UNIVERSITÉ DE MONTRÉAL

CONVERGENCE ET SÉCURITÉ D'ACCÈS DANS LES SYSTÈMES D'ÉDITION COLLABORATIVE MASSIVEMENT RÉPARTIS

AUREL JOSIAS OBOUBÉ RANDOLPH
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
AOÛT 2014

© Aurel Josias Oboubé Randolph, 2014.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

CONVERGENCE ET SÉCURITÉ D'ACCÈS DANS LES SYSTÈMES D'ÉDITION COLLABORATIVE MASSIVEMENT RÉPARTIS

présentée par : RANDOLPH Aurel Josias Oboubé
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. CHAMBERLAND Steven, Ph.D., président
Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche
M. QUINTERO Alejandro, Doct., membre et directeur de recherche
M. GUÉHÉNEUC Yann-Gaël, Doct., membre
M. VILLEMAIRE Roger, Ph.D., membre

À ma famille.

À toutes celles et tous ceux qui m'ont fortifié d'une parcelle de leurs connaissances et savoirs : éducateurs, enseignants et formateurs de tous ordres.

REMERCIEMENTS

De l'idée jusqu'à cette étape d'impétration, j'ai eu l'insigne honneur et le privilège d'être soutenu, accompagné, encouragé et conseillé par plusieurs personnes. Je saisis la présente occasion pour témoigner à toutes et à tous, ma gratitude pour leurs bons offices.

Je tiens à remercier Hanifa Boucheneb et Alejandro Quintéro. Au-delà du rôle de géniteurs dans la lignée scientifique, ils n'ont ménagé aucun effort sur les plans humain et financier. Je prends acte de l'incommensurable dette de reconnaissance contractée envers eux, pour l'encadrement, l'assistance, l'aide et la confiance dont j'ai bénéficié.

J'exprime également toute ma reconnaissance à Abdessamad Imine pour sa collaboration. Nos discussions ne font que m'édifier à chaque fois. Je lui renouvelle mes remerciements, ainsi qu'à Michael Rusinowitch, pour leur accueil lors de mon séjour scientifique à l'INRIA Nancy Grand-Est (FRANCE).

Je remercie les membres de jury pour l'attention particulière accordée à mes travaux.

Je manifeste ma profonde gratitude à ma famille, dans toutes ses dimensions. L'apport quotidien dont je suis gratifié est indicible. Je ne dirai jamais assez merci. Je décerne une mention spéciale à Annick, M.-Gracia, Josias, A.-Yanis, Rita, Floris, William, Claudia, Joanita et Isidore RANDOLPH.

Je sais gré à Eugène ézin, Rock Glitho et Fabien Houéto, sans qui, je n'aurais pas commencé cette thèse. Un merci spécial à Eugène ézin pour son soutien indéfectible et tout ce qu'il fait pour moi, dès lors que nous nous sommes connus.

Je remercie les Professeurs Samuel Pierre, Steven Chamberland, Ronald Beaubrun et Chamseddine Talhi pour leurs encouragements durant tout mon parcours doctoral.

Je ne passe pas sous silence, la convivialité et l'entraide qui ont prévalu dans les laboratoires VeriForm et LARIM. Je remercie plus particulièrement : Moustapha Bandé, Luis Cobo-Campos, Grégory Charlot, Marième Diallo, éric Fafolahan, Reza Gholami, Valérie-D. Justafort, Adjarath Lémamou, Moussa Ouédraogo, Ronald Jean-Julien, Germine Séide.

J'apprécie grandement le diligent concours et les encouragements de Francklin éyélom. Je le remercie infiniment, ainsi que les collègues de l'Académie Moan, en leurs grades et qualités.

Je remercie très sincèment toutes les personnes non distinctement indiquées, dont j'ai bénéficié du soutien durant cette aventure scientifique et humaine.

RÉSUMÉ

Parmi les défis des systèmes d'édition collaborative figure la cohérence des objets partagés. Dans la perspective d'une édition cohérente, le système doit garantir la convergence. Pour assurer la cohérence des objets partagés, la littérature propose plusieurs solutions. Les différentes approches majeures proposées sont : l'approche des types de données commutatives répliquées (CRDT) et l'approche de la transformée opérationnelle (OT). L'approche CRDT considère des opérations commutatives qui peuvent être exécutées dans un ordre différent. L'une des difficultés auxquelles CRDT se bute réside en la commutativité des opérations. Toutes les opérations d'édition doivent être commutatives afin d'être exécutées dans un ordre quelconque. L'approche de la transformée opérationnelle quant à elle propose une transformation des opérations distantes reçues par rapport aux opérations qui lui sont concurrentes ; même si elles sont déjà exécutées. Pour effectuer les transformations, l'approche OT utilise un algorithme de transformation inclusive (IT). Dans la littérature, plusieurs travaux ont prouvé que les principaux algorithmes de transformation inclusive proposés ne satisfont pas le critère de convergence.

Outre la cohérence, la sécurisation des interactions est un autre défi des systèmes d'édition collaborative. Le contrôle d'accès est l'un des modèles de politiques de sécurité applicable dans ce cadre. Il s'agit d'autoriser ou d'interdire l'édition à certains utilisateurs. Le contrôle d'accès doit être uniformément déployé pour éviter de compromettre la cohérence des opérations d'édition. Une opération d'édition valide sur un site doit l'être partout. Une opération refusée sur un site doit être refusée partout. Dans le contexte étudié, le protocole de sécurité est fiable s'il préserve la cohérence du système. Fournir cette preuve de fiabilité est une tâche ardue. Le nombre de cas à examiner est infini. De plus, pour une vérification automatique, le défaut de ressources survient si des techniques appropriées ne sont pas exploitées.

Dans cette thèse, nous nous intéressons aux défis que constituent la convergence et le contrôle d'accès dans les systèmes d'édition collaborative répartis. Nous considérons un objet textuel à structure linéaire qui est massivement édité dans une architecture répartie. L'approche de gestion de cohérence utilisée est la transformée opérationnelle. Ainsi, chaque utilisateur a sa copie locale du document partagé. Les opérations générées sur un site sont aussitôt diffusées aux autres utilisateurs. Elles peuvent être exécutées dans un ordre quelconque. Les types d'opérations d'édition sont : l'insertion d'un caractère et la suppression de caractère. Nous intégrons également un protocole de contrôle d'accès à l'édition collaborative. Notre thèse se présente sous la forme de trois articles scientifiques, chacun traitant d'une problématique bien spécifique.

Dans le premier article, nous abordons la problématique de la convergence. Nous avons adopté une démarche en plusieurs étapes. Une exploration a été initialement faite afin de vérifier s'il est possible d'avoir une fonction IT convergente. En utilisant la méthode de la synthèse de contrôleur et les automates de jeu, nos investigations ont révélé qu'aucune IT basée uniquement sur le caractère et la position ne peut garantir une convergence. L'identification des causes de divergence a permis d'amorcer la résolution du « problème de synthèse de contrôleur ». Ainsi, un troisième paramètre a été ajouté aux opérations d'insertion. Il permet de manipuler le nombre de caractères supprimés avant la position d'insertion indiquée. Une fonction de détermination de la valeur de ce paramètre a été proposée. Une fonction IT a été par la suite proposée, en tenant compte des propositions précédentes. En utilisant la vérification sur modèle (*model-checking*), la preuve a été apportée que notre IT garantit bien la convergence.

Le deuxième article propose l'intégration d'un protocole de sécurité optimiste. L'article aborde la problématique de la fiabilité du protocole dans un espace d'états infini. Il est déployé au dessus de protocole de synchronisation du système d'édition collaborative. Nous faisons l'hypothèse que le système vérifie la propriété de cohérence en l'absence du contrôle d'accès. Pour affronter les difficultés relatives à la preuve de fiabilité, l'approche du *model-checking* symbolique a été préférée. Le *model-checking* borné a été utilisé avec l'outil *Alloy*. L'exploration faite pour des instances dont la taille maximale est de treize « signatures », a permis de conclure la préservation de la cohérence par le protocole de contrôle d'accès. Notons que ces instances ne sont pas massives mais la combinatoire résultante n'est pas négligeable.

Le troisième article aborde la problématique de réduction de système. Des investigations ont été menées afin d'avoir un modèle fini équivalent au système d'édition collaborative, au regard de la propriété de cohérence. Le modèle abstrait proposé comporte trois sites coopératifs, dont l'un est administrateur. Ce modèle à espace d'états fini étant prouvé équivalent par rapport à la propriété de cohérence, au système à espace d'états infini, il a servi de cadre pour la vérification automatique. En utilisant l'outil *Uppaal* et le formalisme d'automate, nous avons prouvé par *model-checking* que le modèle abstrait préserve la cohérence. Par conséquent, le protocole de contrôle d'accès préserve la cohérence de système d'édition collaborative.

Nos travaux comportent quelques limitations liées à leur portée. Nous avons manipulé des objets textuels à structure linéaire sur lesquels ne sont appliquées que des opérations d'insertion et de suppression de caractères. De plus, la gestion des droits d'accès est basée sur un modèle mono-administrateur. La performance du protocole de contrôle d'accès n'a pas non plus été prise en compte. Les travaux auraient sans doute plus d'envergure s'ils couvraient plusieurs types d'objets, plusieurs types d'opérations d'édition, plusieurs administrateurs et

une étude de performance. Nos futures travaux pourraient être consacrés à l'élargissement de la portée de la présente thèse.

ABSTRACT

The consistency of the shared documents is one of the most important challenges in colla–borative editing systems. To achieve consistency, a solution must ensure the convergence criteria. Several solutions are proposed in litterature to achieve consistency of the shared documents. The major approaches are: commutative replicated data type (CRDT) and operational transformation (OT). CRDT considers some commutative operations which could be executed in different order. The main difficulty of CRDT is to compute commutative operations. OT approach proposes to transform remote operations against their concurrent operations, even if they are already executed. An inclusive transformation function is used to compute the transformations. In the litterature, several works show that the main inclusive transformation (IT) functions proposed do not ensure convergence.

Besides consistency, security of the edition is another challenge in distributed collaborative systems. Access control is a model of security policy that could be used. It consists of granting or revoking editing authorizations for users. Access control must be uniformly deployed to not compromise the consistency of the system. A valid editing operation at one site must be valid at all other sites. As the same time, an invalid operation at one site, must be invalid everywhere. In the current context, the security protocol is reliable if it preserves the consistency of the system. Produce the proof of reliability is difficult. It requires examining infinite number of cases. In addition, with automatic verification, ressources become insufficient if appropriate techniques are not used.

This thesis is interested in consistency and access control challenges in distributed collaborative editing systems. It considers a textual object with a linear structure that is massively edited in a distributed architecture. OT is used to manage consistency. Each user has a local copy of the shared document. Locally-generated operations are immediately broadcast to other users. Operations could be executed in any order. Their types are inserting and deleting characters. To ensure security, collaborative edition is combined with an access control protocol. The thesis consists of three scientific articles. Each of them deals with a specific problem.

In the first article, we adress the problem of consistency and proceed in several steps. Initially, we explore the existence of convergent IT functions of OT, which ensure data consistency. Using the controller synthesis approach and game automata, we conclude that there is no IT function, based only on character and position as parameters of insert and delete operations, which ensure data consistency. The investigation of the causes of divergence led to solve the controller synthesis problem. Thus, a new parameter is added to the insert op-

eration signature. It handles the number of characters deleted before the inserting position. The function needed to compute the value of this parameter is provided. Finally, based on these contributions, we propose an IT function and show that it ensures convergence. The proof is achieved by a symbolic model-checking emulated using the tool *Uppaal*.

The second article adresses the reliability of security protocol in an infinite state space. An optimist access control protocol is considered to be deployed over any correct synchronization protocol. The symbolic model-checking approach is choosen to deal with the proof of reliabi–lity. For this purpose, bounded model-checking is used with the tool *Alloy*. Exploration made with instances whose maximum size is thirteen allow to conclude the preservation of consistency by the access control protocol. These instances are not massive but the resulting combinatorial is important.

The third article adresses the problem of system reduction. In this article, we investigate a finite model equivalent to a distributed collaborative editing system, with regard to consistency. The abstract model proposed consists of three cooperative sites including the administrator. This finite state model is proved by model-checking to preserve consistency. Consequently, the access control protocol preserves consistency of any correct distributed collaborative editing system. The model-checking techniques exploits *Uppaal* tool and automata.

Our work has several limitations. We consider textual objects with linear structure. These objects are edited by applying some operations which are inserting and deleting characters. In addition, the management of access rights is based on one-administrator model. The performance study of the access control protocol is not done. The work would probably be more extensive if it covered several types of objects, several types of editing operations, many administrators and the performance study. Our future work could be devoted to the widening of the scope of this thesis.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT viii

TABLE DES MATIÈRES x

LISTE DES TABLEAUX xiv

LISTE DES FIGURES xv

LISTE DES ANNEXES xvi

LISTE DES SIGLES ET ABRÉVIATIONS xvii

CHAPITRE 1 INTRODUCTION 1
- 1.1 Définitions et concepts de base 2
  - 1.1.1 Édition collaborative 3
    - Édition asynchrone et synchrone 3
    - Notion de réplication 3
    - Système réparti, exécution répartie, causalité et concurrence 4
    - État stable et notion de convergence 6
  - 1.1.2 Politique de sécurité et contrôle d'accès 6
  - 1.1.3 Vérification sur modèle et synthèse de contrôleur 6
- 1.2 Éléments de la problématique 8
- 1.3 Objectifs de recherche 12
- 1.4 Esquisse méthodologique 12
- 1.5 Principales contributions de la thèse et leur originalité 13
- 1.6 Plan de la thèse 15

CHAPITRE 2 REVUE DE LITTÉRATURE 17
- 2.1 Analyse sommaire du problème 17
- 2.2 Approches de gestion de la cohérence dans les éditeurs collaboratifs 18

2.2.1 Approche multi-versions 19
2.2.2 Approche sérialisation / Résolution de conflits 20
2.2.3 Approche des types de données commutatives répliquées 21
Algorithme WOOT 22
Algorithmes Logoot et Logoot-Undo 23
TreeDoc 23
Autres applications de l'approche CRDT 24
2.2.4 Approche de la transformée opérationnelle 25
Condition de cohérence 25
Algorithme de Ellis 27
Algorithme de Ressel 27
Algorithme de Sun 28
Algorithme de Suleiman 28
Algorithme d'Imine 29
Algorithme SO6 30
Algorithme MOT2 30
Algorithmes SDT et SDTO 31
Fonctions TTF 32
2.3 Contrôle d'accès dans les systèmes collaboratifs 33
2.4 Conclusion 34

CHAPITRE 3 ARTICLE 1 : On Synthesizing a Consistent Operational Transformation Approach 36
3.1 Introduction 37
3.2 Operational Transformation Approach 40
3.2.1 Background 40
3.2.2 Decentralized integration procedures 42
3.2.3 Inclusive transformation functions 43
Ellis's algorithm 43
Ressel's algorithm 43
Sun's algorithm 44
Suleiman's algorithm 44
Imine's algorithm 44
3.2.4 Consistency criteria 45
3.2.5 Avoiding Property $TP2$ 46
3.3 Controller synthesis of consistent IT function 48

3.3.1 Do there exist IT functions which satisfy TP1? 49
3.3.2 Do there exist IT functions which satisfy TP1 and TP2? 50
3.4 A consistent IT function 54
3.4.1 Extending the insert signature with an extra parameter 54
3.4.2 Our IT function 55
3.4.3 Relationships between positions and the extra parameters 56
3.4.4 Proof of consistency 59
3.4.5 Comparison 61
3.5 Conclusion 63

CHAPITRE 4 ARTICLE 2 : Specification and Verification using Alloy of Optimistic Access Control for Distributed Collaborative Editors 65
4.1 Introduction 66
4.2 Optimistic Access Control Protocol for DCE 67
4.2.1 Generation of Local Cooperative Requests 68
4.2.2 Reception of Remote Cooperative Requests 68
4.2.3 Generation of Administrative Operations 69
4.2.4 Reception of Remote Administrative Requests 69
4.2.5 Verification Issues 70
4.3 Specification and Verification 71
4.3.1 Alloy 72
4.3.2 Formal Specification of ACP 72
4.3.3 Specification of Consistency Property 76
4.4 Related Work 78
4.5 Conclusion 79

CHAPITRE 5 ARTICLE 3 : On Consistency Preservation with Optimistic Access Control for Distributed Collaborative Editors 80
5.1 Introduction 81
5.2 Optimistic Access Control Protocol for DCE 82
5.2.1 Generation of Local Cooperative Requests 83
5.2.2 Reception of Remote Cooperative Requests 84
5.2.3 Generation of Administrative Operations 84
5.2.4 Reception of Remote Administrative Requests 84
5.2.5 Verification Issues 86
5.3 Definitions 88
5.4 A finite abstract model preserving consistency property of ACP 92

5.4.1 Generation and Reception of a Cooperative Operation 92
5.4.2 Execution Process 93
5.4.3 Analyzis of the Execution Process 96
5.5 Model Checking of the Abstract Model 99
5.5.1 Uppaal 99
5.5.2 Description of the system 100
Sites 100
Evolution of Cooperative Operation 100
Evolution of Adminstrative Request 102
ACP 102
5.5.3 Consistency Property and Verification of the system 103
5.6 Related Work 105
5.7 Conclusion 106

CHAPITRE 6 DISCUSSION GÉNÉRALE 107
6.1 Synthèse des travaux 107
6.2 Méthodologie 108
6.3 Analyse des résultats 110

CHAPITRE 7 CONCLUSION 112
7.1 Sommaire des contributions de la thèse 112
7.2 Limitations des travaux 113
7.3 Indication des travaux futurs 114

RÉFÉRENCES 116

ANNEXES 124

LISTE DES TABLEAUX

Table 3.1 IT functions supplied by *Uppaal-Tiga* for TP1 and classical signatures of update operations 51

Table 3.2 Transformation cases for $IT(o_1, o_2)$ and $IT(o_2, o_1)$ 58

Table 3.3 Computing $ND(S \bullet [o_1; o_{21}], p)$ and $ND(S \bullet [o_2; o_{12}], p)$ 58

Table 3.4 Condition of equivalence of $[o_1; o_{21}]$ and $[o_2; o_{12}]$ 61

Table 3.5 Complexity comparison. 62

LISTE DES FIGURES

Figure 2.1 Violation de la propriété TP2 par l'algorithme de Suleiman. 29
Figure 2.2 Violation de la propriété TP2 par SDT. 32
Figure 3.1 Integration without transformation. 40
Figure 3.2 Integration with transformation. 40
Figure 3.3 MV approach 41
Figure 3.4 SRC approach 41
Figure 3.5 CRDT approach 41
Figure 3.6 Violation of TP1 for Ellis's IT. 46
Figure 3.7 Violation of TP1 for Sun's IT. 46
Figure 3.8 Violation of TP1 for Suleiman's IT. 47
Figure 3.9 Violation of TP2 for Ressel's IT (in case $u_2 < u_3$). 47
Figure 3.10 Violation of TP2 for Imine's IT. 47
Figure 3.11 Synthesize an IT for TP1 49
Figure 3.12 Synthesize a consistent IT 49
Figure 3.13 Scenario 1 53
Figure 3.14 Scenario 2 53
Figure 3.15 Applying our IT to scenario 1 53
Figure 3.16 Applying our IT to scenario 2 53
Figure 3.17 The proposed IT function 55
Figure 3.18 Automaton used to verify TP1 and TP2 60
Figure 3.19 View and Model states (Figure taken from [Oster *et al.* (2006a)]). . . . 62
Figure 4.1 Flow of collaboration messages 68
Figure 4.2 Divergence caused by introducing administrative operations 70
Figure 5.1 Flow of collaboration messages 83
Figure 5.2 Processing of a cooperative operation at generation time. 85
Figure 5.3 Processing of a received cooperative operation. 85
Figure 5.4 Processing of received administrative request. 87
Figure 5.5 Divergence caused by introducing administrative operations 87
Figure 5.6 Execution process at any site. 97
Figure 5.7 The model of cooperative operation. 101
Figure 5.8 The model of administrative request. 102
Figure 5.9 The model of ACP. 104
Figure 5.10 Automaton devoted to choose the owner of the cooperative operation. . 104

LISTE DES ANNEXES

ANNEXE A : DÉFINITIONS COMPLÉMENTAIRES 124
ANNEXE B : ALGORITHMES DE TRANSFORMATIONS INCLUSIVES 127

LISTE DES SIGLES ET ABRÉVIATIONS

ACP	*Access Control Protocol*
ADM	*Administrator*
AMM	Modèle de Matrice d'Accès (*Access Matrix Model*)
ANSI	*American National Standards Institute*
ASSERT	*Assertion*
BDD	*Binary Decision Diagram*
C-TMAC	Modèle de contrôle d'accès basé conjointement sur les informations de contexte et les équipes (*Context-Based Team-Based Access Control*)
CAC	Modèle de contrôle d'accès sensible au contexte (*Context-Aware Access Control*)
CCI	Critère de cohérence regroupant la préservation de la causalité, la convergence et la préservation de l'intention (Causality preservation, Convergence and Intention preservation)
CES	*Collaborative Editing System*
CP	*Coordination Protocol*
CP	Problème de contrôle, *Control Problem*
CRDT	Type de données commutatitives répliquées (*Commutative Replicated Data Type*)
CSM	*Causality, single-operation effect, and multi-operation effects relation preservation*
CSP	Problème de la synthèse de contrôleurs, *Controller Synthesis Problem*
CTL	*Computational Tree Logic*
CVS	*Concurrent Version System*
DBM	*Difference Bound Matrice*
DCE	*Distributed Collaborative Editors*
DEL	*Deleting Operation*
DDP	*Data-dependency Precedence Preservation*
DTD	Définition de Type de Document (*Document Type Definition*)
ET	Transformation exclusive (*exclusion transformation*)
FACT	*Fact*
FUN	*Function*
GST-RBAC	*Generalized Spatio-Temporal Role Based Access Control*
IEEE	Institut des Ingénieurs électriciens et électroniciens

INS	*Inserting Operation*
IT	Transformation Inclusive *Inclusive Transformation*
LNCS	*Lecture Notes in Compture Science*
MV	Multi-Versions
NIST	*National Institute of Standards and Technology*
NOP	*No Operation*
OT	Transformée Opérationelle (*Operational Transformation*)
PPS	Sequences Partiellement Persistantes (*Partial Persistent Sequences*)
PRED	*Predicates*
RBAC	Modèle de contrôle d'accès basé sur les rôles (*Role-Based Access Control*)
SAC	Modèle de contrôle d'accès spatial (*Spatial Access Control*)
SAT	*Satisfaibility*
SDT	Transformation par différences d'états (*State Difference Transformation*)
SDTO	Transformation optimisée par différences d'états (*State Difference Transformation Optimized*)
SIG	*Signature*
SMV	*Symbolic Model Verifier*
SRC	*Serialization-Resolution of Conflicts*
SVN	Subversion
TBAC	Modèle de contrôle d'accès basé sur les tâches (*Task-Based Access Control*)
TMAC	Modèle de contrôle d'accès basé sur les équipes (*Team-Based Access Control*)
TTF	*Tombstone Transformation Functions*
XML	*eXtensible Markup Language*

CHAPITRE 1

INTRODUCTION

Le développement des outils et réseaux de communication accroît les interactions entre les êtres humains, équipements et logiciels. Il est donc possible d'interagir avec quelqu'un ou un objet comme s'il se trouve à portée de main ou de voix. L'impression sensorielle est quasi-réelle. Un outil de collaboration, identifié aussi par système collaboratif, est un exemple de cadre d'interactions. C'est un logiciel qui permet à un groupe d'utilisateurs de collaborer dans le cadre d'un projet, afin de répondre à un besoin. Ces utilisateurs partagent un objet commun que chacun manipule en fonction de ses prérogatives. Il peut s'agir par exemple de groupes de chercheurs qui rédigent ensemble un article scientifique. On peut observer la même situation auprès des graphistes qui modifient des illustrations ou des productions visuelles. Peuvent également faire usage de cet outil des ingénieurs du son qui réalisent du mixage de son.

