objet dont la publication par les utilisateurs a réussi ;

des espaces de travail. Chaque utilisateur possède son propre espace de travail. Les modifications apportées dans cet espace restent confidentielles jusqu'à leur publication.

Pour modifier un objet, l'utilisateur en crée une réplique dans son espace de travail. Il peut ensuite librement la modifier. Une fois sa tâche terminée, il publie une nouvelle version dans le référentiel.

Deux utilisateurs peuvent modifier en parallèle deux répliques du même objet. Dans ce cas, le premier qui a fini peut publier sans problème. Le second est forcé d'intégrer les modifications du premier avant de publier à son tour.

La figure 2 illustre le comportement du modèle avec deux utilisateurs "foo" et "bar" :

1. les utilisateurs "foo" et "bar" créent des répliques dans leurs espaces de travail respectifs ;
2. "foo" et "bar" travaillent librement sur leurs répliques ;
3. "bar" valide ses modifications en créant des nouvelles versions des objets dans le référentiel ;
4. si "foo" tente de publier ses modifications, il obtient une erreur lui signifiant qu'il doit obligatoirement consulter les dernières versions publiées.
5. "foo" recopie donc les nouvelles versions des objets validées par "bar". A cet instant, "foo" ne peut toujours pas valider ses changements. En effet, il existe des nouvelles versions des objets dont il n'a pas pris connaissance ;
6. "foo" intègre les modifications de "bar" en fusionnant ses copies locales avec les versions qu'il vient de rapatrier. Pour cela, il utilise un outil dédié à cette tâche comme Diff3 ;

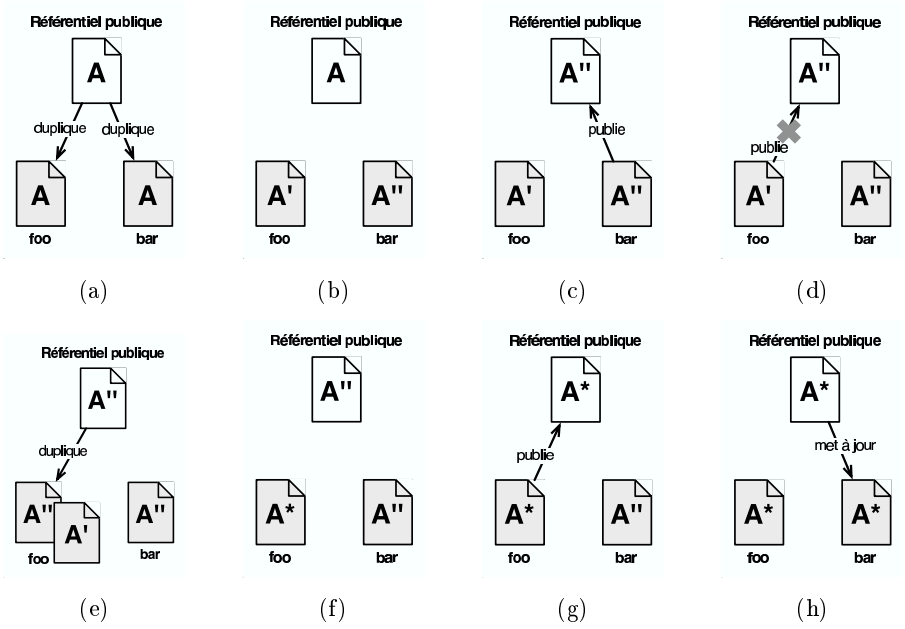


FIG. 2 – Le modèle “Copier-Modifier-Fusionner”

7. “foo” valide ses modifications, et, crée donc de nouvelles versions des objets intégrant les modifications des deux utilisateurs.
8. “bar” intègre les nouvelles modifications publiées. Les deux copies ont convergé.

Cette approche fait l'hypothèse qu'il est possible de fusionner facilement deux documents ayant divergé. Il faut donc disposer d'outils de fusion efficace. Malheureusement, ces outils sont dépendant des types de données partagées. Les outils comme CVS, Subversion, Clearcase disposent juste d'un outil de fusion de fichier texte. Il vont par exemple fusionner un fichier XML comme un fichier texte et générer un fichier XML incorrect. Ils ne sauront pas fusionner des données de type CAO, Calendrier etc...

L'approche multi-version n'est pas réservée aux seuls gestionnaires de configuration et au monde du génie logiciel. Les éditeurs WikiWikiWeb utilisent aussi cette approche.

Les wikiwikiwebs [43] ou Wikis constituent une famille d'éditeurs collaboratifs très populaires. Un wikiwikiweb est une application Web permettant à un utilisateur d'éditer une page Web depuis son navigateur internet.

Une page Wiki peut être modifiée en cliquant sur le bouton "modifier" (voir figure 3)

Une page d'édition s'ouvre alors et permet l'édition selon la syntaxe Wiki. Quand l'édition est terminée, il suffit de cliquer sur le bouton "sauvegarder" (cf figure 4). Chaque sauvegarde crée une nouvelle version de la page Wiki. Les utilisateurs voient seulement la dernière version de la page Wiki et peuvent éventuellement accéder aux versions antérieures si ils le désirent.

Deux utilisateurs peuvent modifier la même page en même temps. Deux sauvegardes seront donc effectuées et donc deux versions seront produites mais l'une après l'autre (voir fig-

[43] Wikiwikiweb. history. http ://c2.com/cgi/wiki ?WikiHistory.

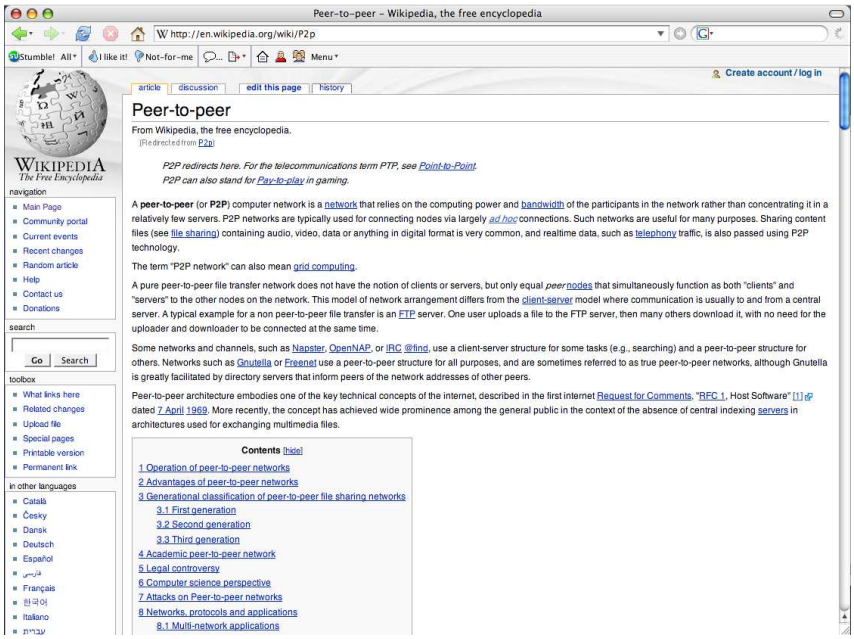


FIG. 3 – Une page Wiki

ure 5). C'est donc le dernier utilisateur qui sauve qui "gagne". Sur notre figure 5, les changements de la version $v11$ masquent les changements de la version $v12$.

Si l'utilisateur de la version $v11$ veut rendre visible ses modifications, il doit produire une nouvelle version qui contiendra les changements de la version $v11$ et de la version $v12$.

Les Wikis font donc aussi l'hypothèse qu'il existe des outils de fusion, que les utilisateur savent s'en servir et qu'ils vont eux-mêmes détecter qu'un problème de modifications concurrente est arrivé.

L'approche sérialisation/résolution de conflits Des données sont partagées entre plusieurs sites. Chaque site exécute des opérations et les stocke dans un journal. Si tous les sites exécutent les mêmes opérations dans le même ordre, alors tous les sites vont converger vers les mêmes valeurs. Le problème est donc d'arriver à un consensus sur l'ordre des opérations à exécuter.

Bayou [27,40] est un système de gestion de données pour des applications collaboratives dans un environnement mobile. Les données sont répliquées entre différents serveurs.

Dès le moment où un serveur reçoit une opération, il tente de l'exécuter. Deux sites peuvent donc recevoir et exécuter les mêmes opérations dans un ordre différent. Pour assurer la convergence, les serveurs doivent exécuter toutes les opérations dans le même ordre. Dans

[27] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the sixteenth ACM symposium on Operating systems principles - SOSP'97*, pages 288–301. ACM Press, 1997.

[40] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles - SOSP'95*, pages 172–182. ACM Press, 1995.

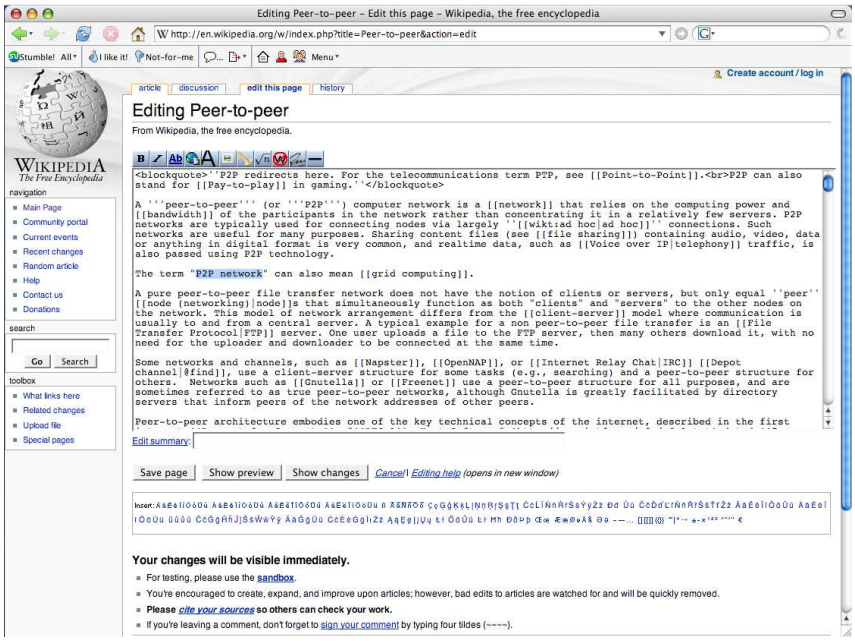


FIG. 4 – Éditer une page Wiki

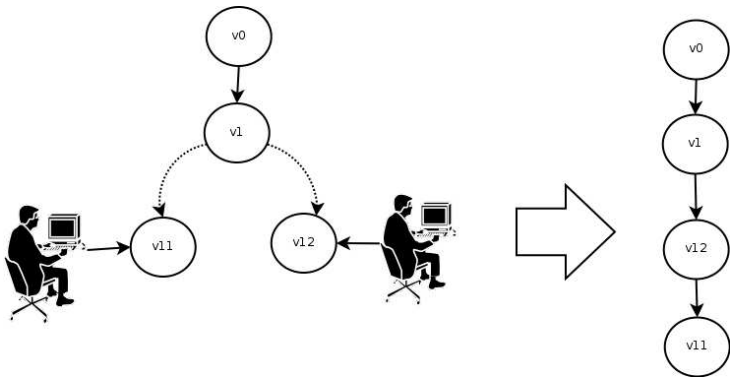


FIG. 5 – Édition concurrente d'une page Wiki

Bayou, les serveurs vont continuellement défaire et rejouer les opérations au fur et à mesure qu'ils prennent connaissance de l'ordre final. Cet ordre final est décidé par un serveur principal désigné au lancement du système.

Ainsi, chaque serveur maintient un journal des opérations exécutées. Ce journal est scindé en deux parties. Un préfixe p qui contient les opérations validées par le serveur principal. Ces opérations sont ordonnées définitivement selon un ordre total validé par le serveur principal. Le reste du journal, qualifié de "provisoire", est ordonné selon un ordre total qui peut être remis en cause au fur et à mesure que de nouvelles opérations sont reçues.

```
1  Bayou_Write = {
     update = {insert, Meeting, 12/18/95, 1:30pm, 60mn, "Budget Meeting"},
3
     dependency_check = {
5      query = "SELECT key FROM Meetings WHERE day = 12/18/95
                  AND start < 2:30pm AND end > 1:30pm",
7      expected_result = EMPTY },

9    mergeproc = {
       alternates = {{12/18/95, 3:00pm}, {12/19/95, 9:30am}};
11     newupdate = {};
       FOREACH a IN alternates {
13       #check if there would be a conflict
         IF (NOT EMPTY(
15             SELECT key FROM Meetings WHERE day = a.date
               AND start < a.time + 60mn AND end > a.time))
17         CONTINUE;
         #no conflict, can shedule meeting at that time
19       newupdate = {insert, Meetings, a.date, a.time, 60mn, "Budget Meeting"};
         BREAK;
21     }
       IF (newupdate = {}) # no alternate is acceptable
23       newupdate = {insert, ErrorLog, 12/18/95, 1:30pm, 60mn, "Budget Meeting"};
       RETURN newupdate;
25   }
   }
```

FIG. 6 – Une opération d'écriture dans Bayou.

La figure 6 présente une opération d'écriture dans Bayou. Cette opération comprend la mise à jour à effectuer (`update`), la pré-condition de l'opération (`dependency check`) qui détermine si l'opération peut être exécutée sur l'état actuel, et la procédure de réconciliation (`mergeproc`). Quand cette opération s'exécute, soit la pré-condition est vraie et la mise à jour est effectuée telle quelle. Soit le pré-condition est fausse, et dans ce cas, la procédure de fusion est exécutée.

Dans cet exemple, cette opération réserve un créneau pour une réunion. Si la pré-condition détermine que ce n'est pas possible, c'est-à-dire qu'il existe d'autres réunions sur ce créneau,

alors la procédure de réconciliation déplace la réunion à une autre date. La pré-condition de cette opération est : "Il n'existe pas d'autres réunions à cette date". Si c'est faux, la procédure de réconciliation déplace la date de réunion à d'autres dates fournies par l'utilisateur. Si aucun créneau libre ne peut être trouvé, alors l'opération est consignée dans un journal dédié. Cette opération sera traitée ultérieurement par un administrateur.

L'approche résolution de contraintes IceCube [20,28,31] est un système de réconciliation générique. Ce système traite la réconciliation comme un problème d'optimisation : celui d'exécuter une combinaison optimale de mises à jour concurrentes.

À partir des différents journaux présents sur les différents sites, IceCube calcule un journal optimal unique contenant le nombre maximum de mises à jour non conflictuelles. Pour cela, IceCube utilise la sémantique, des mises à jour, exprimée sous forme de contraintes.

Dans IceCube, les mises à jour sont modélisées par des actions. Une action réifie une opération et est composée des éléments suivants :

une ou plusieurs cibles : elles identifient les objets accédés par l'action ;

une pré-condition : elle permet de détecter les conflits éventuels lors de l'exécution au cours de la phase de simulation. Les pré-conditions sont exprimées dans un langage de logique du premier ordre ;

une opération : un sous programme accédant aux objets partagés ;

des données privées : une liste de données propres à l'action. Il y a au moins les paramètres et le type de l'opération.

Le système permet de définir des dépendances sémantiques entre les actions sous forme de contraintes. Deux types de contraintes sont disponibles : les contraintes statiques et les contraintes dynamiques. Les contraintes statiques sont évaluées sans utiliser l'état des objets. Les contraintes dynamiques peuvent utiliser l'état des objets.

La réconciliation se déroule en trois phases illustrées par la figure 7 :

une phase d'ordonnancement : durant cette phase, le système construit toutes les exécutions possibles en combinant les différentes actions issues des journaux des différents sites. L'espace de recherche est limité par les contraintes statiques.

une phase de simulation : on exécute les différents ordonnancements trouvés dans la phase précédente. Si une contrainte dynamique est violée alors l'ordonnancement en question est abandonné et rejeté. Après cette phase, il ne reste plus que les ordonnancements respectant les contraintes statiques et dynamiques.

---

[20] Anne-Marie Kermarrec, Antony I. T. Rowstron, Marc Shapiro, and Peter Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing - PODC'01*, pages 210–218. ACM Press, 2001.

[28] Nuno M. Preguiça, Marc Shapiro, and Caroline Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *On The Move to Meaningful Internet Systems 2003 : CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003*, volume 2888 of *Lecture Notes in Computer Science*, pages 38–55. Springer, November 2003.

[31] Marc Shapiro, Nuno M. Preguiça, and James O'Brien. Rufis : Mobile data sharing using a generic constraint-oriented reconciler. In *Proceedings of 5th IEEE International Conference on Mobile Data Management - MDM'04*, pages 146–151. IEEE Computer Society, January 2004.

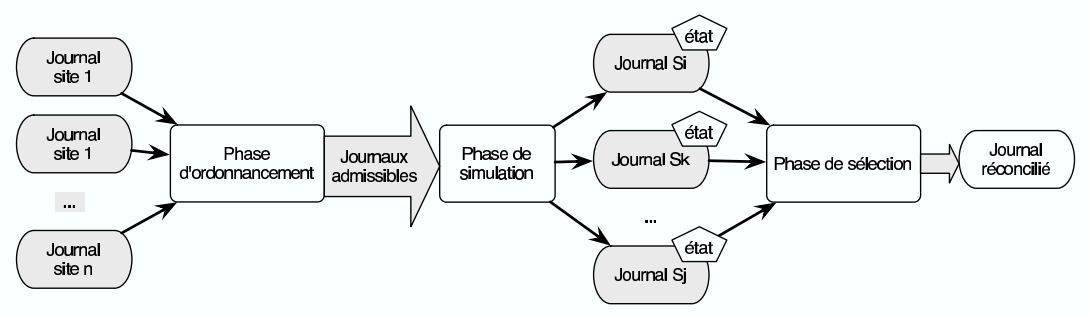


FIG. 7 – Processus de réconciliation dans IceCube.

une phase de sélection : la phase de sélection classe les ordonnancements restants et doit en choisir un. Le critère de sélection est choisi par un administrateur.

Cette description est une vision simplifiée du fonctionnement de IceCube. Dans IceCube, les trois phases décrites précédemment ne sont pas exécutées de manière séquentielle mais elles sont menées en parallèles.

Les contraintes statiques limitent la taille de l'espace de recherche, mais celui-ci reste trop grand pour effectuer une recherche exhaustive. C'est pourquoi, IceCube utilise une heuristique dont les solutions sont proches de l'optimum.

L'approche transformées opérationnelles Le modèle des transformées opérationnelles a été développé par la communauté des éditeurs collaboratifs synchrones. L'approche permet à n'importe quel utilisateur de modifier ses données à tout moment. Le grand intérêt de cette approche est qu'elle est indépendante du type des données partagées.

Le modèle considère n site. Chaque utilisateur peut modifier librement sa copie en exécutant des opérations i.e. insert, delete ... Cette opération est exécutée immédiatement localement.

Chaque opération est ensuite propagée aux autres sites pour y être ré-exécutée. Le problème principal est qu'entre le moment ou l'opération est exécutée localement et qu'elle doit être ré-exécutée sur les sites distants, l'état des sites distants peut avoir changé. Les sites distants doivent alors transformer les opérations qu'ils reçoivent pour qu'elles soient exécutables sur l'état courant. C'est pour cette raison que l'on parle d'approche transformationnelle.

L'architecture générale du modèle des transformées opérationnelles distingue deux composants :

- un algorithme d'intégration. Il est responsable de la réception, de la diffusion et de l'exécution des opérations. Si nécessaire, il fait appel aux fonctions de transformation. Cet algorithme est indépendant du type des données manipulées (chaîne de caractères, document XML, système de fichiers).
- un ensemble de fonctions de transformation. Ces fonctions ont la charge de "fusionner" les mises à jour en sérialisant deux opérations concurrentes. Ces fonctions sont spécifiques à un type de données particulier.

L'algorithme d'intégration est défini comme correct si il assure la Causalité et la Convergence (CC) [35,39,37]. Le problème principal de l'approche transformationnelle est d'écrire

[35] Maher Suleiman, Michèle Cart, and Jean Ferrié. Serialization of concurrent operations in a distributed

des fonctions de transformation correctes. En effet, les fonctions de transformation doivent vérifier deux conditions. Jusqu'à présent, personne n'a été capable d'écrire de telles fonctions de transformations.

J'ai commencé par aborder le problème en utilisant l'approche transactionnelle. J'ai étendu la sérialisabilité classique pour supporter des exécutions non-sérialisables. J'ai ensuite utilisé l'approche des transformées opérationnelles en cherchant la manière d'écrire des fonctions de transformation correctes.

---

collaborative environment. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work : The Integration Challenge - GROUP'97*, pages 435–445. ACM Press, November 1997.

[39] Chengzheng Sun, Yanchun Zhang, Xiahua Jia, and Yun Yang. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work : The Integration Challenge - GROUP'97*, pages 425–434. ACM Press, November 1997.

[37] Chengzheng Sun and Clarence A. Ellis. Operational transformation in real-time group editors : Issues, algorithms, and achievements. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*, pages 59–68, New York, NY, USA, November 1998. ACM Press.

Chapitre 1

Approche transactionnelle

Dans cette approche, nous modélisons un système d'édition collaborative comme des transactions coopératives travaillant en parallèle sur des données partagées.

L'objectif est de garantir la propriété suivante : *Si chaque transaction fait passer la base d'un état correct dans un autre, alors l'exécution en parallèle des transactions n'introduit pas d'incohérence.*

Cette propriété est la propriété classique de cohérence des modèles transactionnels. Par contre les propriétés d'atomicité et d'isolation ne sont pas garanties. La propriété d'isolation est clairement incompatible avec l'édition collaborative. Elle interdit à une transaction de communiquer avec une autre transaction au cours de son exécution. Elle interdit donc la collaboration.

La propriété d'atomicité n'est pas définie dans le cadre de l'édition collaborative. En effet, dans un système transactionnel classique, une transaction est définie comme une séquence d'opérations connues au démarrage de la transaction. La propriété d'atomicité interdit toute exécution partielle d'une transaction. Dans le cadre de l'édition collaborative, la séquence d'opération d'une transaction est dynamique. Elle n'est donc pas fixée au démarrage de transaction. La notion même d'exécution partielle d'une transaction n'est donc pas définie.

Enfin la propriété de durabilité doit aussi être garantie dans le cadre de l'édition collaborative. Les résultats d'une transaction validée sont durables.

La propriété de cohérence des transactions classiques repose sur le critère de sérialisabilité. Une exécution est sérialisable si l'exécution en parallèle de 2 transactions est équivalente à une exécution en série de ces deux transactions. Deux exécutions sont équivalente si les dans les 2 exécutions les transactions réalisent les mêmes lectures et les mêmes écritures.

Clairement les scenarii d'édition collaborative ne sont pas sérialisable. Prenons deux activités de développement : la première est chargée de produire une librairie graphique et la seconde est chargée de produire une application en utilisant cette librairie. Si ces deux activités s'exécutent sans aucun contrôle, le produit de ces activités peut être corrompu. Prenons l'exécution ci-dessous :

La figure 1.1 illustre une exécution possible des deux activités. La flèche symbolise dans quel ordre les opérations se sont effectivement exécutées. Cette exécution est incorrecte puisque l'activité *Produire*(*app*) se termine en ayant construit son application avec une version intermédiaire de la librairie graphique.

La figure 1.2 montre une exécution correcte. En effet, l'application est développée avec la

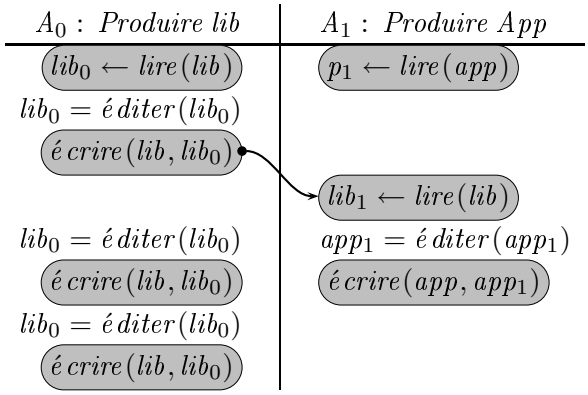


FIG. 1.1 – Exécution incorrecte

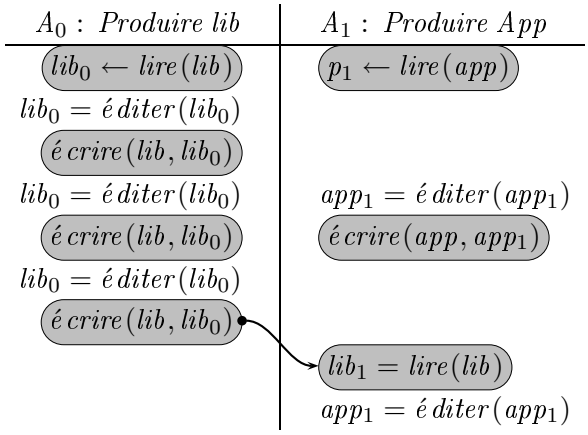


FIG. 1.2 – Exécution équivalente à un exécution en série de A_0 suivie de A_1

dernière version de la librairie produite par A_0.

En l'occurrence cette exécution est équivalente à une exécution en série de A_0 suivie de A_1. Mais cette exécution n'est pas très intéressante. En effet, dans le cadre d'un développement réel, il est plus prudent de démarrer le développement de l'application même avec une version préliminaire de la librairie graphique. Ceci permet de détecter plus tôt un éventuel problème d'intégration de la librairie graphique avec l'application.

La figure 1.3 présente une exécution elle aussi correcte. L'application a bien été produite en ayant connaissance de la dernière version de la librairie graphique. Cependant, A_1 a pu travailler un moment sur un résultat intermédiaire de A_0. Cette exécution n'est pas équivalente à une exécution en série de A_0, A_1 : c'est une exécution coopérative.

Notre approche a défini une extension de la sérialisabilité traditionnelle afin d'accepter les exécutions coopératives. Nous avons donc commencer par caractériser ces exécutions, puis nous avons défini un critère de correction, la COO-sérialisabilité et enfin nous avons écrit un protocole permettant de déterminer incrémentalement si une exécution est COO-sérialisable ou pas.

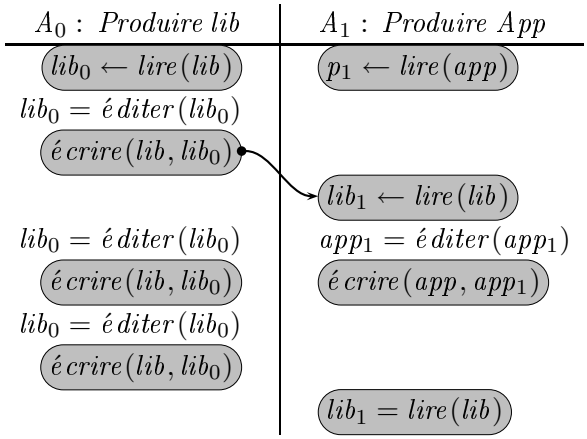


FIG. 1.3 – Exécution coopérative correcte

1.1 Résultats

Mes résultats sont les suivants :

1. deux critères de correction des exécutions coopératives : la CS-sérialisabilité et la COO-sérialisabilité. Ces critères fixent des règles à respecter pour entrelacer correctement des opérations de lecture/écriture sur des données partagées. J'ai formalisé ces critères de correction ainsi que les modèles transactionnels qui en découlent en ACTA, un environnement formel dédié à l'expression de modèles transactionnels. Ce travail a fait l'objet d'une publication majeure[4][Information Sciences].
2. Un protocole permettant d'évaluer de manière incrémentale ces critères [18][ATMA'96] et [38][SCM7].
3. Le protocole COO implanté au sein de l'environnement COO[19][ICSE96] développé par l'équipe ECOO du LORIA. Cet environnement permet de travailler de manière coopérative dans le cadre d'un atelier de génie logiciel.

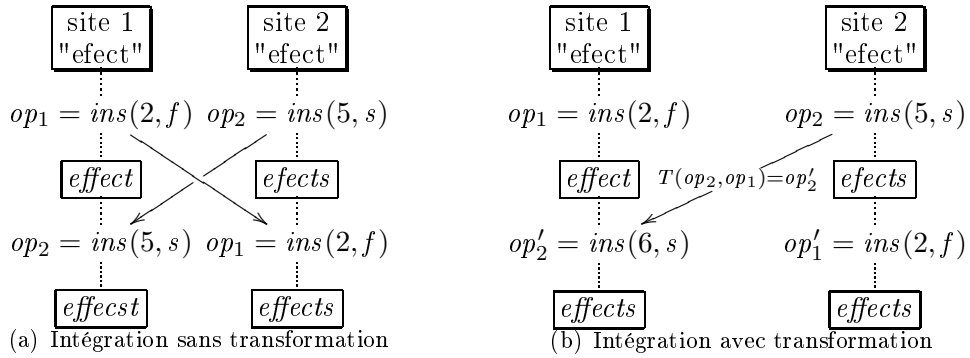
L'approche transactionnelle est intéressante mais ne traite pas le problème essentiel de la resynchronisation des données divergentes. Elle précise juste quand cette réconciliation doit avoir lieu. Pour résoudre ce problème, je me suis tourné vers l'approche transformationnelle.

Chapitre 2

Approche transformationnelle

Le modèle de Ressel [29] considère n sites. Chaque site possède une copie des objets partagés. Quand un objet est modifié sur un site, l'opération correspondante est exécutée immédiatement sur ce site, puis diffusée aux autres sites pour y être également exécutée. Autrement dit, une opération est traitée en quatre étapes :

1. génération sur un site ;
2. diffusion aux autres sites ;
3. réception par les autres sites ;
4. exécution sur ces autres sites.



(a) Intégration sans transformation (b) Intégration avec transformation

FIG. 2.1 – Intégration de 2 opérations concurrentes

Lorsqu'une opération *op* est reçue, son contexte d'exécution peut être différent de celui sur lequel elle a été générée. Dans ce cas, l'intégration de *op* sur les autres sites peut conduire à des incohérences entre les différentes copies. La figure 2.1(a) illustre un tel cas. Soient deux sites $site_1$ et $site_2$ partageant un objet de type *chainedecaracteres*. Un objet de ce type peut être modifié par une opération $Ins(p, c)$ qui a pour effet d'insérer le caractère c à la position p dans la chaîne (0 étant la position du premier caractère). Les utilisateurs $user_1$ et $user_2$ génèrent deux opérations

[29] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'96*, pages 288–297, Boston, Massachusetts, USA, November 1996.

concurrentes $op_1 = Ins(2, f)$ et $op_2 = Ins(5, s)$. Quand op_1 est reçue et exécutée sur $site_2$, elle produit l'état attendu `"effects"`. Mais, lorsque op_2 est reçue sur $site_1$, comme celle-ci ne tient pas compte de l'exécution de op_1 avant elle, l'état obtenu est `"effecst"`. Les deux copies n'ont pas convergé.

Dans le modèle des transformées opérationnelles, les opérations reçues doivent être transformées par rapport aux opérations locales concurrentes avant d'être exécutées. Cette transformation est effectuée en utilisant des fonctions de transformation. Cette fonction notée T prend en paramètre deux opérations concurrentes op_1 et op_2 définies *sur un même état* s de l'objet partagé et calcule une opération op'_1. Cette opération est équivalente à op_1 en terme d'effet, mais elle est définie sur l'état s' résultant de l'exécution de op_2 sur l'état s. La figure 2.1(b) illustre l'effet d'une telle transformation. Ainsi, dans l'exemple précédent, lorsque op_2 est reçue par $site_1$, elle doit être transformée par rapport à op_1. Dans ce cas, la transformation se fait de la manière suivante :

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

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

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

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

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

```
T(Ins(p1,c1),Ins(p2,c2)):−
  if (p1 < p2) then
    return Ins(p1, c1)
  else
    return Ins(p1 + 1, c1)
  endif
```

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

Une description plus théorique du modèle est présente dans [38,34,36]. Il a été montré que pour être correct, un algorithme d'intégration doit assurer deux propriétés générales :

Convergence Lorsque le système est *stable*, c'est-à-dire que toutes les opérations ont été diffusées et intégrées, toutes les copies doivent être identiques. Nous détaillons cette propriété dans la section 2.0.3.

Causalité Si sur un site, une opération op_2 a été exécutée après op_1, alors op_2 doit être exécutée après op_1 sur tous les sites. Nous détaillons cette propriété dans la section 2.0.1.

Plusieurs algorithmes d'intégration ont été proposés tels que SOCT 2,3,4 [34,42], dOPT [14], adOPTed [29], GOTO [37]. Il a été prouvé que leur correction [38,34] repose uniquement sur les fonctions de transformation qui doivent satisfaire les deux conditions suivantes :

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

$$C_1 : S \circ op_1 \circ T(op_2, op_1) = S \circ op_2 \circ T(op_1, op_2)$$

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

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

2.0.1 Respect de la causalité

Pour certaines opérations, il existe une relation de précédence causale qui doit être maintenue. On dit que l'opération op_1 *précède* l'opération op_2 (noté $op_1 \rightarrow op_2$) si et seulement si op_2 a été

[38] 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, March 1998.

[34] 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, Orlando, Florida, USA, February 1998. IEEE Computer Society.

[36] Chengzheng Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(4) :309–361, December 2002.

[42] Nicolas Vidot, Michèle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM conference on Computer supported cooperative work - CSCW'00*, pages 171–180, New York, NY, USA, 2000. ACM Press.

[14] Clarence A. Ellis and Simon J. Gibbs. Concurrency control in groupware systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

[29] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'96*, pages 288–297, Boston, Massachusetts, USA, November 1996.

[37] Chengzheng Sun and Clarence A. Ellis. Operational transformation in real-time group editors : Issues, algorithms, and achievements. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*, pages 59–68, New York, NY, USA, November 1998. ACM Press.

générée et exécutée sur une réplique après l'exécution de op_1 sur cette même réplique. Puisque op_2 a été générée après op_1, elle tient compte implicitement de ces effets, c'est-à-dire que l'on suppose que cette opération op_2 est dépendante des effets produits par l'opération op_1.

Le respect de la causalité garantit que pour les opérations pour lesquelles il existe une relation de causalité, celles-ci seront exécutées dans le même ordre sur toutes les répliques.

Par exemple, nous considérons le scénario présenté à la figure 2.2.

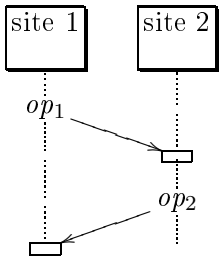


FIG. 2.2 – Exemple de relation de précédence.

Puisque l'opération op_2 a été générée sur le site 2 après que l'opération op_1 se soit exécutée, op_1 précède op_2. Si sur un troisième site, l'opération op_2 est reçue avant l'opération op_1 alors pour garantir l'exécution des opérations selon l'ordre de précédence, l'exécution de op_2 sera différée jusqu'à la réception et l'exécution de op_1.

Généralement, cette dépendance est maintenue en utilisant des vecteurs d'horloges [23,15]. Dans un système à n sites, un tel vecteur V possède n composantes. Chaque composante $V[i]$ compte le nombre d'opérations issues du site i qui ont été déjà exécutées sur le site. Lorsqu'une opération op est générée sur un site i, la composante $V[i]$ est incrémentée de 1. Une copie V_{op}, valeur de V après la génération de op, est alors associée à l'opération avant sa diffusion aux autres sites. Lors de la réception de cette opération sur un site j, si le vecteur V_{s_j} du site "domine" le vecteur V_{op} de l'opération, alors l'opération est prête à être exécutée. Dans le cas contraire, son exécution doit être différée. On dit qu'un vecteur V_1 domine un vecteur V_2 si et seulement si on a $\forall i V_1[i] \geq V_2[i]$. Lors de l'exécution sur un site j d'une opération prête provenant d'un site i, pour maintenir le vecteur V_{s_j} d'horloges du site j, il faut incrémenter sa $i$ème composante de 1.

Deux opérations op_1 et op_2 qui ne sont pas liées par une relation de précédence (ni $op_1 \rightarrow op_2$, ni $op_2 \rightarrow op_1$) sont concurrentes (noté $op_1 \parallel op_2$).

Plus précisément, pour deux opérations quelconques op_1 et op_2, issues respectivement du site S_{op_1} et S_{op_2}, munies de leur vecteur d'horloges respectif V_{op_1} et V_{op_2}, on peut déduire :

- $op_1 \rightarrow op_2$ si et seulement si $V_{op_2}[S_{op_1}] > V_{op_1}[S_{op_1}]$.
- $op_1 \parallel op_2$ si et seulement si $V_{op_2}[S_{op_1}] \leq V_{op_1}[S_{op_1}] \wedge V_{op_2}[S_{op_2}] \geq V_{op_1}[S_{op_2}]$.

---

[23] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Château de Bonas, France, October 1989. Elsevier Science Publishers B. V.

[15] Colin Fidge. Logical time in distributed computing systems. *Computer*, 24(8) :28–33, August 1991.

2.0.2 Préservation de l'intention

La préservation de l'intention ne fait pas partie du modèle de Ressel, mais est introduite en 98 par Sun [38] pour compléter le modèle. Sun définit l'intention comme suit :

Définition 2.1 (Intention) L'intention d'une opération *op* est *l'effet observé* lors de son exécution sur son état de génération.

Supposons un entier initialisé à 3. Une opération fait passer l'entier à 7. L'effet observé dépend du point de vue de l'observateur. On peut observer qu'il y a un changement d'état de 3 vers 7 ou on peut observer une incrémentation de 4 de la valeur de l'entier. Ces deux observations sont différentes. C 'est donc lors de la définition des opérations qu'il faut définir leurs effets.

Considérons maintenant une chaîne de caractères contenant "AB". L'exécution d'une opération transforme la chaîne initiale en "AXB". On peut observer une insertion d'un caractère X à la position 2 $ins(2, X)$, ou l'insertion d'un caractère X entre A et B ($ins(A \prec X \prec B)$), ou l'insertion d'un caractère 'X' après 'A' ($ins(A \prec X)$) ou encore l'insertion d'un caractère 'X' avant 'B' ($ins(X \prec B)$).

Cette définition oblige donc le concepteur d'éditeur collaboratif à définir le type des objets partagés, ses opérations, et pour chaque opération ses intentions.

Sun définit ensuite la préservation de l'intention de la manière suivante :

Définition 2.2 (Préservation de l'intention)

1. Pour toute opération *op*, les effets de l'exécution de *op* sur tous les sites doivent être les mêmes que l'intention de *op*.
2. L'effet d'exécuter une opération *op* ne doit pas changer les effets des opérations concurrentes.

En fonction de la définition des intentions des opérations, la préservation de l'intention est définie ou non. Bien évidemment, si elle ne l'est pas, il n'est pas possible de construire un éditeur collaboratif qui la préserve.

Par exemple, si on prend un objet répliqué de type entier. Cet objet dispose d'une opération d'affectation $=$ avec pour intention de mettre à jour la valeur de l'entier. Il n'est pas possible de préserver l'intention dans ce cas. En effet, si un utilisateur génère une opération $x = 3$ pendant qu'un autre génère en parallèle $x = 7$, l'effet d'exécuter $x = 3$ change l'effet d'exécuter $x = 7$. Cela est en contradiction avec le 2ième point de la définition. Dans ce cas, la notion même de préservation d'intention n'est pas définie pour ce type d'objet avec ce type d'opération.

Prenons un objet de type entier avec une opération "incrémenter" $inc(x)$ qui incrémente l'entier x de 1. Si on définit l'intention comme : "l'entier est incrémenté de 1", alors la notion de préservation de l'intention n'est pas définie. Par contre, si on définit l'intention comme : "la valeur est croissante", alors la notion de préservation de l'intention est définie même si elle est sans grand intérêt.

---

[38] 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, March 1998.

Prenons maintenant un objet de type chaîne de caractères avec une opération d'insertion d'un caractère c à une position p : $ins(p, c)$. Si on définit l'intention de *ins* comme insérer le caractère c exactement à la position p alors la préservation de l'intention n'est pas définie. Si on définit l'intention de *ins* comme l'insertion entre le caractère précédent et le caractère suivant, alors la notion d'intention est définie et donc peut être préservée.

En effet, prenons une chaîne de caractères avec comme valeur initiale "AB". Un utilisateur insère 'X' entre 'A' et 'B' ; $op_1 = ins(1, X)$ et l'intention de op_1 est de faire respecter la contrainte $A \prec X \prec B$, autrement dit 'X' est placé derrière 'A' et devant 'B'. Un autre utilisateur insère en parallèle 'Y' entre 'A' et 'B' ; $op_2 = ins(1, Y)$ et l'intention de op_2 est de préserver $A \prec Y \prec B$.

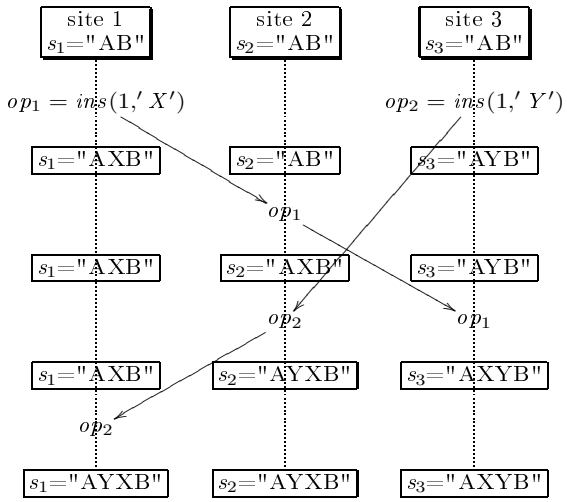


FIG. 2.3 – Préservation de l'intention

La figure 2.3 illustre la propagation de ces deux opérations sur les trois sites. Dès qu'une opération arrive sur un site, elle est exécutée telle quelle sans condition.

Les états finaux "AXYB" et "AYXB" préservent les intentions de op_1 et op_2. En effet, les deux contraintes $A \prec X \prec B$ et $A \prec Y \prec B$ sont bien respectées. Enfin, l'effet de op_1 n'altère pas l'effet de op_2 et réciproquement. Malheureusement, les répliques ne convergent pas.

Il est donc tout à fait possible de préserver les intentions et de ne pas converger et inversement de converger sans respecter les intentions

2.0.3 Convergence des répliques

Le fait de respecter les relations de causalité entre les opérations et de préserver les effets des opérations ne suffit pas pour garantir la convergence des répliques. La figure 2.4 donne un exemple de scénario où les deux critères sont respectés mais où les copies ne convergent pas.

Dans cet exemple, nous considérons deux répliques d'une chaîne de caractères de valeur initiale `ab`. Sur la première réplique, un utilisateur insère le caractère `x` entre les caractères `a` et `b`. En parallèle, sur la seconde réplique, un autre utilisateur insère au même emplacement le caractère `y`.

Pour intégrer les opérations sur les différentes répliques, nous utilisons la fonction de transformation définie à la figure 2.5. Sur la première réplique, l'intégration de l'opération op_2 revient à décaler la position d'insertion d'une position. Nous obtenons alors la chaîne `axyb`. L'effet d'op_2

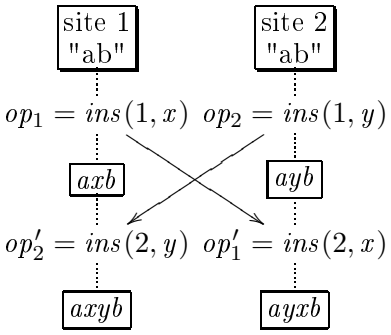


FIG. 2.4 – Divergence des copies due à la violation de C_1

```
T(Ins(p1,c1),Ins(p2,c2)):−
2  if (p1 < p2) then
     return Ins(p1, c1)
4  else
     return Ins(p1 + 1, c1)
6  endif
```

FIG. 2.5 – Fonction de transformation naïves

a été préservé puisque le caractère `y` est bien inséré entre les caractères `a` et `b`. En procédant de la même manière sur la seconde réplique, après avoir intégré l'opération op_1, nous obtenons l'état `ayxb`. L'effet de l'opération op_1 a également été préservé. Cependant, bien que les effets des deux opérations soient préservés, les deux répliques n'ont pas convergé.

Il a été montré [14] que pour garantir la convergence des copies, les fonctions de transformation doivent vérifier la condition suivante.

DÉFINITION 2.3 (CONDITION C_1) Soient op_1 et op_2 deux opérations concurrentes définies sur le même état. La transposée en avant T vérifie la condition C_1 si et seulement si :

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

où $\circ$ dénote un opérateur concaténant deux opérations pour former une séquence d'opérations, et où $\equiv$ signifie que les deux séquences $op_1 \circ T(op_2, op_1)$ et $op_2 \circ T(op_1, op_2)$ produisent le même état.

Dans notre exemple, les opérations op_1 et op_2 sont concurrentes et définies sur le même état `ab`. Pourtant, l'exécution de la séquence d'opération $op_1 \circ T(op_2, op_1)$ sur cet état donne l'état `axyb` tandis que l'exécution de la séquence $op_2 \circ T(op_1, op_2)$ donne l'état `ayxb`. Les deux états ne sont pas égaux ce qui prouve la violation de la condition C_1.

Pour satisfaire cette condition, il faut que la fonction de transposition en avant prenne en compte le cas particulier de deux insertions à la même position dans la chaîne de caractères. Étant donné que rien ne nous permet de décider lequel des deux caractères doit être placé devant

[14] Clarence A. Ellis and Simon J. Gibbs. Concurrency control in groupware systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

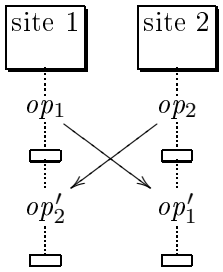


FIG. 2.6 – Condition C_1.

```
T(Ins(p1,c1),Ins(p2,c2)):−
  if (p1 < p2) then
    return Ins(p1, c1)
  else if (p1 > p2) then
    return Ins(p1 + 1, c1)
  else if (p1 = p2) then
      if (c1 < c2) then
         return Ins(p1, c1)
      else
         return Ins(p1 + 1, c1)
      endif
  endif
```

FIG. 2.7 – Fonction de transformation vérifiant la condition C_1.

l'autre, nous devons prendre un choix arbitraire. Et, nous devons garantir que ce choix sera le même sur toutes les répliques.

La figure 2.7 présente un exemple de fonction de transformation vérifiant la condition C_1. Pour "départager" les deux opérations, lors de l'insertion à la même position, nous comparons les caractères. L'insertion dont le caractère est le plus petit selon l'ordre lexicographique s'effectue devant l'autre caractère.

La figure 2.8 illustre l'utilisation de cette nouvelle fonction de transformation sur le scénario précédent. Sur la première réplique, lors de l'intégration de op_2, l'opération voit sa position d'insertion incrémenter puisque le caractère `y` est "plus grand" selon l'ordre lexicographique que le caractère `x` inséré par op_1. Sur la seconde réplique, lors de l'intégration de op_1, le même choix est réalisé – placer `x` devant `y` – l'opération n'est donc pas modifiée. Au final, les deux répliques convergent vers le même état `axyb`.

La condition C_1 n'est pas suffisante pour garantir la convergence des copies en présence de plus de deux opérations concurrentes. La figure 2.9 illustre une telle situation. Dans ce scénario nous utilisons la fonction de transformation satisfaisant la condition C_1 que nous avons présentée précédemment. Nous utilisons également la fonction de transformation donnée par la figure 2.10.