Un éditeur collaboratif est un exemple de système collaboratif. Il permet de manipuler un document textuel selon plusieurs granularités telles les caractères, lignes, paragraphes. Ainsi, des actions comme la lecture, l'écriture, la suppression, l'insertion, la modification sont exécutées par les utilisateurs. GoogleDrive [1], Etherpad [2], Framapad [3], XWiki [4] sont des éditeurs collaboratifs. Dans sa forme la plus simple, la collaboration est synonyme de travail coopératif. Plus précisément, chaque utilisateur doit s'occuper d'une tâche distincte. Le travail final est l'assemblage des travaux réalisés individuellement par chaque utilisateur. Par exemple, chaque membre d'une équipe de projet rédige séparément un ou plusieurs chapitres d'un rapport de projet. Chaque utilisateur peut également être affecté à une fonction distincte tout en manipulant une même partie de l'objet partagé avec d'autres utilisateurs. à l'instar d'un utilisateur qui écrit la version initiale d'un texte tandis qu'il reviendra à un autre utilisateur de la corriger ou de la relire. Les utilisateurs font de la pure collaboration quand chacun peut accéder à n'importe quelle partie de l'objet partagé et l'éditer, c'est-à-dire lire, supprimer des extraits, en modifier ou en rajouter. En plus de leur permettre de conjuguer leurs efforts, les systèmes collaboratifs dispensent les utilisateurs d'un regroupement dans un même lieu physique pendant leurs interactions. Un réseau informatique sert de cadre à ces interactions.

Néanmoins, en dépit de leur performance et la variété de leurs usages, ces systèmes sont

1. `https://drive.google.com`
2. `http://etherpad.org/`
3. `http://framapad.org`
4. `http://www.xwiki.com/`

soumis à plusieurs contraintes. La réactivité locale en est un exemple. En effet, même si le système autorise des interactions simultanées, chaque utilisateur doit avoir l'impression d'être seul à l'exploiter. Cette contrainte est déterminante pour la convivialité du système. L'autonomie des utilisateurs est également l'une des contraintes à satisfaire. Dans ce cas, les utilisateurs doivent être en mesure de manipuler indépendamment les objets partagés. L'autonomie induit une exigence de disponibilité des objets partagés. Pour certains systèmes, le nombre d'utilisateurs pouvant faire les manipulations représente une contrainte importante. Quant à la cohérence, elle incarne l'une des contraintes fondamentales. Le système doit garantir que tous les utilisateurs ont une vue commune des objets partagés. Par ailleurs, il est primordial que le système soit sécuritaire afin de ne laisser collaborer seulement des utilisateurs autorisés, mais aussi que chaque utilisateur agisse dans la limite des rôles qui lui sont dévolus. Une attention particulière doit être accordée à toutes ces contraintes pendant la conception d'un tel système. Les systèmes existant étudiés ne satisfont pas toutes les contraintes. Nous nous proposons donc d'axer nos recherches sur la conception d'un système d'édition collaborative. Ce système doit admettre la simultanéité des interactions, l'autonomie et la collaboration à grande échelle. également, le système doit offrir minimalement une édition cohérente et sécurisée.

Ainsi, dans le présent chapitre, nous commençons par une brève explication de quelques concepts de base que nous estimons nécessaires à la compréhension de cette thèse. Ensuite, nous exposons le cadre théorique en abordant successivement les éléments de la problématique, les objectifs de recherche et la méthodologie adoptée. Par la suite, nous présentons nos principales contributions ainsi que leur originalité. Enfin, nous terminons le chapitre avec un plan de notre thèse.

1.1 Définitions et concepts de base

Dans cette section, nous introduisons quelques concepts liés à une édition collaborative répartie sécurisée et aux preuves formelles. Ces concepts déterminent le socle et sont utilisés tout au long de la thèse. à cet effet, nous commençons par clarifier la notion de synchronisme. Par la suite, nous abordons les concepts de réplication, de système réparti, de causalité et de convergence. Nous définissons aussi la sécurité d'accès dans un système d'édition collaborative. Enfin, nous examinons la vérification sur modèle et l'approche de la synthèse de contrôleur.

1.1.1 Édition collaborative

Édition asynchrone et synchrone

Le modèle d'interaction dans une édition collaborative peut être asynchrone ou synchrone. Il fait référence à la simultanéité des interactions entre les utilisateurs du système. Si les utilisateurs peuvent collaborer en éditant au même moment des objets partagés ; il s'agit dans ce cas d'une collaboration synchrone. Si par conte, les objets doivent être édités à des moments différents, la collaboration est dite asynchrone. Dans ce cas, il n'existe aucun mécanisme qui oblige les utilisateurs à éditer les objets au même moment. Cependant, il faut noter que certains systèmes de collaboration asynchrone tolèrent la collaboration synchrone. Ainsi, bien que les moments d'édition puissent être différents, le fait qu'ils coïncident n'est pas exclu.

Notion de réplication

Les éditeurs collaboratifs constituent une classe de systèmes distribués dans laquelle les utilisateurs sont géographiquement répartis et interagissent en manipulant des objets partagés tels que textes, images, graphiques, etc. [Imine (2010)]. Un réseau informatique est potentiellement exploité lors des manipulations. Dans un système d'édition centralisée, l'objet édité est hébergé sur un serveur central. Une telle édition nécessite une connexion permanente au serveur central. La vulnérabilité aux pannes de l'architecture centralisée peut avoir un impact sur la collaboration. Par exemple, une panne du serveur central ou un problème d'accès à celui-ci rend indisponible l'objet pour son édition.

Pour améliorer la disponibilité des données dans le système, chaque site peut détenir une copie de l'objet partagé. Il s'agit d'une réplication de l'objet partagé. Les copies locales de l'objet partagé sont aussi appelées « répliques ». Cette réplication est surtout mise en œuvre au début d'une collaboration ou dès qu'un nouvel utilisateur rejoint un groupe existant. Le but est de permettre au groupe d'utilisateurs ou de sites répartis dans l'espace géographique, de manipuler parallèlement des répliques. La manipulation d'une réplique se réfère précisément à sa lecture ou aux opérations d'écriture, de suppression ou de modification effectuées par le site concerné. Chaque réplique doit refléter les opérations générées sur tous les sites composant le système. En d'autres termes, une mise à jour réalisée localement par l'une des répliques du fait de la génération d'une opération par l'utilisateur doit être reproduite chez les autres utilisateurs. Pour ce faire, la mise à jour est propagée aux autres sites. Le mécanisme de distribution d'opérations localement générées aux sites distants est appelé « réplication d'opérations ». Dans la pratique, il existe deux scénarii de réplication d'opérations : pessimiste et optimiste.

La réplication pessimiste suppose bien sûr l'utilisation de plusieurs copies mais donne l'illusion aux utilisateurs de l'existence d'une copie unique [Bernstein *et al.* (1987)]. Les lectures sont faites sur les copies de manière indépendante tandis qu'une opération d'écriture doit être exécutée sur toutes les copies de manière atomique [Imine (2006)]. L'écriture est faite sur toutes les copies simultanément, sans être interrompue jusqu'à son terme. Ceci nécessite la mise en œuvre des techniques de synchronisation par un site central et une connexion permanente à ce site central.

Dans le cas de la réplication optimiste, chaque site a sa copie mais dispose d'une autonomie [Saito et Shapiro (2005)]. En vertu de cette autonomie, il peut la manipuler (lecture, écriture) à volonté indépendamment des autres sites. Les opérations générées et exécutées sur un site doivent également être envoyées aux autres sites pour être exécutées. Cet état des choses induit l'indépendance des mises à jour car les opérations ne peuvent être ni exécutées dans le même ordre, ni sur les mêmes copies [Imine (2010)].

Système réparti, exécution répartie, causalité et concurrence

Un système réparti s'appelle aussi système distribué. Il constitue un ensemble de processus (ou machines, ordinateurs, tablettes, sites) autonomes et distants qui œuvrent dans un but commun, en s'échangeant des données à travers un réseau informatique. L'autonomie des processus dans un système réparti vient du fait qu'ils ne partagent ni mémoire ni horloge commune. Il est important ici de faire le lien entre un système réparti et un système parallèle. Un système parallèle encore appelé machine parallèle est composé de machines qui permettent le traitement de plusieurs opérations ou tâches en parallèle. Il met en œuvre principalement deux modes de communication : le passage de messages et la mémoire partagée. Un système réparti est donc un système parallèle à la différence que les machines qui le composent sont autonomes, les distances entre elles sont plus importantes et elles nécessitent l'utilisation d'un réseau informatique pour les échanges. Dans un système réparti, les processus sont caractérisés par une exécution en concurrence et une interaction par passage de messages. D'après Lamport [Lamport (1978)], (i) l'envoi d'un message m produit un événement d'émission $send(m)$; (ii) la réception d'un message m produit un événement de réception $receive(m)$; et enfin, (iii) un événement interne est produit par l'exécution d'instructions n'induisant ni envoi ni réception de message. Faisons une analogie avec un système réparti d'édition collaborative. Il utilise un modèle de communication par messages. Chaque site qui participe à la collaboration est considéré comme un processus. La génération d'une opération et l'exécution locale d'une opération générée ou reçue sont des événements internes. L'envoi et la réception d'une opération d'édition sont considérés comme événements d'émission et de réception, respectivement.

L'histoire locale d'un processus est définie comme la séquence de tous les événements qu'il a produits. L'union de toutes les histoires locales, munie d'une relation d'ordre (voir Annexe A.) particulière sur les événements constitue une exécution répartie. Dans un modèle à communication par messages, une exécution répartie se caractérise par la production d'événements par les divers processus. La relation d'ordre implicite sur les événements d'une exécution répartie est la préséance causale. Elle permet de définir une relation de type « e_1 est arrivé avant e_2 » sans avoir recours à la synchronisation d'horloges, e_1 et e_2 étant deux événements. Elle est notée $e_1 \rightarrow e_2$ et se lit « e_1 précède e_2 ». Formellement, la préséance causale se définit par trois relations : *Program order* (Expression 1.1), *Receive-from* (Expression 1.2) et transitivité (Expression 1.3).

- La relation *Program order* indique que deux événements e_{i1} et e_{i2} produits par un même processus P_i sont totalement ordonnés. Ainsi donc, soit « e_{i1} précède e_{i2} », soit « e_{i2} précède e_{i1} ». En considérant l'opérateur logique $\oplus$ qui permet d'exprimer l'alternative (ou exclusif), nous avons :

$$\forall e_{i1}, e_{i2} \in P_i, (e_{i1} \rightarrow e_{i2}) \oplus (e_{i2} \rightarrow e_{i1}) \tag{1.1}$$

- La relation *Receive-from* indique que l'envoi d'un message $send(m)$ précède la réception du message $receive(m)$.

$$send(m) \rightarrow receive(m) \tag{1.2}$$

- La relation de transitivité exprime le fait que si l'événement e_1 précède l'événement e_2 et que l'événement e_2 précède l'événement e_3, alors e_1 précède e_3.

$$((e_1 \rightarrow e_2) \wedge (e_2 \rightarrow e_3)) \Rightarrow (e_1 \rightarrow e_3) \tag{1.3}$$

Quand la préséance causale ne peut être établie entre deux événements e_1 et e_2, c'est-à-dire que l'un ne précède causalement l'autre et vice-versa, on dit qu'il y a indépendance causale entre ces deux événements. Ils sont alors dits concurrents et notés $e_1 \,||\, e_2$. Formellement la notion de concurrence entre ces deux événements se définit tel qu'indiqué dans l'expression mathématique (1.4).

$$(e_1 \,||\, e_2) \Leftrightarrow (\neg(e_1 \rightarrow e_2) \wedge \neg(e_2 \rightarrow e_1)) \tag{1.4}$$

État stable et notion de convergence

Lors d'une édition, au fil de l'écoulement du temps donc de l'évolution de l'horloge locale, le système d'édition collaborative connaît une succession d'événements de génération, d'envoi, de réception et d'exécution d'opérations d'édition. La réalisation de ces événements a pour effet de modifier localement non seulement l'histoire et le document partagé, mais aussi toutes autres structures de données (queues, files d'attente, variables, etc.) que manipule le système. Il est donc dans une dynamique de changement que nous qualifions de changement d'états. Un changement d'état survient dès lors qu'un événement se produit. Le système d'édition collaborative est dit dans un état stable, quand toutes les opérations générées sur les différents sites sont exécutées sur tous les sites. Dans un tel état, il est possible d'évaluer la qualité des différentes répliques qui existent sur les sites grâce à la notion de convergence. Ainsi, lorsque le système est dans un état stable, les copies du document partagé convergent si et seulement si elles sont identiques sur tous les sites. La notion de convergence suppose donc que l'état du système dépend de l'ensemble des opérations exécutées et non de la séquence d'exécution distincte sur chaque site [Ressel *et al.* (1996)].

1.1.2 Politique de sécurité et contrôle d'accès

Parmi les états d'un système, certains sont désirables tandis que d'autres sont potentiellement non désirables. Les états désirables sont également appelés états autorisés. La définition des états autorisés d'un système constitue ses objectifs de sécurité. La manière de satisfaire les objectifs est la stratégie de sécurité. Les solutions techniques utilisées pour satisfaire ces objectifs sont les mécanismes de sécurité. Une politique de sécurité se définit alors comme une description simple des objectifs de sécurité d'un système, appuyée d'une stratégie de haut niveau pour les satisfaire. Un modèle de politiques de sécurité est une définition abstraite de la façon de définir une politique. Parmi les modèles de politiques de sécurité, nous avons le contrôle d'accès. Il vise à permettre à une entité autorisée d'avoir accès à une certaine ressource et interdire l'accès à une autre, conformément aux objectifs de sécurité. Il permet de déterminer les droits et privilèges courants d'une entité. En général, ceux-ci sont déterminés à partir de l'identité de l'entité et de la politique de sécurité. Pour une édition collaborative répartie, il s'agit d'autoriser ou non des utilisateurs ou des sites à exécuter des opérations d'édition sur un document partagé.

1.1.3 Vérification sur modèle et synthèse de contrôleur

La conception d'un système requiert des preuves de fiabilité. Pour cette raison, la validation du bon fonctionnement par des tests et techniques formelles sont nécessaires. Alors que

les tests explorent juste des états jugées critiques du système, la vérification formelle couvre tous ses états possibles. L'approche classique de vérification des systèmes est la vérification par modèle (*model-checking*). Celle-ci considère un système complet, évoluant sans influence extérieure et prenant en compte l'environnement à contrôler et son contrôleur. Pour vérifier formellement le système, il faut le modéliser ainsi que les propriétés qu'il doit satisfaire. Par la suite, il faut apporter la preuve que le système modélisé possède bien les propriétés attendues en procédant à une vérification effective. Pour toute propriété non vérifiée par le système, un contre-exemple est généré et sert à corriger le système. Ce processus est itératif. Il est répété jusqu'à l'obtention d'un système répondant aux propriétés attendues. L'approche de la synthèse de contrôleur introduite par Wonham et Ramadge (1988) évite les itérations. Elle propose de modéliser le système à contrôler et de calculer ou de synthétiser un contrôleur s'il en existe, de sorte que la propriété à vérifier (appelée objectif de contrôle) soit satisfaite. On part donc d'un système ouvert qu'on essaie de fermer en lui ajoutant en parallèle un contrôleur afin que le système complet satisfasse la propriété voulue. De façon pratique, le comportement du système à contrôler est forcé de manière à ce qu'il satisfasse la propriété indiquée. Pour y parvenir, la démarche consiste à résoudre le « problème de contrôle » (*Control Problem*, CP) puis en cas de succès, résoudre le « problème de la synthèse de contrôleur » (*Controller Synthesis Problem*, CSP). Le problème de contrôle pour un système ouvert S se définit formellement comme suit : *étant donné un système S, et une propriété φ, existe-t-il un contrôleur C tel que, le système fermé obtenu en mettant en parallèle S et C satisfait la propriété φ* ? Elle peut être formellement réécrite comme ci-après.

$$CP : S \,||\, C \models \varphi? \tag{1.5}$$

Le problème de synthèse de contrôleur, est posé après le CP. Il se formule comme suit : *Si la réponse au problème de contrôle est oui, peut-on construire un tel contrôleur C* ? Elle peut être réécrite comme ci-après.

$$CSP : CP,\ peut-on\ construire\ C\ tel\ que\ S \,||\, C \models \varphi? \tag{1.6}$$

Pour répondre au CP, il est nécessaire de restreindre la classe des modèles dans laquelle le contrôleur est recherché. Cette restriction est faite en choisissant un formalisme tel que automates finis, automates temporisés, réseaux de pétri. L'approche de la synthèse de contrôleur a pour but de restreindre le comportement du système à un comportement admis par le contrôleur, afin d'assurer les objectifs de contrôle. Cette restriction se réalise par le biais du contrôleur. Celui-ci observe le système et lui interdit certaines évolutions. Il force ainsi le système à rester dans l'ensemble des comportements prescrits pour atteindre un objectif donné.

Généralement, le défi consiste à construire un contrôleur à moindre coût et assez permissif.

Après avoir défini ces quelques concepts de base, nous nous intéressons aux problèmes relatifs à une édition collaborative sécurisée dans la quelle la convergence des répliques doit être assurée.

1.2 Éléments de la problématique

Les systèmes d'édition collaborative répartis se caractérisent par des échanges d'opérations entre sites, une forte concurrence et une coordination décentralisée. Dans ce contexte, ils doivent garantir la réactivité locale, le passage à l'échelle et la cohérence [Imine (2008)]. Les échanges d'opérations entre sites constituent une caractéristique fondamentale pour une collaboration en temps réel [Ahmed et Shirmohammadi (2006)]. Cependant l'exécution des opérations dans un ordre différent, sur des copies différentes a pour conséquence une divergence de ces copies [Tlili *et al.* (2008), Tlili *et al.* (2010)].

La contrainte de convergence est plus forte selon le contexte applicatif considéré. Par exemple, un système collaboratif d'édition d'images dont les opérations sont réduites à celles qui modifient les valeurs des pixels, présente une exigence de convergence moindre qu'un système collaboratif d'édition textuelle. En effet, pour deux valeurs différentes d'un même pixel, si ces deux valeurs sont relativement proches l'une de l'autre ou de celles des pixels du voisinage, il est difficile pour l'oeil humain de faire la différence entre elles. Il y a une perception de convergence qui résulte de l'imperfection de la vision. La contrainte de convergence n'est pas si forte car il n'est point besoin d'avoir des valeurs égales de pixels, mais un taux, seuil ou marge de convergence acceptable par l'oeil humain. Il n'en est pas ainsi pour un document textuel. Le fait d'avoir un mot ou un caractère à la place d'un autre peut produire un autre effet et soulever des problèmes de sémantique. Pour un objet textuel édité de manière collaborative, la sémantique des phrases ou des mots ajoute donc une complexité.

L'approche de la transformée opérationnelle (OT) [Ellis et Gibbs (1989)] représente le principal cadre proposé pour résoudre la divergence [Molli *et al.* (2003a), Tlili *et al.* (2010)]. Dans ce cadre, chaque opération exécutée localement est aussitôt diffusée aux autres sites pour être exécutée [Saito et Shapiro (2005), Oster *et al.* (2006b)]. Les répliques sont alors temporairement divergentes, jusqu'à l'exécution des opérations concernées. En pratique, chaque site procède à la transformation des opérations reçues des autres sites par rapport aux opérations locales avant de les exécuter [Li et Li (2004), Imine (2010)]. Les propriétés sémantiques des opérations sont exploitées pour réaliser ces transformations d'opérations [Molli *et al.* (2003a), Sun et Ellis (1998), Li et Li (2008b)]. Les transformations ont pour finalité la construction de l'histoire de chaque copie de l'objet partagé. Il ne s'agit pas d'obtenir des

histoires identiques pour chaque site mais des histoires équivalentes (voir Annexe A) conduisant à un même état final. Pour prendre en compte une opération distante o, l'histoire locale h est réorganisée en deux sous-histoires hh et hc, représentant respectivement la séquence des opérations qui précèdent causalement o et les opérations concurrentes à o. En considérant la sémantique de l'objet partagé, l'opération o est transformée par rapport à hc puis exécutée [Imine (2006)]. Les traitements relatifs aux opérations sont réalisés par deux composants [Boucheneb et Imine (2009)]. Le premier, dénommé algorithme d'intégration, s'occupe de la diffusion, la réception et l'exécution des opérations. Il a également pour rôle, de déterminer les opérations concurrentes déjà exécutées (hc), qui doivent être considérées lors d'une transformation ; ainsi que l'ordre d'application des transformations. Le deuxième composant fait la transformation d'une opération reçue par rapport à une opération concurrente déjà exécutée. Il s'agit de l'algorithme de transformation inclusive (IT). Les opérations reçues étant transformées vis-à-vis des opérations concurrentes déjà exécutées, avant d'être exécutées à leur tour, il n'est plus nécessaire de respecter l'ordre dans lequel les opérations sont exécutées sur leurs sites d'origine. Il suffit juste de détecter et gérer la concurrence des opérations. Le défi pour l'algorithme d'intégration est donc celui de l'utilisation d'un mécanisme approprié de détection de la concurrence, tandis que celui de l'algorithme de transformation inclusive est une transformation cohérente. La cohérence suppose le respect du principe de causalité et la convergence. Les ITs recensés dans la littérature modifient les paramètres des opérations en guise de transformation mais aussi afin de les rendre exécutables dans un ordre quelconque [Preguiça *et al.* (2009)].

Pour respecter le principe de la causalité, la plupart des algorithmes basés sur une approche OT utilisent les méthodes d'ordonnancement classiques : estampille, vecteur d'horloge et vecteur d'état. Les algorithmes qui utilisent les vecteurs d'horloge, ou leurs variantes pour résoudre la concurrence, associent à chaque site un vecteur d'horloge dont la taille varie en fonction du nombre de sites. L'utilisation du vecteur d'horloge impose la connaissance préalable du nombre de sites, qui ne peut ni être réduit, ni croître indéfiniment dans le temps. Dans un contexte de passage à l'échelle, le nombre de sites connectés change continuellement au rythme des connexions et déconnexions. Il est difficile de maintenir un vecteur d'horloge dans ces conditions. De plus, à grande échelle, la difficulté de la représentation de l'horloge peut constituer un handicap. En somme, les algorithmes basés sur l'approche de la transformée opérationnelle utilisant les vecteurs d'horloge souffrent de faiblesses liées à l'autonomie des sites, la topologie variable et la grande échelle [Tlili *et al.* (2008), Bakhshi et Gurov (2007), Tlili *et al.* (2010), Akbarinia *et al.* (2007)].

Dans l'approche OT, l'édition collaborative est convergente si l'algorithme de transformation inclusive utilisé satisfait deux propriétés TP1 et TP2 [Ressel *et al.* (1996)]. On parle

de respect du critère de convergence. La propriété TP1 stipule que pour deux opérations concurrentes o_1 et o_2 définies sur le même état, l'histoire obtenue en considérant o_1, suivi de la transformation de o_2 par rapport à o_1 est équivalente à l'histoire obtenue en considérant o_2, suivi de la transformation de o_1 par rapport à o_2. En d'autres termes, l'exécution de o_1 suivie de celle de o_2 conduit au même état que l'exécution de o_2 suivi de l'exécution de o_1. La propriété TP2 quant à elle indique que la transformation par rapport à une séquence d'opérations concurrentes déjà transformées du résultat de la transformation d'une opération par rapport à toute opération concurrente considérée dans ladite séquence ne dépend pas de l'ordre dans lequel ces opérations concurrentes ont été transformées dans la séquence.

$$TP1 : [o_1; IT(o_2, o_1)] \equiv [o_2; IT(o_1, o_2)] \tag{1.7}$$

$$TP2 : IT^*(o_3, [o_1; IT(o_2, o_1)]) = IT^*(o_3, [o_2; IT(o_1, o_2)]), \tag{1.8}$$

avec IT^* l'extension de IT telle que définie à l'annexe A.
L'analyse des deux propriétés montre que la propriété TP1 définit une identité d'état alors que la propriété TP2 définit une identité d'opérations. L'identité d'état peut être satisfaite conceptuellement. Il s'agit d'ailleurs la seule propriété à vérifier dans une édition centralisée car elle permet de préserver les intentions des utilisateurs.