Le respect de la condition C_1 nous garantit la convergence des copies après l'intégration de

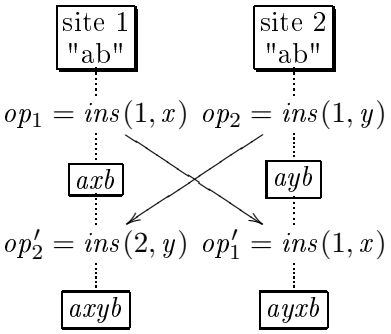


FIG. 2.8 – Respect de la condition C_1

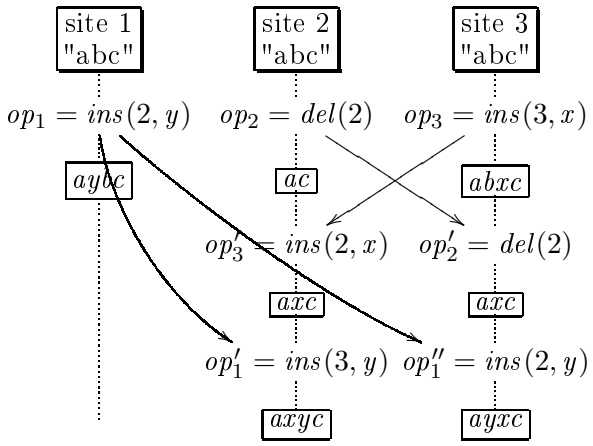


FIG. 2.9 – Insuffisance de la condition C_1.

```
T(Ins(p1,c1),Del(p2)):−
  if (p1 ≤ p2) then
    return Ins(p1, c1)
  else
    return Ins(p1 − 1, c1)
  endif
```

FIG. 2.10 – Une autre fonction vérifiant la condition C_1.

op_2 sur le site 2 et de op_3 sur le site 3. Cependant, un problème survient lors de l'intégration de op_1. Sur le site 2, on obtient l'opération $op'_1 = Ins(3, y)$ tandis que sur le site 3 on obtient l'opération $op''_1 = Ins(2, y)$. Ces deux opérations étant différentes lorsqu'elles sont exécutées sur le même état donnent forcément deux états différents. Les copies ne convergent donc pas.

Pour éviter cette situation un certain nombre d'algorithmes d'intégration, tels que GOT [], SOCT3, SOCT4 [42], SOCT5 [41], imposent un ordre de sérialisation unique. Ainsi dans notre exemple, sur le site 3, op_2 est toujours transformée par rapport à op_3 et sur le site 2 op_3 est toujours transformée par rapport à op_2. Par contre, l'intégration sur ces deux sites de l'opération op_1 est réalisée en transformant par rapport à la même séquence $[op_2; T(op_3, op_2)] = [op_2; op'_3]$ et non plus par rapport à deux séquences équivalentes. Au final, l'équivalence entre les deux séquences exécutées sur les sites 2 et 3 est bien assurée. En effet, sur ces deux sites nous avons exécuté, tout d'abord, une des deux séquences équivalentes $[op_2; op'_3]$ ou $[op_3; op'_2]$, puis la même opération op'_1 ; cette opération résultant de la transformation de op_1 par rapport à la séquence $[op_2; op'_3]$.

Malheureusement, la construction d'un ordre de sérialisation unique dans un contexte réparti est coûteux et passe difficilement à l'échelle. C'est pourquoi, d'autres algorithmes d'intégration, tels que adOPTed [29], GOTO [37] SOCT2 [34], ne requièrent pas d'ordre total, mais en contrepartie imposent aux fonctions de transformation de satisfaire une condition supplémentaire [29], la condition C_2.

Définition 2.4 (Condition C_2) Soient trois opérations op_1, op_2 et op_3 concurrentes définies sur le même état, la fonction de transposition en avant T vérifie la condition C_2 si et seulement si :

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

On notera que par définition, l'expression $T(op_x, op_y \circ op_z)$ est égale à l'expression $T(T(op_x, op_y), op_z)$. Ainsi, l'égalité de la condition C_2 peut également s'écrire :

$$T(T(op_3, op_1), T(op_2, op_1)) = T(T(op_3, op_2), T(op_1, op_2))$$

Contrairement à la condition C_1, il s'agit bien ici d'une égalité sur les opérations obtenues après transformation.

---

[42] Nicolas Vidot, Michèle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM conference on Computer supported cooperative work - CSCW'00*, pages 171–180, New York, NY, USA, 2000. ACM Press.

[41] Nicolas Vidot. Convergence des copies dans un environnements collaboratifs répartis. In *Thèse de Doctorat de l'Université de Montpellier II*, 2002.

[29] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'96*, pages 288–297, Boston, Massachusetts, USA, November 1996.

[37] Chengzheng Sun and Clarence A. Ellis. Operational transformation in real-time group editors : Issues, algorithms, and achievements. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*, pages 59–68, New York, NY, USA, November 1998. ACM Press.

[34] 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, Orlando, Florida, USA, February 1998. IEEE Computer Society.

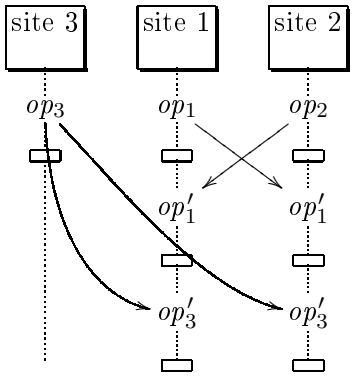


FIG. 2.11 – Condition C_2.

En généralisant cette dernière condition, on peut montrer [29] que si les fonctions de transformation vérifient la condition C_2 alors le résultat de la transformation d'une opération par rapport à une séquence d'opérations concurrentes ne dépend pas de l'ordre selon lequel les opérations de la séquence ont été transformées.

Enfin, il est possible de démontrer [35,33] que les conditions C_1 et C_2 sont deux conditions suffisantes pour assurer la convergence des copies. Autrement dit, si l'on considère n opérations $op_1, \ldots, op_n$ concurrentes et un ensemble de fonctions de transformation vérifiant les conditions C_1 et C_2, quel que soit l'ordre dans lequel on transforme ces opérations, les séquences obtenues sont équivalentes, c'est-à-dire qu'exécutées à partir du même état, elles produisent le même état.

2.1 Résultats

2.1.1 Environnement de développement Multi-Synchrone

Nous avons développé un concept original d'environnement SAMS pour Synchrone, Asynchrone et Multi-Synchrone. Un tel environnement permet d'adapter la façon de travailler du groupe aux conditions de travail.

Le travail synchrone semble plus adapté pour les phases de résolutions de conflits, le travail asynchrone pour les phases d'intégration et le travail multi-synchrone pour les phases de production. Dans cet environnement, il est possible à tout moment, *pour tout ou partie de groupe* de travailler dans le mode de son choix tout en garantissant les propriétés de convergence, causalité et intention.

Pour ce faire, nous avons étendu l'approche transformationnelle pour y intégrer le travail déconnecté [42]. Nous avons ensuite construit un premier éditeur SAMS sur données partagées XML [25]. Ce prototype peut-être testé en ligne à l'adresse suivante : `http://woinville.loria.fr/sams`.

---

[35] 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 : The Integration Challenge - GROUP'97*, pages 435–445. ACM Press, November 1997.

[33] Maher Suleiman. Sérialisation des opérations concurrentes dans les systèmes collaboratifs répartis. In *Thèse de Doctorat de l'Université de Montpellier II*, 1998.

2.1.2 Synchroniseurs sûrs et génériques

SO6 est un gestionnaire de configuration reposant sur le modèle des transformées opérationnelles. Cet outil est actuellement diffusé sous forme de logiciel libre et également intégré dans une offre commerciale.

SO6 montre qu'il est possible d'adapter les algorithmes OT, développés pour des éditeurs collaboratifs temps-réels, afin de réaliser un outil comparable à CVS [11] ou Subversion. SO6 met en avant l'aspect générique de l'approche OT. En séparant l'algorithme d'intégration des fonctions de transformation, il est possible de construire de façon simple un gestionnaire de configuration. En démontrant que les fonctions de transformation vérifie la propriété dite C_1, on obtient immédiatement un outil assurant la convergence des copies dans tous les cas. Cet outil reste extensible à de nouveaux types de données répliquées en ajoutant les fonctions de transformation correspondantes.

SO6 est basé sur l'algorithme SOCT4 [42]. Cet algorithme utilise un estampilleur pour ordonner totalement les opérations et requiert ainsi uniquement la vérification de la condition C_1. Les opérations sont estampillées selon un ordre total. Il se démarque des autres algorithmes d'intégration OT du même type en utilisant un mécanisme de diffusion différée. Pour envoyer une opération, il faut d'abord avoir reçu toutes les opérations estampillées et transformer les opérations locales en conséquence. Il est ensuite possible d'estampiller ses opérations locales et de les envoyer. Le fonctionnement de cet algorithme s'apparente au paradigme "copier-modifier-fusionner" utilisé dans CVS où un utilisateur ne peut publier ses modifications que si il a pris en compte toutes les modifications déjà publiées.

L'architecture générale du système est centralisée autour d'un estampilleur qui délivre une estampille à chaque opération. Une application cliente s'exécute sur chaque machine des utilisateurs. Les applications clientes ne s'échangent pas directement les opérations, elles doivent les faire transiter par l'estampilleur.

SO6 met en œuvre l'estampilleur sous forme d'une file d'opérations. Cette file maintient, de manière persistante, les opérations publiées par les différents sites dans l'ordre des estampilles. Elle correspond à l'histoire commune des différentes copies.

Un estampilleur est constitué par :

une file d'opérations Q où les opérations sont rangées selon l'ordre des estampilles ;

une estampille *lastTicket* qui représente la dernière estampille délivrée ;

une méthode *publish*(*op*) qui attribue une nouvelle estampille à l'opération *op* et stocke cette opération dans la file Q ;

```
int publish(Operation op) {
2   lastTicket++
    Q[lastTicket] = op
4   return lastTicket
}
```

[11] Per Cederqvist. *Version Management with CVS*. Network Theory Ltd., December 2002.

[42] Nicolas Vidot, Michèle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM conference on Computer supported cooperative work - CSCW'00*, pages 171–180, New York, NY, USA, 2000. ACM Press.

une méthode *retrieve*(*ticket*) qui retourne l'opération dont l'estampille a pour valeur *ticket* ;

```
1 Operation retrieve(int ticket) {
    return Q[ticket]
3 }
```

Nous avons développé l'estampilleur de deux manières différentes : une version utilisant un disque partagé pour une utilisation sur un réseau local et une autre version utilisant un serveur J2EE [19] pour un déploiement à plus large échelle.

Un client SO6 doit détecter les modifications locales sous forme d'opérations, intégrer les opérations concurrentes, et publier les opérations résultant des modifications locales.

Un client est composé des éléments suivants :

une estampille *siteTicket* qui représente l'estampille de la dernière opération intégrée par le client. Autrement cette estampille est égale à celle de la dernière opération publiée par le site ou à celle de la dernière opération rapatriée par le site.

deux états *currentState* **et** *referenceState* . Ils permettent de calculer la séquence des opérations réalisées par l'utilisateur. *currentState* est l'état sur lequel l'utilisateur travaille. *referenceState* est l'état résultant de l'exécution de toutes les opérations qui ont été intégrées par le client ;

une séquence d'opérations *Hg* qui stocke les opérations qui ont été intégrées par l'application client. Elle contient donc d'une part les opérations publiées par les autres utilisateurs et qui ont été intégrées, et d'autre part, les opérations produites par l'utilisateur et qu'il a publiées. Les opérations sont rangées selon l'ordre de leur estampille.

L'exécution de cette séquence d'opérations *Hg* sur un état vide calcule toujours un état égal à l'état *referenceState* ;

une méthode *computeDifference*(*state*1, *state*2) qui calcule par différenciation la séquence d'opération permettant de générer l'état *state*2 à partir de l'état *state*1. Pour réaliser ce calcul sur les documents textuels nous utilisons l'algorithme Diff [24,25]. Pour les documents XML, nous utilisons l'algorithme XyDiff [13].

une méthode *commit*() qui publie les opérations locales vers l'estampilleur ;

```
1 commit() {
    if (estampilleur.lastTicket > siteTicket) then
3     abort "***_interdiction_de_publier_sans_être_à_jour_***"
    Operation[] locals = computeDifference(referenceState, currentState)
5   for (int i=0; i<locals.length; i++) {
      int ticket=estampilleur.publish(locals[i])
7     execute(locals[i], referenceState)
```

---

[19] Java 2 Platform Enterprise Edition. http ://java.sun.com/j2ee/. June 2005.

[24] Webb Miller and Eugene W. Myers. A file comparison program. *Software—Practice and Experience*, 15(11) :1025–1040, 1985.

[25] Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1(2) :251–266, 1986.

[13] Gregory Cobena, Serge Abiteboul, and Amélie Marian. Detecting changes in xml documents. In *Proceedings of the eighteenth International Conference on Data Engineering - ICDE'02*, pages 41–52, San Jose, California, USA, February 2002. IEEE Computer Society.

```
        Hg[ticket] = locals[i]
    }
    siteTicket = estampilleur.lastTicket
}
```

Cette méthode commence par vérifier que le site a intégré toutes les opérations qui ont déjà été publiées et qui sont disponibles sur l'estampilleur. Ensuite, elle calcul la séquences des opérations locales qui ont été réalisées par l'utilisateur (et qui n'ont pas encore été publiées). Chaque opération est ensuite envoyée pour être estampillée, puis exécutée sur l'état de référence, et enfin, ajoutée à l'histoire *Hg* selon l'ordre des estampilles.

une méthode *update*() qui met à jour la réplique locale en rapatriant les opérations qui ont déjà été publiées ;

```
update() {
    Operation[] remotes
    int i=0
    while (siteTicket < estampilleur.lastTicket) {
        siteTicket++
        i++
        remotes[i] = estampilleur.retrieve(siteTicket)
    }
    Operation[] locals = computeDifference(referenceState, currentState)
    merge(remotes, locals)
}
```

Cette méthode commence par rapatrier toutes les opérations disponibles sur l'estampilleur et qui n'ont pas encore été intégrées. Elle calcule ensuite la séquence des opérations locales. Les deux séquences d'opérations obtenues, étant concurrentes, sont fusionnées.

une méthode *merge*(*Operation*[], *Operation*[]) qui "fusionne" deux séquences d'opérations concurrentes en utilisant la fonction de transformation T.

```
merge(Operation[] remotes, Operation[] locals) {
    for (int i=0; i<remotes.length; i++) {
        Operation opr = remotes[i]
        int ticket = remotes[i].ticket
        for (int j=0; j<locals.length; j++) {
            Operation opl = locals[j]
            locals[j] = T(opl, opr)
            opr = T(opr, opl)
        }
        execute(remotes[i], referenceState)
        Hg[ticket] = remotes[i]
        execute(opr, currentState)
        siteTicket = ticket
    }
}
```

Cette méthode reprend le principe d'intégration développé dans l'algorithme SOCT4. Chaque opération à intégrer, *remote*[*i*] doit être transformée par rapport à la séquence des opérations locales *locals* en une opération *opr*. Cette opération peut être exécutée sur l'état

actuel *currentState* sur lequel l'utilisateur travaille. L'opération $remote[i]$ non transformée est ajoutée à la séquence Hg. Puis cette opération est exécutée sur l'état de référence *referenceState* pour maintenir la cohérence entre cet état et l'histoire Hg.

△ △ △

Convergence SO6 assure la convergence des répliques. Celle-ci est garantie d'une part par la preuve [41] de correction de l'algorithme SOCT4, et d'autre part la preuve de la vérification de la condition C_1 par nos fonctions de transformation.

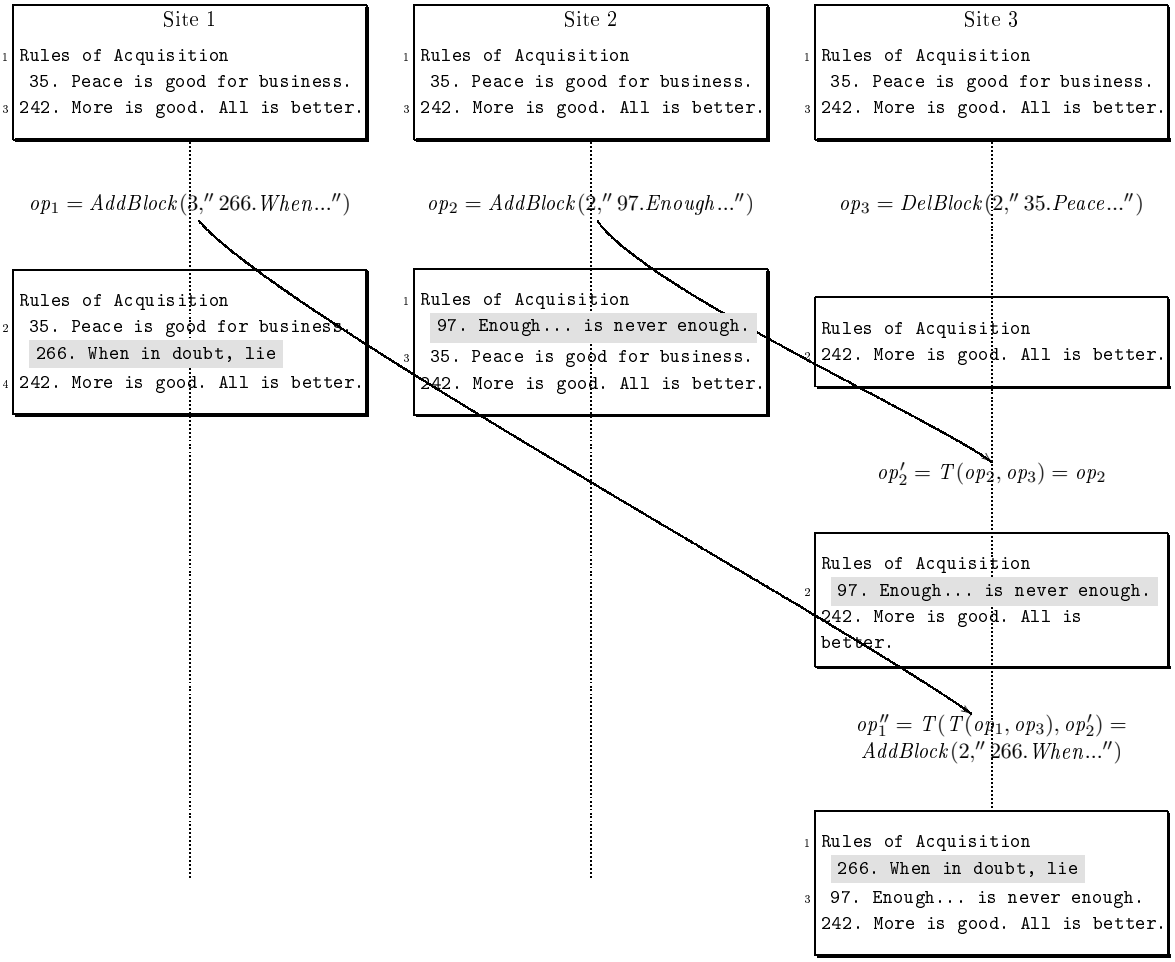


FIG. 2.12 – Violation de l'intention dans SO6

Intention Le scénario de la figure 2.12 conduit à une violation de l'intention dans SO6 (voir définition 2.2). En effet, la ligne 266 a été insérée après la ligne 35 et la ligne 97 avant la ligne 35. Logiquement, dans le résultat final, la ligne 266 doit apparaître après la ligne 97. Le

[41] Nicolas Vidot. Convergence des copies dans un environnements collaboratifs répartis. In *Thèse de Doctorat de l'Université de Montpellier II*, 2002.

scénario présenté montre comment il est possible d'arriver au résultat inverse dans SO6. Ce scénario considère un document textuel répliqué qui peut être modifié par deux opérations : $AddBlock(l, b)$ qui insère le bloc de texte b à la ligne l du document ; $DelBlock(l, b)$ qui détruit le bloc b situé à la ligne l. Dans ce scénario, il est clair que sur l'état initial, op_1 insère sa ligne après celle insérée par op_2. Pourtant, si on suppose que l'estampilleur ordonnance les opérations dans l'ordre op_3, op_2, op_1, alors l'intégration de ces opérations conduit à une violation des intentions. Par exemple, on s'intéresse à l'intégration de ces opérations sur le site 3. On intègre op_2 en calculant :

$$
\begin{aligned}
& T(op_2, op_3) \\
= \; & T(AddBlock(2,''\,97.Enough...''), DelBlock(2,''\,35.Peace...'')) \\
= \; & AddBlock(2,''\,97.Enough...'') \\
= \; & op_2'
\end{aligned}
$$

On intégre ensuite op_1 en calculant les transformées suivantes :

$$
\begin{aligned}
& T(T(op_1, op_3), op_2') \\
= \; & T(T(AddBlock(3,''\,266.When...''), DelBlock(2,''\,35.Peace...'')), op_2') \\
= \; & T(AddBlock(2,''\,266.When...''), op_2') \\
= \; & T(AddBlock(2,''\,266.When...''), AddBlock(2,''\,97.Enough...'')) \\
= \; & AddBlock(2,''\,266.When...'')
\end{aligned}
$$

si l'on suppose que $\texttt{code}(''266.When...'')$ est inférieur à $\texttt{code}(''97.Enough...'')$

Au final, l'état obtenu ne respecte pas les intentions, puisque la ligne insérée par op_1 se retrouve devant ligne insérée par op_2. La vérification de la condition C_1, nous garantit quand même que toutes les répliques convergeront vers cet état.

SO6 ne préserve donc pas les intentions.

2.1.3 Vérification des fonctions de transformation existantes

L'approche des transformées opérationnelles permet de construire des environnements de travail de groupe temps réel. Les algorithmes comme aDOPTed, GOTO, SOCT 2,3,4 font l'hypothèse que les fonctions de transformations T vérifient les conditions suivantes :

- $C1 : op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$.
  $T(op_2, op_1)$ signifie que op_2 est transformée selon op_1.
- $C2 : T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$

La preuve de ces conditions est longue et complexe. Nous coopérons avec l'équipe CASSIS à la preuve automatique de ces propriétés en utilisant le prouveur automatique SPIKE. Afin de valider notre approche, nous avons formalisé l'ensemble des proposition existantes et vérifié leur correction.

Fonctions de transformation d'Ellis

Ellis et Gibbs [14] sont les pionniers de l'approche des transformées opérationnelles. Ils ont défini les fonctions de transformation ci-dessous. Les opérations $ins()$ et $del()$ sont étendues avec

[14] Clarence A. Ellis and Simon J. Gibbs. Concurrency control in groupware systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

un nouveau paramètre pr représentant la priorité. Les priorités sont basées sur l'identificateur du site sur lesquelles les opérations sont générées. Id représente l'opération identité. L'exécution de cette opération n'affecte pas l'état.

$T(ins(p_1, c_1, pr_1), ins(p_2, c_2, pr_2)) :-$
2 **if** $(p_1 < p_2)$ **return** $ins(p_1, c_1, pr_1)$
else if $(p_1 > p_2)$ **return** $ins(p_1 + 1, c_1, pr_1)$
4 **else if** $(c_1 == c_2)$ **return** $Id()$
else if $(pr_1 > pr_2)$ **return** $ins(p_1 + 1, c_1, pr_1)$
6 **else return** $ins(p_1, c_1, pr_1)$

8 $T(ins(p_1, c_1, pr_1), del(p_2, pr_2)) :-$
if $(p_1 < p_2)$ **return** $ins(p_1, c_1, pr_1)$
10 **else return** $ins(p_1 - 1, c_1, pr_1)$

12 $T(del(p_1, pr_1), ins(p_2, c_2, pr_2)) :-$
if $(p_1 < p_2)$ **return** $del(p_1, pr_1)$
14 **else return** $del(p_1 + 1, pr_1)$

16 $T(del(p_1, pr_1), del(p_2, pr_2)) :-$
if $(p_1 < p_2)$ **return** $del(p_1, pr_1)$
18 **else if** $(p_1 > p_2)$ **return** $del(p_1 - 1, pr_1)$
else return $Id()$

FIG. 2.13 – Fonctions de transformation d'Ellis

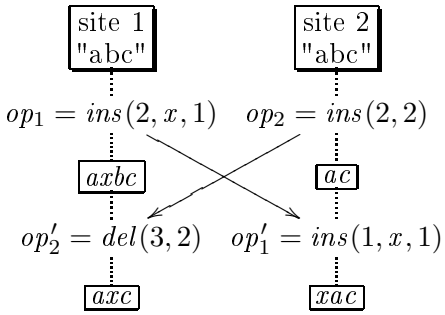


FIG. 2.14 – Contre-exemple violant la condition C_1.

Ces fonctions de transformation sont connues pour être fausses [29,38,34]. Nous les avons tout

[29] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'96*, pages 288–297, Boston, Massachusetts, USA, November 1996.

[38] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality

de même spécifiées et vérifiées avec SPIKE . SPIKE trouve le contre-exemple de la figure 2.14 en quelques secondes. Cet exemple montre une violation de la condition C_1. Si nous modifions la fonction de transformation $T(ins(), del())$ en remplaçant $p_1 < p_2$ par $p_1 \leq p_2$, la condition C_1 est alors vérifiée. Malheureusement, la condition C_2 est violée et SPIKE trouve le contre-exemple présenté dans la section suivante.

Fonctions de transformation de Ressel

Ressel modifie en 1996 les fonctions proposées par Ellis. Il modifie principalement la fonction de transformation pour le couple $(ins(), ins())$. Quand deux opérations d'insertion insèrent un caractère à la même position p, alors le caractère produit par le site de rang inférieur est inséré à la position p. L'autre caractère est inséré à la position $p + 1$. Cela suppose que tous les sites sont rangés selon un ordre total strict.

$T(ins(p_1, c_1, u_1), ins(p_2, c_2, u_2)) :-$
  if $(p_1 < p_2)$ or $(p_1 == p_2$ and $u_1 < u_2)$ **return** $ins(p_1, c_1, u_1)$
  else return $ins(p_1 + 1, c_1, u_1)$

$T(ins(p_1, c_1, u_1), del(p_2, u_2)) :-$
  if $(p_1 \leq p_2)$ **return** $ins(p_1, c_1, u_1)$
  else return $ins(p_1 - 1, c_1, u_1)$

$T(del(p_1, u_1), ins(p_2, c_2, u_2)) :-$
  if $(p_1 < p_2)$ **return** $del(p_1, u_1)$
  else return $del(p_1 + 1, u_1)$

$T(del(p_1, u_1), del(p_2, u_2)) :-$
  if $(p_1 < p_2)$ **return** $del(p_1, u_1)$
  else if $(p_1 > p_2)$ **return** $del(p_1 - 1, u_1)$
  else return $Id()$

FIG. 2.15 – Fonctions de transformation de Ressel.

Les fonctions de transformation de Ressel vérifient C_1 mais SPIKE trouve un contre-exemple violant C_2 (cf Figure 2.16). Ce scénario nécessite trois opérations concurrentes : $op_1 = ins(2, y)$, $op_2 = del(1)$ et $op_3 = ins(1, x)$.

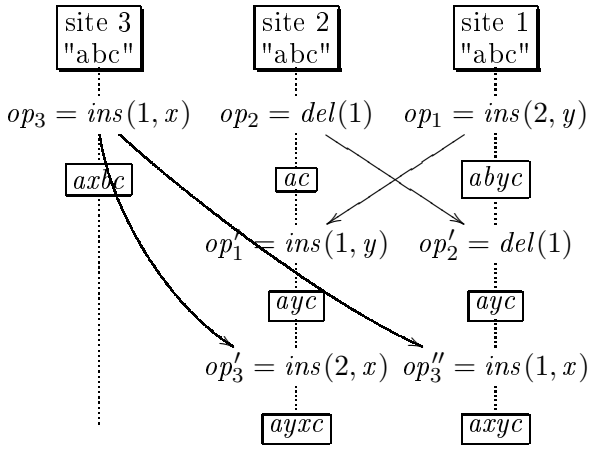


FIG. 2.16 – Contre-exemple violant C_2.

Fonction de transformation d'IMOR

Dans [26], nous proposons d'ajouter un nouveau paramètre ip_i à l'opération d'insertion. Ce paramètre représente la position initiale d'insertion avant toute transformation. Par example, si l'utilisateur insère un caractère à la position 3, ip sera égal à 3 et restera invariant. Les fonctions de transformation vont influer sur p_i mais non sur ip_i. Les fonctions de transformations sont présentées dans la figure 2.17

Dans [26], nous avons affirmé que ces fonctions vérifient C_2. Malheureusement, nous avions fait une erreur élémentaire de spécification. Nous avions défini l'opération d'insertion *ins* comme suit :

operation
$(p_1 == p_2)$ and $(p \leq length())$: $ins(nat\, p_1, nat\, p_2, char\, c)$;
$p < length()$: $del(nat\, p)$;

En spécifiant $p_1 == p_2$ comme pré-condition de *ins*, nous avons forcé le prouveur de théorème à ne pas explorer les cas où $p_1 \neq p_2$. Or, ces cas peuvent arriver comme nous l'illustrons dans la figure 2.18. Cette mésaventure démontre qui si la preuve de correction des fonctions de transformation est complexe et génératrice d'erreur, la spécification de ces fonctions l'est aussi. Nous avons tout de même persévéré en observant qu'il est tout de même bien plus facile de vérifier la spécification que la preuve. Nous avons donc ré-écrit la définition de *ins* comme suit :

operation
$p_1 \leq length()$: $ins(nat\, p_1, nat\, p_2, char\, c)$;
$p < length()$: $del(nat\, p)$;

Avec cette nouvelle spécification, SPIKE trouve un nouveau contre-exemple présenté dans la figure 2.18.

preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1) :63–108, March 1998.

[34] 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, Orlando, Florida, USA, February 1998. IEEE Computer Society.

$T(ins(p_1, ip_1, c_1), ins(p_2, ip_2, c_2)) :-$
2 **if** $(p_1 < p_2)$ **return** $ins(p_1, ip_1, c_1)$
else if $(p_1 > p_2)$ **return** $ins(p_1 + 1, ip_1, c_1)$
4 **else if** $(ip_1 < ip_2)$ **return** $ins(p_1, ip_1, c_1)$
else if $(ip_1 > ip_2)$ **return** $ins(p_1 + 1, ip_1, c_1)$
6 **else if** $(code(c_1) < code(c_2))$ **return** $ins(p_1, ip_1, c_1)$
else if $(code(c_1) > code(c_2))$ **return** $ins(p_1 + 1, ip_1, c_1)$
8 **else return** $Id()$

10 $T(ins(p_1, ip_1, c_1), del(p_2)) :-$
if $(p_1 > p_2)$ **return** $ins(p_1 - 1, ip_1, c_1)$
12 **else return** $ins(p_1, ip_1, c_1)$

14 $T(del(p_1, pr_1), ins(p_2, ip_2, c_2)) :-$
if $(p_1 < p_2)$ **return** $del(p_1)$
16 **else return** $del(p_1 + 1)$

18 $T(del(p_1), del(p_2)) :-$
if $(p_1 < p_2)$ **return** $del(p_1)$
20 **else if** $(p_1 > p_2)$ **return** $del(p_1 - 1)$
else return $Id()$

FIG. 2.17 – Fonctions de transformation d'IMOR

A la lumière de ces contre-exemples, il est clair que l'approche consistant à garder la position originale ne peut résoudre le TP2-puzzle.

Fonctions de transformation de Suleiman

Suleiman et al. [35] a proposé le jeu de transformées présenté dans la figure 2.20. Suleiman ajoute 2 nouveaux paramètres à l'opération d'insertion a_i et b_i pour obtenir la signature suivante : $ins(p_i, c_i, b_i, a_i)$. a_i est l'ensemble des opérations concurrentes qui ont détruit un caractère après p_i. b_i est l'ensemble des opérations concurrentes ayant détruit un caractère avant p_i.

Avec ces nouvelles informations, si nous prenons deux opérations concurrentes $ins(p_1, c_1, b_1, a_1)$ and $ins(p_2, c_2, b_2, a_2)$ définies sur le même état, nous pouvons déduire les informations suivantes :

- si $(b_1 \cap a_2) \neq \emptyset$ alors c_2 a été inséré avant c_1,
- si $(a_1 \cap b_2) \neq \emptyset$ alors c_2 a été inséré après c_1,
- si $(b_1 \cap a_2) = (a_1 \cap b_2) = \emptyset$ alors c_1 et c_2 ont été insérés à la même position. Dans cas les caractères doivent être placés dans l'ordre de leur code alphanumérique.

Suleiman, dans sa thèse, fait la preuve complète de vérification de C_1 et C_2 à la main. Nous

[35] 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 : The Integration Challenge - GROUP'97*, pages 435–445. ACM Press, November 1997.

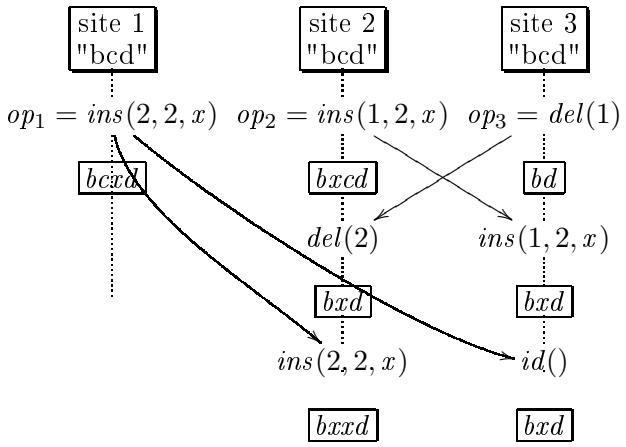


FIG. 2.18 – Contre-exemple pour IMOR

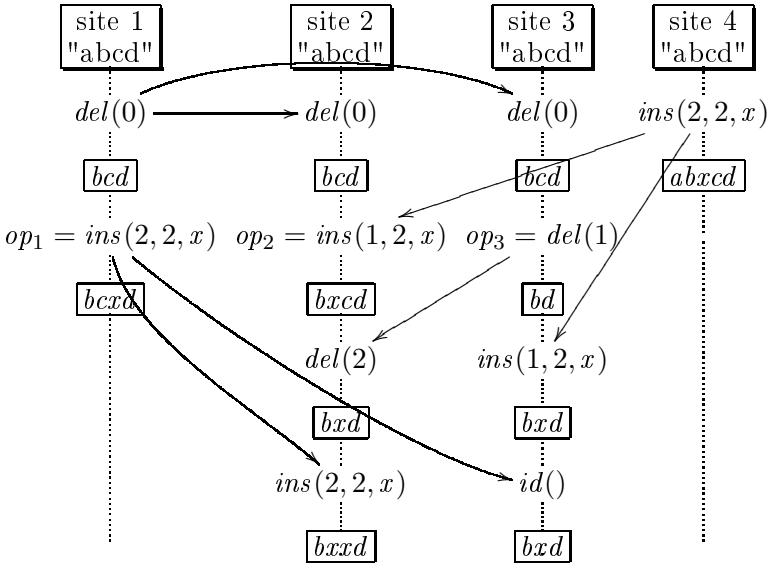


FIG. 2.19 – Contre-exemple complet pour IMOR

```
1  T(ins(p1, c1, b1, a1), ins(p2, c2, b2, a2)) :−
     if (p1 < p2) return ins(p1, c1, b1, a1)
3    else if (p1 > p2) return ins(p1 + 1, c1, b1, a1)
     else if (b1 ∩ a2) ≠ ∅ return ins(p1 + 1, c1, b1, a1)
5    else if (a1 ∩ b2) ≠ ∅ return ins(p1, c1, b1, a1)
     else if (code(c1) > code(c2)) return ins(p1, c1, b1, a1)
7    else if (code(c1) < code(c2)) return ins(p1 + 1, c1, b1, a1)
     else return Id()
9
   T(ins(p1, c1, b1, a1), del(p2)) :−
11   if (p1 > p2) return ins(p1 − 1, c1, b1 ∪ {del(p2)}, a1)
     else return ins(p1, c1, b1, a1 ∪ {del(p2)})
13
   T(del(p1, pr1), ins(p2, c2, b2, a2)) :−
15   if (p1 < p2) return del(p1)
     else return del(p1 + 1)
17
   T(del(p1), del(p2)) :−
19   if (p1 < p2) return del(p1)
     else if (p1 > p2) return del(p1 − 1)
21   else return Id()
```

FIG. 2.20 – Fonctions de transformation définies par Suleiman (97)

avons refait cette preuve automatiquement dans [26]. Malheureusement, nous avons fait la même erreur de spécification que dans la section précédente 2.1.3.0. Nous avons donc écrit :

operation
$p \leq length()$ and $b = \{\}$ and $a = \{\}$: $ins(nat\ p, char\ c, setop\ b, setop\ a)$;
$p < length()$: $del(nat\ p, opid\ i)$;

Avec cette spécification, SPIKE ne génère des opérations d'insertion qu'avec des ensembles vides, ce qui semble logique de prime abord. Malheureusement, cela signifie que nous traitons que des opérations non transformées. Cette hypothèse est bien entendue fausse. Les ensembles des fonctions de Suleiman peuvent être remplis de manière arbitraire. Si nous ré-écrivons la spécification comme suit :

operation
$p \leq length()$: $ins(nat\ p, char\ c, setop\ b, setop\ a)$;
$p < length()$: $del(nat\ p, opid\ i)$;

Avec cette nouvelle spécification, SPIKE trouve le contre-exemple présenté dans la figure 2.21. Dans ce scénario, nous faisons l'hypothèse que $code('x') < code('y')$.

Les fonctions de Suleiman ne vérifient donc pas C_2. Et pourtant Suleiman avait fait la preuve de correction à la main. Cet exemple montre bien combien ces preuves sont génératrices d'erreur et

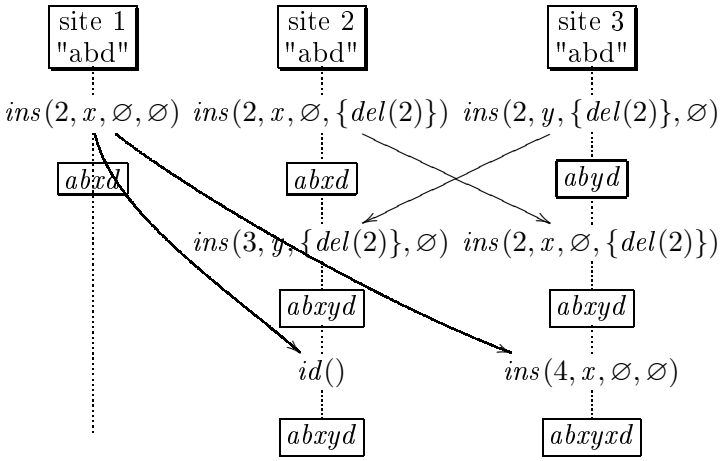


FIG. 2.21 – Contre-exemple pour Suleiman (3 sites)

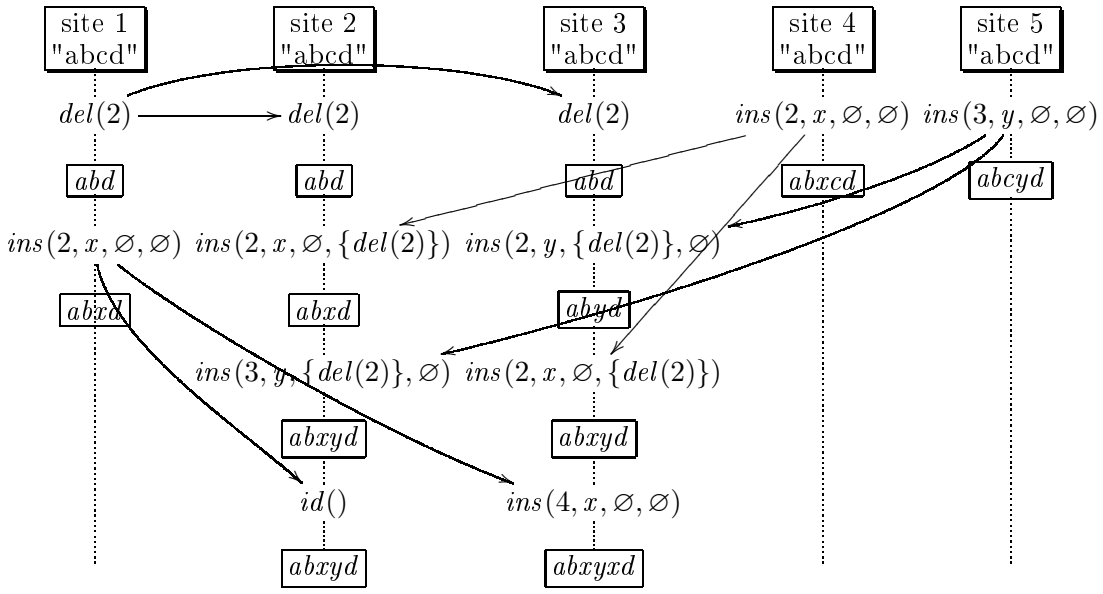


FIG. 2.22 – Contre-exemple complet pour Suleiman

combien la spécification est délicate. Cependant, la spécification des fonctions de transformation de Suleiman prend 20 lignes contre 10 pages pour la preuve. L'approche de la preuve automatique des fonctions de transformation reste donc une approche viable.

Fonction de transformation SDT

En 2004, Li [21] propose un nouveau jeu de transformées avec la preuve à la main de vérification de C_2. Ce nouveau jeu de transformées repose sur l'approche SDT pour "State Difference Transformation". La fonction de transformation pour le couple $(ins(), ins())$ est décrit dans la figure 2.23.

```
1 T(op1 = ins(p1, c1), op2 = ins(p2, c2)) :−
  if β(op1) < β(op2) return op1
3 else if  β(op1) > β(op2) return ins(p1 + 1, c1)
  else
5   if p1 < p2 return ins(p1, c1)
    else if p1 > p2 return ins(p+1, c1)
7   else
      if id(op1) < id(op2) return op1
9      else return ins(p1 + 1, c1)
```

FIG. 2.23 – Fonctions de transformation pour SDT

Les fonctions de transformation SDT introduisent la fonction $\beta(op)$. Cette fonction calcule la position de op sur un état appelé l'état de dernière synchronisation (Last Synchronization Point (LSP). Cet état est identifié par le vecteur d'état $V_{min} = min(v(op_1), v(op_2))$, où $v(op_1), v(op_2)$ sont les vecteurs d'état de op_1 et op_2. V_{min} est formé de la valeur minimale de chaque composant des vecteurs d'état de op_1 et op_2.

Pour calculer la position d'insertion sur l'état LSP d'une opération op définie sur un état s, SDT commence par calculer la séquence d'opération SQ qui mène de LSP vers s. Puis SDT calcule la séquence minimale d'opérations SD équivalente à SQ. Enfin, SDT exclut l'effet de SD sur op_1 et op_2 pour obtenir la position d'insertion sur LSP. La comparaison de $\beta(op_1)$ et $\beta(op_2)$ permet de lever toute ambiguïté.

Nous avons spécifié les fonctions de transformation SDT avec SPIKE sans pour autant spécifier la fonction $\beta()$. SPIKE va alors vérifier tous les scenarii possibles avec toutes les valeurs de retour possible de $\beta()$. SPIKE trouve alors le contre-exemple de la figure 2.24 avec $\beta(op_1) = \beta(op_2) = \beta(op_4)$.

Sur le site 1, sur l'état s_0, pendant $T(op_4, op_1)$ on a $\beta(op_4) = \beta(op_1)$ and $p(op_4) < p(op_1)$. Dans cette situation, les positions d'insertion déterminent le résultat de la transformation. Sur le site 2, sur l'état s_1 et pendant $T(op_4, op_1')$, on a $\beta(op_4) = \beta(op_1')$ and $p(op_4) = p(op_1')$. Maintenant ce n'est plus les positions d'insertion qui résolvent le conflit mais les identificateurs

[21] Du Li and Rui Li. Ensuring content and intention consistency in real-time group editors. In *Proceedings of the Twenty Fourth International Conference on Distributed Computing Systems - ICDCS'04*, pages 748–755, Hachioji, Tokyo, Japan, March 2004. IEEE Computer Society.

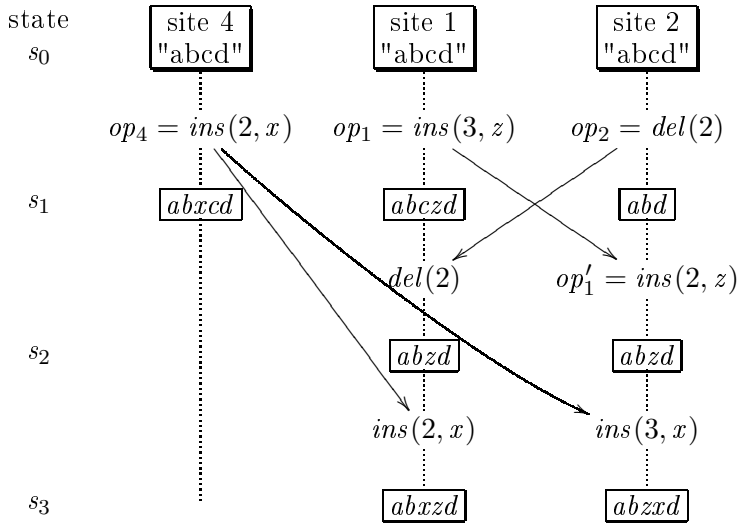


FIG. 2.24 – Contre-exemple potentiel pour SDT

de site. Une même ambiguïté est alors résolue par deux mécanismes différents : les positions et les identificateurs de sites. Si ces deux mécanismes retournent des résultats différents alors il y a divergence.

Comme nous n'avons pas spécifié la fonction β, un tel cas ne peut, peut être, jamais se produire. Le scénario de la figure 2.25 montre que ce cas peut arriver.

Le scénario de la figure 2.25 démontre le problème suivant :

- Pendant $T(op_{41}, op_3)$ sur l'état s_2 du site 1, si $\beta(op_{41}) = \beta(op_3) = 2$, alors les positions d'insertion lèvent l'ambiguïté.
- Pendant $T(op_{41}, op_3)$ sur l'état s_3 du site 2, si $\beta(op_{41}) = \beta(op_3) = 2$, alors les identificateurs de sites lèvent l'ambiguïté.

Nous avons vérifié que ce contre-exemple conduit bien à une divergence sur l'implantation du SDTO fournie par les auteurs [12].

Ce contre-exemple démontre le besoin de preuve automatique. Les fonctions de transformation deviennent de plus en plus compliquées et les contre-exemples sont quasiment impossibles à trouver sans l'aide de l'ordinateur. En effet, les fonctions de transformation sont petites, mais elles génèrent une combinatoire de cas extrêmement importante.

2.1.4 Fonctions de transformation TTF

Toutes les fonctions de transformation présentées précédemment ne peuvent résoudre un même problème récurrent. Ce problème, présenté dans la figure 2.26, met en jeu 3 opérations concurrentes : $ins(1, x)$, $del(1)$ and $ins(2, y)$.

Quand $op_1 = ins(1, x)$ est reçue sur le site 1, elle est transformée par rapport à op_2 et retourne l'opération $op'_1 = ins(1, x)$. Quand $op_3 = ins(2, y)$ est reçue sur le site 2, elle est transformée par rapport à op_2 et devient $op'_2 = ins(1, y)$.