Cependant, sur le plan conceptuel, il n'est pas évident de trouver un algorithme de transformation satisfaisant l'identité d'opérations [Imine (2006)], encore moins dans un contexte asynchrone. D'ailleurs, les algorithmes traditionnels de réplication optimiste [Molli *et al.* (2003a), Sun *et al.* (1998), Ressel *et al.* (1996), Suleiman *et al.* (1997), Ellis et Gibbs (1989), Imine *et al.* (2003), Suleiman *et al.* (1998), Sun et Ellis (1998), Vidot *et al.* (2000)] sont pour la plupart des algorithmes synchrones. Leur utilisation dans un contexte asynchrone leur impose la satisfaction de la propriété TP2. La littérature a révélé qu'aucun de ces algorithmes n'a pu satisfaire la propriété TP2 avec les objets ayant une structure linéaire [Imine (2006), Boucheneb et Imine (2009)].

Par ailleurs, la sécurité des données partagées est d'une importance capitale dans un système collaboratif. Cette dernière peut être intégrée dans le processus de conception de système collaboratif temps-réel ou être étudiée séparément. Dans tous les cas, les exigences de sécurité doivent s'aligner sur les enjeux du système d'édition considéré. à cet effet, le défi de sécurisation d'une édition collaborative est de concilier les deux objectifs de convergence et de sécurité. Ceci requiert inéluctablement un équilibre entre les contraintes de collaboration et le contrôle d'accès. L'édition collaborative vise à rendre disponibles les documents partagés à tous les membres participants alors que le contrôle d'accès vise à restreindre cette disponibilité à certains membres. La gestion de ces deux objectifs contradictoires fait face à

des exigences structurelles et opérationnelles non négligeables. La topologie de la collaboration étant variable car les membres peuvent joindre ou quitter le groupe collaboratif à leur guise, il faut permettre une gestion dynamique des changements de droits d'accès, d'où une généricité et une flexibilité du modèle de contrôle d'accès.

Cependant, le contrôle d'accès avec des changements dynamiques des droits d'accès ne doit pas nuire à la réactivité locale, au temps de réponse et à la gestion des répliques. Cet impact signifierait une dégradation des performances du système. De plus, la création paradoxale de trous de sécurité est un risque patent selon le modèle de contrôle d'accès retenu. Il pourrait s'agir de la non application uniforme de la politique de contrôle d'accès sur tous les sites ; avec en prime, l'autorisation de certaines opérations qui ne devraient pas l'être. Ou encore, du refus de certaines opérations qui devraient être autorisées sur certains sites. Une telle situation serait également source de divergence des répliques de documents partagés.

à l'origine, la politique de sécurité doit donc être formulée de sorte que les stratégies à mettre en place soient facilement applicables et les mécanismes faciles d'utilisation. Mais au delà de la généricité, la flexibilité, l'aspect dynamique, le maintien des indicateurs de performance à des seuils acceptables, la conception d'un modèle de contrôle d'accès qui tient compte des spécificités d'une édition collaborative répartie doit offrir une spécification de haut niveau des droits d'accès [Tolone *et al.* (2005)]. En plus, le modèle de contrôle d'accès doit également être strictement examiné en vue de fournir la preuve de sa fiabilité. Fournir une preuve de fiabilité pour un modèle de sécurité est une tâche difficile [Imine *et al.* (2009), Jayaraman *et al.* (2013)]. Le contexte complexe de l'édition collaborative répartie en fait tout un défi dont témoigne le nombre de cas à observer. Un système d'édition collaborative massivement réparti a un espace d'états potentiellement infini. Il serait pratique de trouver un système qui lui est équivalent en termes de fiabilité, mais dont l'espace d'états est fini. La preuve de fiabilité sera alors apportée en considérant le modèle fini.

De tout ce qui précède, nous convenons que dans un contexte d'édition collaborative répartie qui considère un document à structure linéaire, la poursuite des objectifs de convergence forte dans une approche OT et de contrôle d'accès est un défi multidimensionnel. Aux préoccupations de détection et de gestion de la concurrence s'ajoutent celles d'un algorithme de transformation inclusive qui doit garantir la cohérence, de l'atteinte d'un équilibre entre la disponibilité du document partagé et la gestion des droits d'accès, sans compromettre la cohérence de l'édition. La preuve de la fiabilité du contrôle d'accès doit couvrir aussi bien l'application uniforme de la politique que la préservation de cette cohérence. Cette thèse est une tentative de résolution de certaines dimensions de ce défi.

1.3 Objectifs de recherche

L'objectif général de cette thèse est de proposer des mécanismes de réplication sécurisés pour les systèmes d'édition collaborative massivement répartis. Ils sont basés sur une approche OT. Le but est, d'une part, d'améliorer l'« expérience utilisateur » dans le cadre du travail collaboratif et, d'autre part, de faciliter le développement de nouveaux environnements collaboratifs et de nouvelles applications intéressantes qui prennent en compte l'autonomie, le facteur d'échelle et l'instabilité topologique. Plus précisément, il s'agit :

1. D'investiguer l'existence de mécanismes de réplication optimiste convergents ;
2. De concevoir un algorithme de transformation inclusive (IT) devant garantir la convergence dans un contexte réparti de réplication optimiste ;
3. De prouver formellement que l'algorithme de transformation inclusive proposé conduit bien à une édition convergente ;
4. D'intégrer un modèle de contrôle d'accès adapté aux spécificités du contexte d'édition collaborative considéré ;
5. De prouver formellement que le modèle de contrôle d'accès préserve la cohérence d'un système d'édition collaborative réparti à grande échelle, si ce dernier garantit initialement la cohérence.

Nous nous intéressons en particulier aux systèmes d'édition collaborative répartis qui manipulent un objet textuel à structure linéaire.

1.4 Esquisse méthodologique

Pour atteindre les objectifs de recherche, nous utilisons dans un premier temps l'approche du problème de contrôle pour investiguer formellement l'existence de mécanismes de réplication optimiste convergents. à cet effet, le mécanisme de réplication basé sur la transformée opérationnelle est assimilé à un contrôleur. Nous nous basons sur la théorie des jeux et les automates de jeux pour concevoir des modèles décrivant les comportements attendus d'un système d'édition collaborative. Ensuite la propriété de convergence est spécifiée en tant qu'objectif de contrôle. Nous procédons à la vérification effective, en prenant en compte les questions de sûreté et d'accessibilité.

Une fois le problème du contrôleur résolu, nous essayons de résoudre celui de la synthèse de contrôleur. Nous cherchons donc à synthétiser des stratégies gagnantes. Il s'agit ensuite d'exploiter ces stratégies pour formuler les spécifications auxquelles doit répondre le contrôleur recherché afin que l'objectif de convergence soit atteint. Partant des spécifications, un algo-

rithme de transformation inclusive est conçu. Nous nous assurons ensuite que le contrôleur ainsi obtenu (IT) réponde aux besoins en faisant une vérification formelle.

En considérant un protocole de synchronisation qui assure initialement la convergence, nous lui intégrons sous la forme d'une couche supérieure, un protocole de contrôle d'accès. Dans un premier temps, ce protocole est spécifié grâce à un formalisme de description de comportements. Sa préservation de la convergence est vérifiée de façon bornée en exploitant un formalisme de description des requis. Par la suite, nous investiguons un modèle abstrait d'espace d'états fini qui nous permet de prouver la préservation de la propriété de convergence quelle que soit la taille du système.

1.5 Principales contributions de la thèse et leur originalité

Les principales contributions de cette thèse portent sur deux volets : (i) la proposition d'une approche de transformées opérationnelles qui assure la cohérence dans les éditeurs collaboratifs répartis et (ii) l'intégration d'un protocole de contrôle d'accès qui préserve la cohérence d'un système d'édition collaborative répartie. Ces deux principales contributions facilitent la résolution de deux problématiques que sont : la convergence et la sécurité dans les systèmes d'édition collaborative répartis. Elles favorisent ainsi l'amélioration du travail collaboratif, en général, et de l'édition collaborative à grande échelle, en particulier. Les contributions peuvent être détaillées de la manière suivante :

1. **La preuve par model-checking que les approches OT basées sur les signatures classiques des opérations d'édition ne peuvent pas assurer la convergence**. Plusieurs travaux [Boucheneb *et al.* (2010), Boucheneb et Imine (2009), Imine *et al.* (2006), Imine (2006)] ont démontré que les approches OT proposées dans la littérature n'assurent pas la convergence dans les éditeurs collaboratifs répartis. Partant de ce constat, nous nous sommes posés la question de savoir s'il est possible de concevoir des systèmes d'édition collaborative répartis, basés sur la réplication optimiste et qui assurent la convergence des documents édités. En considérant l'édition d'un document textuel à structure linéaire pour laquelle les seuls types d'opérations autorisés sont l'insertion et la suppression de caractères, nous avons alors apporté la preuve formelle qu'aucune approche OT basée sur les signatures classiques de ces deux types d'opérations ne peut assurer la convergence. Ce résultat très important car toute piste de recherche basée uniquement sur les signatures classiques est désormais prouvée vouée à l'échec. Outre l'impact de cette contribution, son originalité vient également de l'approche utilisée. En effet, le questionnement sur l'existence de solution au verrou de la convergence a été transformé en un problème de contrôle dont le but est de rechercher

l'existence ou non d'un contrôleur qui pourrait gouverner un système spécifique. Nous n'avons trouvé dans la littérature, aucun travail appliquant le problème de contrôle aux éditeurs collaboratifs répartis.

2. **La conception d'une approche OT en utilisant la technique de synthèse de contrôleur**. Nous avons proposé une fonction de transformation des répliques d'opérations. La fonction admet un paramètre supplémentaire pour l'insertion, en plus des paramètres classiques que sont le caractère à insérer et la position dans laquelle il sera inséré. Le nouveau paramètre permet de spécifier le nombre de caractères supprimés avant la position d'insertion indiquée dans la signature. Cette contribution est appuyée par la proposition d'un mécanisme de détermination automatique de la valeur du nouveau paramètre. L'originalité de cette contribution réside également dans l'approche de résolution utilisée pour aborder l'étude de la convergence. Il s'agit de l'approche de la synthèse de contrôleur. En effet, à la suite de la résolution du problème de contrôle, qui a révélé l'inexistence de fonction de transformation pour les signatures classiques de l'insertion et de la suppression, la synthèse de contrôleur a été utilisée pour identifier les causes de la divergence, d'une part, et proposer la fonction de transformation, d'autre part. L'identification des causes de la divergence a permis d'avoir les balises nécessaires à la formulation des spécifications qui ont conduit à la conception d'un mécanisme qui assure la convergence. De surcroît, à notre connaissance, une telle approche n'a pas été exploitée jusqu'alors dans le cadre des éditeurs collaboratifs, sauf dans notre cas.

3. **Preuve par model-checking que l'approche proposée assure la convergence**. Pour consolider l'approche OT proposée pour la convergence des répliques, nous avons formellement prouvé son exactitude. La vérification symbolique sur modèle basée sur la technique des matrices de bornes (DBM, *Difference Bound Matrices*) a été utilisée, en combinaison avec un modèle d'automate. L'utilisation des matrices de bornes pour exprimer les contraintes sur les positions des caractères et traduire les transformations des répliques d'opérations est très originale dans la démarche de preuve. Elle permet de manipuler symboliquement les paramètres, à domaines infinis, des opérations d'édition.

4. **Intégration d'une politique de contrôle d'accès aux systèmes d'édition collaborative et preuve par *model-checking* de la préservation de la convergence**. Dans le but de sécuriser une édition collaborative répartie, un protocole de contrôle d'accès a été proposé [Imine *et al.* (2009)]. Nous avons dans un premier temps élaboré une première spécification du protocole. Elle se base sur la logique de premier ordre et la théorie des ensembles. Nous avons utilisé une analyse bornée pour prouver que jusqu'à concurrence d'un certain seuil, le protocole préserve la cohérence du protocole de synchronisation au dessus duquel il est déployé. Traitant d'un système à espace

d'états infini, du fait du nombre arbitraire d'utilisateurs, d'opérations coopératives par utilisateur et d'opérations administratives, l'outil d'analyse n'a pas permis de couvrir tout l'espace d'états. Pour y remédier nous avons proposé un modèle abstrait à espace d'états fini et prouvé que ce dernier préserve, sur tout l'espace d'états, la cohérence du protocole de synchronisation. Nous en avons déduit que le modèle abstrait est équivalent au système à espace d'états infini. Ce résultat est très important car il permet de manipuler un modèle fini dans lequel un nombre réduit d'utilisateurs sera utilisé. Ainsi n'importe quel outil pourrait être utilisé pour vérifier le protocole. De plus la démarche utilisée pour obtenir le modèle abstrait à espace d'états fini peut servir de cadre d'analyse à d'autres problèmes partageant des problématiques similaires de réduction.

1.6 Plan de la thèse

Le reste de cette thèse est organisé comme suit. Le chapitre 2 expose une revue de littérature sur les approches de gestion de la cohérence dans les éditeurs collaboratifs et les protocoles de contrôle d'accès proposés pour les éditeurs collaboratifs.

Les trois chapitres suivants sont des articles qui traitent aussi bien de la cohérence d'une édition collaborative répartie, que de la préservation de cette propriété de cohérence par un protocole de contrôle d'accès.

Le premier article, dont le titre est *On Synthesizing a Consistent Operational Transformation Approach*, étudie dans un premier temps la possibilité d'avoir une édition cohérente à partir des signatures classiques des opérations de suppression et d'insertion. Par la suite, en se basant sur l'approche de la synthèse de contrôleur, il propose une fonction de transformation des opérations d'édition avec une nouvelle signature pour l'opération d'insertion, puis prouve formellement que son utilisation assure une édition concurrente cohérente. Cet article a été accepté pour publication dans la revue *IEEE Transactions on Computer* et fait l'objet du chapitre 3.

Le chapitre 4 intitulé *Specification and Verification using Alloy of Optimistic Access Control for Distributed Collaborative Editor* est un article publié dans *Formal Methods for Industrial Critical Systems, Lecture Notes in Compture Science* (LNCS). Il propose une spécification d'un protocole flexible de contrôle d'accès pour les éditeurs collaboratifs. L'analyse de cette spécification a été faite dans le but de s'assurer que le protocole préserve la cohérence de tout protocole de synchronisation au-dessus duquel il serait déployé, si ce dernier garantit indépendamment la cohérence. La vérification faite lors de l'analyse est bornée.

Le système collaboratif étant à espace d'états infini, la vérification doit se baser sur un modèle fini qui est équivalent au système collaboratif, par rapport à la propriété de cohérence.

Cette préoccupation a été abordée par le chapitre 5 dans l'article intitulé *On Consistency Preservation with Optimistic Access Control for Distributed Collaborative Editors*. Ce dernier est soumis à la revue *ACM Transactions on Information and System Security*. Il propose un modèle abstrait à espace d'états fini qui est utilisé pour analyser la préservation de la propriété de cohérence par le protocole de contrôle d'accès.

Dans le chapitre 6, nous présentons une discussion générale sur les aspects méthodologiques et les résultats obtenus lors des travaux présentés dans cette thèse. Pour finir, une conclusion est proposée au chapitre 7. Elle présente une synthèse des travaux qui ont été effectués. En outre, la conclusion expose les limitations des travaux et une esquisse des travaux de recherche futurs.

CHAPITRE 2

REVUE DE LITTÉRATURE

Dans ce chapitre, nous passons en revue les travaux majeurs qui ont été réalisés dans le cadre d'une édition collaborative. Plus précisément, la gestion de la cohérence et le contrôle de l'accès seront couverts.

2.1 Analyse sommaire du problème

Un système d'édition collaborative se rapporte à un groupe d'utilisateurs. Outre les états correspondant à la création ou à la suppression d'un tel groupe, le système évolue de manière dynamique. Cette dynamique se traduit aussi bien en taille (nombre d'utilisateurs), qu'en volume (nombre d'opérations d'édition exécutées). En effet, une fois que le groupe collaboratif existe, les opérations qu'un utilisateur peut exécuter par rapport à ce groupe sont les suivantes : (i) joindre le groupe ; (ii) partciper à l'animation du groupe, (iii) quitter le groupe. Dès lors que l'utilisateur joint le groupe, il récupère une copie du document (celle d'un utilisateur avec son état). Cette action finalise son processus d'admission. Une telle copie du document partagé est une réplication tel qu'expliqué dans la section 1.1.1. La disposition de cette copie marque également le début de la collaboration. En effet, l'utilisateur peut désormais participer à l'animation du groupe en générant des opérations d'édition, en les exécutant localement et en les répliquant au profit des autres participants. En outre, il participe à l'animation du groupe en exploitant les opérations reçues des autres participants. Les répliques d'opérations coopératives s'appliquent donc sur des répliques d'objets partagés, ce qui maintient et développe le système. Il évolue au rythme de la génération et de l'exécution des opérations.

Au-delà de la performance et de la disponibilité des répliques d'objets partagés, l'un des principaux objectifs poursuivis pour un système d'édition collaboratif, est celui de la cohérence des répliques, quand le système est dans un état stable. Il serait aisé d'observer cette propriété du système, s'il était possible d'établir un ordre total (voir Annexe A) sur l'ensemble des opérations générées dans le système. Malheureusement, dans un environnement réparti, l'ordre total sur les événements est difficilement concevable. En effet, les nœuds qui composent un système réparti ne partagent ni mémoire, ni horloge commune (caractéristiques fondamentales de ces systèmes). Il est possible d'avoir lors d'une collaboration, des opérations concurrentes (voir Section 1.1.1) et des conflits, ce qui augmente le niveau de complexité pour

avoir un système cohérent. De surcroît, il existe une incertitude liée au délai de transit des messages dans un système réparti. Les défis de la réplication dans un système distribué sont non seulement la propagation efficace des répliques d'opérations (n'est pas étudiée dans cette thèse) mais aussi l'ordonnancement des opérations, la détection et la résolution des conflits et l'élimination de la divergence des répliques [Saito et Shapiro (2005)]. Dans les systèmes distribués, il est possible de synchroniser les horloges et les processus des différents nœuds avec une précision adaptée au contexte d'application (synchronisation temporelle et événementielle) afin de définir un ordonnancement. Une telle approche n'est pas adaptée aux systèmes d'édition collaborative. En effet, de tels systèmes doivent être réactifs — l'utilisateur doit avoir l'impression qu'il est seul à travailler — tout en brassant un nombre élevé d'opérations coopératives. En somme, le "pseudo-ordonnancement" (l'ordre total est difficile à obtenir) qui pourrait être défini sur les opérations coopératives n'élimine pas les conflits qui pourraient survenir du fait de la concurrence des opérations. Il paraît donc utile d'étudier les différentes solutions proposées dans la littérature pour affronter le défi de la cohérence engendré par les opérations concurrentes.

Une fois que la frontière des approches de gestion des cohérences est bien délimitée, il s'en suit la nécessité de jeter un regard sur les tentatives de sécurisation des systèmes d'édition collaborative. En effet, l'approche de gestion des aspects de sécurité a, sans aucun doute, une incidence sur la collaboration. Les mécanismes qui gèrent la collaboration sur chaque site sont sensés brasser les mêmes opérations. Or, une mesure de sécurité non adaptée pourrait causer l'acceptation de certaines opérations par certains sites alors que d'autres sites se verraient refuser les mêmes opérations. En conséquence, des opérations valides seraient refusées par certains sites et réciproquement, des opérations non valides pourraient être exécutées sur d'autres sites. Une telle situation conduirait inéluctablement à une divergence de copie, d'où une incohérence. Nous examinons les solutions proposées dans la littérature au sujet de la gestion de la sécurité dans un système comme celui en étude.

2.2 Approches de gestion de la cohérence dans les éditeurs collaboratifs

Dans la littérature, la nature des objets répliqués considérés dans les différents travaux sont : système de fichiers, fichiers textes avec ou sans image, fichiers XML, etc. En fonction de la nature des objets, plusieurs niveaux de granularité sont ciblés : caractère, ligne, chaîne de caractères, atomes, etc. Plusieurs techniques ont alors été élaborées comme support à la gestion de la cohérence. Ces techniques sont pour la plupart des techniques de contrôle de concurrence dont le but principal est d'éviter la divergence ; la convergence étant considérée comme une cohérence partielle. Elles peuvent être réparties en deux groupes : les techniques

pessimistes et les techniques optimistes [Imine (2006)]. Les techniques pessimistes évitent l'apparition de la divergence en utilisant des verrous. Par contre les techniques optimistes permettent l'apparition de la divergence et ensuite la résolvent à l'aide de procédures. La technique de synchronisation est un exemple de technique optimiste. Elle se définit comme un processus qui prend en entrée deux répliques divergentes et les modifie afin qu'elles soient identiques en sortie [Molli *et al.* (2003a)]. Le but est donc de faire converger grâce au synchroniseur (ou outil de fusion selon le contexte) deux copies initialement divergentes. Les approches proposées afin de garantir la cohérence sont essentiellement : « multi-versions » (MV) [Bernstein et Goodman (1983)], « sérialisation / résolution de conflits » (SRC) [Ellis et Gibbs (1989)], transformée opérationnelle (OT) [Ellis et Gibbs (1989)] et celle des types de données commutatives répliquées (*Commutative Replicated Data Type*, CRDT) [Preguiça *et al.* (2009)].

2.2.1 Approche multi-versions

Cette approche [Bernstein et Goodman (1983)] se base sur le paradigme «Copier–Modifier–Fusionner». Chaque utilisateur a sa copie locale de l'objet, la modifie à sa guise et décide du moment de la fusion. Les opérations locales ne sont donc pas automatiquement diffusées aux autres utilisateurs. La fusion a pour effet de rendre l'ancienne version obsolète et de considérer la version courante fusionnée comme dernière version [Bernstein et Goodman (1983)]. Il existe de ce fait, plusieurs versions d'un même objet. En effet, outre la nouvelle version, les anciennes sont également sauvegardées. Chaque copie se fait à partir de la dernière version disponible. Toutes ces versions sont stockées sur un site central. L'architecture utilisée est de type client / serveur. C'est le site central qui détermine l'ordre dans lequel deux événements (génération d'opérations) se sont produits. Pour assurer la cohérence, le site central gère des estampilles matérialisées par un vecteur d'horloge qui sert à décompter les opérations. Ainsi, pour chaque opération générée en local (événement interne), le site central génére une estampille. La fusion est faite opération par opération. Une fusion n'est applicable pour une opération que si l'estampille correspondante est attribuée plus tôt que celle de toute autre opération en instance de demande de fusion [Bernstein et Goodman (1981)]. Dans le cas contraire, un conflit est détecté. Un conflit est également détecté si deux opérations concernent le même élément (caractère, ligne, etc., selon la granularité), même si l'une des opérations est déjà fusionnée. En l'absence de conflit, la fusion est effective. Une nouvelle version est ainsi créée. à la suite de la fusion, la nouvelle version pourra être visible par les utilisateurs. Cette approche est utilisée dans *Concurrent Version System*[1] (CVS),

1. `http://www.cvshome.org/`

Subversion [2] (SVN) et les produits de la suite Rational ClearCase [3] qui permettent de gérer le cycle de vie d'applications logicielles.

Dans CVS, toutes les copies stockées sur le site central sont gérées grâce à une structure appelée «dépôt» (*repository*). Le dépôt contient l'historique des versions du document partagé ainsi que des métadonnées. Elles sont par exemple, l'horodatage de l'opération et l'identifiant de l'utilisateur. Chaque utilisateur a localement un espace de travail qui contient une copie du dépôt située sur le site central. Lors de la fusion, si un conflit est détecté au niveau du système de fichier, CVS a recours à l'utilisateur pour sa résolution [Molli *et al.* (2003b)]. Si par contre le conflit concerne le contenu d'un fichier texte (l'élément définissant la granularité), la fusion est faite automatiquement par le système. Toutefois, l'utilisateur peut corriger d'éventuelles mauvaises décisions prises par le système lors de la synchronisation. Les anciennes versions qui ont été sauvegardées peuvent être restaurées à cet effet. SVN est une amélioration de CVS. L'un des apports concerne le volume d'informations échangées entre le client et le serveur. En effet, les échanges sont différentiels. Seules les opérations engendrant des modifications effectives sur les objets sont envoyées au serveur lors de la fusion. Cette pratique est contraire à celle de CVS. Ce dernier prend systématiquement en compte toutes les opérations. SVN apporte donc un gain en terme de bande passante. Cependant, le principe de gestion reste le même. Les principes de SVN sont ǵalement ceux utilisés dans Rational Clear Case multi-sites [4].