[12] SDTO est une version optimisée de SDT. Elle est fournie par les auteurs sur leur site WEB `http://cocasoft.csdl.tamu.edu/~lidu/projects/CE/`

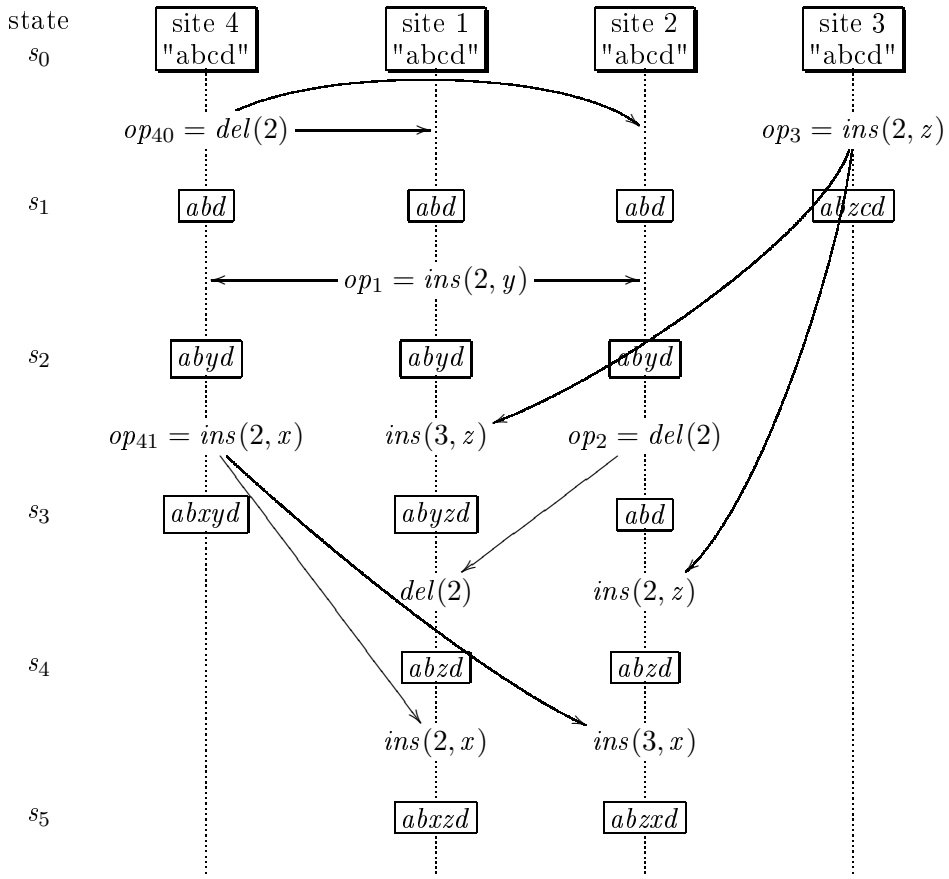


FIG. 2.25 – Contre-exemple pour SDT

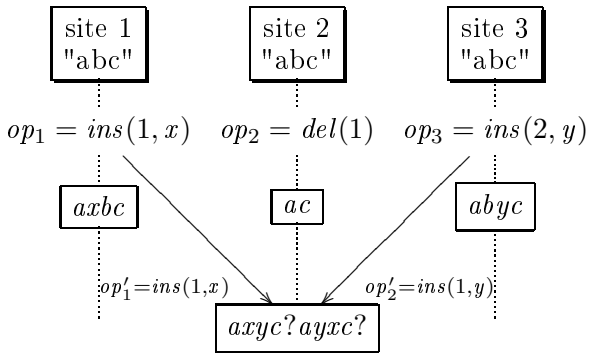


FIG. 2.26 – Problème récurrent.

Les positions d'insertion de op'_1 et op'_2 deviennent égales par transformation, mais il est clair que 'x' a été inséré avant 'y' sur l'état initial. En effet 'x' a été inséré entre 'a' et 'b' et 'y' a été inséré entre 'b' et 'c'. Le fait de détruire 'b' génère le problème.

Quand op'_1 est transformé par rapport à op'_2 ou vice-versa, les positions sont égales, donc il faut trancher. Mais cela doit être fait en tenant compte du fait que sur l'état initial, 'x' était avant 'y'. Si le conflit est tranché sans tenir compte de cette information, alors nous obtenons une violation de l'intention.

IMOR[26] tente de garder les positions pour trancher le faux-conflit, Suleiman et al.[34] maintient des ensembles d'opération concurrente, Li et al.[22] tente de régénérer l'état initial, mais finalement toutes les tentatives, même les plus complexes, échouent. Tous les contre-exemples que nous avons trouvés sont finalement des instances de ce problème récurrent.

Après des années d'effort, nous avons finalement analysé le problème de la façon suivante. La destruction de caractères pivots comme le 'b' dans notre exemple est la cause fondamentale des ambiguïtés.

Nous avons développé l'idée qu'il fallait absolument garder ce caractère, même détruit, sous forme de pierre tombale. Si un caractère est détruit, le système garde une trace de ce caractère concernant sa position. L'idée des pierres tombales vient des systèmes distribués et est largement utilisé dans Usenet[13], et Active Directory[30].

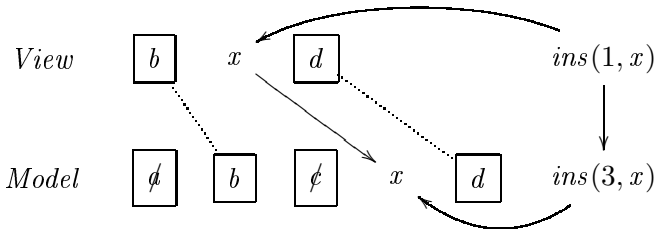


FIG. 2.27 – Modèle d'une chaîne de caractères pour TTF

Si nous appliquons cette approche à notre problème, cela a pour conséquence de garder des caractères cachés dans le modèle d'une chaîne de caractère sur un site. Ces caractères sont présents dans le modèle mais pas visible dans la vue de l'utilisateur. Le modèle d'une chaîne de caractère devient une séquence de paires ordonnées $(caractre, visible)$ où visible est un attribut de type booléen. Ce modèle et son comportement sont illustrés dans la figure 2.27. L'opération $ins(1, x)$ est générée sur la vue mais exécutée sur le modèle comme $ins(3, x)$. C'est l'opération $ins(3, x)$ qui est transmise aux autres sites pour y être transformées.

Pour faire la conversion entre la position d'insertion dans la vue et la position d'insertion dans le modèle, le système calcule la fonction suivante :

```
1 view2model(int iview) : int
```

[13] Les news en français...

[34] 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, Orlando, Florida, USA, February 1998. IEEE Computer Society.

[22] Du Li and Rui Li. Preserving operation effects relation in group editors. In *Proceedings of the ACM conference on Computer supported cooperative work - CSCW '04*, pages 457–466. ACM Press, 2004.

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

```
begin
 int n=0, j=0;
 while(n < iview or (not model[j]. visible )) do
   if (model[j]. visible ) then n++
   endif
   j++
 endwhile
 return j
end
```

Cette fonction ne fait que retrouver le n-ième caractère visible dans le modèle. Il suffit pour cela de compter les pierres tombales. Cette stratégie nous conduit à écrire les fonctions de transformation triviale présentées dans la figure 2.28. Nous les avons appelées TTF, pour "Tombstone Transformation Functions".

Un paramètre additionnel sid_i a été ajouté aux opérations. Il identifie de manière unique le site sur lequel cette opération a été générée. Quand un vrai conflit d'insertion est détecté, l'ordre total des sites est utilisé pour trancher le conflit.

$T(ins(p_1, c_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  if $(p_1 < p_2)$ **return** $ins(p_1, c_1, sid_1)$
  else if $(p_1 == p_2$ and $sid_1 < sid_2)$ **return** $ins(p_1, c_1, sid_1)$
  else return $ins(p_1 + 1, c_1, sid_1)$

$T(del(p_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  if $(p_1 < p_2)$ **return** $del(p_1, sid_1)$
  else return $del(p_1 + 1, sid_1)$

$T(ins(p_1, c_1, sid_1), del(p_2, sid_2)) :-$
  return $ins(p_1, c_1, sid_1)$

$T(del(p_1, sid_1), del(p_2, sid_2)) :-$
  return $del(p_1, sid_1)$

FIG. 2.28 – Fonctions de transformation TTF.

Puisque l'exécution d'une opération *del* remplace le caractère par une pierre tombale, elle n'affecte pas la position des autres caractères dans la chaîne de caractère. Transformer une opération par rapport à $del()$ ne modifie donc pas la position de cette opération.

Nous avons vérifié ces transformations avec le prouveur de théorème, et ces fonctions vérifient effectivement C_1 et C_2. Nous avons donc trouvé le premier jeu de transformées qui vérifient C_1 et C_2. Ces fonctions utilisées avec l'algorithme Adopted permettent de construire un éditeur collaboratif multi-synchrone.

Adopted a cependant un inconvénient, il utilise une matrice multi-dimensionnelle proportionnelle au nombre de sites. Si le nombre de sites devient important, cette matrice devient énorme.

Les algorithmes Soct2 et Goto ne requièrent pas cette matrice, mais nécessitent de définir des fonctions de transformation inverses.

Étant donné que les fonctions TTF sont injectives, la définition des fonctions inverses est triviale. Ces fonctions sont présentées dans la figure 2.29

```
T^{-1}(ins(p1, c1, sid1), ins(p2, c2, sid2)) :-
  if (p1 < p2) return ins(p1, c1, sid1)
  else if (p1 == p2 and sid1 < sid2) return ins(p1, c1, sid1)
  else return ins(p1 − 1, c1, sid1)

T^{-1}(del(p1, sid1), ins(p2, c2, sid2)) :-
  if (p1 < p2) return del(p1, sid1)
  else return del(p1 − 1, sid1)

T^{-1}(ins(p1, c1, sid1), del(p2, sid2)) :-
  return ins(p1, c1, sid1)

T^{-1}(del(p1, sid1), del(p2, sid2)) :-
  return del(p1, sid1)
```

FIG. 2.29 – Fonctions de transformation inverse pour TTF

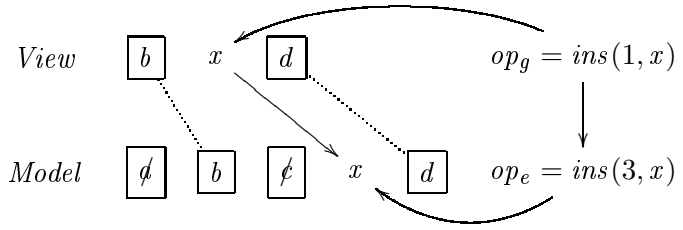
Ces fonctions inverses permettent d'utiliser les fonctions de transformations TTF avec Soct2 et GOTO.

Bien que les pierres tombales constituent une solution très simple au problème récurrent de l'approche transformationnelle, elles présentent aussi l'inconvénient de marquer et de se souvenir des caractères détruits. Cela a pour conséquence que l'espace mémoire occupé les pierres tombales croit indéfiniment. Deux approches existent dans les systèmes distribués pour prévenir ce problème. La première approche repose sur une période d'expiration liée à chaque pierre tombale. Lorsque cette période est écoulée, alors la pierre tombale est définitivement supprimée. Malheureusement, cette méthode n'est pas sûre. Si une opération arrive et suppose l'existence d'une pierre tombale expirée, alors le système va diverger.

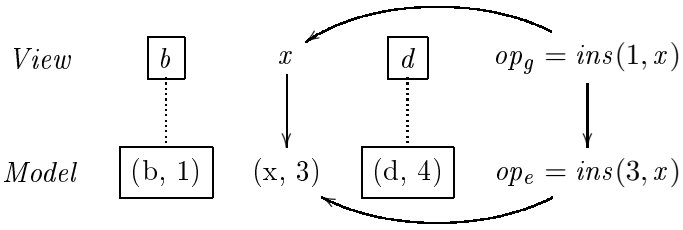
La seconde approche repose sur un protocole à deux phases pour supprimer définitivement les pierres tombales [30]. Malheureusement, cette approche suppose que tous les sites soient vivant pour permettre à l'algorithme de progresser. Cela signifie que chaque participant doit rester connecté pendant l'exécution du ramasse-miette. Cette hypothèse n'est pas adaptée pour les éditeurs collaboratif asynchrone ou multi-synchrone.

Nous proposons une nouvelle approche originale en changeant le modèle de données pour une chaîne de caractère. Dans le modèle présenté précédemment, les pierres tombales sont représentées comme les caractères barrés de la chaîne $\not{a}b\not{c}d$ (cf figure 2.30(a)). Dans le modèle modifié, nous ne gardons que les caractères visibles avec leurs positions absolues. Ce nouveau modèle est illustré dans la figure 2.30(b). Une chaîne de caractère est donc une séquence de paires ordonnées

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



(a) Modèle originel



(b) Modèle compacté

FIG. 2.30 – Différence entre le modèle original et le modèle compacté des TTF.

(*caractre*, *position*). La fonction *view2model* utilisée pour recalculer les positions de la vue dans le modèle doit être ré-écrite comme suit :

```
view2model(int iview) : int
2 begin
  return (model[iview]).position
4 end
```

L'insertion d'un caractère dans le modèle à la position p incrémente la position de chaque caractères situé après le caractère inséré. La destruction d'un caractère à la position p détruit juste la paire correspondante sans affecter les autres caractères. En utilisant ce nouveau modèle, l'espace occupé par les pierres tombales est nul. Si ce modèle demande un peu plus de calculs, la complexité en temps dans le pire cas d'une intégration reste en $O(n^2)$.

Les fonctions de transformation TTF sont actuellement les seules fonctions de transformation vérifiant C_1 et C_2. Nous avons également défini les fonctions inverses et prouvé la propriété d'inversibilité. Nous n'avons pas prouvé que les fonctions de transformation TTF préservent les intentions. Pour cela il faut commencer par définir les intentions des opérations *ins* et *del* sur une chaîne de caractère.

Nous exprimons l'intention de l'opération $ins(p, c)$ en utilisant la relation de précédence $\prec$ comme dans[22]. Si un utilisateur génère une opération $op = ins(p, c)$ sur un site ou x est visible à une position inférieure à p et y visible à une position supérieure ou égale à p alors la relation $x \prec c \prec y$ est définie. L'intention d'une opération $del(p)$ exécutée sur une chaîne S est de détruire le caractère $S[p]$.

[22] Du Li and Rui Li. Preserving operation effects relation in group editors. In *Proceedings of the ACM conference on Computer supported cooperative work - CSCW '04*, pages 457–466. ACM Press, 2004.

Si nous voulons préserver l'intention d'une opération $ins()$, cela signifie qu'une fois une relation de précédence définie, cette relation existe sur tous les états successifs. Puisque notre opération $del()$ n'affecte pas la position des caractères, les relations seront toujours conservées sur le site de génération de l'opération op. Maintenant, étant donné que les fonctions de transformation TTF assure la convergence, cela implique que pour toute opération $ins(p, c)$, la relation d'ordre $x \prec c \prec y$ sera préservée. Donc, les fonctions TTF vérifient C_1, C_2 et préservent les intentions définies sur une séquence de caractères.

Chapitre 3

Conclusions et Perspectives

Les fonctions de transformation TTF représentent une percée importante. Elles résolvent enfin un problème soulevé dès 1989. La manière dont nous avons résolu le problème peut être réutilisé pour construire d'autres fonctions de transformation pour des types plus complexe. Nous travaillons déjà avec des résultats prometteurs, sur des fonctions de transformation XML. En 96, Ressel a publié modèle avec un gros potentiel mais personne jusqu'à présent n'avait réussi à l'instancier correctement. Nous l'avons fait simplement et nous sommes maintenant prêt à construire des éditeurs collaboratifs multi-synchrones décentralisés reposant sur l'idée originale de Ressel.

3.1 Édition collaborative sur réseau Pair-à-Pair

Ces dernières années ont connue un fort développement dans l'utilisation des environnements collaboratifs. De nombreuses applications collaboratives sont maintenant très populaires comme les messageries instantanées, les blogs, les environnements de e-learning, les gestionnaires de contenus, les calendriers partagés ...

L'édition collaborative s'est particulièrement illustrée par l'adoption par le grand public des éditeurs Wikis. L'encyclopédie Wikipédia est le fer de lance de cette démocratisation de l'édition collaborative. La wikipédia est un formidable exemple de ce qui peut être réalisé par un effort collaboratif mondial.

Cependant, ces éditeurs collaboratifs ont un défaut intrinsèque : ils reposent sur une architecture centralisée. Cette architecture est un obstacle majeur à plusieurs titres :

- Un serveur central peut tomber en panne et dans ce cas les données ne sont bien sûr plus disponibles. Il faut remarquer que les données rédigées collaborativement sont souvent des données importantes. De nombreuses entreprises utilisent des Wikis pour faire de la gestion de connaissances. Toute panne est alors très pénalisante.
- Un serveur central peut être facilement attaqué. Si la sécurité du serveur est compromise, il ne reste plus qu'à arrêter le serveur jusqu'à réparation de la panne. Le temps nécessaire à cette réparation est indéterminé, les données sont alors indisponibles pour un temps indéterminé.
- Un serveur central représente un goulet d'étranglement pour le passage à l'échelle et les performances. La seule solution est alors d'investir dans des machines coûteuses et de louer

la bande passante nécessaire. A titre indicatif, l'hébergement de la wikipédia a coûté 198000 Dollards pour le trimestre du 1er juillet au 30 septembre 2005 [14]. Une architecture centrale ne permet pas de partager les coûts.
- Un serveur central est contrôlé par une personne, une entreprise ou une organisation et permet donc la censure. L'hébergeur de l'application collaborative contrôle alors le contenu.

Les architectures pair-à-pair représentent une évolution logique pour les outils d'édition collaboratif. Si les données partagées sont répliquées sur un réseau pair-à-pair, alors le système devient tolérant aux pannes, la charge peut être distribuée sur l'ensemble du réseau et ainsi résoudre les problèmes de performance et de bande passante. Le contenu étant disponible sur chaque noeud, les risques de censure sont écartés. Enfin, il devient possible de partager les coûts d'exploitation en apportant dans le réseau des ressources en terme de machine et de bande passante.

3.1.1 L'approche WOOT

Le modèle des transformées est à première vue bien adapté aux réseaux pair-à-pair. En effet, le modèle des transformées n'impose pas de site central. Malheureusement, les algorithmes de transformées existant s'appuient sur des vecteurs d'états. Cela suppose que chaque noeud du réseau connaît la taille du réseau, ce qui, bien sûr, n'est pas le cas sur un réseau P2P. En effet, le "churn" ou l'agitation dans les réseaux P2P rend l'utilisation de techniques de base des systèmes distribués inadéquates.

Je travaille actuellement sur des algorithmes d'édition collaborative où le nombre de participants n'est pas un paramètre. L'objectif est toujours d'assurer la convergence et le respect des intentions mais en faisant abstraction du nombre de participants. Pour résoudre ce problème, nous avons décidé de quitter l'approche OT pour une nouvelle approche que nous avons baptisée WOOT . L'idée de cette approche est simple : au lieu de recalculer les relations d'ordre a posteriori comme dans OT, nous diffusons les relations d'ordre avec les opérations. Ainsi, nous sommes déjà sûr de préserver les intentions. Par exemple, quand un utilisateur observe la chaîne de caractères `"ABCDE"` et qu'il insère `"12"` à la position 2, on peut supposer qu'il veut insérer `"12"` entre `"A"` et `"B"`. Respecter les effets de cette opération consiste à préserver les relations `"A"` $\prec$ `"12"` $\prec$ `"B"` sur tous les futurs états de la chaîne. Nous adaptons le profil des opérations. Au lieu de diffuser l'opération $insert(2, \texttt{"12"})$, nous diffusons l'opération $insert(\texttt{"A"} \prec \texttt{"12"} \prec \texttt{"B"})$. Un premier problème se pose : comment exécuter une opération $insert(\texttt{"A"} \prec \texttt{"12"} \prec \texttt{"B"})$ si nous avons localement supprimé le caractère `"A"` ? Nous choisissons de ne pas détruire les caractères, nous les marquons comme invisibles. Cependant la complexité mémoire de notre algorithme reste inférieure ou égale à celle des algorithmes existants. En effet, nous ne conservons pas le journal des opérations et nous n'utilisons pas de vecteur d'horloges.

Chaque opération d'insertion génère deux nouvelles relations. L'ensemble des relations constitue un ordre partiel. Considérons trois sites qui génèrent chacun une opération comme décrite dans la figure 3.1. Le diagramme de Hasse de la figure 3.2 représente l'ensemble des relations entre les caractères.

Les états de convergence où les intentions sont préservées correspondent à l'ensemble des

[14] `http://wikimediafoundation.org/wiki/Budget/fr/2005`

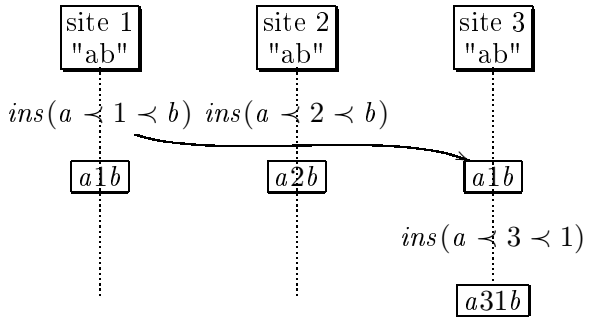


FIG. 3.1 – Génération d'un ordre partiel.

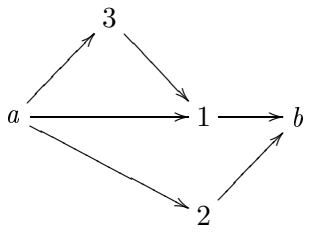


FIG. 3.2 – Diagramme de Hasse des relations d'ordre entre les caractères.

extensions linéaires de cet ordre partiel i.e. `a312b`, `a321b`, `a231b`. Pour assurer la convergence des répliques, tous les sites doivent trouver la même extension linéaire. L'objectif de cette nouvelle approche est donc de garantir la convergence en utilisant une fonction de linéarisation [32].

3.1.2 Procédés d'édition collaboratifs sur réseaux P2P

Si l'édition collaborative sur réseaux P2P remet en cause les algorithmes existant de maintient de la cohérence des données partagées, il remet aussi en cause les modèles de coordination et de conscience de groupe.

La convergence et le respect des intentions sont nécessaires pour construire un éditeur collaboratif mais pas suffisant. L'encyclopédie collaborative Wikipédia en est un bon exemple. Au début de son existence, n'importe qui pouvait modifier n'importe quoi, n'importe quand. Mais la croissance vertigineuse de wikipédia a remis en cause ce principe fondateur. Il existe maintenant des rôles et des procédés pour modifier les articles considérés comme stables. Il existe aussi des procédés pour catégoriser, fusionner et détruire des articles.

Ces procédés sont actuellement dédiés à la problématique d'une encyclopédie collaborative déployée sur une architecture centralisée. L'objectif est de modéliser ces procédés, de les généraliser et les déployer sur une architecture P2P.

3.1.3 Conscience de groupe dans l'édition collaborative sur réseaux P2P

Il est clair que la connaissance des actions passées, présentes et à venir des autres participants rend l'action d'un acteur de la coopération bien plus efficace. Elle permet de développer une coordination informelle à la différence des modèles formels de coordination comme le workflow ou le dataflow.

Dans le cadre de l'édition collaborative, je m'intéresse plus particulièrement aux artefacts de conscience de groupe asynchrone. En effet, de nombreux artefacts ont été développé pour les interactions synchrones, mais très peu sont à destination des utilisateurs travaillant à des moments différents ou en isolation.

J'ai déjà travaillé sur des artefacts liés à la divergence entre les différent utilisateurs. On peut mesurer la divergence en estimant l'importance des conflits qu'il faudra résoudre pour obtenir la convergence[40]. On peut localiser les points de divergence pour provoquer des interactions comment dans les "state treemap"[43].

J'ai développé un algorithme de quantification de la divergence pour des objets typés [40, 24]. Cette mesure a été déployée dans le prototype MOTU. La métrique développée est intrinsèquement indépendante des types d'objet manipulés. Il est donc possible de fournir une métrique grossière basée des objets génériques comme XML ou d'affiner la métrique en fonction de critère très fin. En ce sens, la métrique est plutôt un environnement de développement de métrique. Nous avons développé la métrique dans le cadre d'objet XML générique. Nous avons proposé plusieurs modes de visualisation de cette métrique à destination des utilisateurs finaux [43].

Malheureusement, nous n'avions pas à cette époque les outils que nous avons développés depuis. Nous n'avions pas alors les moyens de développer rapidement des métriques fiables. Une perspective de recherche est d'utiliser les algorithmes développés dans la section 3.1.1 non pas pour assurer la convergence mais pour calculer une métrique de divergence. Cette métrique pourra alors être visualisée pour provoquer des interactions et générer de la coordination informelle.

Troisième partie

Transferts industriels, logiciels, prototypes

Depuis la création de l'équipe, nous avons toujours cherché à expérimenter et valider nos propositions de recherches par des prototypes. Cette caractéristique forte de l'équipe nous a amené à écrire de nombreux prototypes. Ces prototypes nous ont beaucoup apporté. Ils nous ont fait découvrir de nouvelles problématiques, ils nous ont maintenus en contact avec les dernières avancées technologiques, ils nous ont permis de comprendre les difficultés liées aux transferts industriels : propriété intellectuelle, communication, travail avec des partenaires industriels, pérennité du logiciel, animation de communautés ... Ils nous ont donné de l'expérience en développement pour pouvoir le transmettre dans nos activités d'enseignements.

Je me suis personnellement beaucoup investi dans ces développements. D'abord en tant que programmeur, puis en tant qu'architecte, puis en tant qu'interlocuteur des projets.

1.1 P-RooT & COO (1993-96)

COO est un environnement coopératif de développement de logiciels[35, 19, 38, 3]. Il est basé sur les COO-transactions. Le prototype COO est articulé autour de services de base :

- P-RooT est un référentiel orienté objet développé au dessus d'un référentiel PCTE [15]. Il assure en premier lieu le stockage des objets persistants. Il gère les accés concurrents à ces objets à travers des transactions ACID classiques.
- Le composant bases/sous-bases émule sur P-RooT le partitionnement de la base selon le modèle bases/sous-bases [8]. La définition des opérateurs de transfert *checkout*, *checkin* et *update* se trouve à ce niveau.
- Les transactions coopératives s'appuient sur les bases/sous-bases pour gérer leurs mémoires locales. Les opérateurs de transfert sont surchargés pour implanter le protocole COO.
- Un service de verrouillage flexible permet de définir des modèles de verrous. Il est alors possible de restreindre les exécutions autorisées par le protocole COO selon diverses politiques. Une de ces politiques peut être un protocole isolant à l'image du 2PL.
- Enfin, un service de contraintes temporelles permet d'exprimer des contraintes d'intégrité mais aussi des contraintes de transition. Ces contraintes permettent de caractériser les états corrects de la base ainsi que la manière d'y parvenir.

Dans ce prototype, j'ai personnellement réalisé le service de transactions, et une bonne partie de la base P-Root sur une idée de François Charoy, aujourd'hui Maître de Conférences à l'Université Henri Poincaré. Notre prototype n'est pas construit en empilant des couches de services mais en fusionnant de manière dynamique des services pour former l'environnement d'une session.

Nous présentons quelques photos d'écrans du prototype P-RooT&COO. Ces photos ont été prises lors de la démonstration à BDA'95.

La figure 1.3 montre 4 fenêtres. La première représente l'arbre des COO-transactions. Dans cette situation, il y a trois COO-transactions ; la transaction racine et deux sous-transactions *MODULEA* et *MODULEB*. Dans le cadre d'un développement logiciel, *MODULEA* doit développer le module A et *MODULEB*, le module B.

[15] Programme Common Tool Environment. Norme ISO JTC1/SC22/WG22.

[8] F. Bancilhon, W. Kim, and H. Korth. A Model for CAD Transactions. In *Proceedings of the 11th international conference on VLDB*, pages 25–33, Stockholm, august 1985.

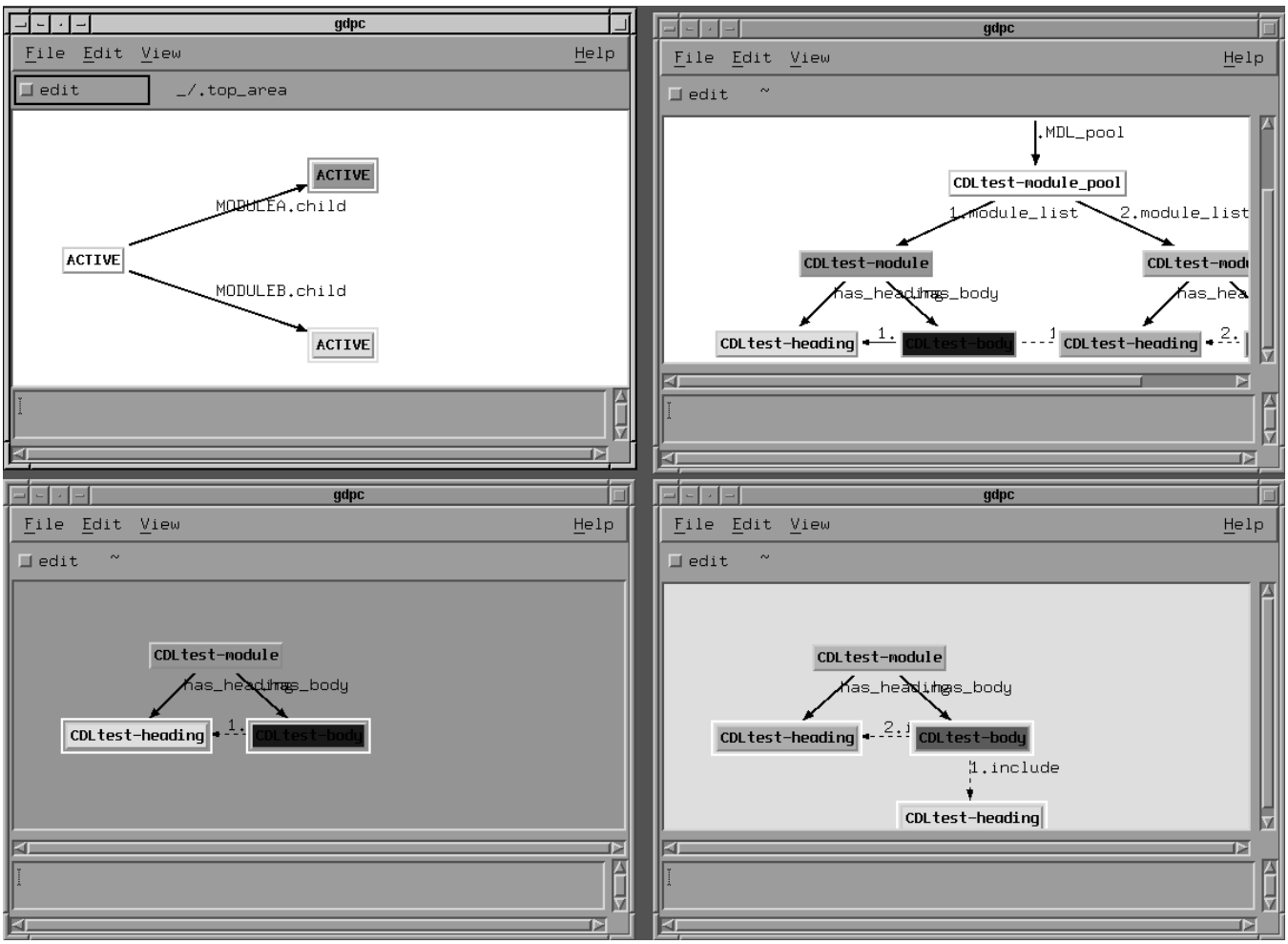


FIG. 1.3 – P-RooT&COO : Un état initial

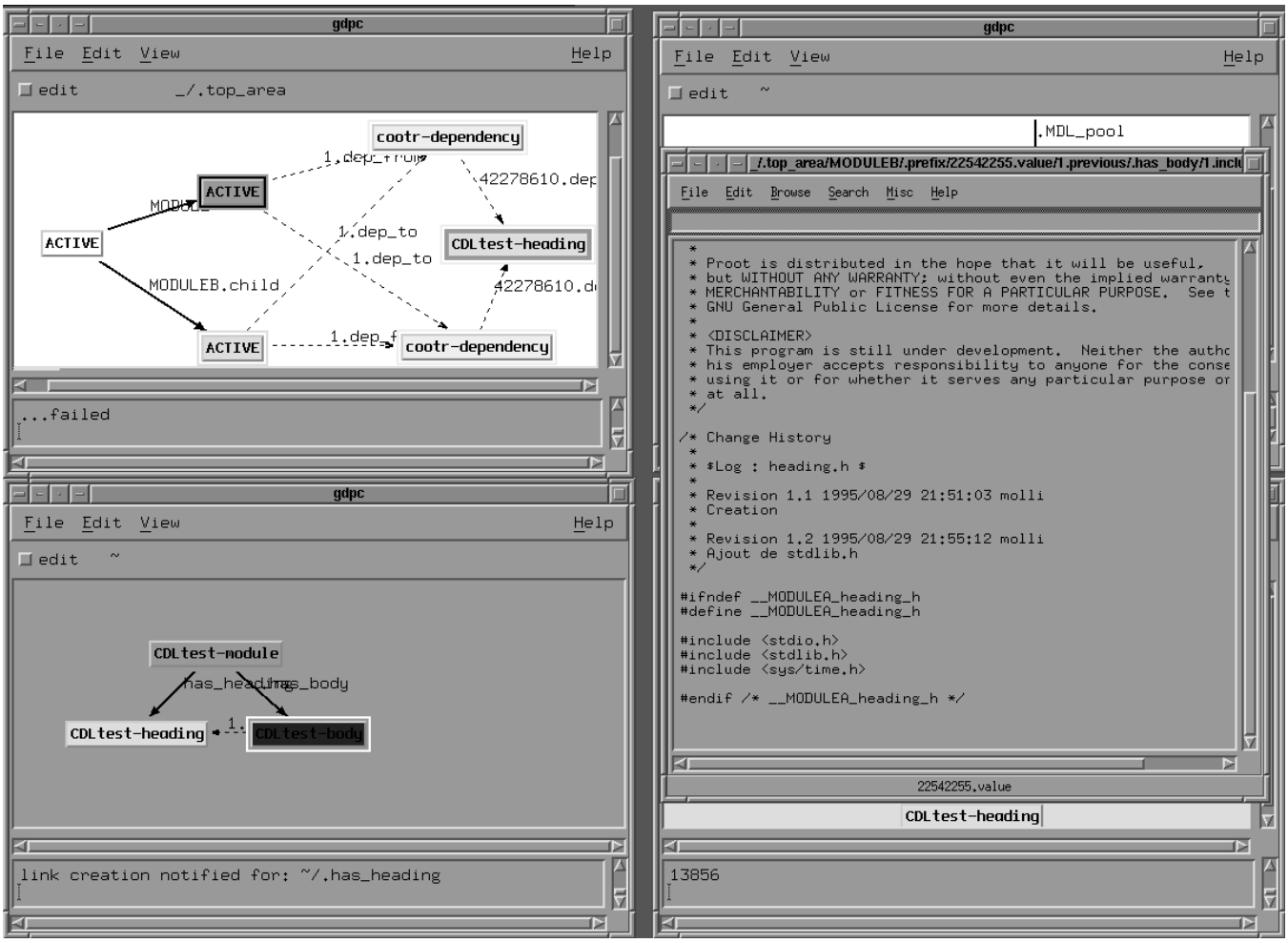


FIG. 1.4 – P-RooT&COO : appel de l'opération "edit" dans une COO-transaction

La fenêtre en haut à droite représente la mémoire locale de la transaction racine, la fenêtre en bas à gauche, celle de la COO-transaction *ModuleA* et la dernière la mémoire de locale de *MODULEB*

Les noeuds du graphe des mémoires locales représentent des objets. L'intitulé du noeud correspond au type de l'objet, les arcs symbolisent les relations entre objets. La mémoire de la transaction racine contient les deux modules tandis que la mémoire de *MODULEA* contient le module A et la mémoire de *MODULEB* contient le module B.

La figure 1.4 montre l'appel d'un outil sur un objet présent dans la mémoire de la transaction racine. Les échanges d'objets entre les sous-transactions ont développé un réseau de dépendances entre *MODULEA* et *MODULEB*. Ces dépendances sont représentées dans la fenêtre en haut à gauche.

La figure 1.5 montre comment le protocole COO réagit en cas de violation de la COO-sérialisabilité. Une exception est déclenchée pour avertir l'utilisateur que sa demande de terminaison de transaction est refusée.

Le prototype COO était construit sur la plate-forme PCTE. La plate-forme PCTE n'a jamais percé, nous avons du nous résoudre à abandonner ce développement qui proposait certaines

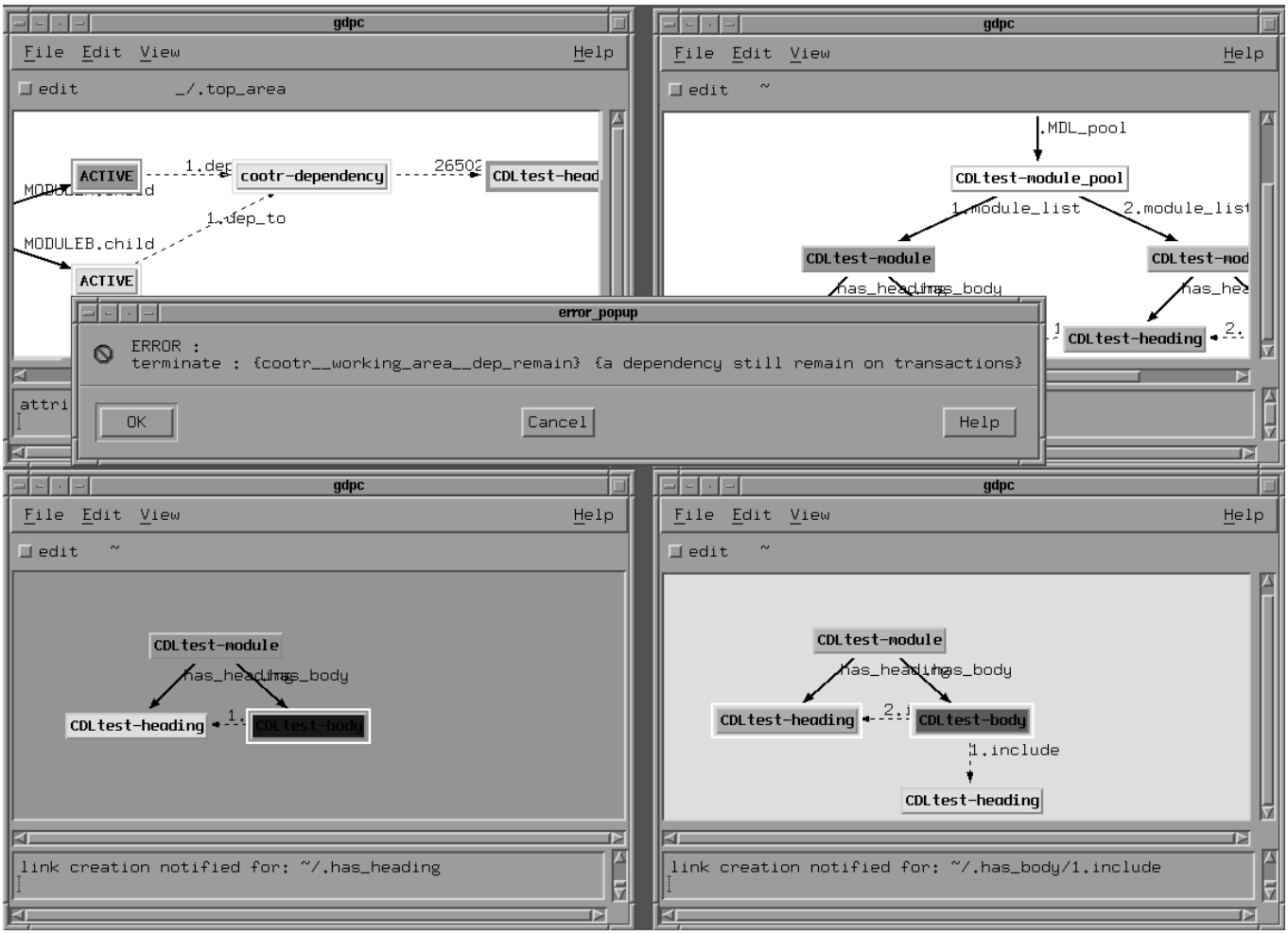


FIG. 1.5 – P-Root&COO : le protocole COO détecte une erreur

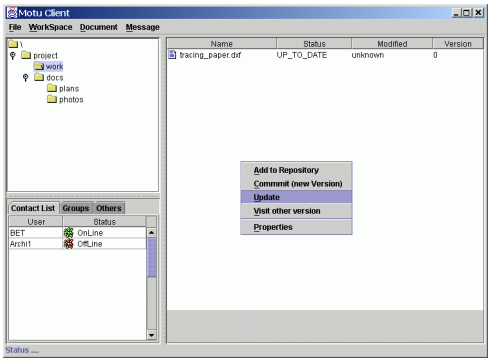


FIG. 1.6 – Interface générale des Motu

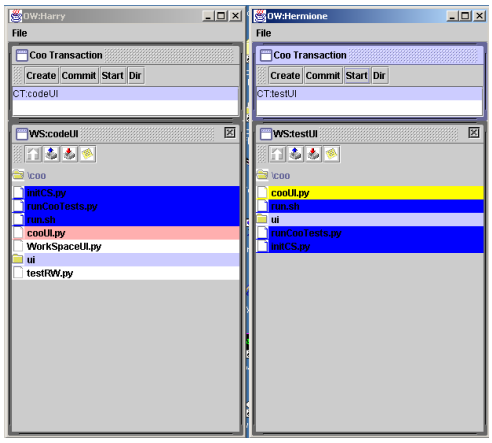


FIG. 1.7 – Interface des COO-Transactions

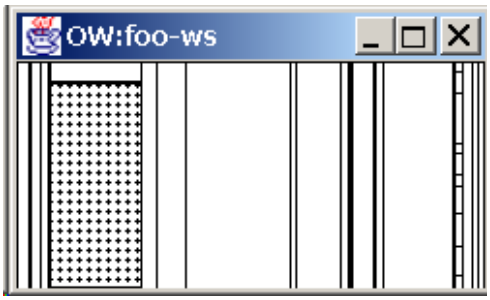


FIG. 1.8 – State Treemap

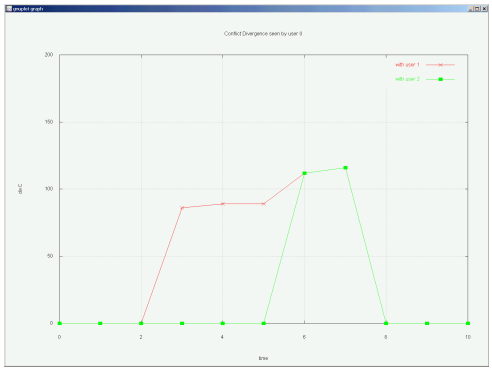


FIG. 1.9 – Métrique de divergence

fonctionnalités que nous ne sommes toujours pas en mesure de reproduire aujourd'hui.

1.2 MOTU (1996-1998)

MOTU est une plate-forme de collaboration [39]. Elle a servi de plate-forme d'expérimentation pour le projet COCAO (voir page 8).

Alors que COO était essentiellement tourné vers la cohérence des données et les procédés de développement, nous nous somme rendus compte qu'un environnement collaboratif devait aussi intégrer des fonctionnalités de communication et de conscience de groupe. J'ai développé la majeure partie de prototype MOTU en utilisant exclusivement la technologie Java.

Motu est diffusé en "open-source" et peut être téléchargé depuis motu.sourceforge.net. Motu fournit les services suivant :

Données partagées La plate-forme Motu fournit des services de gestion de versions non-linéaire. La concurrence d'accés est gérée par un gestionnaire de transactions coopératives développé dans ECOO. Les COO-transactions (cf figure 1.7) autorisent des exécutions non-sérialisables tout en garantissant les propriétés de cohérence et de durabilité.

Communication La plate-forme Motu fournit un service de visio-conférences intégré (cf fig-

FIG. 1.10 – Visio-conférence

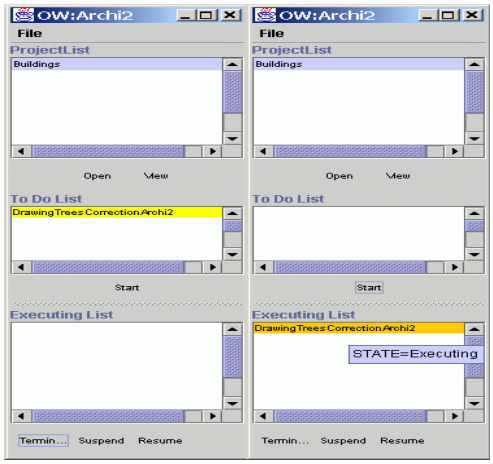


FIG. 1.11 – moteur de Workflow

ure 1.10) et une gestion des messages instantanés. L'intérêt d'une intégration à la plate-forme est la facilité de lancement d'une visio-conférence à n personnes.

Coordination La plate-forme Motu fournit un service de workflow original développé dans l'équipe ECOO (cf figure 1.11). Celui-ci permet une exécution de processus flexibles adaptée au travail en groupe.

Conscience de groupe La plate-forme Motu permet aux utilisateurs d'être conscients :

1. de la présence des membres à travers un outil intégré de gestion de présence,
2. De la divergence entre les données répliquées à travers deux résultats de recherche originaux : les "State-Treemap" [43] (cf figure 1.8) et des métriques de divergence [24] (cf figure 1.9).

Le projet COCAO a fait l'objet d'un film réalisé par l'INRIA montrant le prototype MOTU en action.

1.3 JlibDiff (Diffusée depuis 1999)

jLibDiff est une librairie java pour calculer le chemin d'édition minimal entre deux fichiers textes. Elle est diffusée sous licence LGPL et disponible sur Sourceforge.

Nous avons réalisé cette librairie essentiellement dans le but de founir les données de base pour calculer la métrique de divergence dans MOTU.

La librairie est trés facile à utiliser comme le montre l'exemple ci-dessous :

```
import diff.*;
import java.io.*;
class test{
  public static void main(String[] args)throws IOException{

    if((args.length)<2)
      System.out.println("diff requires two file names.");
```

```
8               else
        {
10                      differ  d=new differ(args[0], args[1]);
                        d.print ();
12                      d.save("diff.txt");
        }
14          }
}
```

1.4 3GBooster, SERIALI (2001)

A force de développer des protocoles de synchronisation de données, nous avons développé une expertise dans le domaine. La société 3GBooster nous a contacté en 2001 pour résoudre des problèmes de prise de rendez-vous par internet par des artisans ou des professions libérales.

Le problème consistait à synchroniser des prises de rendez vous faites par des clients par internet et d'autres prises de rendez effectuées localement dans le cabinet ou par le commerçant.

Nous avons appliqué notre savoir-faire pour résoudre ce problème. 3GBooster est devenue SERIALI entre temps, et son slogan est "SERIALI : Optimisation and Synchronisation processes".

1.5 TOXIC (2002)

Toxic est une plate-forme de collaboration [44, 28, 9]. Elle a servi de base d'expérimentation pour le project COOPERA (voir page 8).

Toxic Farm reprends l'approche de MOTU. Les services de partage de données, de communication, de coordination et de conscience de groupe ont cette fois été implantés sous forme d'une application WEB (cf figure 1.12) :

Données partagées La plate-forme Toxic fournit une première approche d'un synchroniseur de données. Ce synchroniseur reprend l'approche de Pierce [7]. C'est en construisant ce service de synchronisation que je me rend compte qu'il est possible de construire un synchroniseur de fichiers en suivant l'approche des transformées opérationnelles.