L'approche multi-versions présente une faiblesse liée au coût de stockage des diverses versions. à cela s'ajoute la décision de fusion qui est dépendante du bon vouloir de l'utilisateur. Il serait intéressant qu'il soit débarrassé de cette tâche afin qu'une vue plus réaliste du document partagé soit visible par chaque utilisateur assez rapidement. De plus, le modèle de contrôle de concurrence centralisé utilisé, et son recours aux estampilles pour gérer la cohérence, rendent cette approche incompatible avec un environnement où le nombre d'utilisateurs est massif. Dans un tel environnement, il ne saurait exister un serveur central et une estampille à large échelle serait difficile à gérer.

2.2.2 Approche sérialisation / Résolution de conflits

Dans l'approche SRC [Ellis et Gibbs (1989)], un site de référence est désigné au lancement du système. Pour obtenir la cohérence, chaque site est tenu d'exécuter la même séquence d'opérations, c'est-à-dire que les opérations doivent être dans le même ordre. L'approche SRC vise donc à établir un ordre total sur les opérations avant de les exécuter. Le mécanisme

2. `http://subversion.apache.org/`
3. `http://www-03.ibm.com/software/products/fr/category/SW860`, accès : 15 mai 2014
4. `http://www-03.ibm.com/software/products/fr/ccmulticoll`, accès : 15 mai 2014

de détermination de cet ordre total sur les opérations est appelé «sérialisation». L'ordre est défini par le site désigné au début de la collaboration. Dès qu'il y a un nouvel ordre défini, chaque site doit annuler toutes les exécutions précédemment faites (qui ne correspondent pas à l'ordre établi). Les opérations doivent être réexécutées suivant le résultat de la nouvelle sérialisation. Ce processus itératif est très coûteux car il faut continuellement faire et défaire les exécutions selon le rythme de la génération de nouvelles opérations dans le système. Aussi, le délai pour réaliser la sérialisation à chaque nouvelle opération constitue un coût supplémentaire. En outre, le site désigné pour sérialiser les opérations fonctionne comme un nœud central. Au vue de ces faiblesses, la définition d'un ordre total des opérations sur un serveur avant l'exécution et l'obligation faite aux sites d'exécuter la même séquence d'opérations n'est ni adaptée aux éditeurs collaboratifs ni à un contexte de grande échelle. Toutefois, cette approche est associée à d'autres techniques pour résoudre la divergence. Tel est le cas dans MOT2 [Cart et Ferrié (2007)] et SOCT2 [Suleiman *et al.* (1997)].

Dans SOCT2, Suleiman et *al.* sérialisent les opérations concurrentes en ayant pour critère le respect des intentions des utilisateurs. Dans un environnement réparti, ils proposent l'adjonction de méthodes complémentaires pour le respect de la causalité entre deux opérations. En fait, les auteurs proposent une transformation de la deuxième opération afin de tenir compte de l'effet produit par la première et ainsi satisfaire les intentions des utilisateurs. Ce faisant un ordre partiel (voir Annexe A) est défini sur les opérations. C'est un ordre local équivalent qui est construit dès lors qu'il y a des opérations concurrentes. Cet ordre tient compte aussi bien des opérations natives que des transformations d'opérations natives. L'histoire et la sémantique des opérations ont été utilisées à cette fin.

2.2.3 Approche des types de données commutatives répliquées

La classe émergente des algorithmes de réplication est celle des types de données commutatives répliquées (CRDT) [Preguiça *et al.* (2009)]. Shapiro et Preguiça ont formalisé cette approche en utilisant pour fondement la thèse selon laquelle, sous de simples et standards hypothèses, les répliques convergent vers une valeur correcte, quels que soient leurs types de données, si les opérations concurrentes sont commutatives [Shapiro et Preguiça (2007)]. Ainsi, il ne sera pas nécessaire de détecter la concurrence entre les opérations afin d'assurer la cohérence [Weiss *et al.* (2010)]. De plus, les algorithmes conçus dans cette approche n'ont point besoin de satisfaire la condition TP2 (expression 1.8, page 10). En pratique, ces algorithmes s'appuient sur des opérations nativement commutatives (voir Annexe A), définies sur des types de données abstraites, tels que des listes, des arbres ordonnés, etc. Les opérations, une fois générées sur un site, sont diffusées vers les autres sites pour être réexécutées sans quelque mécanisme complexe de fusion ou d'intégration. Il n'y a pas d'ordre total sur

les opérations, ce qui permet à des opérations présumées concurrentes d'être exécutées dans n'importe quel ordre. De plus, les précurseurs de cette approche ont montré que de tels types de données supportent les transactions à très faible coût. Dans [Shapiro et Preguiça (2007)], en partant du principe selon lequel toutes les opérations concurrentes doivent commuter, il a été prouvé que tout type de CRDT converge si les opérations se réfèrent à des identifiants uniques différents. Le défi revient donc à concevoir les types de données et les mécanismes appropriés pour garantir la commutativité des opérations. Les questions sous-jascentes sont celles de l'identification unique des atomes et de la performance. Pour ce qui est de la gestion de l'identifiant unique, le choix du domaine de valeurs de l'identifiant doit être fait de sorte qu'il soit compact. Ainsi, entre deux identifiants donnés, il doit toujours être possible de générer un nouvel identifiant. Pour ce qui est de la performance, si par exemple une structure de données arborescente est utilisée, il faut veiller à mettre en place des mécanismes pour équilibrer l'arbre à moindre coût. Ces mécanismes ne doivent pas compromettre la gestion de l'identification unique des atomes. De plus, ils doivent être conçus pour faciliter les recherches et les faire aussi à moindre coût.

Quelques approches et techniques ont été identifiées pour assurer la commutativité. En termes d'approches, il s'agit de la coalescence des opérations et de la préséance. La coalescence suppose que, pour deux opérations concurrentes, l'exécution de l'une préserve les effets de l'autre et vice-versa. Selon Shapiro et Preguiça, bien que la coalescence soit la meilleure approche, elle n'est pas toujours possible. La préséance (ordre implicite sur les opérations), par contre, serait beaucoup plus facile à réaliser que la coalescence. En ce qui concerne les techniques, le concept de mises à jour non destructives a été couplé avec une identification invariante. De plus, si le consensus est nécessaire, il est réalisé en arrière-plan, mais juste pour des opérations non essentielles. Il est interrompu si elles rentrent en conflit avec une opération essentielle. Parmi les algorithmes de CRDT nous distinguons : WOOT [Oster *et al.* (2006b)], Logoot [Weiss *et al.* (2009)], Logoot-Undo [Weiss *et al.* (2010)], TreeDoc [Preguiça *et al.* (2009)]. Nous présentons ci-après chacun de ces algorithmes.

Algorithme WOOT

WOOT est une méthode de réplication asynchrone qui exploite un tampon de caractères répliqués de type CRDT. Les objets répliqués sont considérés avoir une structure linéaire et la définition des opérations est basée sur les caractères (éléments) et non leur position. à cet effet, un identifiant est géré pour chaque caractère utilisé dans le système. La relation d'ordre induite par une opération relative à un caractère est maintenue localement et lui est associée lors de la diffusion. Les opérations sont supposées commutatives car les mises à jour sont non destructives et l'identifiant d'un caractère ne change pas avec les modifications concurrentes.

WOOT n'utilise pas un vecteur d'horloge et ne dépend pas du nombre de sites, permettant ainsi son utilisation à grande échelle. Cependant, l'usage de la méthode dite des «pierres tombales» (*tombstones*) pour enregistrer les caractères supprimés a pour conséquence une consommation de ressource (espace mémoire). De surcroît, aucun mécanisme de vidange des plus anciens caractères supprimés, qui ne va pas compromettre la cohérence, n'est défini. Il est à noter également que WOOT ne supporte pas les opérations groupées comme le «copier-coller».

Algorithmes Logoot et Logoot-Undo

Logoot [Weiss *et al.* (2009)] est un CRDT qui considère que les objets répliqués ont une structure linéaire. L'élément de granularité considéré ici est la ligne, en plus d'un contenu. La ligne est caractérisée par un identificateur unique de position. Les identificateurs de position sont gérés grâce à une liste ordonnée dans l'ordre lexicographique. Logoot utilise un arbre n-aire comme structure de données. N'utilisant pas de «pierres tombales», il n'a pas besoin d'un outil de vidange. Il n'implémente pas le concept d'«annulation n'importe où n'importe quand» et peut supporter la grande échelle. Logoot-Undo est une version améliorée de Logoot pour supporter l'« annulation n'importe où n'importe quand ». En termes d'améliorations, on note une table d'identificateurs ajoutée au modèle, une structure de données «cimetière» qui permet de gérer le degré de visibilité de la ligne, un tampon historique par site afin de stocker les rustines. La surcharge générée par la prise en compte de la propriété d'«annulation» pourrait être dommageable au système à grande échelle.

TreeDoc

La conception de TreeDoc [Preguiça *et al.* (2009)] repose sur le fait que les documents partagés consistent en une séquence linéaire d'«atomes». Un atome représente un caractère ou tout autre élément non éditable comme des figures insérées dans l'objet. Deux opérations d'édition sont possibles sur les atomes : l'insertion et la suppression. L'insertion prend trois paramètres à savoir la position d'insertion, l'atome à insérer et l'identifiant du site générateur de l'opération. La suppression prend deux paramètres dont le premier indique la position à laquelle l'atome sera supprimé et le second, l'identifiant du site ayant initié l'opération. Malgré la suppression de l'atome, elle est maintenue dans la structure de données sous-jacente mais n'est plus visible par l'utilisateur. La commutativité des opérations est assurée par coalescence. TreeDoc utilise un tampon d'atomes dans lequel chaque position a un identifiant unique qui ne change pas pendant toute la durée de vie du document. Un ordre total est défini sur les identificateurs de positions. Toutefois, il est toujours possible de générer un nouvel

identifiant unique entre deux identifiants existants. Une structure d'arbre binaire étendu est utilisée pour les identifiants. Chaque nœud de l'arbre contient ou non un atome. L'identifiant de position associé à l'atome est le chemin dans l'arbre pour arriver à ce nœud. Pour construire le chemin, on considère que le « fils gauche » correspond à 0 et le « fils droit » à 1, le nœud « racine » correspondant à une chaîne vide représentée par le caractère ϵ. Deux stratégies sont alors proposées pour gérer les identifiants. La première est compacte mais utilise une « pierre tombale » pour garder la trace des suppressions. De ce fait, il n'est pas adaptable à un P2P car gourmant en espace mémoire, bien qu'une procédure soit proposée pour la vidange. La seconde ne garde pas trace des suppressions. Pour équilibrer l'arbre et ainsi éviter la surcharge, un mécanisme d'optimisation a été proposé.

Autres applications de l'approche CRDT

Bien que WOOT, TreeDoc, Logoot et Logoot-Undo considèrent un document XML comme ayant une structure linéaire et donc pris en charge par ces derniers, Martin et *al.* lui accordent une attention particulière dans [Martin *et al.* (2010)]. Cette attention est due à l'importante place qu'occupe de plus en plus le format XML dans les systèmes informatiques pour ce qui est du stockage, de la possibilité de faire des requêtes et des échanges de données. Le document XML n'a pas été assimilé à un document textuel linéaire, comme le faisaient les algorithmes cités ci-dessus ; mais il est réellement considéré par les auteurs comme un document semi-structuré avec une arborescence. à cet effet, pour un nœud de l'arborescence, ses attributs éventuels, ses nœuds éléments (fils) ainsi que leurs attributs respectifs sont considérés dans la conception. L'arborescence XML est considérée comme un ensemble d'arcs, chacun défini par un identifiant unique, un ensemble d'arcs fils et un ensemble d'attributs. Trois types d'opérations sont supportées par cette arborescence : l'ajout d'un arc vide avec en paramètre l'identifiant de l'arc à ajouter et celui sous lequel il sera ajouté ; la suppression d'un arc dont l'identifiant est indiqué en paramètre ; et l'affectation d'attribut à un arc. L'identifiant, l'attribut, sa valeur ainsi que l'estampille de l'opération sont les paramètres de la fonction réalisant l'affectation. La suppression d'un attribut revient à lui affecter une valeur nulle. L'estampille est utilisée afin de gérer la commutativité des opérations mais en réalité un ordre total est défini sur ces estampilles. Martin et *al.* prennent en compte les annulations afin de mieux gérer les erreurs et les conflits d'édition. à cet effet, les auteurs ont utilisé les « pierres tombales » pour garder la trace de toutes les opérations exécutées sur le document. Cependant, ils se limitent aux dernières opérations car un mécanisme a été prévu pour la vidange des vieilles opérations en se basant sur les estampilles. Bien qu'offrant une gestion compacte qui facilite l'ordonnancement des « nœuds éléments » et malgré la prise en compte d'une procédure pour la vidange, la proposition de Martin et *al.* n'est pas

très adaptée à cause de la grande échelle et sa consommation d'espace. De plus, ces travaux considèrent seulement un document XML bien formé mais pas un document XML valide. Le document peut être vérifié avec un parseur non validant (analyse de la syntaxe concrète seulement). Il n'est pas vérifiable avec un parseur validant (offre un support comme la DTD, une grammaire qui permet de vérifier la conformité du document XML). Le modèle décrit par les auteurs inclut les « pierres tombales » et les opérations. Ces dernières ne doivent pas être visibles pour les applications donc il y a nécessité d'un traitement particulier par les parseurs validant.

Wu et *al.* ont proposé un modèle de cohérence basé sur l'approche CRDT [Wu et Pui (2009), Wu *et al.* (2010)]. Il est composé de deux propriétés : la convergence et la préservation de la préséance de la dépendance des données (*Data-dependency Precedence Preservation*, DDP). La propriété DDP considère une restriction de la préservation de la préséance causale présentée à la Section 2.2.4. Elle exploite une structure de données appelée séquences partiellement persistantes (*partial persistent sequences* , PPS) que nous présentons à la Section 3.4.5.

2.2.4 Approche de la transformée opérationnelle

L'approche de la transformée opérationnelle a été proposée par la communauté des éditeurs collaboratifs synchrone pour résoudre la divergence causée par la présence d'opérations concurrentes. Elle considère plusieurs sites, chacun ayant sa copie de l'objet partagé. Les opérations générées localement sont envoyées aux autres sites pour leur prise en compte. Chaque copie est modifiée localement par l'exécution d'une opération générée localement ou reçue d'un site distant. Les sites procèdent à la transformation des opérations reçues des autres sites par rapport aux opérations locales avant de les exécuter. L'édition est possible grâce aux deux composants que sont l'algorithme d'intégration et l'algorithme de transformation inclusive (IT). Cette dernière s'occupe de la transformation des opérations reçues par rapport aux opérations concurrentes déjà exécutées. Nous présentons ici une revue des requis que doivent satisfaire les IT pour l'obtention d'une édition cohérente et les principales IT proposées dans la littérature.

Condition de cohérence

D'après Sun et *al.*, la cohérence d'un système d'édition collaborative n'est obtenue que si les critères de préservation de la causalité (préséance causale ou critère de causalité), de convergence et de préservation de l'intention sont respectés [Sun *et al.* (1998)]. Ces critères sont appelés CCI. Il y a préservation de la préséance causale, si pour deux opérations o_1 et

o_2 tel que o_1 a préséance sur o_2, leur exécution dans n'importe quel ordre conserve l'effet de o_1 mais pas nécesairement celui de o_2. Ces opérations doivent être ordonnées suivant la relation de préséance au sens de Lamport (voir Section 1.1.1, page 4). Cette propriété peut être également formulée comme indiquée dans l'expression 2.1.

$$\forall o_1, o_2 \in \mathcal{O}, o_1 \rightarrow o_2 \Rightarrow \forall s \in \mathcal{S},\ execution(o_1) \rightarrow\ execution(o_2) \tag{2.1}$$

Avec

- $\mathcal{O}$: l'ensemble des opérations générées dans le système ;
- $\mathcal{S}$: l'ensemble des sites participant à la collaboration ;
- $\rightarrow$: la relation de préséance ;
- $execution(o_i)$ qui signifie « *exécution de l'opération* o_i ».

Le critère de convergence est respecté lorsque tous les sites exécutent le même ensemble d'opérations dans un ordre quelconque et que les copies du document partagé sont identiques. L'intention d'une opération o est l'effet qu'aurait produit son exécution sur la copie locale du document, au moment de sa génération [Sun *et al.* (1998)]. Il y a donc préservation de l'intention quand son exécution produit sur tous les sites, le même effet que celui qui est produit sur le site où elle a été générée, et ceci, au moment de sa génération. Li et Li ont indiqué que la préservation de l'effet des opérations implique le respect de la convergence [Li et Li (2004)]. La preuve a été apportée dans [Li et Li (2008a)]. Autrement dit, la préservation de l'intention est suffisante pour conduire à la convergence. Pour qu'un système d'édition collaborative soit cohérent, il suffit juste que les critères de causalité et celui de préservation d'intention soient réalisés. Ils définissent un nouveau concept de relation d'effet pour les opérations. Ce concept de relation d'effet est en fait une reformulation de la préservation d'intention. Ils proposent alors un nouveau modèle de cohérence appellé CSM (*Causality, single-operation effect, and multi-operation effects relation preservation*) qui pourrait être traduite littéralement par « préservation de la causalité, préservation de l'effet d'une opération et préservation des effets de plusieurs opérations ». La préservation de l'effet d'une opération stipule que l'effet de l'exécution de toute opération sur n'importe quel état d'exécution, produit le même effet que dans son état de génération. Par exemple, un caractère supprimé (dans l'état de génération de la suppression), doit l'être dans tous les états du système postérieur à la génération. Pour une opération d'insertion, l'ordre total doit être maintenu pour deux caractères quelconques quels que soient les états du système (états postérieurs à l'état de génération de l'insertion). La préservation de l'effet de multiples opérations stipule quant à lui que, pour tout état, la relation d'effet entre deux opérations quelconques doit être maintenue après leur exécution.

Dans l'approche OT, le critère de convergence est respecté si l'algorithme de transformation (IT) utilisé satisfait les propriétés TP1 et TP2 [Ressel *et al.* (1996)] définies à la Section 1.2 (expressions 1.7 et 1.8, page 10). à l'analyse de ces propriétés, on peut retenir que la TP1 peut être satisfaite conceptuellement, alors que la TP2 n'est pas triviale. Nos travaux dans le chapitre 3 en donne d'ailleurs les preuves.

Plusieurs algorithmes ont été recensés dans la littérature. Un inventaire et une comparaison de quleques uns de ces algorithmes est disponible dans [Kumawat et Khunteta (2010)]. Les principaux algorithmes basés sur OT sont : algorithme de Ellis [Ellis et Gibbs (1989)], algorithme de Ressel [Ressel *et al.* (1996)], algorithme de Sun [Sun *et al.* (1998)], algorithme de Suleiman [Suleiman *et al.* (1998)], algorithme d'Imine [Imine *et al.* (2003)] et SO6 [Molli *et al.* (2003b)].

Algorithme de Ellis

L'algorithme de Ellis [Ellis et Gibbs (1989)] (voir Annexe B.1.) considère un objet textuel, donc à structure linéaire, pour lequel seul deux fonctions peuvent être utilisées : insertion et suppression d'un caractère. L'insertion prend trois paramètres : la position p dans laquelle l'insertion sera faite, le caractère c à insérer et une priorité pr qui est utilisée pour résoudre le conflit né de deux opérations concurrentes d'insertion de caractères différents à la même position. La priorité est déterminée sur chaque site et il ne saurait avoir collision (même valeur de priorité sur deux sites différents). La suppression quant à elle, prend deux paramètres : la position p à laquelle il faut supprimer un caractère et la priorité pr définie comme précédemment. L'analyse de l'algorithme de Ellis montre qu'il exploite la propriété sémantique des opérations. Selon Suleiman et *al.*, il existerait bien des situations d'incohérence, de non préservation de l'intention et d'annulation suivie de réexécution de certaines opérations [Suleiman *et al.* (1998)]. Boucheneb et Imine ont entériné cette thèse en prouvant que l'algorithme de Ellis ne respecte pas la propriété de convergence et par conséquent n'assure pas la cohérence des données [Boucheneb et Imine (2009)]. Selon Ressel [Ressel *et al.* (1996)], l'algorithme de Ellis entraîne une incohérence quand un site exécute plus d'une opération concurrente par rapport à une opération générée sur un autre site. Il propose alors une amélioration de celui de Ellis.

Algorithme de Ressel

L'algorithme de Ressel (voir Annexe B.2.) est utilisé actuellement dans bien de systèmes centralisés d'édition collaborative populaires tel que XWiki [4]. Outre les différences réperto-

4. `http://www.xwiki.com/`

riées à la section 3.2.3, l'approche de Ressel pour gérer la cohérence diffère de celle de Ellis par l'introduction d'un modèle d'interactions multi-dimensionnelles pour garder la trace des transformations valides. Le nombre de dimensions est égal au nombre de sites participant à la collaboration. Son approche prend en compte les «annulations groupées». L'algorithme de Ressel ne satisfait pas la propriété TP2. Imine le prouve dans [Imine (2006)] en exhibant un contre-exemple. La preuve qu'il ne permet pas la convergence a été également faite par Boucheneb et Imine en utilisant le model-checking [Boucheneb et Imine (2009)]. Des contre-exemples prouvant que l'algorithme de Ellis ne respecte pas la TP1 et que celui de Ressel ne respecte pas la TP2 sont présentés à la figure 3.6 (page 46) et à la figure 3.9 (page 47), respectivement.

Algorithme de Sun

Sun et *al.* ont proposé un algorithme de transformation inclusive qui se base sur une chaîne de caractères comme élément de granulatité. Les opérations acceptées sont : (i) l'insertion d'une chaîne de caractères d'une certaine longueur dans une position donnée et (ii) la suppression d'une chaîne de caractères d'une longueur donnée à partir d'une position donnée. Dans le cas de l'insertion, la position dans laquelle l'insertion doit être faite, la chaîne de caractère à insérer et sa longueur sont les paramètres. Pour une suppression, seule la position et la longueur sont les paramètres (voir Annexe B.3.). L'algorithme de Sun et *al.* ne satisfait ni la TP1 ni la TP2 [Boucheneb *et al.* (2010)].

Algorithme de Suleiman

L'algorithme de Suleiman [Suleiman *et al.* (1998)] considère des objets textuels sur lesquels les utilisateurs agissent en insérant ou en supprimant des caractères. En cherchant à assurer la satisfaction de la TP2, Suleiman et *al.* proposent une méthode synchrone qui exploite aussi les propriétés sémantiques des opérations (voir Annexe B.4.). Leur approche bâtit l'historique associé à un objet de manière incrémentale sans prendre en compte l'annulation et la réexécution de certaines opérations. La fonction de suppression ne prend en paramètre que la position p du caractère à supprimer. La fonction d'insertion quant à elle prend quatre paramètres qui sont respectivement : la position p à laquelle l'insertion sera faite, le caractère c à insérer, un paramètre av qui contient les opérations de suppression qui ont eu pour effet d'enlever un caractère en avant de la position d'insertion et un paramètre ap qui contient les opérations qui ont effacé un caractère en arrière de la position d'insertion. Comparativement aux précédents algorithmes présentés, celui de Suleiman et *al.* n'utilise pour l'insertion ni de paramètre de priorité ni d'identifiant de site mais, en lieu et place, deux paramètres servant

à mémoriser l'ensemble des opérations de suppression concurrentes à l'opération d'insertion. De plus la suppression ne requiert que la position comme seul paramètre. La solution de Suleiman et *al.* a permis de résoudre bien des cas que les précédents algorithmes n'ont pas pu résoudre. Cependant, il existe des cas où l'algorithme ne conduit pas à la satisfaction de la condition TP2. Cette preuve a été apportée dans [Imine (2006)] par contre-exemple. Boucheneb et Imine [Boucheneb et Imine (2009)] en utilisant le model-checking ont également établit que l'algorithme de Suleiman et *al.* ne respecte pas la propriété de convergence. Des contre-exemples qui témoignent du non respect des propriétés TP1 et TP2 par l'algorithme de Suleiman sont exhibés à la figure 3.8 (page 47) et à la figure 2.1.

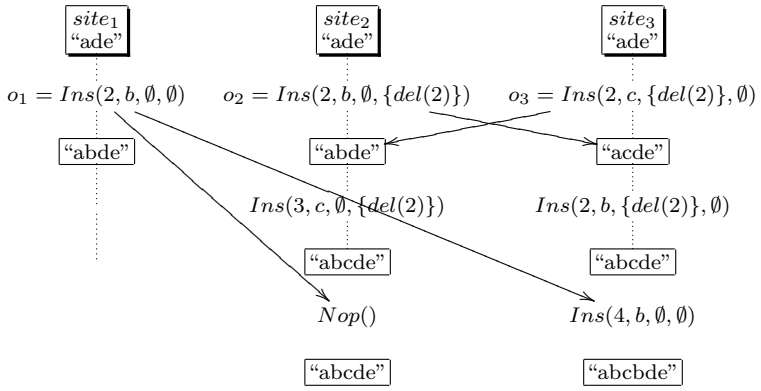

Figure 2.1 Violation de la propriété TP2 par l'algorithme de Suleiman.

Le contre-exemple de la figure 2.1 considère trois sites : $site_1$, $site_2$ et $site_3$. Les copies du document partagé sont initialement identiques sur les sites et contiennent le texte "ade". Les sites génèrent chacun une opération d'insertion en position 2, qui est en concurrence avec chacune des autres opérations. Sur les trois sites, l'historique associée à l'insertion contient comme paramètres av et ap : $(\emptyset, \emptyset)$, $(\emptyset, \{del(2)\})$, $(\{del(2)\}, \emptyset)$, respectivement. Ainsi, $site_1$ produit une insertion sans historique. Sur $site_2$, l'historique n'indique que la suppression en position 2 après la position courante. Quant à $site_3$, son insertion n'indique que la suppression en position 2 avant la position courante. Il est à remarquer que la position courante d'insertion et les positions indiquées dans l'historique sont égales. L'application des différentes transformations suggérées par Suleiman et *al.* donne comme résultats sur les $site_2$ et $site_3$ les textes "abcde" et "abcbde", respectivement.

Algorithme d'Imine

Imine et *al.* ont essayé d'apporter une amélioration à l'algorithme de Suleiman [Suleiman *et al.* (1998)] en associant à chaque caractère sa position initiale [Imine *et al.* (2003)]. Cette position reste inchangée même si la transformation de l'opération afférente affecte la posi-

tion courante du caractère. Dans leur algorithme (voir Annexe B.5.), la fonction d'insertion comporte trois paramètres : la position courante d'insertion p, la position initiale d'insertion définie à la génération de l'opération o et le caractère c à insérer. à la génération, p et o sont de valeurs égales. La fonction de suppression ne prend que le paramètre indiquant la position du caractère à supprimer. Imine et *al.*, en faisant l'hypothèse que les opérations d'insertion considérées n'auraient pas subi de transformation auparavant, avaient conclu que leur algorithme satisfaisait TP2. Cependant, il n'est pas réaliste de considérer que o et p sont de valeurs égales avant toute vérification de TP2. En faisant fi de cette hypothèse, leur algorithme ne conduit pas à la convergence. Imine en exhibe d'ailleurs lui-même un contre-exemple dans [Imine (2006)]. Ce constat a été renforcé par les travaux de Boucheneb [Boucheneb et Imine (2009)]. Un exemple de la violation du TP2 par l'algorithme d'Imine est proposé à la figure 3.10 (page 47).

Algorithme SO6

SO6 [Molli *et al.* (2003b)] est une méthode de réplication asynchrone basée sur la transformée opérationnelle. Il implémente un synchroniseur de système de fichier qui s'appuie sur un séquenceur pour faire l'ordonnancement des opérations concurrentes. Ces opérations sont relatives à plusieurs niveaux de granularité : système de fichier, fichier texte, fichier XML, texte, etc. SO6 allie l'approche «copier-modifier-fusionner» et l'approche de la transformée opérationnelle. Le séquenceur définit un ordre total sur les opérations en les estampillant. Ce mécanisme est déployé par une machine centrale. La propagation des opérations se fait en se basant sur le mécanisme de diffusion différée proposé dans SOCT4 [Vidot *et al.* (2000)]. Cette méthode ne peut pas être utilisée à grande échelle du fait de l'usage d'un séquenceur. La variabilité de la topologie rend aussi inutilisable le séquenceur.

Algorithme MOT2

MOT2 [Cart et Ferrié (2007)] est un algorithme de réconciliation asynchrone basée sur la transformée opérationnelle. Inspiré de MOT1 [Bernstein et Goodman (1981)], qui considère une copie de référence, MOT2 suppose plutôt une relation d'ordre entre les sites ou entre les copies dès leur création. La relation entre les sites peut dériver des noms de ces sites si leur unicité est garantie. Celle entre les copies peut également dériver de leur nom en utilisant un schéma d'identification hiérarchique. Cette relation d'ordre est utilisée à des fins de sérialisation en conduisant à la définition d'un ordre total unique entre les opérations. Cette stratégie dispense MOT2 d'un serveur central, de vecteurs d'états, d'estampilles ou de séquenceur. Il procède par appariement des copies (deux à deux) et ne privilégie aucune

copie au cours du processus. La propagation est faite librement dès que deux sites décident de réconcilier leurs copies. Il serait intéressant d'affranchir les sites de cette décision. De plus, l'absence d'un serveur central et de vecteurs rendent certes MOT2 compatible au facteur échelle en architecture répartie, mais appliquer une stratégie d'appariement à cette échelle entraînera la dégradation des performances en termes de temps nécessaire pour que le système soit stable.

Algorithmes SDT et SDTO

à l'appui de leur nouveau modèle de cohérence (CSM) qui prend en compte les critères de préservation de causalité et de préservation d'intention, Li et Li proposent, afin de satisfaire la condition TP2, le concept de «transformation par différences d'états» (*state difference transformation*, SDT). L'algorithme SDT [Li et Li (2004), Li et Li (2008a)] a été proposé sur la base de ce concept. Selon les auteurs, l'algorithme SDT (voir Annexe B.6.) assure la convergence dans une architecture «pure Pair-à-Pair». Dans SDT, les auteurs ont recours à deux fonctions. La première est une fonction de transformation inverse de la transformation IT. Elle est notée ET (*exclusion transformation*) et permet d'exclure l'effet d'une opération par rapport à une autre. La deuxième fonction, notée β, permet d'obtenir pour toute opération, la position qu'elle aurait pu avoir sur un état précédent de convergence. Cette dernière est utilisée avec une fonction δ qui détermine les opérations ayant les mêmes valeurs de retour pour β [Li et Li (2008a)]. Cette solution a été proposée dans le seul but de résoudre la TP2 et ne tient pas compte des questions de performance. Pour pallier ce problème, le SDTO [Li et Li (2008b)] a été élaboré et constitue une optimisation du SDT. Cette optimisation a permis de réduire considérablement la complexité (en temps et en espace) de ce dernier. Il a également été étendu par l'ajout d'une troisième primitive outre l'insertion et la suppression. Cette primitive ajoute une fonctionnalité de mise à jour et permet de traiter les documents formatés en complément aux documents textuels que prenait en charge SDT. La vérification de l'algorithme de Li et Li réalisée par Imine [Imine (2006)] a montré un contre-exemple et permet de conclure que cet algorithme ne conduit pas à la convergence. Un exemple de cas de violation de la TP2 est fourni à la figure 2.2.

Le contre-exemple de la figure 2.2 considère trois sites : $site_1$, $site_2$ et $site_3$. Les copies du document partagé sont initialement identiques sur les sites et contiennent le texte "adf". Une opération d'insertion du caractère b en position 2 est générée par $site_1$. Sur $site_2$, une opération d'insertion est également générée. Il s'agit d'ajouter le caractère e en position 3. Enfin, $site_3$ produit une opératoion de suppression en position 2. Les trois opérations sont concurrentes. Leurs exécutions locales donnent respectivement : "abdf", "adef" et "af". La transformation selon SDT, de l'opération reçue de $site_3$ par $site_2$, suivie de la transformation

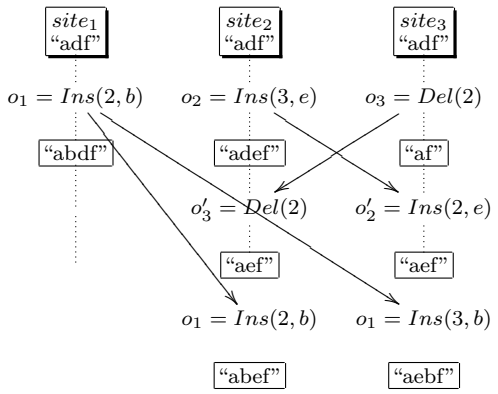

Figure 2.2 Violation de la propriété TP2 par SDT.

de celle reçue de $site_1$ par $site_2$, conduit à une insertion en position 2. Par contre, $site_3$ obtient une insertion en position 3, après avoir transformé les opérations reçues de $site_2$ et $site_1$, prises dans cet ordre. En comparant les résultats des transformations de l'opération reçue de $site_1$ par chacun des deux autres sites, il est aisé de conclure que les deux sites n'ont pas le même résultat. Ce qui montre que la TP2 n'est pas vérifiée.

Fonctions TTF

Pour assurer la cohérence des documents textuels, Oster et *al.* ont proposé un ensemble de fonctions dénommé TTF (*Tombstone Transformation functions*) [Oster *et al.* (2006a)]. Leur solution considère le caractère comme élément de granularité manipulé lors d'une édition. La suite de fonctions est composée d'une fonction de recherche de position d'un caractère et d'une fonction de transformation inclusive. Le TTF s'appuie sur deux modèles représentant le document édité : la vue et le modèle de données. La vue est le document auquel l'utilisateur a accès tandis que le modèle de données est transparent pour lui. Le lien entre les deux modèles est assuré par le principe de la pierre tombale (*tombstone*) et l'usage de la fonction de transformation. En effet, lorsque l'utilisateur insère un caractère dans la vue, il est aussitôt aussi inséré dans le modèle de données. Quand il supprime un caractère dans la vue, celui est effectivement supprimé de la vue, le rendant ainsi invisible, mais simplement déclaré masqué dans le modèle de données sous-jascent, qui est de ce fait un modèle persistant. C'est le fait de masquer un caractère dans le modèle persistant qui lui fait valoir l'attribut de pierre tombale. La fonction de transformation sert alors à faire le lien entre une position de la vue et la position correspondante dans le modèle persistant, en prenant en compte en plus des caractères visibles, tous les caractères supprimés (masqués) avant cette position. De façon pratique, quand l'utilisateur insère un caractère dans la vue, la fonction de recherche de position est utilisée pour trouver la position dans laquelle le caractère doit être inséré de

manière cohérente dans le modèle persistant. De la même manière, quand il supprime un caractère de la vue, la fonction permet de trouver ce caractère dans le modèle persistant en retournant la position de ce dernier. La nouvelle position déterminée est celle qui est envoyée aux autres utilisateurs lors de la réplication de l'opération. Ainsi, lorsqu'une opération distante est reçue, elle est transformée sur la base du modèle persistant. Il faut noter que la fonction de recherche de position est bien comparable à la fonction β de SDT ou SDTO (Section 2.2.4), qui permet d'obtenir pour toute opération, la position qu'elle aurait pu avoir sur un état précédent de convergence. En termes de ressources, la proposition de Oster et *al.* manipule deux structures de données en plus de l'histoire des opérations, ce qui nécessite des dispositions particulières pour leur gestion et une quantité de mémoire supplémentaire. De plus, toute mise à jour sollicitée par l'utilisateur implique une opération de mise à jour sur chacune des deux structures de données et un calcul de position. En outre, la sémantique de la suppression de caractère est redéfinie par l'usage de la pierre tombale avec pour conséquence la persistance du caractère et une croissance de la taille du modèle de données. La complexité de TTF est consignée dans le tableau 3.4 (page 61).

2.3 Contrôle d'accès dans les systèmes collaboratifs

Les principaux modèles de contrôle d'accès dédiés aux systèmes collaboratifs sont pour la plupart répertoriés dans [Tolone *et al.* (2005)]. Il s'agit de : la matrice d'accès (*Access Matrix Model*, AMM) [Sandhu et Samarati (1994)], le contrôle d'accès basé sur les rôles (*Role Based Access Control*, RBAC) [Sandhu *et al.* (1996)], le contrôle d'accès basé sur les tâches (*Task Based Access Control*, TBAC) [Thomas et Sandhu (1997),Thomas et Sandhu (1994)], le contrôle d'accès basé sur les équipes (*Team Based Access Control*, TMAC) [Thomas (1997)], le contrôle d'accès basé conjointement sur les informations de contexte et les équipes (*Context Based Team-Based Access Control*, C-TMAC) [Georgiadis *et al.* (2001)], le contrôle d'accès spatial (*Spatial Access Control*, SAC) [Bullock et Benford (1999)] et le contrôle d'accès sensible au contexte (*Context Aware Access Control*) [Covington *et al.* (2001)]. En se basant sur l'idée selon laquelle la conception d'un modèle de contrôle d'accès qui tient compte des spécificités d'une édition collaborative répartie est sujette aux exigences telles que la spécification de haut niveau des droits d'accès, la généricité et la flexibilité du modèle, l'aspect dynamique du modèle, le maintien des indicateurs de performance à des seuils acceptables, Tolone et *al.* ont déterminé des critères qui leur ont permis de faire une étude comparative sur ces modèles de contrôle d'accès. Les auteurs ont mis en lumière les forces et les faiblesses de chacun d'eux au regard du contexte complexe de l'édition collaborative répartie. De cette étude, nous pouvons retenir que le TMAC est le plus adapté, en ce qui concerne une édition

collaborative sécurisée. Cependant, TMAC a un niveau d'applicabilité assez moyen. L'une des faiblesses de TMAC est qu'il ne permet pas d'ajouter de nouveaux droits d'accès aux membres d'un groupe, autres que ceux découlant directement des droits fonctionnnels du groupe. C'est l'une des raisons pour lesquelles il a été étendu dans TMAC04 [Alotaiby et Chen (2004)] pour tenir compte des contextes et des organisations ayant un grand nombre de groupes d'utilisateurs qui doivent interagir entre eux. C-TMAC est une variante de TMAC qui tire partie des rôles, des équipes et des informations de contexte, dans un modèle flexible de contrôle d'accès. Cette démarche a été également suivie dans le modèle STRAC [Kawagoe et Kasai (2011)] consacré au milieu médical. STRAC est en fait une variante de C-TMAC qui considère la situation médicale comme contexte déterminant le modèle. Pour gérer les accès, RBAC utilise pour chaque utilisateur la notion de session. Bien qu'il ait permis de résoudre certains problèmes liés à la gestion dynamique des droits d'accès, RBAC ne rend pas aussitôt effectif le changement du rôle d'un utilisateur. Ce dernier doit se déconnecter et se reconnecter afin d'être authentifié à nouveau sous son nouveau rôle, sans quoi il serait dans l'illégalité. Cette faiblesse a été comblée avec SAC, mais il reste aussi non adapté au contexte en étude à cause du mécanisme de verrou à deux phases qu'il utilise et qui est plus adapté pour un contexte de base de données. Dans [Imine *et al.* (2009)], un modèle flexible de contrôle d'accès a été proposé (FACMDCE). Il est basé sur la réplication du document partagé ainsi que ses autorisations d'accès dans la mémoire locale de chaque utilisateur. Pour faire face à la latence et au changement dynamique des droits d'accès, une technique optimiste a été utilisée et consiste à appliquer de manière rétroactive les autorisations d'accès.

Malheureusement, nous n'avons pas trouvé d'étude consacrée à la vérification de la préservation de la propriété de cohérence pour les modèles cités précédemment. Cependant, il existe plusieurs études telles que [Abdunabi *et al.* (2013), Jayaraman *et al.* (2013), Toahchoodee *et al.* (2009),Hu et Ahn (2008),Samuel *et al.* (2007)], relatives à la satisfaction de certaines propriétés par RBAC et quelques unes de ses variantes, dans des cadres spécifiques.

2.4 Conclusion

Plusieurs approches ont été proposées dans la littérature pour gérer la cohérence des documents lors d'une édition collaborative. Celles qui sont plus adaptées à un contexte réparti sont l'approche des types de données commutatives répliquées et l'approche de la transformée opérationnelle. Dans l'approche CRDT, il n'est pas nécéssaire de détecter la concurrence entre les opérations afin d'assurer la cohérence et les opérations peuvent être exécutées dans un ordre quelconque. De plus, cette approche permet de contourner le défi que représente la satisfaction de la propriété TP2. Les répliques peuvent converger vers une valeur correcte,

quels que soient leurs types de données, mais à condition que les opérations soient nativement commutatives. Cependant, obtenir des opérations commutatives est tout un autre défi. L'autre défi réside dans la définition de la structure de données abstraite qu'il faut pour gérer les atomes ou éléments de granularité considérés. Elle nécessite une identification unique de ces atomes mais également un domaine d'identification compact qui puisse toujours permettre la génération d'un nouvel identifiant entre deux identifiants quelconques. L'approche CRDT a été utilisée dans plusieurs solutions ; parmi celles-ci figurent TreeDoc, qui gère la commutativité par coalescence, WOOT, Logoot, Logoot-Undo qui se basent sur la préséance causale.

L'approche de la transformée opérationnelle présente aussi plusieurs avantages. Elle accorde aux sites de modifier librement leurs copies et d'échanger leurs modifications dans n'importe quel ordre. Les opérations transformées peuvent également être exécutées dans un ordre différent sur chaque site. Ceci est possible grâce à l'équivalence établie entre les histoires. L'approche OT ne définit donc pas un ordre total sur les opérations. De plus, elle produit un état de convergence sans perte de mise à jour. Elle offre également l'avantage de résoudre les conflits au fur et à mesure de l'exécution des opérations. Cependant, l'utilisation d'un mécanisme appropriée de détection de la concurrence et d'une fonction de transformation correcte qui doit satisfaire les propriétés TP1 et TP2 sont des défis. L'approche de la transformée opérationnelle a été utilisée par la plupart des méthodes synchrones [Ellis et Gibbs (1989)], [Ressel *et al.* (1996)], [Suleiman *et al.* (1998)], [Li et Li (2004), Li et Li (2008a)], [Li et Li (2008b)], [Imine *et al.* (2003)], mais aussi par des algorithmes asynchrones [Molli *et al.* (2003b)], [Cart et Ferrié (2007)], [Oster *et al.* (2006a)]. Parmi les ITs proposées dans la littérature, certaines se heurtent à la satisfaction de la condition TP2 en considérant le modèle CCI [Ellis et Gibbs (1989)], [Ressel *et al.* (1996)], [Suleiman *et al.* (1998)], [Sun *et al.* (1998)], [Imine *et al.* (2003)], ou à la préservation de l'intention des opérations, en considérant le modèle CSM [Li et Li (2004), Li et Li (2008a)], [Li et Li (2008b)]. Il en existe qui ne satisfont pas la TP1 [Ellis et Gibbs (1989)], [Suleiman *et al.* (1998)], [Sun *et al.* (1998)]. La solution TTF quant à elle considère une sémantique différente pour l'opération de suppression. Sa fonction de transformation inclusive est dite correcte par les auteurs mais aucune étude n'a prouvé son exactitude par rapport aux propriétés TP1 et TP2.

Plusieurs modèles ont également été proposés dans la littérature pour gérer le contrôle d'accès dans les systèmes collaboratifs. Parmi eux, très peu répondent convenablement aux spécificités d'une édition massivement répartie (TMAC, FACMDCE). Nous notons à l'état actuelle de nos connaissances, l'absence d'une étude qui montre que ces modèles préservent la cohérence d'un système d'édition collaborative réparti.

CHAPITRE 3

ARTICLE 1 : On Synthesizing a Consistent Operational Transformation Approach

Aurel Randolph * , Hanifa Boucheneb *, Abdessamad Imine †, and Alejandro Quintero*

Abstract

The Operational Transformation (OT) approach, used in many collaborative editors, allows a group of users to concurrently update replicas of a shared object and exchange their updates in any order. The basic idea is to transform any received update operation before its execution on a replica of the object. Concretely, OT consists of a centralized / decentralized integration procedure and a transformation function. In the context of decentralized integration, designing transformation functions for achieving convergence of object replicas is a critical and challenging issue. Indeed, the transformation functions proposed in the literature are all revealed inefficient.

In this paper, we investigate the existence of transformation functions. From the theoretical point of view, two properties, named TP1 and TP2, are necessary and sufficient to ensure convergence. Using controller synthesis technique, we show that there are some transformation functions, which satisfy TP1 for the basic signatures of insert and delete operations. But, there is no transformation function, which satisfies both TP1 and TP2. Consequently, a transformation function which satisfies both TP1 and TP2 must necessarily have additional parameters in the signatures of some update operations.

We propose, in this paper, a new transformation function and show formally that it ensures convergence.

Keywords Collaborative editors, operational transformation, proof of convergence, symbolic model checking, controller synthesis.

∗. A. Randolph, H. Boucheneb and A. Quintero are with the Department of Computer and Software Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Québec, Canada, H3C 3A7. E-mail : {aurel.randolph, hanifa.boucheneb, alejandro.quintero}@polymtl.ca

†. A. Imine is with Lorraine University and INRIA Nancy-Grand-Est, France. E-mail : abdessamad.imine@inria.fr

3.1 Introduction

Collaborative editing systems (CESs for short) constitute a class of distributed systems where dispersed users interact by manipulating some shared objects like texts, images, graphics, XML documents, etc. To improve data availability, these systems are based on data replication. Each user has a local private copy of the shared object that he can access and update. The update operations executed locally are propagated to other users. The execution of these operations in different orders may lead to a divergence (object replicas are not identical). As an example, suppose two users u_1 and u_2 working on their own copies of a text containing the word *"efecte"*, starting at position 0. User u_1 inserts 'f' at position 1, to change the word into *"effecte"*. Concurrently, user u_2 deletes element at position 5 (i.e., the last $'e'$), to change the word into *"efect"*. Each user will receive an update operation that was applied on a different version of the text. Applying naively the received update operations will lead to divergent replicas (*"effece"* for user u_1 and *"effect"* for user u_2, see Figure 3.1). Moreover, users may generate concurrently conflicting or identical operations. The challenge in such situations is to ensure convergence of replicated data whilst preserving the intention of users.

Several approaches are proposed in the literature, to deal with the convergence of replicated data: Multi-Version (MV) [Bernstein et Goodman (1983)], Serialization-Resolution of Conflicts (SRC) [Ellis et Gibbs (1989)], Commutative Replicated Data Type (CRDT) [Preguiça *et al.* (2009), Weiss *et al.* (2010)], Operational Transformation (OT) [Ellis et Gibbs (1989)], etc.

The multi-version approach, used in CVS, Subversion and ClearCase, is based on the paradigm "Copy-Modify-Merge". In this approach, update operations made by a user are not automatically propagated to the others. They will be propagated only when the user explicitly calls the merge function. It would be interesting to propagate automatically, to all others, each update operation performed by a user. This is the basic idea of SRC.

To achieve convergence, SRC imposes to execute the operations in the same order at every site. Therefore, sites may have to undo and execute again operations, as they receive the final execution order of update operations. This order is determined by a central server fixed when the system is launched. For the previous example, this approach requires that sites of both users execute the two operations in the same order. However, even if we obtain an identical result in both sites, the execution order imposed by the central server may not correspond to the original intention of some user. For instance, executing, in both sites, the operation of u_1 followed by the one of u_2 results in the text *"effece"*, which is inconsistent with the intention of u_2.