Coordination La plate-forme Toxic s'interface pour la première fois avec le gestionnaire de workflow Bonita développé dans l'équipe..

Conscience de groupe La plate-forme Motu permet aux utilisateurs d'être conscients reprend l'idée des "state treemap" développé dans Motu.

Je suis à l'origine de ce développement. Ce projet nous a servi de vitrine technologique. Il a montré notre savoir-faire en matière de plate-forme de collaboration. Il a été le laboratoire du projet Libresource et le sujet de l'étude du projet COOPERA.

---

[7] Sundar Balasubramaniam and Benjamin C. Pierce. What is a file synchronizer? In *Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking - MobiCom'98*, October 1998.

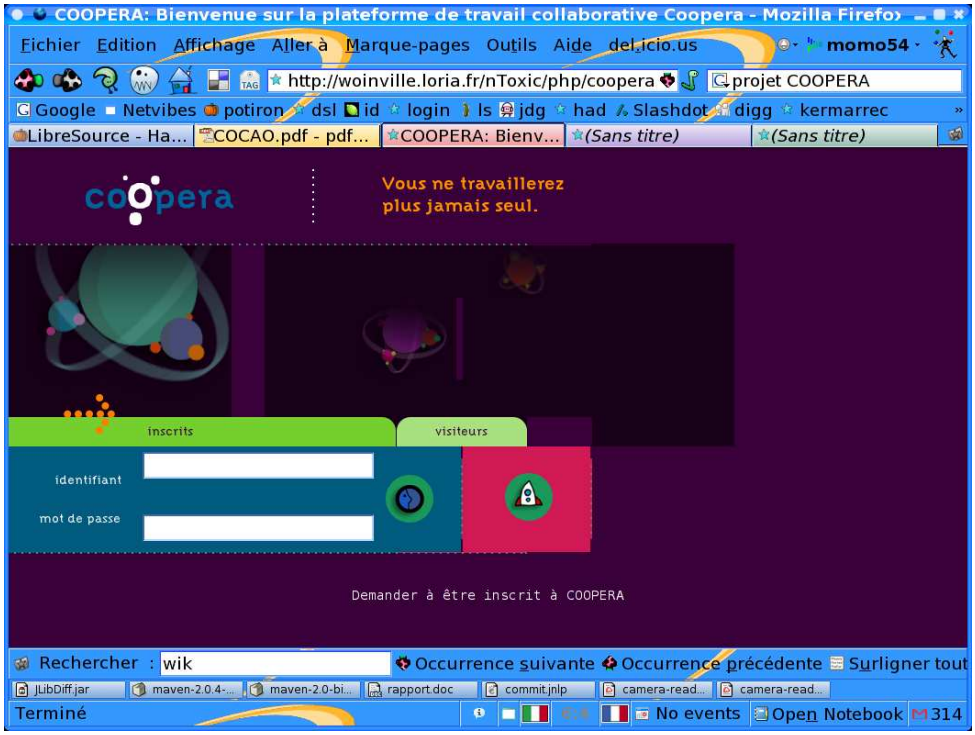


FIG. 1.12 – écran d'acceuil de Toxic/Coopera

1.6 SAMS (2002)

SAMS est le premier éditeur de groupe multi-synchrone [25]. Il contient un éditeur de CRC-cards et un éditeur XML. Il est accessible à l'adresse suivante : `http://woinville.loria.fr/sams/`. Il est le résultat de l'action de recherche VXP (voir page 8).

Les figures 1.13, 1.14, 1.15 représentent les différentes parties ou composants de l'environnement SAMS.

L'interface se décompose en plusieurs parties :

Dom Preview : Cette zone représente l'espace de travail de l'utilisateur. C'est la zone de travail où l'utilisateur peut interagir avec le programme de manière à créer et manipuler des documents XML ou des cartes CRC.

Object Cette partie représente l'état local des objets partagés, c'est-à-dire ici l'arbre XML.

Reception Queue Cette zone affiche la queue de réception. Celle-ci montre les opérations reçues des autres sites et non encore intégrées.

Log Cette partie représente le journal des opérations appliquées sur l'objet partagé. Il peut s'agir d'opérations générées localement ou reçues des autres sites.

Synchrone / Commit / Update Ces boutons permettent de choisir le mode d'interaction souhaité avec le reste du groupe. Si la case Synchrone est cochée, les opérations engendrées par l'utilisateur sont immédiatement propagées aux autres sites et les opérations reçues immédiatement intégrées. Si la case Synchrone n'est pas cochée, alors c'est le mode multi-synchrone qui est activé. Les opérations locales sont uniquement envoyées aux autres sites

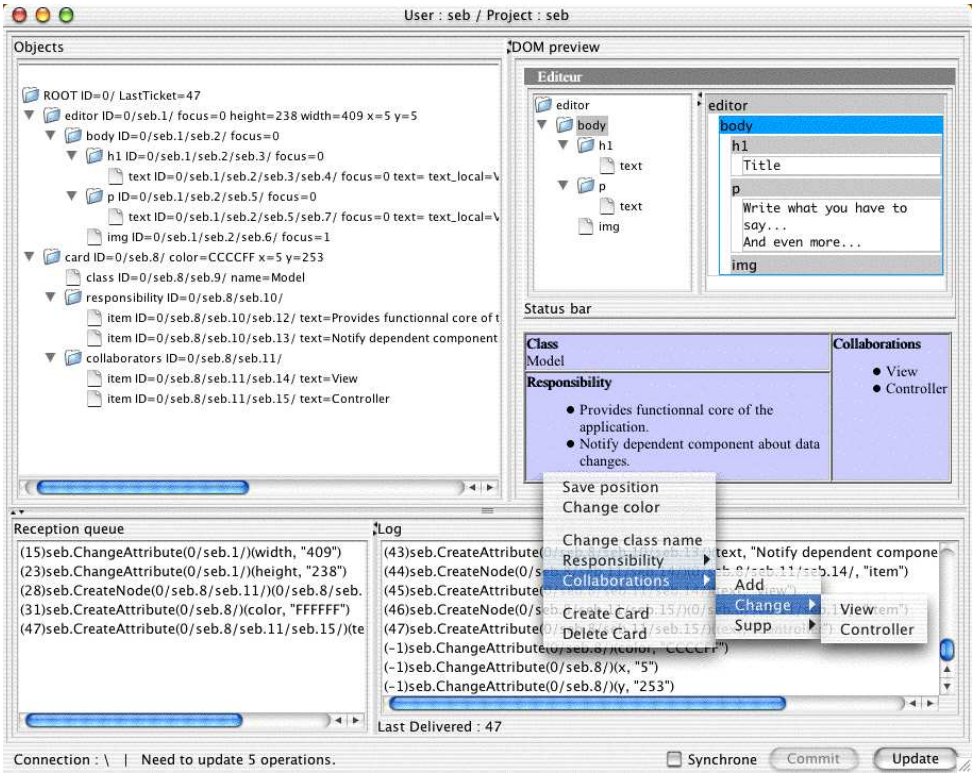


FIG. 1.13 – Interface générale de notre environnement SAMS

lorsque l'utilisateur clique sur Commit, et les opérations reçues sont intégrées lorsque l'utilisateur clique sur Update. Cependant, l'utilisateur ne peut envoyer ses opérations locales qu'après avoir donné l'ordre d'intégrer toutes les opérations reçues.

1.7 Libresource (Diffusé depuis Juin 2005)

Libresource [16] [34] est une plate-forme de travail collaboratif diffusée sous licence libre en QPL et commercialisée par la société Artenum. Libresource a reçu le prix spécial du jury 2006, de la conférence ObjectWebCon. J'ai été récompensé par la médaille du LORIA 2005 pour cette réalisation.

Libresource est une forge diffusée sous licence libre depuis juin 2005. Une forge est un environnement de développement collaboratif de logiciel. Elle fournit a des équipes de développement des services de partage de données, de communication, de coordination, de conscience de groupe et d'indexation. Le service de partage de données peut être un service FTP ou un gestionnaire de configuration. Le service de communication peut être un service de liste de diffusion ou des forums. Le service de coordination peut être un gestionnaire de tâches ou un gestionnaire de workflow. Le service de conscience de groupe peut utiliser des notifications par mél ou un bus de message. Le service d'indexation peut être une recherche par catégories ou par mot clefs.

Les forges existantes comme Sourceforge, Gforge, Savannah, Codehaus, Trac sont largement

[16] `www.libresource.org`

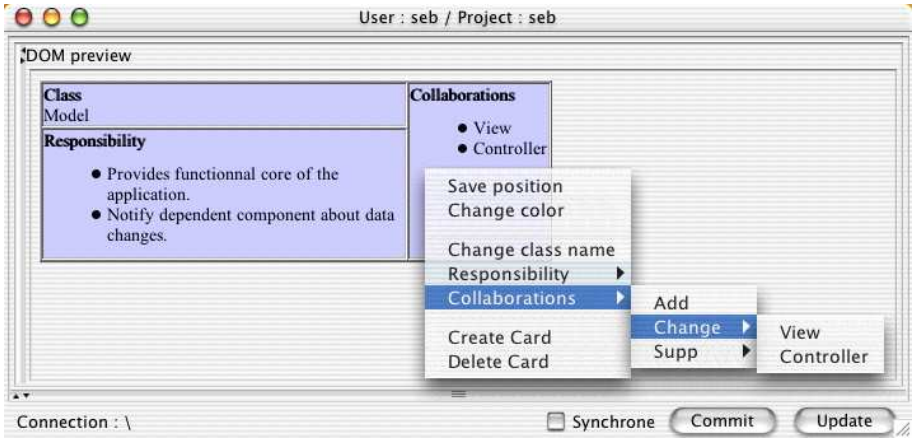


FIG. 1.14 – Interface de gestion des cartes CRC

utilisées et appréciées par la communauté des logiciels libres. En effet, une équipe de développement distribuée dans le temps et l'espace peut rapidement trouver sur ces forges les ressources nécessaire à leurs activités. Cependant, ces forges souffrent de problèmes récurrents : difficulté d'installation et de maintenance, passage à l'échelle, tolérance aux pannes, intéropérabilité des forges, pérennité du service... Ces problèmes ont des conséquences naturelles : finalement peu de sites hébergent des forges, un projet hébergé sur un site peut difficilement migrer sur un autre, un problème de sécurité ou de charge des serveurs peut provoquer l'arrêt du service. Enfin, que se passe-t-il si l'hébergeur décide simplement de ne plus fournir le service ?

Libresource a été conçu pour pouvoir être installé simplement et pour faciliter la migration d'un projet d'un serveur vers un autre. L'équipe de développement n'est plus alors dépendante du service de forge. Si l'hébergeur est défaillant, l'équipe peu migrer ses données vers un autre hébergeur ou installer son propre serveur.

La vision privilégiee pour le déploiement de Libresource est donc une fédération de serveurs plutôt qu'un gros site d'hébergement monolithique. Cela répartit les coûts de maintenance, limite les risques technologiques et politiques et favorise l'appropriation des serveurs par les communautés.

Pour arriver à ce résultat, Libresource adopte une architecture complétement différente des autres forges. Les forges existantes sont construites en intégrant des outils existants. Cette intégration n'est pas toujours facile car les outils utilisés n'ont pas été conçus pour coopérer. Libresource est basé sur une architecture à composant. Les composants de partage de données, de communication et de coordination ont été écrits pour être instanciés, composés, déplacés, exportés, notifiés, sécurisés etc...

Libresource fournit actuellement des composants pour les forums, les suivis de bogues, les pages wikis, les zones de téléchargement, la gestion de configuration, le suivi d'événements etc... Libresource est une architecture ouverte, il est facile d'ajouter un nouveau composant ou de modifier un composant existant. De manière transversale tout composant instancié dans Libresource bénéficie des services de sécurité, de notification et d'indexation.

Un serveur Libresource gère un arbre de composants. Cet arbre est trés similaire à une arborescence de répertoires et de fichiers, simplement au lieu de créer des fichiers, on crée des

FIG. 1.15 – Interface d'édition HTML

forums ou des gestionnaires de configuration. Il est possible de créer un forum comme fils d'une page wiki, ou une page wiki comme fils d'une zone de téléchargement. Toutes les combinaisons sont permises. Pour chaque composants, il faudra ensuite fixer ses droits d'accès en utilisant des ACLs et des groupes. Enfin il faudra s'abonner ou pas aux événements produits par ce nouveau composant.

Le serveur Libresource est écrit sous forme de composants J2EE. Nous utilisons le serveur d'application Jonas du consortium ObjectWeb. Les données sont stockées dans Postgres. Les événements sont stockés dans des queues JORAM. Nous bénéficions donc des avantages du serveur d'application : sécurité, passage à l'échelle, tolérance aux pannes, facilité d'installation, clustering...

Si Libresource a des atouts technologiques sur ses concurrents, Libresource dispose aussi d'un support fournit par la société Artenum [17]. La société Artenum vend un serveur Libresource à destination les entreprises (Libresource Enterprise Edition) et assure la support pour la version libre (Libresource Community Edition).

Libresource reste cependant un produit jeune, il n'est disponible que depuis quelques mois. Le retour que nous avons eu de la part de la communauté a déjà permis de faire de nombreuses corrections et nous permet de dégager les fonctionnalités que nous voulons intégrer dans les prochaines versions. La prochaine version de Libresource devrait entre autre délivrer des performances multipliées par 10, une synchronisation des documents XML et un installeur. En conclusion, Libresource est une forge innovante technologiquement, disposant d'un support pour sa pérennité et proposant un modèle déploiement différent de celle des forges existantes.

[17] www.artenum.org

1.7.1 So6

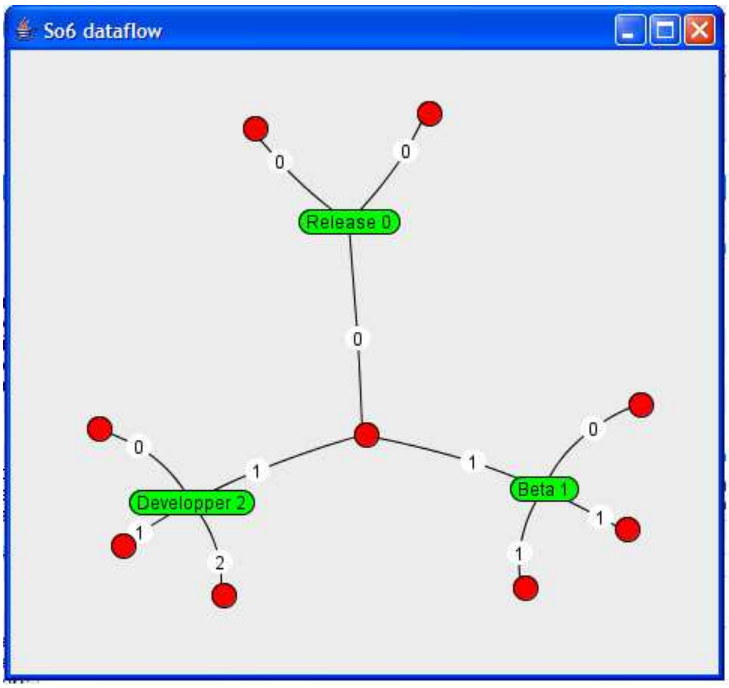


FIG. 1.16 – Réseau de synchronisation So6

So6 est le composant de gestion de configuration de Libresource. A l'origine, So6 a été développé pour montrer comment l'approche des transformées opérationnelles peut être appliquée à la synchronisation de données divergentes. Le modèle des transformées opérationnelles a été développé pour réaliser des éditeurs collaboratifs temps réels. Le souci principal était alors la réactivité de l'éditeur. Nous avons réalisé que les algorithmes utilisés n'était finalement pas trés éloignés des algorithmes de fusion utilisés dans les gestionnaires de configuration. Mieux, nous avons montré que les algorithmes de fusion utilisé dans les gestionnaires de configuration peuvent être vu comme une instance du modèle des transformées opérationnelles. Pour le prouver, nous avons réalisé So6. L'approche des transformées opérationnelles simplifie énormément l'écriture d'un tel outil et surtout nous avons pu prouver la correction des fonctions de transformation utilisées à l'aide du prouveur automatique de théorème SPIKE du projet CASSIS. Nous avons donc prouvé que les différentes répliques d'un objet partagé par So6 finissent par converger quand le système est au repos [27].

Aujourd'hui, So6 assure la convergence pour des objets partagés de type système de fichier, fichiers textes et fichiers XML. Contrairement aux outils concurrents, c'est le même algorithme qui resynchronise ces différents types de données. Un développeur So6 peut rajouter ses propres types de données. Pour cela, il doit écrire ses fonctions de transformation et prouver leur correction en utilisant l'environnement de preuve VOTE développé par l'équipe ECOO.

Contrairement aux outils concurrents, un espace de travail So6 peut être synchronisé avec plusieurs serveur So6. Un utilisateur peut donc se mettre à jour avec un serveur so6 et publier les nouvelles modifications venant de ce serveur vers un autre. Cette fonctionnalité permet de développer des réseaux de synchronisation. Ces réseaux peuvent être utilisé pour modéliser facile-

ment des procédés de développement. Par exemple, une équipe de développement travaille selon un procédé classique : développement, test, exploitation. Un serveur So6 sert le développement, un second les tests et un dernier l'exploitation (cf figure 1.16). Pour assurer la propagation des opérations selon le procédé, un espace de travail est relié à la fois au serveur de développement et au serveur de test. Pour faire passer la dernière version de développement en test, il suffit pour l'utilisateur de cet espace de travail de se mettre à jour avec le serveur de développement et publier le résultat vers le serveur de test.

Actuellement, So6 impose que le réseau de synchronisation ait une topologie en arbre. Les derniers algorithmes que nous avons développés autorise n'importe quelle topologie. La nouvelle version de So6 pourra donc s'exécuter sur un réseau P2P pur, nous obtiendrons alors une sorte de CVS pair-à-pair avec des qualités trés intéressantes : passage à l'échelle et tolérance aux pannes.

1.8 jXYDiff (Diffusé depuis mars 2006)

jXYDiff est une librairie java pour calculer les diffèrences entre deux fichiers XML. Elle est diffusée sous licence libre QPL. jXYDiff a été réalisé en coopération avec Eric Cobena sur la base de ses travaux de thèse dirigés par S. Abiteboul.

Couplée avec So6, elle permet de fusionner des fichiers XML. Actuellement, seul So6 dispose de cette fonctionnalité en exploitant les qualités naturelles de l'approche des transformées opérationelles.

1.9 Conclusions et perspectives

Comme je l'ai déjà souligné, l'équipe ECOO attache une grande importance à la validation des propositions de recherche par le prototypage et le transfert industriel.

J'ai largement participé à cette politique. Il y actuellement 3 développements en cours dans lesquels je suis impliqué :

- Le projet WOOKI est l'implantation d'un Wiki sur un réseau pair-à-pair. Il exploite nos derniers résultats sur l'édition collaborative massive en conjonction avec l'ARC RECALL.
- Le projet Graveyard est l'implantation d'un éditeur multi-synchrone en exploitant les fonctions de transformation TTF détaillées dans la partie recherche de ce document.
- Le projet européen Qualipso va permettre de donner une suite au projet Libresource. En effet, l'objectif est de concevoir une nouvelle plate-forme de collaboration en exploitant les dernières architectures et notamment les architectures orientées service et les architectures pair-à-pair.

Quatrième partie

Activités d'enseignement

Je suis Maitre de Conférences à l'Université Henri Poincaré (UHP), Nancy 1 depuis le 1er septembre 1997. Depuis 2006, je suis membre du conseil d'UFR STMIA de la faculté des sciences et techniques de l'Université Henri Poincaré, Nancy 1.

J'enseigne principalement sur les thèmes suivants en Master M1 et M2 :

programmation objet, conception objet, génie logiciel. L'expérience acquise au cours des développements logiciels accompagnant mes recherches est un atout précieux.

programmation concurrente, programmation distribuée, gestion de la réplication . Ces enseignements sont dans la ligne directe de mes recherches.

1.1 Programmation objet

J'ai créé ce cours en 1996 en coopération avec Laurent Andrey, Maître de Conférence à Nancy 2. Ce cours fut le premier cours à utiliser Java comme langage support à l'UHP. J'ai créé ce cours alors que j'étais encore ATER à l'ESIAL.

L'objectif de ce cours est d'illustrer les paradigmes objets en prenant Java comme langage support. Je montre comment sont réalisés en java les paradigmes d'encapsulation, généricité, héritage, polymorphisme, liaison dynamique... Je compare tout au long du cours la représentation en Java des paradigmes avec d'autres représentations comme C++, Eiffel, Smalltalk et Self.

Le cours est accompagné de TP. Les notes de cours sont disponibles au centre de documentation [61].

J'ai assuré ce cours devant de nombreux publics :

- Au début, Java était un langage nouveau. J'ai commencé par l'enseigner au DESS. Comme peu de cours existaient sur le sujet, j'ai aussi assuré ce cours dans le cadre de l'école "Objets et Réseaux" du CIMPA [18]. Cette formation s'est déroulée en septembre 97 à Hanoï.
- J'ai assuré ce cours à des étudiants en formation continue. Devant ce public plus agé, il a fallu adapter le niveau de discours pour obtenir un cours moins scolaire.
- Avec la démocratisation de Java, j'ai ensuite assuré ce cours devant des débutants en informatique. De nouveaux problèmes sont apparus ; vocabulaire, compréhension des mécanismes de base des langage objet. J'ai du progressivement ne plus utiliser les transparents du cours originels. Le rythme du cours était trop élevé et les résultats aux examens se dégradaient. En revenant au tableau, en amenant une machine ainsi qu'un rétro-projecteur en cours, j'ai pu largement améliorer la compréhension générale de ce cours.

1.2 Conception orientée objet

J'ai créé ce cours en 1996 en complément du cours de conception de bases de données. Ce cours fut le premier à utiliser UML comme formalisme de conception. Ce cours est destiné à des étudiants en master informatique 1ère année.

Autour d'études de cas, je montre concrêtement ce qui doit être fait à chaque étape du développement d'un logiciel ; cahier des charges, analyse, conception, codage, tests. L'intérêt

[18] Centre International de Mathématiques Pures et Appliquées

d'une telle approche est de partir du cahier des charges jusqu'aux tests. Bien sûr, je m'appuie sur les notations UML. UML est ici utilisé comme un outil de communication entre moi et les étudiants.

J'introduis déjà dans ce cours les notions d'architectures logicielles, de programmation par composant et de patrons de conception. Ces notions sont reprises plus tard dans le cours "Patrons de conception et d'architecture" (voir section 1.3).

Ce cours est certainement le plus difficile des cours dont j'ai la charge. Il ne s'agit pas seulement d'apporter des connaissances mais véritablement de convaincre les étudiants du bien-fondé des étapes de conception.

Le cours a évolué de simples présentations UML vers des mises en application concrêtes devant les étudiants des apports d'une bonne conception. Un moyen efficace de faire passer le message consiste à d'abord à écrire une application comme ils pourraient le faire. Puis à montrer les problèmes de couplage, de cohésion, de tests et d'évolution d'une telle application. Enfin, étape par étape, on introduit la notion de composants en montrant les apports d'une telle solution, en essayant de mesurer les nouvelles qualités du programme. Il faut aller jusqu'à manipuler le code devant eux en cours pour les convaincre de l'intérêt des phases de conception. Finalement, les progrés récents des environnements de développement où l'environnement de codage et de conception sont disponibles au sein du même outil rendent possible de telles expérimentations pendant les cours.

1.3 Patrons de conception et d'architecture

J'ai créé ce cours en 97. Il vient en complément du cours de conception orienté objet. Depuis le passage au LMD, ces deux cours sont d'ailleurs fusionnés en Master M1.

Le cours est basé sur les livres d'Erich Gamma et al "design patterns" pour la partie patrons de conception et sur le livre de Buschmann et al "A System of Patterns" pour la partie patrons d'architecture.