The Commutative Replicated Data Type (CRDT) is a data type where all concurrent operations commute with each other [Preguiça *et al.* (2009), Weiss *et al.* (2010)]. In such a case, to ensure convergence of replicas it suffices to respect the causality principle (i.e., whenever an operation o' is generated after executing another operation o, o is executed before o' at every site). The main challenge of CRDT is designing commutative operations for the data type. The commonly used idea consists in associating a unique identifier with the position of each symbol, line or atom of the shared document and when an operation is generated, a unique identifier is also associated with the inserting/deleting position. The position identifiers do not change and are totally ordered with regard to $<$. Symbols, lines or atoms of the document appear in increasing order with regard to their identifiers. Let us apply this paradigm to the previous example. A unique identifier is associated with each symbol of the initial text: *"(e,3) (f,6) (e, 8) (c,9) (t,9.5) (e,10)"*. A unique identifier between 3 and 6 is affected to position 1 of the operation of u_1. Let 4.5 be the selected identifier. The identifier affected to position 5 of the delete operation of u_2 is 10. Both execution orders of operations of u_1 and u_2 lead to the text *"(e,3) (f,4.5) (f,6) (e, 8) (c,9) (t,9.5)"*. CESs like TreeDoc [Preguiça *et al.* (2009)], Logoot [Weiss *et al.* (2009)], Logoot-Undo [Weiss *et al.* (2010)] and WOOT [Oster *et al.* (2006b)] are based on CRDT paradigm. Managing position identifiers is a very important issue in this approach as the correctness is based on the uniqueness of position identifiers. Ensuring uniqueness may induce space and time overheads. Moreover, the identifier space increases regardless of the size of the document.

Unlike CRDT, in OT approach, proposed by Ellis et *al.* [Ellis et Gibbs (1989)], there is no need to associate a unique identifier with each symbol and the generated concurrent operations are not necessarily commutative. Their commutativity is forced by transformation of operations before their execution. More precisely, when a site receives an update operation, it is first transformed with regard to concurrent operations already executed on the site. The transformed operation is then executed on the local copy. This transformation aims at ensuring the convergence of copies even if users execute the same set of operations in different orders but with regard to causality principle. Concretely, OT consists of a centralized / decentralized integration procedure and a transformation function, called Inclusive Transformation (IT).

The integration procedure is in charge of executing update operations, broadcasting local update operations to other sites, receiving update operations from other sites, and determining transformations to be performed on a received operation before its execution. Several integration procedures have been proposed in the groupware research area, such as dOPT [Ellis et Gibbs (1989)], adOPTed [Ressel *et al.* (1996)], SOCT2,4 [Suleiman *et al.* (1998), Vidot *et al.* (2000)], GOTO [Sun et Ellis (1998)] and COT [Sun et Sun (2009)]. There are two

kinds of integration procedures: centralized and decentralized. In the centralized integration procedures such as SOCT4 and COT, there is a central node which ensures that all concurrent operations are executed in the same order at all sites. In the decentralized integration procedures such as adOPTed, SOCT2 and GOTO, there is no central node and the operations may be executed in different orders by different sites. We focus, in the following, on the decentralized integration procedures as they do not need any central server and then are more appropriate to P2P systems.

The IT function transforms an update operation with regard to another one. For the previous example, when u_1 receives the operation of u_2, it is first transformed with regard to the local operation as follows: $IT(Del(5), Ins(1, f)) = Del(6)$. The deletion position is incremented because u_1 has inserted a character at position 1, which is before the character deleted by u_2. Next, the transformed operation is executed on the local copy of u_1. In a similar way, when u_2 receives the operation of u_1, it is transformed as follows before its execution on the local copy of u_2: $IT(Ins(1, f), Del(5)) = Ins(1, f)$ (see Figure 3.2). We can find, in the literature, several IT functions: *Ellis*'s algorithm [Ellis et Gibbs (1989)], *Ressel*'s algorithm [Ressel *et al.* (1996)], *Sun*'s algorithm [Sun *et al.* (1998)], *Suleiman*'s algorithm [Suleiman *et al.* (1997)] and *Imine*'s algorithm [Imine *et al.* (2003)]. However, all these functions fail to ensure convergence [Imine *et al.* (2006), Imine (2006), Boucheneb et Imine (2009), Boucheneb *et al.* (2010)]. More precisely, from the theoretical point of view, to ensure data convergence, the IT function has to satisfy two properties named TP1 and TP2 [Ressel *et al.* (1996)]. Intuitively, IT satisfies TP1 means that IT forces commutativity of any pair of update operations executed in different order, on the same state. IT satisfies TP2 means that an update operation is transformed in the same manner with regard to two different but equivalent sequences of update operations. Some IT functions, proposed in the literature, satisfy TP1 but there is no IT function which satisfies TP2. To overcome this problem, some solutions impose a total ordering on the integration of operations, fixed in general by a central site. In this context, property TP1 is used to preserve the user intentions. For instance, in Figure 3.4 both update operations are executed in the same order in both sites but the intention of u_2 is not preserved. The transformation of the update operation of u_2 with regards of the operation of u_1: $IT(o_2, o_1) = Del(6)$ will preserve the intention of u_2. Such solutions are used in some tools such as Git [1], Joint Emacs [Ressel *et al.* (1996)], CoWord [Sun *et al.* (2006)], CoPowerPoint [Sun *et al.* (2006)] and Google Wave [2]. However, an IT function which satisfies both TP1 and TP2 will be of great interest and value for developing interesting distributed applications in the context of P2P architectures.

1. git.smc.com
2. http://www.waveprotocol.org/whitepapers/operational-transform.

In this paper, we first investigate the existence of IT functions, based on the classical signatures of update operations, which satisfy TP1 and TP2. Then, we propose a new IT function and show formally that it ensures convergence.

Section 3.2 is devoted to OT and IT functions proposed in the literature. For each IT function, we provide a counterexample for the convergence property. In Section 3.3, we show, using a controller synthesis technique, that for the classical signatures of update operations, there are some IT functions, which satisfy TP1 but there is no IT function, which satisfies both TP1 and TP2. Exploring the reasons of this failure identifies two problematic scenarios, which need two different transformation functions to ensure convergence. Consequently, a transformation function which satisfies TP1 and TP2 must have additional parameters in the operation signatures. Finding the appropriate parameters to be added to the signatures of the update operations, so as to ensure convergence with low overhead, is very challenging. Indeed, almost all IT functions, proposed in the literature, are based on extending the signature of the insert operation by one or two parameters but fail to ensure data convergence. We propose, in Section 3.4, to extend the signature of the insertion with the number of symbols deleted before the inserting position and establish a new transformation algorithm. We end, this section, by proving formally, using a symbolic model checking technique, that our IT function satisfies both TP1 and TP2 and then ensures convergence. Conclusion goes in Section 3.5.

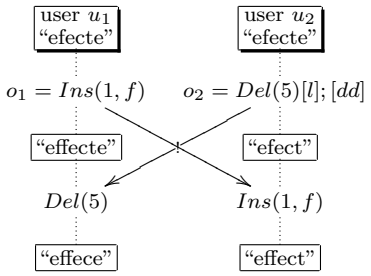

Figure 3.1 Integration without transformation.

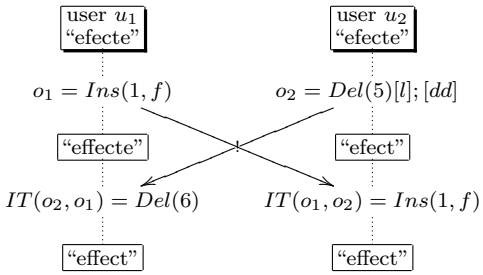

Figure 3.2 Integration with transformation.

3.2 Operational Transformation Approach

3.2.1 Background

OT considers n sites, where each site has a copy of the shared object which is a finite sequence of elements from a data type $\mathcal{A}$. It is assumed here that the shared object can only be modified by the following primitive operations:

$$\mathcal{O} = \{Ins(p, c) | c \in \mathcal{A} \text{ and } p \in \mathbb{N}\} \cup \{Del(p) | p \in \mathbb{N}\} \cup \{Nop()\}$$

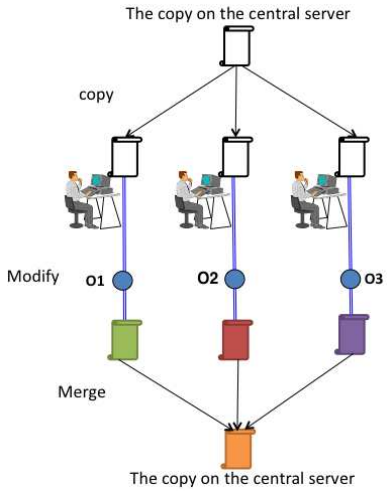

Figure 3.3 MV approach

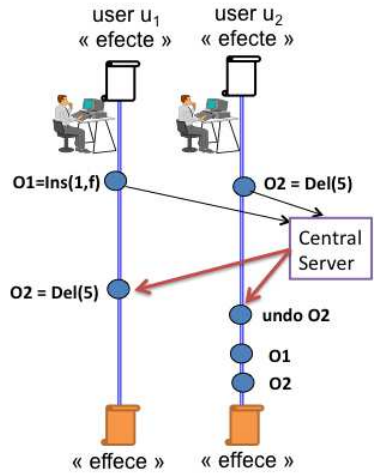

Figure 3.4 SRC approach

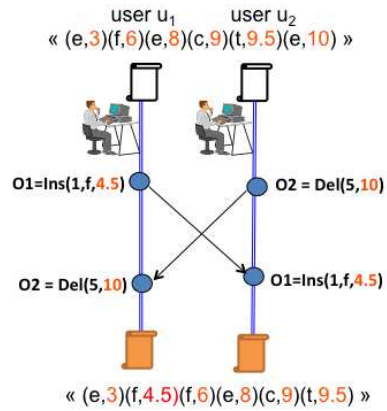

Figure 3.5 CRDT approach

where $Ins(p, c)$ inserts the element c at position p; $Del(p)$ deletes the element at position p, and $Nop()$ is the idle operation that has null effect on the object.

Each site can concurrently update its copy of the shared object. Its local updates are then propagated to other sites. When a site receives an update operation, it is first transformed before its execution. Since the shared object is replicated, each site will own a local state l that is altered only by operations executed locally. The initial state of the shared object, denoted l_0, is the same for all sites. Let $\mathcal{L}$ be the set of states. The function $Do : \mathcal{O} \times \mathcal{L} \rightarrow \mathcal{L}$, computes the state $Do(o, l)$ resulting from applying operation o to state l. We denote by $S = [o_1; o_2; \ldots; o_m]$ an operation sequence and $S = S_1 \bullet S_2$ a sequence obtained by concatenating sequences S_1 and S_2. Applying an operation sequence to a state l is defined as follows: (i) $Do^*([], l) = l$, where [] is the empty sequence and (ii) $Do^*(S \bullet [o], l) = Do(o, Do^*(S, l))$, S being a sequence of operations.

Two sequences of operations S_1 and S_2 are *equivalent*, denoted $S_1 \equiv S_2$, iff $Do^*(S_1, l) = Do^*(S_2, l)$ for every state l.

OT consists of the integration procedure and the transformation function (IT function). The integration procedure is in charge of executing update operations, broadcasting local update operations to other sites, receiving update operations from other sites, and determining transformations to be performed on a received operation before its execution. The transformation function transforms an update operation o with regard to another update operation o' ($IT(o, o')$). Let $S = [o_1; o_2; \ldots; o_m]$ be a sequence of operations. Transforming any update operation o with regard to S, denoted $IT^*(o, S)$, is recursively defined by:
$IT^*(o, []) = o$ and
$IT^*(o, [o_1; o_2; \ldots; o_m]) = IT^*(IT(o, o_1), [o_2; \ldots; o_m])$.

By definition: $IT(Nop(), o) = Nop()$ and $IT(o, Nop()) = o$ for every operation o.

3.2.2 Decentralized integration procedures

The integration procedure is based on two notions: concurrency and dependency of operations. Let o_1 and o_2 be operations generated at sites i and j, respectively. Operation o_2 is said to *causally dependent* on o_1, denoted $o_1 \rightarrow o_2$, iff: (i) $i = j$ and o_1 was generated before o_2; or, (ii) $i \neq j$ and the execution of o_1 at site j happened before the generation of o_2. Operations o_1 and o_2 are said to be *concurrent*, denoted $o_1 \parallel o_2$, iff neither $o_1 \rightarrow o_2$ nor $o_2 \rightarrow o_1$.

As a long established convention in OT-based collaborative editors [Ellis et Gibbs (1989), Sun et Ellis (1998)], the *timestamp vectors* are used to determine the causality and concurrency relations between operations. A timestamp vector is associated with each site and each generated operation. Every timestamp is a vector of integers with a number of entries equal to the number of sites. For a site j, each entry $V_j[i]$ returns the number of operations generated at site i that have been already executed on site j. When an operation o is generated at site i, a copy W_o of V_i is associated with o before its broadcast to other sites. The entry $V_i[i]$ is then incremented by 1. Once o is received at site j, if the local vector V_j "dominates"[3] W_o, then o is ready to be executed on site j. In this case, $V_j[i]$ will be incremented by 1 after the execution of o. Otherwise, o's execution is delayed until the causality condition holds. Let W_{o_1} and W_{o_2} be timestamp vectors of o_1 and o_2, respectively. Using these timestamp vectors, the causality and concurrency relations are defined as follows:
(i) $o_1 \rightarrow o_2$ iff W_{o_2} dominates W_{o_1} and $W_{o_1} \neq W_{o_2}$.
(ii) $o_1 \parallel o_2$ iff neither W_{o_1} dominates W_{o_2} nor W_{o_2} dominates W_{o_1}.

In the decentralized integration procedures, every site generates operations sequentially and stores these operations in a stack also called a *history*. When a site receives a remote operation o, the integration procedure executes the following steps:

1. From the local history S, it determines the equivalent sequence S' that is the concatenation of two sequences S_h and S_c where (i) S_h contains all operations happened before o (according to the causality relation), and (ii) S_c consists of operations that are concurrent to o.
2. It calls the transformation component in order to get operation o' that is the transformation of o according to S_c (*i.e.* $o' = IT^*(o, S_c)$).
3. It executes o' on the current state and then adds o' to local history S.

3. Let V_1 and V_2 be two timestamp vectors. We say that V_1 dominates V_2 iff for each site i, $V_1[i] \geq V_2[i]$.

The integration procedure allows history of executed operations to be built on every site, provided that the causality relation is preserved. When all sites have executed the same set of operations (stable states), their histories are not necessarily identical because the concurrent operations may be executed in different orders. Nevertheless, they must be equivalent in the sense that they must lead to the same final state.

3.2.3 Inclusive transformation functions

We can find, in the literature, several IT functions: *Ellis*'s algorithm [Ellis et Gibbs (1989)], *Ressel*'s algorithm [Ressel *et al.* (1996)], *Sun*'s algorithm [Sun *et al.* (1998)], *Suleiman*'s algorithm [Suleiman *et al.* (1997)] and *Imine*'s algorithm [Imine *et al.* (2003)]. They differ in the manner that conflict situations are managed. A conflict situation occurs when two concurrent operations insert, at the same position, different characters. To deal with such conflicts, all these algorithms, except the one proposed by Sun et *al.*, add some extra parameters to the insert operation signature.

Ellis's algorithm

Ellis and Gibbs [Ellis et Gibbs (1989)] are the pioneers of OT approach. They extend operation Ins with another parameter pr representing its priority. Always, concurrent operations have different priorities. The four transformation cases for Ins and Del proposed by Ellis and Gibbs are illustrated in the appendix.

Ressel's algorithm

Ressel et *al.* [Ressel *et al.* (1996)] proposed an algorithm that provides two modifications to Ellis's algorithm. The first modification consists in replacing priority parameter pr by another parameter u, which is simply the *identifier* of the issuer site. Similarly, u is used for tie-breaking when a conflict occurs between two concurrent insert operations. As for the second modification, it concerns how a pair of insert operations is transformed. When two concurrent insert operations add at the same position two (identical or different) elements, only the insertion position of operation having a higher identifier is incremented. In other words, both elements are inserted even if they are identical. This is the opposite to the solution proposed by Ellis and Gibbs, which keeps only one element in case of identical concurrent insertions. Apart from these modifications, the other cases remain similar to those of Ellis and Gibb. The transformation cases given by the algorithm of Ressel et *al.* [Ressel *et al.* (1996)] can be found in the appendix.

Sun's algorithm

Sun et *al.* [Sun *et al.* (1998)] have designed another IT algorithm, which is slightly different in the sense that it is defined for stringwise operations. Indeed, the following operations are used: $Ins(p, s, l)$ (insert string s of length l at position p) and $Del(p, l)$ (delete string of length l from position p). To compare with other IT algorithms, we suppose that $l = 1$ for all update operations. The IT function in this case is reported in the appendix .

Suleiman's algorithm

Suleiman et *al.* [Suleiman *et al.* (1997)] proposed another solution that modifies the signature of insert operation by adding two parameters av and ap. For an insert operation $Ins(p, c, av, ap)$, av contains operations that have deleted a character before the insertion position p. The set ap contains operations that have removed a character after or at position p. When an insert operation is generated, the parameters av and ap are empty. They will be filled during transformation steps. The IT function of Suleiman and *al.* is given in the appendix. To resolve the conflict between two concurrent insert operations $Ins(p, c_1, av_1, ap_1)$ and $Ins(p, c_2, av_2, ap_2)$, three cases are possible:
1) $(av_1 \cap ap_2) \neq \emptyset$: character c_2 is inserted before character c_1,
2) $(ap_1 \cap av_2) \neq \emptyset$: character c_2 is inserted after character c_1,
3) $(av_1 \cap ap_2) = (ap_1 \cap av_2) = \emptyset$: in this case, characters c_1 and c_2 are compared (for instance according to the lexicographic order) to choose the one to be added before the other. Like the site identifiers and priorities, parameters av, ap, comparison of characters are used to tie-break conflict situations. Note that when two concurrent operations insert the same character (*e.g.* $c_1 = c_2$) at the same position, only one is executed. The other one is ignored by returning the idle operation $Nop()$. In other words, like the solution of Ellis and Gibb [Ellis et Gibbs (1989)], only one character is kept.

Imine's algorithm

In [Imine *et al.* (2003)], Imine and *al.* proposed another IT algorithm which again enriches the signature of insert operation with parameter ip which is the initial (or the original) insertion position given at the generation stage. Thus, when transforming a pair of insert operations having the same current position, they compare first their initial positions in order to recover the position relation at the generation phase. If the initial positions are identical, then like Suleiman and *al.* [Suleiman *et al.* (1997)] they compare symbols to tie-break an eventual conflict. The IT function of Imine can be found in the appendix.

3.2.4 Consistency criteria

An OT-based collaborative editor is *consistent* iff it satisfies the following properties:

1. *Causality preservation:* if $o_1 \rightarrow o_2$ then o_1 is executed before o_2 at all sites.
2. *Convergence:* when all sites have performed the same set of updates, the copies of the shared document are identical.

To preserve the causal dependency between update operations, timestamp vectors are used. In [Ressel *et al.* (1996)], the authors have established two properties $TP1$ and $TP2$ that are necessary and sufficient to ensure data convergence for *any number* of operations executed in *arbitrary order* on copies of the same object: For all o_1, o_2 and o_3 pairwise concurrent operations defined on the same state (initial state or state reached from the initial state by executing equivalent sequences):

– $TP1$: $[o_1\,; IT(o_2, o_1)] \equiv [o_2\,; IT(o_1, o_2)]$.

– $TP2$: $IT^*(o_3, [o_1\,; IT(o_2, o_1)]) = IT^*(o_3, [o_2\,; IT(o_1, o_2)])$.

Property $TP1$ defines a *state identity* and ensures that if o_1 and o_2 are concurrent, the effect of executing o_1 before o_2 is the same as executing o_2 before o_1. Property $TP2$ ensures that transforming o_3 along equivalent and different operation sequences will give the same operation. By abuse of language, an IT function satisfying properties TP1 and TP2 is said to be consistent.

Accordingly, by these properties, it is not necessary to enforce a global total order between concurrent operations because data divergence can always be repaired by operational transformation. However, finding an IT function that satisfies $TP1$ and $TP2$ is considered to be a hard task, because this proof is often unmanageably complicated.

All IT functions proposed, in the literature, do not ensure data converge [Imine *et al.* (2006), Boucheneb et Imine (2009), Boucheneb *et al.* (2010)]. We report, in the following, a counterexample for each IT function.

IT functions of Ellis and Sun do not satisfy the property TP1 (see Figure 3.6 and Figure 3.7) [Imine *et al.* (2003)]. The pairs of concurrent operations violating TP1 are $(o_1 = Ins(1, f, pr_1), o_2 = Del(1))$ and $(o_1 = Ins(1, f), o_2 = Ins(1, e))$, respectively.

Suleiman's IT satisfies neither TP1 nor TP2 [Imine *et al.* (2003), Boucheneb *et al.* (2010)]. The counterexample for TP1 is given by the pair of operations $(o'_1 = Ins(2, f, \{o_3\}, \{o_5\}), o'_2 = Ins(2, c, \{o_5\}, \{o_3\}))$. The corresponding scenario, reported at Figure 3.8, consists of 4 users u_1, u_2, u_3 and u_4 on different sites. Users u_1, u_2 and u_3 have generated and executed locally sequences $S_1 = [o_1 = Ins(3, f, \emptyset, \emptyset)]$, $S_2 = [o_2 = Ins(2, c, \emptyset, \emptyset)]$ and $S_3 = [o_3 = Del(2); o_4 = Ins(2, e, \emptyset, \emptyset); o_5 = Del(2)]$, respectively. Then, user u_3 receives successively operations o_1 and o_2. User u_4 receives consecutively operations of S_3, o_2 and o_1. The IT function of

Suleiman fails to ensure convergence. Property TP1 is violated for $o'_1 = IT^*(o_1, S_3) = Ins(3, f, \{o_3\}, \{o_5\})$ and $o'_2 = IT^*(o_2, S_3) = Ins(2, c, \{o_5\}, \{o_3\})$.

Ressel's IT does not satisfy TP2 but satisfies TP1 [Imine *et al.* (2003), Boucheneb *et al.* (2010)]. In Figure 3.9, we report a scenario violating property TP2 for the triplet of concurrent operations ($o_1 = Del(1), o_2 = Ins(2, c_2, u_2)$, $o_3 = Ins(1, c_3, u_3)$).

Imine's IT function satisfies TP1 but does not satisfy TP2. In Figure 3.10, we report a scenario violating TP2. In this scenario, there are 4 users u_1, u_2, u_3 and u_4 on different sites. Users u_1, u_2 and u_3 have generated sequences $S_1 = [o_1 = Del(2)]$, $S_2 = [o_0 = Del(2); o_2 = Ins(2, c, 2)]$ and $S_3 = [o_3 = Ins(2, e, 2)]$, respectively. User u_2 executes operations o_0 and o_2 then it receives successively operations o_1 and o_3. User u_4 receives successively operations o_0, o_1, o_2 and o_3. For this scenario, the IT function of Imine fails to ensure convergence for copies of users u_2 and u_4. The property $TP2$ is violated for $o'_1 = IT(o_1, o_0)$, o_2 and $o'_3 = IT(o_3, o_0)$ (see Figure 3.10).

3.2.5 Avoiding Property $TP2$

The violation of property $TP2$ gave birth to several works that have tried to avoid this problem, often at the expense of genericity and efficiency. These works may be categorized in two approaches.

The first approach falls in the category of works, such as GOT [Sun *et al.* (1998)], SOCT4 [Vidot *et al.* (2000)] and COT [Sun et Sun (2009)], which enforce a total order on operations in order to maintain the same transformation path at all sites. Although these algorithms ensure data convergence, they do not allow a high concurrency degree because of the global order of execution. In the second approach, some works build a particular class of transformation paths. For instance, OPTIC [Imine (2006), Imine (2009)] and ABT [Li et Li (2010)] integrate by transformation remote operations on logs organized in such a way that insertion operations are always before deletion operations. Building these transforma-

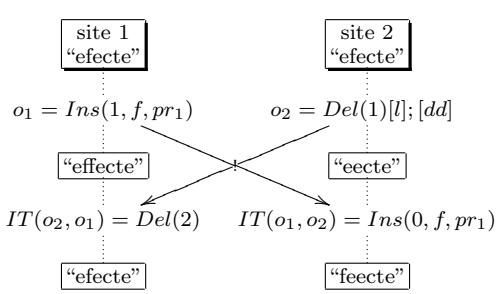

Figure 3.6 Violation of TP1 for Ellis's IT.

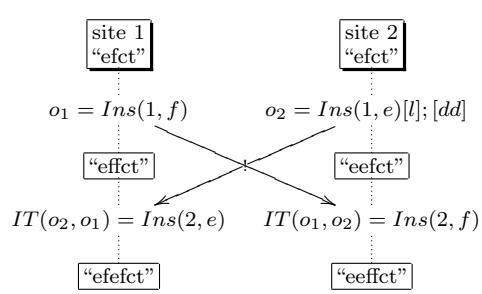

Figure 3.7 Violation of TP1 for Sun's IT.

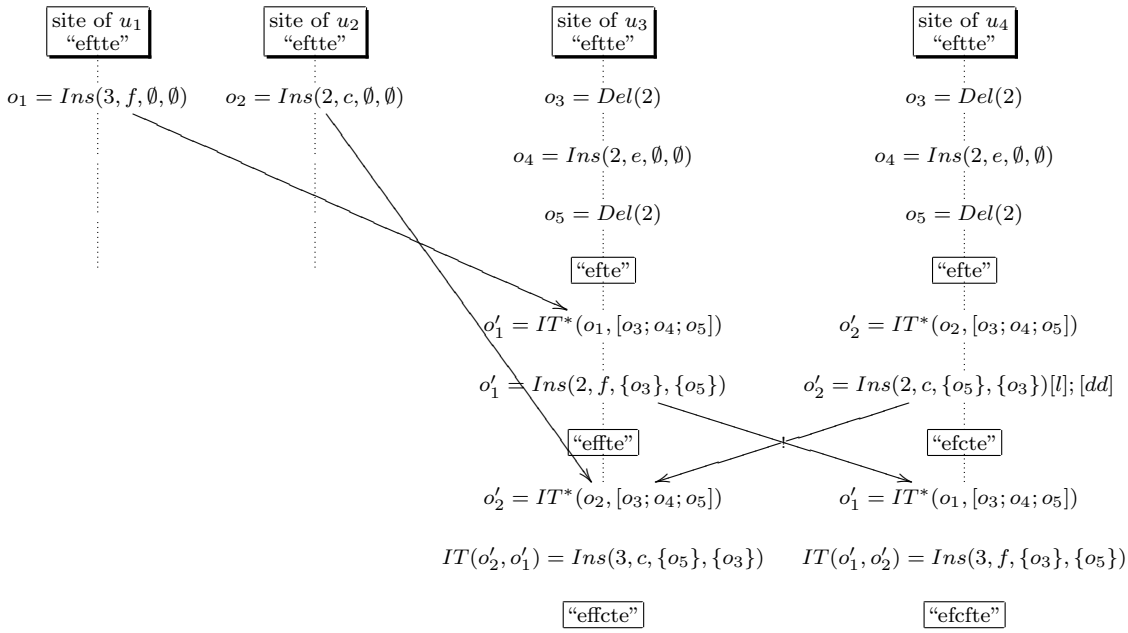

Figure 3.8 Violation of TP1 for Suleiman's IT.

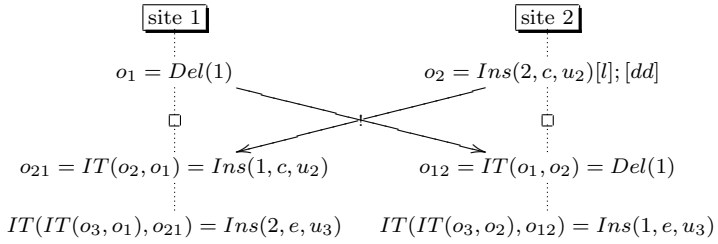

Figure 3.9 Violation of TP2 for Ressel's IT (in case $u_2 < u_3$).

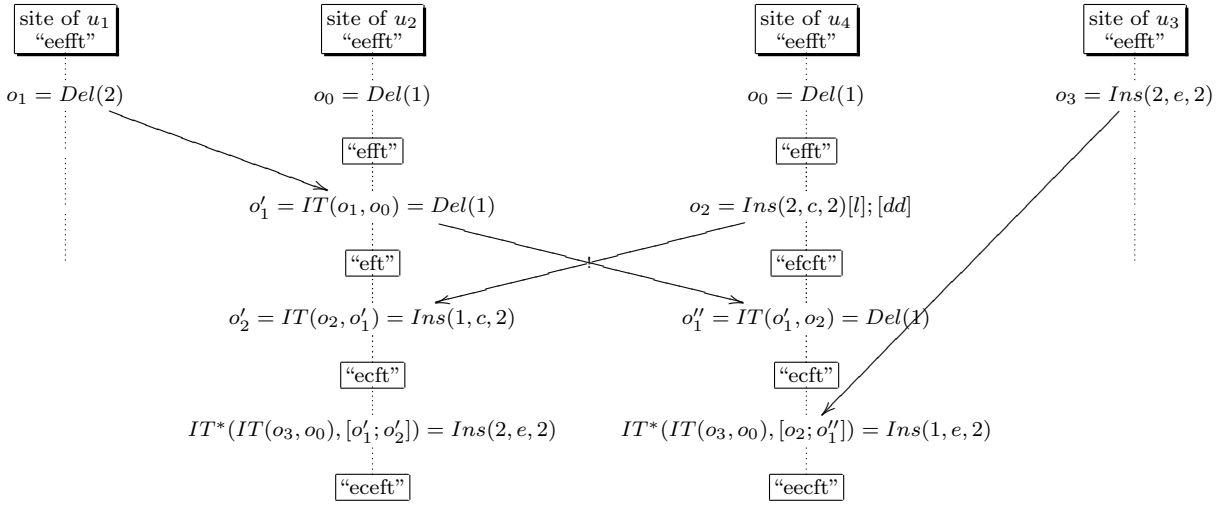

Figure 3.10 Violation of TP2 for Imine's IT.

tions needs, unfortunately, to devise new integration algorithms. Other works are based on changing the semantics of the delete operation [Oster *et al.* (2006a)]. The delete operation does not remove symbols but make them invisible (see Section 3.4.5 for comparison with the approach proposed here).

3.3 Controller synthesis of consistent IT function

In [Imine *et al.* (2006), Boucheneb et Imine (2009), Boucheneb *et al.* (2010)], the authors used a theorem proving or a model checking technique to verify whether or not a given IT is consistent. They showed that all IT functions, proposed in the literature, do not ensure consistency and a counterexample is provided for each IT function. From this fact, a question arises concerning the existence of consistent IT functions. Model checking and theorem proving are useful to prove whether or not a given system satisfies some properties but not appropriate to verify whether or not there exist a system, which satisfies some properties. The controller synthesis techniques solve a more general problem than the model checking and theorem proving techniques, since they verify whether or not the system can be modified (forced) so as to meet the properties of interest. In such a framework, the system consists, in general, of controllable and uncontrollable transitions. The control objective is to find, if it exists, a strategy to force the properties of interest, by choosing appropriately controllable actions to be executed, no matter what uncontrollable transitions are executed.

We are interested in applying the principle of controller synthesis to design an IT function which satisfies properties TP1 and TP2. The idea is to look for consistent IT in the set of allowed functions. There are four allowed transformations for each operation o: 1) increment its position, 2) decrement its position, 3) transform it into an idle operation ($Nop()$), and 4) leave it unchanged. We first investigate whether or not there exist in such a set, some IT functions which satisfy property TP1. If it is the case, we investigate whether or not there exist some IT functions, among those satisfying TP1, which also satisfy TP2.

For these investigations, we use the game automata formalism 'à la UPPAAL' [Cassez *et al.* (2005)]. A game automaton is an automaton with two kinds of transitions: controllable and uncontrollable. Each transition has a source location, a destination location and is annotated with selections, guards and blocks of actions. Selections bind non-deterministically a given identifier to every value in a given range. The other labels of a transition are within the scope of this binding. A state is defined by the current location and the current values of all variables. A transition is enabled in a state iff the current location is the source location of the transition and its guard evaluates to true. The firing of the transition consists in reaching its destination location and executing atomically its block of actions. The side effect of this

block changes the state of the system. To force some properties, the enabled transitions that are controllable can be delayed or simply ignored. However, the uncontrollable transitions can neither be delayed nor ignored.

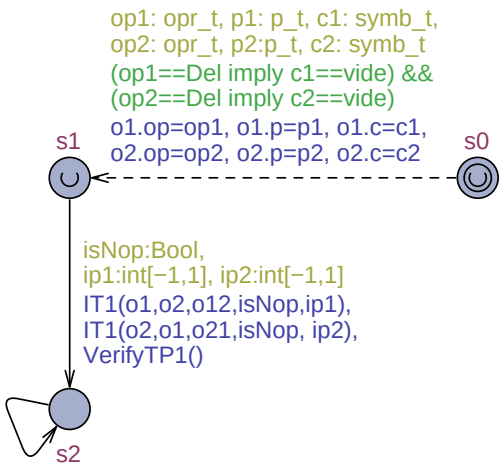

Figure 3.11 Synthesize an IT for TP1

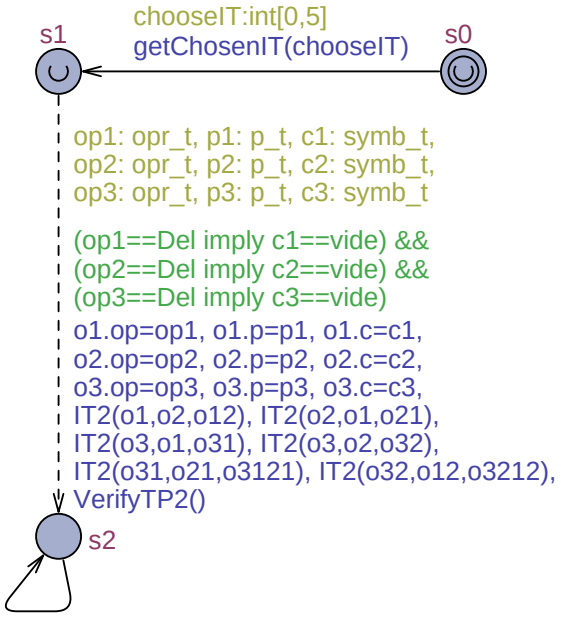

Figure 3.12 Synthesize a consistent IT

3.3.1 Do there exist IT functions which satisfy TP1?

An IT function satisfies property $TP1$ iff for any pair of concurrent operations o_1 and o_2 defined on the same state, it holds that $[o_1; IT(o_2; o_1)] \equiv [o_2; IT(o_1, o_2)]$. To verify whether or not there are some IT functions which satisfy property TP1, we have represented in the game automaton, depicted at Figure 3.11, the generation of operations o_1 and o_2, the computation of $IT(o_1; o_2)$ and $IT(o_2, o_1)$, and the verification of $[o_1; IT(o_2; o_1)] \equiv [o_2; IT(o_1, o_2)]$. The generation of operations is specified by the uncontrollable transition (s_0, s_1), since we have no control on the kinds of operations generated by users. The operational transformations and the verification of TP1 are represented by the controllable transition (s_1, s_2). The model starts by selecting two operations o_1 and o_2. The domain of operations is fixed so as to cover all cases of transformations. Afterwards, the model chooses two transformations to apply to o_1 with regard to o_2 and o_2 with regard to o_1 and applies them by invoking function $IT1$. Function $IT1(o_1, o_2, o_{12}, IsNop, ip_1)$ returns in o_{12} the result of transformation of o_1 with regard to o_2. If $IsNop = true$ then $o_{12} = Nop()$, otherwise the transformation of o_1 consists in updating the parameter position $(o_{12}.p = o_1.p + ip_1)$. It means that 4 possibilities

are offered for transforming an operation o_1 with regard to another operation o_2: $Nop()$, decrementing, maintaining, or incrementing the position of o_1. Finally, the model verifies whether or not the property TP1 is satisfied. No matter what operations o_1 and o_2 generated by the uncontrollable transition, the controller synthesis aims to force property TP1 by choosing appropriately the operational transformations.

We have used the tool *Uppaal-Tiga* [Cassez *et al.* (2005)] to verify whether or not there exist some IT functions, which satisfy TP1. The safety control objective for TP1 is $AG\ TP1$, where $TP1$ is defined in the model as a boolean variable whose value is *true* while the property TP1 is satisfied. The boolean variable TP1 is set to false by the function VerifyTP1 if $[o_1; IT(o_2, o_1)] \not\equiv [o_2; IT(o_1, o_2)]$. *Uppaal-Tiga* concludes that the property is satisfied, which means that there is, at least, a strategy to force property TP1. We report in Table 3.1 the different IT functions (satisfying TP1) extracted from the output file of the tool *verifytga* of *Uppaal-Tiga*.

Note that even if some operational transformations satisfy TP1, they are unacceptable from the semantic point of view. For instance, if $p_1 = p_2$, the operational transformations $IT(Del(p_1), Del(p_2)) = Del(p_1 - 1)$, $IT(Del(p_1), Del(p_2)) = Del(p_1)$ and $IT(Del(p_1), Del(p_2)) = Del(p_1 + 1)$ mean that if two users generate concurrently the same delete operation, two symbols will be deleted in each site, which is unacceptable from the semantic point of view. The only operational transformation which makes sense for this case is $IT(Del(p_1), Del(p_2)) = Nop()$. It means that only the symbol at position p_1 is deleted in each site. After eliminating these incoherent operational transformations, there remain 2 possibilities for $IT(Ins(p_1, c_1), Ins(p_2, c2)), p_1 = p_2, c_1 \neq c_2$, and 3 possibilities for $IT(Ins(p_1, c_1), Ins(p_2, c2)), p_1 = p_2, c_1 = c_2$. Therefore, we can extract 6 IT functions which satisfy TP1. These IT functions differ in the way that conflicting operations are managed.

3.3.2 Do there exist IT functions which satisfy TP1 and TP2?

An IT function satisfies property $TP2$ iff for any triplet of pairwise concurrent operations o_1, o_2 and o_3 defined on the same state, it holds that $IT(IT(o_3, o_1), IT(o_2, o_1)) = IT(IT(o_3, o_2), IT(o_1, o_2))$. To verify whether or not there are some IT functions which satisfy properties TP1 and TP2, we have used the game automaton depicted at Figure 3.12. This model starts by selecting an IT function, which satisfies property TP1 (the range of *chooseIT* corresponds to the 6 IT functions satisfying TP1). Afterwards, it selects three operations o_1, o_2 and o_3, and performs the transformations needed to verify TP2. Function $IT2(o_1, o_2, o_{12})$ applies the selected IT function to o_1 with regard to o_2 and returns the result of this transformation in o_{12}. Finally, the model calls function VerifyTP2. The control aims to force to choose the appropriate IT function so as to satisfy property TP2. The control

Table 3.1 IT functions supplied by *Uppaal-Tiga* for TP1 and classical signatures of update operations

o_1	o_2	$Cnd(p_1, p_2, c_1, c_2)$	$IT(o_1, o_2)$	$IT(o_2, o_1)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 < p_2$	$Ins(p_1, c_1)$	$Ins(p_2 + 1, c_2)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 = p_2 \wedge c_1 < c_2$	$Ins(p_1 + 1, c_1)$	$Ins(p_2, c_2)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 = p_2 \wedge c_1 < c_2$	$Ins(p_1, c_1)$	$Ins(p_2 + 1, c_2)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 = p_2 \wedge c_1 = c_2$	$Ins(p_1 + 1, c_1)$	$Ins(p_2 + 1, c_2)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 = p_2 \wedge c_1 = c_2$	$Ins(p_1, c_1)$	$Ins(p_2, c_2)$
$Ins(p_1, c_1)$	$Ins(p_2, c_2)$	$p_1 = p_2 \wedge c_1 = c_2$	$Nop()$	$Nop()$
$Del(p_1)$	$Del(p_2)$	$p_1 < p_2$	$Del(p_1)$	$Del(p_2 - 1)$
$Del(p_1)$	$Del(p_2)$	$p_1 = p_2$	$Del(p_1 - 1)$	$Del(p_2 - 1)$
$Del(p_1)$	$Del(p_2)$	$p_1 = p_2$	$Del(p_1 + 1)$	$Del(p_2 + 1)$
$Del(p_1)$	$Del(p_2)$	$p_1 = p_2$	$Del(p_1)$	$Del(p_2)$
$Del(p_1)$	$Del(p_2)$	$p_1 = p_2$	$Nop()$	$Nop()$
$Ins(p_1, c_1)$	$Del(p_2)$	$p_1 < p_2$	$Ins(p_1, c_1)$	$Del(p_2 + 1)$
$Ins(p_1, c_1)$	$Del(p_2)$	$p_1 = p_2$	$Ins(p_1, c_1)$	$Del(p_2 + 1)$
$Del(p_1)$	$Ins(p_2, c_2)$	$p_1 < p_2$	$Del(p_1)$	$Ins(p_2 - 1, c_2)$
$Del(p_1)$	$Ins(p_2, c_2)$	$p_1 = p_2$	$Del(p_2 + 1)$	$Ins(p_1, c_1)$

objective is specified by the CTL formula $AG\ TP2$, where $TP2$ is a boolean variable whose value is *true* while the property TP2 is satisfied. This variable is set to false by the function VerifyTP2 if $IT(IT(o_3, o_1), IT(o_2, o_1)) \neq IT(IT(o_3, o_2), IT(o_1, o_2))$.

Uppaal-Tiga concludes that the property $AG\ TP2$ cannot be forced, which means that there is no strategy to force property TP2. In other words, there is no IT function, based on classical parameters of delete and insert operations, which satisfies both TP1 and TP2. We investigated why a consistent IT function does not exist for the basic parameters of delete and insert operations. This investigation led to the identification of two symbolic pairwise scenarios which prevent a consistent IT function from being obtained. We report in Figure 3.13 and Figure 3.14 these two pairwise sequences named scenario 1 and scenario 2, respectively. For scenario 1, to verify TP2, the performed transformations are:

$o_{21} = IT(o_2, o_1) = IT(Ins(p_1, c_2), o_1) = Ins(p_1, c_2)$,

$o_{12} = IT(o_1, o_2) = IT(Del(p_1), Ins(p_1, c_2)) = Del(p_1 + 1)$,

$o_{31} = IT(o_3, o_1) = Ins(p_1, c_3)$,

$o_{32} = IT(o_3, o_2) = Ins(p_1 + 2, c_3)$,

$IT(o_{32}, o_{12}) = IT(Ins(p_1 + 2, c_3), Del(p_1 + 1)) = Ins(p_1 + 1, c_3)$ and $IT(o_{31}, o_{21}) = IT(Ins(p_1, c_3), Ins(p_1, c_2))$.

For the last transformation, we have different possibilities (see Table 3.1). To satisfy TP2, we must choose $IT(Ins(p_1, c_3), Ins(p_1, c_2)) = Ins(p_1 + 1, c_3)$.

For scenario 2, the performed transformations are:

$o_{21} = IT(o_2, o_1) = Ins(p_1, c_2)$,

$o_{12} = IT(o_1, o_2) = Del(p_1)$,
$o_{31} = IT(o_3, o_1) = Ins(p_1, c_3)$,
$o_{32} = IT(o_3, o_2) = Ins(p_1, c_3)$,
$IT(o_{32}, o_{12}) = IT(Ins(p_1, c_3), Del(p_1)) = Ins(p_1, c_3)$ and $IT(o_{31}, o_{21}) = IT(Ins(p_1, c_3), Ins(p_1, c_2))$.
To satisfy TP2, for the last operational transformation, we must use $IT(Ins(p_1, c_3), Ins(p_1, c_2)) = Ins(p_1, c_3)$.

Consequently, a consistent IT function, if it exists, must have additional parameters in its operation signatures. We have seen, in the previous section, different IT functions based on extending the insert signature with priority, issuer site, initial position or sets of delete operations before and after the inserting position. We have provided for each IT function a divergent scenario violating either TP1 or TP2. It means that the suggested additional parameters are insufficient or inappropriate to ensure convergence. Indeed, adding priority (as in Ellis's IT) or owner identifier (as in Ressel's IT) to the insert signature fails to ensure convergence for scenarios 1 and 2. Scenario 1 violates TP1 for Ellis's IT (see Figure 3.6). Scenario 2 violates TP2 for Ressel's IT (see Figure 3.9). For Suleiman's IT and Imine's IT, scenarios 1 and 2 satisfy TP1 and TP2 but the added parameters introduce other cases of divergence (see Figure 3.8 and Figure 3.10). All these failed tentatives show that designing a consistent IT function is a hard task.

The main difference between both scenarios resides in the position of the deleted symbol relatively to the symbols inserted by o_2 and o_3 (see Figure 3.13 and Figure 3.14). In scenario 1, the deleted symbol is before the inserting position of o_3, whereas, in scenario 2, it is before the inserting position of o_2. Extending the signature of the insert operation with the number of symbols deleted before its position allows the definition of appropriate transformations so as both scenarios 1 and 2 satisfy TP1 and TP2 (see Figure 3.15 and Figure 3.16). Indeed, in scenario 1, the symbol of $o_{31} = IT(o_3, o_1) = Ins(p_1, c_3, 1)$ should precede the one of $o_{21} = Ins(p_1, c_2, 0)$, as o_{31} has the biggest number of symbols deleted before its position ($1 > 0$). In scenario 2, $o_{21} = IT(o_2, o_1) = Ins(p_1, c_2, 1)$ has the greatest number of symbols deleted before its position. Its symbol should be inserted after the one of $o_{31} = Ins(p_1, c_3, 0)$. The idea of our IT function is to use this extra parameter to deal with conflicting operations. In the following, we propose based on this idea an IT function and show formally its consistency.

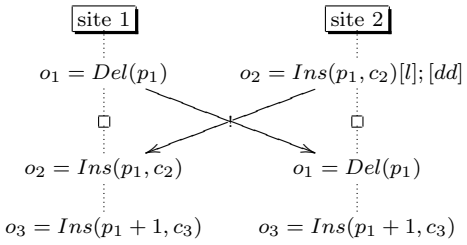

Figure 3.13 Scenario 1

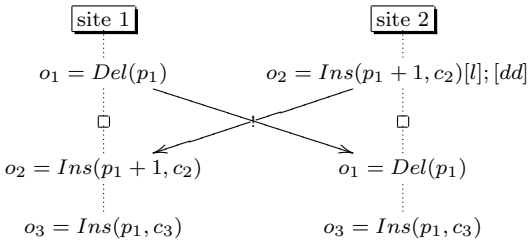

Figure 3.14 Scenario 2

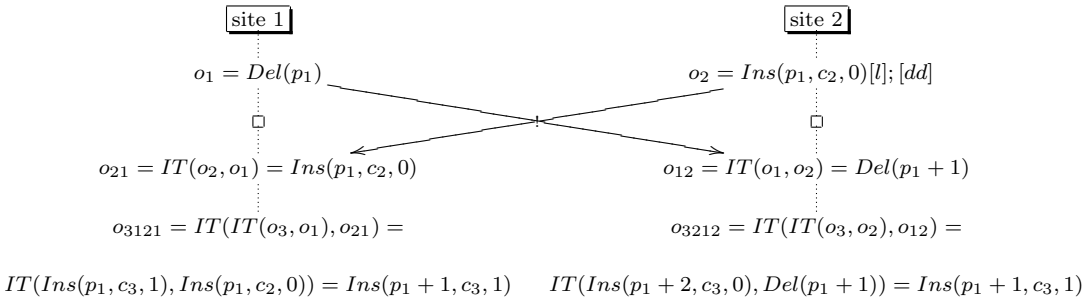

Figure 3.15 Applying our IT to scenario 1

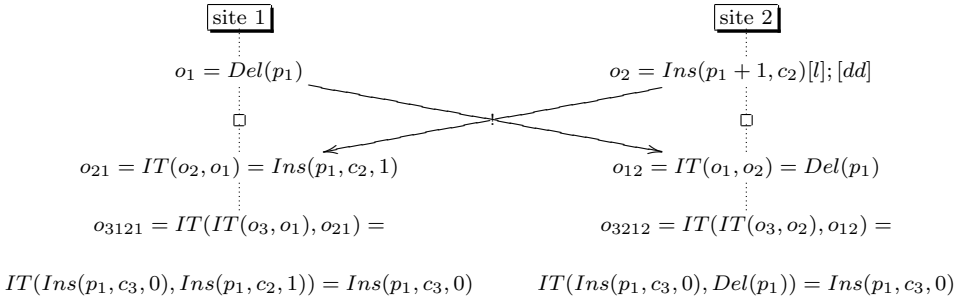

Figure 3.16 Applying our IT to scenario 2

3.4 A consistent IT function

3.4.1 Extending the insert signature with an extra parameter

We propose to add a new parameter, named nd, to the insert operation signature. This extra parameter is filled with the number of symbols deleted before the inserting position. When an insert operation o is generated, its parameter nd is set appropriately to the number of symbols deleted before the position of o. Afterwards, this parameter is incremented whenever it is transformed with regard to a delete operation whose position is before its inserting position.