Je commence par situer l'utilisation des patrons de conception dans le cycle de vie d'un logiciel. Autour d'exemples et d'études de cas, j'illustre les différents patrons de conception individuellement mais surtout collectivement en les composant.

Enfin, j' aborde la partie plus complexe de patrons d'architecture en prenant la même démarche ; études des différents modèles d'architecture pris individuellement puis nous essayons de les composer.

Les exemples sont quasiment tous pris en utilisant les librairies Java ou l'on peut retrouver la quasi-totalité des patterns. Le fait de montrer que ces patterns sont réellement implantés au sein de librairies qu'ils utilisent permet aux étudiants de prendre conscience qu'ils tirent parti de logiciels qui ont été pensés. Ils comprennent alors que l'étude de l'architecture des logiciels qu'il utilisent est trés importante pour qu'ils puissent eux même concevoir des logiciels de qualité.

1.4 Outils pour le génie logiciel

Ce cours existait déjà en DESS. Son objectif était de montrer aux étudiants les différents outils à leur disposition pour faciliter le développement logiciels.

J'ai longtemps maintenu une page CVS à l'époque où ce logiciel était méconnu. A ma grande surprise, ces pages m'ont données une certaine notoriété. Pendant longtemps, j'ai fais des présentations au sein de laboratoire pour encourager l'utilisation. J'ai même été contacté par CODITIEL (UPS CNRS 856) pour faire une présentation sur la gestion de configurations en général et sur l'outil CVS en particulier dans le cadre d'une école thématique intitulée : "Génie Logiciel pour la Modélisation" qui s'est déroulée en avril 1997.

C'est fort logiquement que j'ai utilisé ce savoir pour dans le cadre de ce cours. CVS est en fait pour moi un outil d'édition collaborative qui fait partie de l'état de l'art de mon domaine de recherche.

L'objectif du cours est de faire connaêtre les outils de gestion de versions et de configurations, quelles sont leurs fonctionnalités, comment ils se placent dans un environnement de développement et quel est l'apport lié à l'utilisation de tels outils par rapport à des modéles de référence de génie logiciel comme le *Capability Maturity Model (CMM)*.

Le cours est découpé en deux parties : une introduction générale sur les gestionnaires de configurations qui présente l'état de l'art dans ce domaine. Vient ensuite une partie plus technique liée à des produits particuliers : RCS [19] et CVS [20]. La présentation de ces outils est suivie de séances de TP où les étudiants utilisent et configurent ces outils.

1.5 Programmation concurrente

J'ai cré ce cours en 99. Jusqu'alors, les problèmes de programmation concurrente étaient abordés comme des problèmes "système" et abordé dans le cadre de l'étude d'UNIX. J'ai été le premier à utiliser le support Multi-thread de JAVA pour aborder ces problèmes. Le cours est inspiré du livre de Doug Lea "Concurrent Programming in Java". Ces cours est destiné à des étudiants en Master 1ère année.

Nous abordons les problèmes de sûreté et de vivacité en étudiant les problèmes classiques de verrous mortels, famine, incohérence d'états...

La encore, le cours a évolué. Il était d'abord partie intégrante du cours Java. Ensuite, j'ai décidé qu'il fallait séparer ces deux parties. En effet, l'apprentissage du langage Java suivi de l'apprentissage de la programmation multi-thread constituait un obstacle majeur pour les étudiants. Malgré tout, les résultats restaient médiocre. Certains étudiants ont beaucoup de mal se représenter plusieurs flots de contrôle au sein d'une même programme.

C'est en passant à l'expérimentation lors des cours i.e. en écrivant réellement les exemples classiques de programmation concurrente pendant les cours que j'ai pu mieux faire passer mon message.

Cela est rendu possible par la simplicité de l'écriture de programmes multi-thread en Java, par la disponibilité d'environnements de développement libres tels que Eclipse ou Netbeans, et l'introduction des portables et des vidéo-projecteurs en cours.

[19] Revision Control System
[20] Concurrent Version System

1.6 Programmation distribuée

J'ai créé ce cours en 98 suite à une demande de l'IUP GEII. Il fallait faire découvrir aux étudiants les principes de bases de la programmation distribuée.

J'ai basé ce cours sur le programme de l'école d'été "Construction d'Applications Réparties" organisée par L'IMAG, l'INRIA et le LIFL. Le cours commence par aborder les problèmes généraux : passage à l'échelle, tolérance aux pannes, sécurité...

Je montre comment ces notions sont traitées dans une approche comme les RMI en JAVA, puis avec les Servlet Java. La encore, faire les manipulations devant les étudiants avant d'aller en TP s'avère trés efficace.

Je continue ensuite le cours en leur montrant les modéles d'exécutions existant : Client/serveur classique, à objet, WEB, modèles à messages, à événements, à composant, à mémoires partagées, P2P.

1.7 Réplication

J'ai créé ce cours en 2005. Ce cours est destiné à des étudiants en 2ième année de Master Informatique.

Les systèmes de réplication sont au coeur de l'informatique actuelle. Ils permettent aux serveurs de tolérer les pannes et d'augmenter leurs performances. Ils sont à la base des systèmes P2P ou la réplication massive des données permet le passage à l'échelle.

L'objectif du cours est de connaître ces systémes et leurs limites. Le cours présente les différents systèmes et algorithmes utilisés en réplication pessimiste et optimiste. L'accent est mis sur la réplication optimiste qui en prise directe avec mes activités de recherche.

1.8 Perspectives d'enseignement

Il est évident qu'un bon cours ne peut être assuré et maintenu que si il est en prise directe avec les activités de recherches :

- Mes différentes réalisations en matière de développement me donnent de l'expérience, des arguments, des cas concrêts pour mes cours de programmation et conception objet.
- Mon activité de recherche autour des systèmes collaboratifs me donne les connaissances pour enseigner les bases de la programmation concurrente et distribuée ainsi que les systèmes de réplication.

Le fait de rester actif sur ces deux fronts me permet de maintenir la qualité de ces cours. Je vois malgré tout une carence chez les étudiants que j'accueille parfois dans l'équipe. Ils ne savent pas gérer les tests d'un logiciel. J'ai pu me rendre compte de l'importance de ces tests au cours du développement de Libresource.

J'aimerai donc compléter le cours de conception avec un cours centré sur l'ingénierie du test avec les notions de test unitaires, tests d'intégration, tests fonctionnels, tests de non-régression, tests non-fonctionnels. Ce cours n'existe pas pour l'instant au département informatique de l'Université Henri Poincaré.

Cinquième partie

Bibliographie

Chapitres de livres

[1] Fahrad Daneshgar, Pradeep Ray, Fethi Rahbi, Hala Molli, Pascal Molli, Claude Godart. – « Knowledge Sharing Infrastructures for Teams within Virtual Communities ». – *In : e-Collaborations and Virtual Organizations*, Dr. Michelle Fong (réd.). IGP/Infosci/IRM Press, august 2004.

Mémoires

[2] Pascal Molli. – *Environnements de développement coopératifs.* – Thèse d'université, Université Henri Poincaré - Nancy I, 1996.

Articles dans revues avec comité de lecture

[3] Gérôme Canals, Claude Godart, François Charoy, Pascal Molli, Hala Skaf. – « COO Approach to Support Cooperation in Software Developments ». – *IEE Software Engineering 145*, 2-3 (1998), pp. 79–84.

[4] Gérôme Canals, Claude Godart, Pascal Molli, Manuel Munier. – « A Criterion to Enforce Correctness of Indirectly Cooperating Applications ». – *Information Sciences 110* (1998), pp. 279–302.

[5] Gérôme Canals, Pascal Molli, Claude Godart. – « Support for end user participation using replicated versions and group communication ». – *SIGGROUP Bull. 20*, 1 (1999), pp. 5–9.

[6] Guy Bernard, Jalel Ben-Othman, Luc Bouganim, Gérôme Canals, Sophie Chabridon, Bruno Defude, Jean Ferrié, Stéphane Gançarski, Rachid Guerraoui, Pascal Molli, Philippe Pucheral, Claudia Roncancio, Patricia Serrano-Alvarado, Patrick Valduriez. – « Mobilité et bases de données : état de l'art et perspectives ». – *TSI : Technique et science informatiques 22*, 3 (2003).

[7] Guy Bernard, Jalel Ben-Othman, Luc Bouganim, Gérôme Canals, Sophie Chabridon, Bruno Defude, Jean Ferrié, Stéphane Gançarski, Rachid Guerraoui, Pascal Molli, Philippe Pucheral, Claudia Roncancio, Patricia Serrano-Alvarado, Patrick Valduriez. – « Mobilité et bases de données : état de l'art et perspectives ». – *TSI : Technique et science informatiques 22*, 4 (2003).

[8] Guy Bernard, Jalel Ben-Othman, Luc Bouganim, Gérôme Canals, Sophie Chabridon, Bruno Defude, Jean Ferrié, Stéphane Gançarski, Rachid Guerraoui, Pascal Molli, Philippe Pucheral, Claudia Roncancio, Patricia Serrano-Alvarado, Patrick Valduriez. – « Mobile Databases : a Selection of Open Issues and Research Directions. ». – *SIGMOD Record 33*, 2 (2004), pp. 78–83.

[9] Claude Godart, Pascal Molli, Gérald Oster, Olivier Perrin, Hala Skaf-Molli, Pradeep Ray, Fethi Rabhi. – « The ToxicFarm Integrated Cooperation Framework for Virtual Teams ». – *Distributed and Parallel Databases 14* (january 2004).

[10] Abdessamad Imine, Michaël Rusinowitch, Gérald Oster, Pascal Molli. – « Formal design and verification of operational transformation algorithms for copies convergence. ». – *Theoretical Computer Science 351*, 2 (2006), pp. 167–183.

[11] Olivera Marjanovic, Hala Skaf-Molli, Pascal Molli, Claude Godart. – « Innovative Learning Designs Enabled by Process-Driven Collaborative Editing ». – *Journal of Educational Technology and Society (endorsed by IEEE Learning Technology Task Force) Accepted for publication* (2006).

Congrès internationaux avec comité de lecture

[12] Claude Godart, Pascal Molli, Gérôme Canals, Jacques Lonchamp. – « Extending Cooperative Transaction Model to Support Software Engineering Activities ». – *In : Proceedings Second International Conference on Systems Integration*, P. A. Ng, L. C. Siefert, C. V. Ramamoorthy, R. T. Yeh (réd.), IEEE Computer Society Press, pp. 184–192. – Morristown, NJ, USA, june 1992.

[13] Pascal Molli. – « Partage d'objets complexes dans un environnement de développement de logiciels-Mise en oeuvre sous PCTE ». – *In : Inforsid 92, informatique des organisations et systèmes d'information et de décision*, pp. 187–206. – Clermond-Ferrand, FRANCE, november 1992.

[14] François Charoy, Pascal Molli. – « Experimenting the PCIS Triggers on P-RooT ». – *In : Proceedings PCTE'94 Conference*, pp. 389–398. – San Francisco, USA, november 1994.

[15] Claude Godart, Gérôme Canals, François Charoy, Pascal Molli. – « An Introduction to Cooperative Software Development ». – *In : Proceedings 3rd International Conference on System Integration (ICSI 1994)*, IEEE Press. – august 1994.

[16] Gérôme Canals, François Charoy, Claude Godart, Frank Juillard, Pascal Molli, Alexandre Rossel, Hala Skaf. – « P-Root and COO : un système de gestion d'objets et des services de mise en oeuvre de procédé de développement. ». – *In : Onzièmes Journées Bases de Données Avancées*, Geneviève Jomier (réd.), INRIA, pp. 477–. – Nancy, France, september 1995.

[17] Gérôme Canals, François Charoy, Claude Godart, Pascal Molli. – « P-RooT and COO : Building a Cooperative Software Development Environment ». – *In : Proceedings 7th Conference on Software Engineering Environments*, IEEE Computer Society Press. – march 1995.

[18] Gérôme Canals, Pascal Molli, Claude Godart. – « Concurrency Control for Cooperating Software Processes ». – *In : Advanced Transaction Models and Architectures*. – Goa, India, september 1996.

[19] Claude Godart, Gérôme Canals, François Charoy, Pascal Molli, Hala Skaf. – « Designing and Implementing COO : Design Process, Architectural Style, Lessons learned ». – *In : International Conference On Software Engineering (ICSE 1996)*, IEEE Publishing Computer Society Press, pp. 342–352. – Berlin,Germany, march 1996.

[20] Gérôme Canals, Christophe Bouthier, Claude Godart, Pascal Molli. – « Tuamotu : Une infrastructure distribuée pour le support des entreprise-projets ». – *In : NOTERE'98, Montréal, Canada*, M. Kadoch R. Dssouli, P. Dini (réd.), CRIM, pp. 103–118. – Montréal, 1998.

[21] C. Godart, C. Bouthier, P. Canalda, F. Charoy, P. Molli, O. Perrin, H. Saliou, J. C. Bignon, G. Halin, O. Malcurat. – « Asynchronous coordination of virtual teams in creative applications (co-design or co-engineering) : requirements and design criteria ». – *In : ITVE '01 : Proceedings of the workshop on Information technology for virtual enterprises*, IEEE Computer Society, pp. 135–142. – Washington, DC, USA, 2001.

[22] Claude Godart, Gilles Halin, Jean-Claude Bignon, Christophe Bouthier, Pascal Malcurat, Pascal Molli. – « Implicit or Explicit Coordination of Virtual Teams in Building Design. ». – *In : Computer-Aided Architectural Design Research In Asia (CAADRIA 2001)*. – Sydney, Australia, april 2001.

[23] Pascal Molli, Hala Skaf-Molli, Claude Godart, Pradeep Ray, Rajan Shankaran, Vijay Varadharajan. – « Integrating Network Services for Virtual Teams ». – *In : International Conference on Enterprise Information Systems - ICEIS 2001*. – Setùbal, Portugal, july 2001.

[24] Pascal Molli, Hala Skaf-Molli, Gérald Oster. – « Divergence Awareness for Virtual Team through the Web ». – *In : Integrated Design and Process Technology, IDPT 2002*, Society for Desing and Process Science. – Pasadena, CA, USA, june 2002.

[25] Pascal Molli, Hala Skaf-Molli, Gérald Oster, Sébastien Jourdain. – « SAMS : Synchronous, Asynchronous, Multi-Synchronous Environments ». – *In : The Seventh International Conference on CSCW in Design.* – Rio de Janeiro, Brazil, september 2002.

[26] Abdessamad Imine, Pascal Molli, Gérald Oster, Michaël Rusinowitch. – « Proving Correctness of Transformation Functions in Real-Time Groupware ». – *In : Proceedings of the 8th European Conference on Computer-Supported Cooperative Work (ECSCW 2003)*, ACM. – Helsinki, Finland, september 2003.

[27] Pascal Molli, Gérald Oster, Hala Skaf-Molli, Abdessamad Imine. – « Using the Transformational Approach to Build a Safe and Generic Data Synchronizer ». – *In : Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003*, ACM Press, pp. 212–220. – Sanibel Island, Florida, USA, november 2003.

[28] Hala Skaf-Molli, Pascal Molli, Gérald Oster, Claude Godart, Pradeep Ray, Fethi Rabhi. – « Toxic Farm : A Cooperative Management Platform For Virtual Teams And Enterprises ». – *In : 5th International Conference on Enterprise Information Systems (ICEIS 2003).* – Angers, France, april 2003.

[29] Abdessamad Imine, Pascal Molli, Gérald Oster, Michaël Rusinowitch. – « Deductive Verification of Distributed Groupware Systems ». – *In : Tenth International Conference on Algebraic Methodology and Software Technology - AMAST 2004, Stirling, Scotland, United Kingdom*, *Lecture Notes in Computer Science*, *3116*, A04-R-187, Springer. – july 2004.

[30] Abdessamad Imine, Pascal Molli, Gérald Oster, Michaël Rusinovitch. – « Towards Synchronizing Linear Collaborative Objects with Operation Transformation ». – *In : FORTE 2005*, *LNCS*, 3731, pp. 411–427. – october 2005.

[31] Olivera Marjanovic, Hala Skaf-Molli, Pascal Molli, Fethi Rabhi, Claude Godart. – « Supporting Complex Collaborative Learning Activities : The LIBRESOURCE Approach. ». – *In : 8th International Conference on Enterprise Information Systems - ICEIS'06, Paphos, Chypre.* – may 2006.

[32] Gérald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. – « Data Consistency for P2P Collaborative Editing ». – *In : Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, November 4-8, 2006*, ACM. – 2006.

[33] Gérald Oster, Pascal Urso, Pascal Molli, 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)*, IEEE Press. – Atlanta, Georgia, USA, november 2006.

[34] Hala Skaf-Molli, Pascal Molli, Olivera Marjanovic, Claude Godart. – « LibreSource : Web Based Platform for Supporting Collaborative Activities ». – *In : 2nd IEEE International Conference on Information and Communication Technologies : from Theory to Applications, Umayyad Palace, Damascus, Syria.* – april 2006.

Colloques et workshops internationaux avec comité de lecture

[35] Gérôme Canals, François Charoy, Claude Godart, Pascal Molli. – « P-RooT and COO : Extending the PCTE OMS with New Capabilities ». – *In : Proceedings ICSE'94 Workshop on the Intersection between Databases and Software Engineering*, IEEE Press. – Sorrento, Italy, 1994.

[36] François Charoy, Pascal Molli. – « Subjectivity and Software Engineering Environment Framework ». – *In : OOPSLA'95 Workshop on Subjectivity in Object Oriented System.* – Austin, TX, USA, october 1995.

[37] Claude Godart, Gérôme Canals, François Charoy, Pascal Molli. – « About Some Relationships between Configuration Management, Software Process and Cooperative Work : the COO Environment ». – *In : Proceedings 5th Workshop on Software Configuration Management - ICSE 95 - Selected Papers*, *Lecture Notes in Computer Science*, 1005, Springer Verlag, pp. 173–178. – april 1995.

[38] Pascal Molli. – « COO-Transactions : Supporting Cooperative Work ». – *In : 7th International Workshop on Software Configuration Management (SCM7)*, *Lecture Notes in Computer Science*, *1235*, Springer-Verlag, pp. 128–141. – Boston, USA, may 1997.

[39] Gérôme Canals, Pascal Molli, Claude Godart. – « TuaMotu : Support for Telecooperative Engineering Applications using Replicated Versions ». – *In : IGROUP Workshop, Seattle*, T. Tuikka M. Divitini, B. Farshcian (réd.), *Working Papers Series*, B 56, Oulu University Press. – Oulu, Finland, 1998.

[40] Abdelmajid Bouazza, Hala Skaf-Molli, Pascal Molli. – « Coordinating Virtual Teams by Measuring Group Divergence ». – *In : Workshop on Groupware related Task Design at GROUP'99 Conference, Phoenix, Arizona, USA.* – november 1999.

[41] Claude Godart, Pascal Molli, Olivier Perrin. – « Modeling and enacting processes. Some difficulties. ». – *In : International Process Techgnology Workshop.* – september 1999.

[42] Abdelmajid Bouazza, Pascal Molli. – « Unifying coupled and uncoupled collaborative work in virtual teams ». – *In : ACM CSCW'2000 workshop on collaborative editing systems, Philadelphia, Pennsylvania, USA.* – december 2000.

[43] Pascal Molli, Hala Skaf-Molli, Christophe Bouthier. – « State Treemap : an Awareness Widget for Multi-Synchronous Groupware ». – *In : 7th International Workshop on Groupware - CRIWG'2001.* – Darmstadt, Germany, september 2001.

[44] François Charoy, Claude Godart, Pascal Molli, Gérald Oster, Marc Patten, Miguel Valdes. – « Services for Virtual Teams Hosting : ToxicFarm Introduction ». – *In : Proceedings of the 2nd International Workshop on Cooperative Internet Computing*, ACM Hong Kong. – Hong Kong, China, august 2002.

[45] Pascal Molli, Gérald Oster, Michaël Rusinowitch, Abdessamad Imine. – « Development of Transformation Functions Assisted by Theorem Prover ». – *In : The Fourth International Workshop on Collaborative Editing.* – ACM CSCW 2002, New Orleans, Louisiana, USA, november 2002.

[46] Abdessamad Imine, Pascal Molli, Gérald Oster, Pascal Urso. – « VOTE : Group Editors Analyzing Tool ». – *In : International Workshop on First-Order Theorem Proving (FTP 2003).* – Valencia, Spain, june 2003.

[47] Hala Skaf-Molli, Pascal Molli, Gérald Oster. – « Semantic Consistency for Collaborative Systems ». – *In : the Fifth International Workshop on Collaborative Editing.* – Helsinki, Finland, september 2003.

Congrès et colloques nationaux ou francophones

[48] Gérald Oster, Pascal Molli. – « Réconciliation de Données dans un Environnement Mobile ». – *In : Colloque Informatique Mobile.* – Nancy, France, december 2002.

[49] Gérald Oster, Pascal Molli, Hala Skaf-Molli, Abdessamad Imine. – « Un modéle sûr et générique pour la synchronisation de données divergentes ». – *In : Premières Journées Francophones : Mobilité et Ubiquité - UbiMob'04.* – Nice, France, june 2004.

[50] Pascal Molli Gérald Oster, Pascal Urso, Abdessamad Imine. – « Edition collaborative sur réseau pair-à-pair à large échelle ». – *In : CDUR 2005 : Journées Francophones sur la Cohérence des Données en Univers Réparti*. – CNAM, Paris, november 2005.

Publications internes et rapports de recherche

[51] Claude Godart, Gérôme Canals, Jacques Lonchamp, Pascal Molli, Jean-Claude Derniame. – « Using Software Process Modeling to Support Consistent Cooperation and Competition of Software Engineering ». – *Rapport interne*, Centre de Recherche en Informatique de Nancy, Vandoeuvre-lès-Nancy, 1991.

[52] Pascal Molli. – « Partage d'objets entre activités dans un environnement de développement de logiciels ». – *Rapport de dea*, Centre de Recherche en Informatique de Nancy (CRIN), 1991.

[53] François Charoy, Pascal Molli. – « An Experience with Subjectivity : the P-RooT Software Engineering Framework ». – *Rapport interne*, Centre de Recherche en Informatique de Nancy, Vandoeuvre-lès-Nancy, 1995.

[54] Pascal Molli, Gérôme Canals. – « A Core Transaction Protocol for Cooperative Software Engineering Activities ». – *Rapport interne*, Centre de Recherche en Informatique de Nancy, Vandoeuvre-lès-Nancy, 1995.

[55] Pascal Molli, Manuel Munier, Gérôme Canals, François Charoy, Claude Godart. – « COO-serializability : A Correctness Criterion for Cooperative Executions ». – *Rapport de recherche*, LORIA, 1997.

[56] Pascal Molli, Gérald Oster, Hala Skaf-Molli, Abdessamad Imine. – « Safe Generic Data Synchronizer ». – *Research Report A03-R-062*, LORIA – INRIA Lorraine, may 2003.

[57] Abdessamad Imine, Pascal Molli, Gérald Oster, Michaël Rusinowitch. – « Achieving Convergence with Operational Transformation in Distributed Groupware Systems ». – *Rapport de recherche RR-5188*, INRIA, may 2004.

[58] Gérald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. – « Proving correctness of transformation functions in collaborative editing systems ». – *Rapport RR-5795*, INRIA, december 2005.

[59] Gérald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. – « Real time group editors without Operational transformation ». – *Rapport RR-5580*, INRIA, may 2005.

[60] Gérald Oster, Pascal Urso, Pascal Molli, Hala Skaf-Molli, Abdessamad Imine. – « Optimistic Replication for Massive Collaborative Editing ». – *Rapport RR-5719*, INRIA, october 2005.

Divers

[61] Pascal Molli, Laurent Andrey. – « Java Lectures Notes », 1998.

Soumissions en cours d'examen

[62] Gérald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. – « Proving Correctness of Transformation Functions in Collaborative Editing Systems ». – *Journal of CSCW (Submitted)* (2006).