Définition 3.1 *Let S be a sequence of operations executed on the shared document before generating an insert operation o. Let p be the inserting position of o, we denote $ND(S,p)$ the number of symbols deleted before p by the sequence S. The parameter nd of o set to $ND(S,p)$ is inductively defined as follows:*

$$ND(S,p) = \begin{cases} 0 & \text{if } S = [] \\ ND(S',p) & \text{if } S = S' \bullet [o'] \wedge p < p' \\ nd' & \text{if } S = S' \bullet [Ins(p',c',nd')] \wedge \ p = p' \\ ND(S',p-1) & \text{if } S = S' \bullet [Ins(p',c',nd')] \ \wedge \ p > p' \\ ND(S',p+1)+1 & \text{if } S = S' \bullet [Del(p')] \ \wedge \ p \geq p' \end{cases}$$

where $o' \in \{Ins(p',c',nd'), Del(p')\}$

Intuitively, if S is empty, it is trivial that there is no symbol deleted by S. Then, $ND([],p) = 0$. For $S = S' \bullet [Ins(p',c',nd')]$, the symbol of $Ins(p',c',nd')$ is inserted at position p'. If $p < p'$, the execution of $Ins(p',c',nd')$ does not affect the part before position p' (i.e., $ND(S,p) = ND(S',p)$). If $p = p'$, the symbol c is inserted at position $p = p'$, shifts c' to position $p+1$ but does not affect the number of symbols deleted before position p' (i.e., $ND(S,p) = nd'$). If $p > p'$, the symbol at position $p-1$ before executing $Ins(p',c',nd')$ will be shifted to position p (i.e., $ND(S,p) = ND(S',p-1)$). Similarly, for $S = S' \bullet [Del(p')]$, the execution of $Del(p')$ will delete the symbol at position p'. If $p < p'$, the execution of $Del(p')$ does not affect the part before position p' (i.e., $ND(S,p) = ND(S',p)$). Otherwise, symbols deleted before p by S are symbols deleted before $p+1$ by S' plus the one deleted by $Del(p')$ (i.e., $ND(S,p) = ND(S',p+1)+1$).

As an example, let us compute the number of symbols deleted before position 2, by the sequence $[Del(2); Ins(2,a); Ins(1,b)]$ (i.e., $ND([Del(2); Ins(2,a); Ins(1,b)],2)$). This

number is equal to the number of deleted symbols by $[Del(2); Ins(2,a)]$ before position 1, because $Ins(1,b)$ will shift the symbol at position 1 to position 2. Therefore, $ND([Del(2); Ins(2,a); Ins(1,b)], 2) = ND([Del(2); Ins(2,a)], 1)$. The number of deleted symbols by $[Del(2); Ins(2,a)]$ before position 1 is equal to $ND([Del(2)], 1)$ as $Ins(2,a)$ does not affect symbols before position 2. Finally, since $1 < 2$, it follows that $ND([Del(2)], 1) = ND([], 1) = 0$. Therefore, there is no symbol deleted by $[Del(2); Ins(2,a); Ins(1,b)]$ before position 2. Consider now the sequence $[Ins(2,a); Del(3); Ins(1,b)]$, which is equivalent to the previous one. Let us compute $ND([Ins(2,a); Del(3); Ins(1,b)], 2)$, using Definition 3.1:
$ND([Ins(2,a); Del(3); Ins(1,b)], 2) = ND([Ins(2,a); Del(3)], 1) =$
$ND([Ins(2,a)], 1) = ND([], 1) = 0$. Note that, the number of deleted symbols before position 2 is the same for the two equivalent sequences.

Since in OT, operations are, in general, executed after integration in different orders, to ensure consistency, our computation procedure of ND should give the same result for all equivalent sequences and any position p (i.e., $ND(S,p) = ND(S',p)$, for any pair of equivalent sequences S and S'). We first define our IT function then we establish some nice properties of ND which will be useful to show consistency of our IT function.

3.4.2 Our IT function

$$\mathrm{IT}(Ins(p_1,c_1,nd_1), Ins(p_2,c_2,nd_2)) = \begin{cases} Ins(p_1,c_1,nd_1) & \text{if } \mathrm{p_1} < \mathrm{p_2} \vee (\mathrm{p_1} = \mathrm{p_2} \wedge \mathrm{nd_1} < \mathrm{nd_2}) \\ & \vee (p_1 = p_2 \wedge nd_1 = nd_2 \wedge c_1 \leq c_2) \\ Ins(p_1+1,c_1,nd_1) & \text{if } \mathrm{p_1} > \mathrm{p_2} \vee (\mathrm{p_1} = \mathrm{p_2} \wedge \mathrm{nd_1} > \mathrm{nd_2}) \\ & \vee (p_1 = p_2 \wedge nd_1 = nd_2 \wedge c_1 > c_2) \end{cases}$$

$$\mathrm{IT}(Ins(p_1,c_1,nd_1), Del(p_2)) = \begin{cases} Ins(p_1,c_1,nd_1) & \text{if } \mathrm{p_1} \leq \mathrm{p_2} \\ Ins(p_1-1,c_1,nd_1+1) & \text{otherwise} \end{cases}$$

$$\mathrm{IT}(Del(p_1), Ins(p_2,c_2,nd_2)) = \begin{cases} Del(p_1) & \text{if } \mathrm{p_1} < \mathrm{p_2} \\ Del(p_1+1) & \text{otherwise} \end{cases} \qquad \mathrm{IT}(Del(p_1), Del(p_2)) = \begin{cases} Del(p_1) & \text{if } \mathrm{p_1} < \mathrm{p_2} \\ Del(p_1-1) & \text{if } \mathrm{p_1} > \mathrm{p_2} \\ Nop() & \text{otherwise} \end{cases}$$

Figure 3.17 The proposed IT function

In our IT function, the signatures of the insert and delete operations are $Ins(p,c,nd)$ and $Del(p)$, respectively, where p is a position, c is a symbol and nd is the number of symbols deleted before position p. The conflicting situations between two concurrent insert operations are handled using their extra parameters nd, in a similar way as in other IT functions. More precisely, when transforming a pair of insert operations having the same current position, their parameters nd are first compared in order to recover the position relation at the generation

phase. If their parameters nd are equal, then their symbols are compared to tie-break an eventual conflict. The IT function proposed here is reported at Figure 3.17.

3.4.3 Relationships between positions and the extra parameters

We establish, in Lemma 1 and Theorem 1, some relationships between positions of insert operations generated on the same state and their parameters nd. We first suppose, in Lemma 1, that the operations are generated after executing the same sequence of operations. Then, we consider, in Theorem 1, the case where the operations are generated after executing different but equivalent sequences (i.e., they consist of the same set of original operations executed, after integration, in different orders). Intuitively, these relationships mean that for any pair of insert operations generated on the same state, the order relation of their positions is the same as the order relation of their extra parameters nd.

Lemma 1 *Let $o_1 = Ins(p_1, c_1, nd_1)$ and $o_2 = Ins(p_2, c_2, nd_2)$ be two insert operations generated on the same state, just after executing the same sequence of operations S. Then: $p_1 < p_2 \Rightarrow nd_1 \leq nd_2$.*

Proof 1 *By definition, $nd_1 = ND(S, p_1)$ and $nd_2 = ND(S, p_2)$. We show by induction on the length of S that $p_1 < p_2 \Rightarrow ND(S, P_1) \leq ND(S, P_1)$. For $S = []$, by definition, $ND(S, p_1) = ND(S, p_2)$. For $S = S' \bullet [o']$, assume that $p_1 < p_2 \Rightarrow ND(S', p_1) \leq ND(S', p_2)$ and let us show that $ND(S, p_1) \leq ND(S, p_2)$. Let p' be the position of o'. We consider 9 cases:*
1) $p_1 < p_2 < p'$: By definition, $ND(S, p_1) = ND(S', p_1)$, $ND(S, p_2) = ND(S', p_2)$. By assumption, $ND(S', p_1) \leq ND(S', p_2)$. Then $ND(S, p_1) \leq ND(S, p_2)$.
2) $p_1 < p_2 = p'$ and $o' = Ins(p', c', nd')$: $ND(S, p_1) = ND(S', p_1)$ and $ND(S, p_2) = nd'$. Since $p_1 < p'$, it follows that $ND(S', p_1) \leq nd'$ and then $ND(S, p_1) \leq ND(S, p_2)$.
3) $p_1 < p_2 = p'$ and $o' = Del(p')$: $ND(S, p_1) = ND(S', p_1)$ and $ND(S, p_2) = ND(S', p_2 + 1) + 1$. Then $ND(S, p_1) \leq ND(S, p_2)$.
4) $p_1 < p' < p_2$ and $o' = Ins(p', c', nd')$: $ND(S, p_1) = ND(S', p_1)$ and $ND(S, p_2) = ND(S', p_2 - 1)$. Then $ND(S, p_1) \leq ND(S, p_2)$.
5) $p_1 < p' < p_2$ and $o' = Del(p')$: $ND(S, p_1) = ND(S', p_1)$ and $ND(S, p_2) = ND(S', p_2 + 1) + 1$. Then $ND(S, p_1) \leq ND(S, p_2)$.
6) $p_1 = p' < p_2$ and $o' = Ins(p', c', nd')$: $ND(S, p_1) = nd'$ and $ND(S, p_2) = ND(S', p_2 - 1)$. Since $p' \leq p_2 - 1$, it follows that $nd' \leq ND(S', p_2 - 1)$ and then $ND(S, p_1) \leq ND(S, p_2)$.
7) $p_1 = p' < p_2$ and $o' = Del(p')$: $ND(S, p_1) = ND(S', p_1 + 1) + 1$ and $ND(S, p_2) = ND(S', p_2 + 1) + 1$. It follows that $ND(S, p_1) \leq ND(S, p_2)$.
8) $p' < p_1 < p_2$ and $o' = Ins(p', c', nd')$: $ND(S, p_1) = ND(S', p_1 - 1)$ and $ND(S, p_2) =$

$ND(S', p_2 - 1)$. Then $ND(S, p_1) \leq ND(S, p_2)$.
9) $p' < p_1 < p_2$ and $o' = Del(p')$: $ND(S, p_1) = ND(S', p_1 + 1) + 1$ and $ND(S, p_2) = ND(S', p_2 + 1) + 1$. Then $ND(S, p_1) \leq ND(S, p_2)$.

Theorem 1 *Let $o_1 = Ins(p_1, c_1, nd_1)$ and $o_2 = Ins(p_2, c_2, nd_2)$ be two insert operations generated on the same state after executing two different but equivalent sequences of operations S_1 and S_2. Then: 1) $p_1 = p_2 \Rightarrow nd_1 = nd_2$ and 2) $p_1 < p_2 \Rightarrow nd_1 \leq nd_2$.*

Proof 2 *By definition $nd_1 = ND(S_1, p)$ and $nd_2 = ND(S_2, p)$.*
1) By assumption, sequences S_1 and S_2 consist of the same set of original operations executed, after integration, in two different orders. The different execution orders can be obtained by successive pairwise permutations of concurrent operations. If one permutation of concurrent operations preserves the value given for the number of deletes, then any sequence of such swaps preserve also this value. As a result, it is sufficient to consider only the case of one permutation: $S_1 = S \bullet [o_1; o_{21}] \bullet S'$ and $S_2 = S \bullet [o_2; o_{12}] \bullet S'$, where $o_{21} = IT(o_2, o_1)$ and $o_{12} = IT(o_1, o_2)$. So, to show that $ND(S_1, p) = ND(S_2, p)$, it suffices to show that $ND(S \bullet [o_1; o_{21}], p) = ND(S \bullet [o_2; o_{12}], p)$. Table 3.3 reports values of $ND(S \bullet [o_1; o_{21}], p)$ and $ND(S \bullet [o_2; o_{12}], p)$ for every transformation case (see Table 3.2). They are equal for each transformation case.
2) According to Lemma 1, if $p_1 < p_2$ then $ND(S_1, p_1) \leq ND(S_1, p_2)$. Part 1) of Theorem 1 states that $ND(S_1, p_1) = ND(S_2, p_1)$ and $ND(S_1, p_2) = ND(S_2, p_2)$. It follows that: $p_1 < p_2 \Rightarrow ND(S_1, p_1) \leq ND(S_2, p_2)$.

Concretely, these nice relationships established above ensure some consistency to the computation procedure of ND given in Definition 3.1. Indeed, all insert operations with the same position, generated on the same state at different sites, will have the same parameter nd, even if this state has been reached by different but equivalent sequences. Furthermore, if the inserting position p_1 of an operation o_1 is on the left of the inserting position p_2 of another operation o_2, generated on the same state, then the number of symbols deleted before p_1 is less or equal to the number of symbols deleted before p_2 (i.e., $nd_1 \leq nd_2$). Note that $nd_1 = nd_2$ in case there is no deleted symbol between p_1 and p_2, including position p_1. This extra parameter seems then to be appropriate to handle conflicting operations.

Note that the above theorem is very important in the sense that it enables us to avoid recomputing the parameter nd of an insert operation in the remote sites. It suffices to compute this parameter when the insert operation is generated and then include it in the operation sent to all other sites.

Table 3.2 Transformation cases for $IT(o_1, o_2)$ and $IT(o_2, o_1)$

k	IT cases for $IT(o_1, o_2)$ and $IT(o_2, o_1)$
0	$p_1 < p_2$
1	$p_1 > p_2$
2	$o_1 = Ins(p_1, c_1, nd_1) \wedge o_2 = Ins(p_2, c_2, nd_2) \wedge p_1 = p_2 \wedge nd_1 = nd_2 \wedge c_1 < c_2$
3	$o_1 = Ins(p_1, c_1, nd_1) \wedge o_2 = Ins(p_2, c_2, nd_2) \wedge p_1 = p_2 \wedge nd_1 = nd_2 \wedge c_1 > c_2$
4	$o_1 = Ins(p_1, c_1, nd_1) \wedge o_2 = Ins(p_2, c_2, nd_2) \wedge p_1 = p_2 \wedge nd_1 < nd_2$
5	$o_1 = Ins(p_1, c_1, nd_1) \wedge o_2 = Ins(p_2, c_2, nd_2) \wedge p_1 = p_2 \wedge nd_1 > nd_2$
6	$o_1 = Ins(p_1, c_1, nd_1, s_1) \wedge o_2 = Ins(p_2, c_2, nd_2, s_2) \wedge p_1 = p_2 \wedge nd_1 = nd_2 \wedge c_1 = c_2$
7	$(o_1 = Del(p_1) \vee o_2 = Del(p_2)) \wedge p_1 = p_2$

Table 3.3 Computing $ND(S \bullet [o_1; o_{21}], p)$ and $ND(S \bullet [o_2; o_{12}], p)$

Transformation cases	$ND(S \bullet [o_1; o_{21}], p)$	$ND(S \bullet [o_2; o_{12}], p)$
Ins, Ins, $k \in \{0, 2, 4, 6\}$, $nd_{12} = nd_1, nd_{21} = nd_2$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ nd_1 & \text{if } p = p_1 \\ ND(S,p-1) & \text{if } p_1 < p \leq p_2 \\ nd_{21} & \text{if } p = p_{21} \\ ND(S,p-2) & \text{if } p > p_{21} \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ nd_{12} & \text{if } p = p_1 \\ ND(S,p-1) & \text{if } p_1 < p \leq p_2 \\ nd_2 & \text{if } p = p_{21} \\ ND(S,p-2) & \text{if } p > p_{21} \end{cases}$
Ins, Ins, $k \in \{1, 3, 5\}$, $nd_{12} = nd_1, nd_{21} = nd_2$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ nd_{21} & \text{if } p = p_2 \\ ND(S,p-1) & \text{if } p_2 < p \leq p_1 \\ nd_1 & \text{if } p = p_{12} \\ ND(S,p-2) & \text{if } p > p_{12} \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ nd_2 & \text{if } p = p_2 \\ ND(S,p-1) & \text{if } p_2 < p \leq p_1 \\ nd_{12} & \text{if } p = p_{12} \\ ND(S,p-2) & \text{if } p > p_{12} \end{cases}$
Del, Del, $k = 0$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ ND(S,p+1)+1 & \text{if } p_1 \leq p < p_{21} \\ ND(S,p+2)+2 & \text{if } p \geq p_{21} \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ ND(S,p+1)+1 & \text{if } p_1 \leq p < p_{21} \\ ND(S,p+2)+2 & \text{if } p \geq p_{21} \end{cases}$
Del, Del, $k = 1$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ ND(S,p+1)+1 & \text{if } p_2 \leq p < p_{12} \\ ND(S,p+2)+2 & \text{if } p \geq p_{12} \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ ND(S,p+1)+1 & \text{if } p_2 \leq p < p_{12} \\ ND(S,p+2)+2 & \text{if } p \geq p_{12} \end{cases}$
Del, Del, $k = 7$	$ND(S \bullet [o_1], p)$	$ND(S \bullet [o_2], p)$
Ins, Del, $k \in \{0, 7\}$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ nd_1 & \text{if } p = p_1 \\ ND(S,p-1) & \text{if } p_1 < p \leq p_2 \\ ND(S,p)+1 & \text{if } p \geq p_{12} \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_1 \\ nd_{12} & \text{if } p = p_1 \\ ND(S,p-1) & \text{if } p_1 < p \leq p_2 \\ ND(S,p)+1 & \text{if } p \geq p_{12} \end{cases}$
Ins, Del, $k = 1$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ ND(S,p+1)+1 & \text{if } p_2 \leq p < p_1 \\ nd_1+1 & \text{if } p = p_{12} \\ ND(S,p)+1 & \text{if } p \geq p_1 \end{cases}$	$\begin{cases} ND(S,p) & \text{if } p < p_2 \\ ND(S,p+1)+1 & \text{if } p_2 \leq p < p_1 \\ nd_{12} & \text{if } p = p_{12} \\ ND(S,p)+1 & \text{if } p \geq p_1 \end{cases}$

3.4.4 Proof of consistency

To prove consistency, it suffices to show that our IT satisfies both properties TP1 and TP2. In [Imine *et al.* (2006)], the authors proposed a formal framework for modelling and verifying IT functions with algebraic specifications. For checking the properties TP1 and TP2, they used an automatic theorem proving. However, this theorem proving approach has some shortcomings: (i) the model of the system is sound but not complete w.r.t. TP1 and TP2 (i.e., it does not guarantee that the violation of property TP1 or TP2 is really feasible); (ii) there is no guidance to understand the counterexamples (when the properties are not verified); (iii) it requires some interaction (by injecting new lemmas) to complete the verification. In [Boucheneb *et al.* (2010)], the authors addressed these drawbacks and proposed a symbolic model-checking technique based on difference bound matrices (DBMs) to verify whether an IT function satisfies properties TP1 and TP2. The verification of these properties is performed automatically and symbolically without carrying out different copies of the shared object and executing explicitly the updates. Moreover, unlike the approach proposed in [Imine *et al.* (2006)], the approach proposed in [Boucheneb *et al.* (2010)] provides faisable and complete symbolic counterexamples. As in [Boucheneb *et al.* (2010)], we use a symbolic model-checking technique, where the shared objects are abstracted and their update operations are handled symbolically using difference bound matrices (DBMs). In our context, DBMs are used to encode sets of constraints of the form $p_i - p_j \leq c$, where p_i, p_j are integer variables and c is an integer constant. From the practical point of view, a DBM is a square matrix P indexed by variables. Each entry P_{ij} represents the atomic constraint $p_i - p_j \leq P_{ij}$. If there is no upper bound on $p_i - p_j$ with $i \neq j$, P_{ij} is set to ∞. Entry P_{ii} is set to 0. Constraints $p_i - p_j = c$ and $p_i - p_j \geq c$ are considered as abbreviations of atomic constraints. In the following, we use invariantly atomic constraints or their abbreviations.

A set of constraints is consistent iff they represent a non empty domain. Although the same non empty domain may be encoded by different DBMs, they have a canonical form. The canonical form of a DBM is the representation with tightest bounds on all differences between variables, computed by propagating the effect of each entry through the DBM. Canonical forms are much more useful to verify consistency and test of equivalence.

Two sets of atomic constraints are equivalent iff the canonical forms of their DBMs are identical. To verify the consistency of a DBM (i.e., a set of atomic constraints), it suffices to apply a shortest-path algorithm and to stop the algorithm as soon as a negative cycle is detected. The presence of negative cycles means that the set of atomic constraints is inconsistent.

The model depicted at Figure 3.18 is used to verify whether or not our IT function satisfies properties TP1 and TP2. The model starts by selecting types of the 3 update operations o_1, o_2

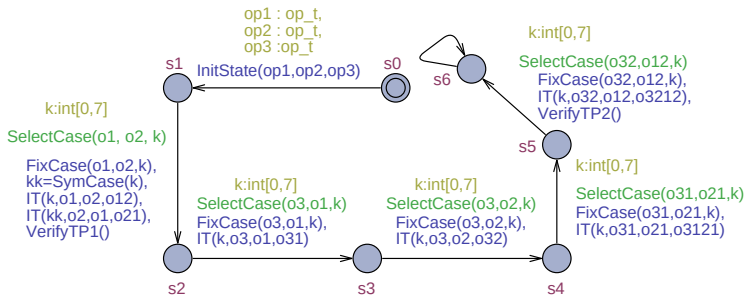

Figure 3.18 Automaton used to verify TP1 and TP2

and o_3 and initializing the DBMs of their parameters (initial DBMs are free from constraints). This is the role of the edge (s_0, s_1). Afterwards, the model successively selects for each operational transformation, a feasible transformation case among the 8 cases shown at Table 3.2, fixes and applies the transformation case, and verifies TP1 or TP2 as soon as the needed transformations are computed. For instance, the edge (s_1, s_2) selects a transformation case k for o_1 with regard to o_2 ($SelectCase(o_1, o_2, k)$), fixes the selected case ($FixCase(o_1, o_2, k)$) and applies it to transform o_1 with regard to o_2. The symmetrical case $kk = SymCase(k)$ is used to transform o_2 with regard to o_1. After these transformations ($IT(k, o_1, o_2, o_{12})$ and $IT(kk, o_2, o_1, o_{21})$), the property TP1 is verified. Concretely, function $FixCase(o_i, o_j, k)$ adds to the DBMs the constraints corresponding to the transformation case k to be applied to o_i with regard to o_j. Function $IT(k, o_i, o_j, o_{ij})$ applies the transformation case k to transform o_i with regard to o_j and returns the transformation result in o_{ij}. Functions $verifyTP1()$ and $verifyTP2()$ set in boolean variables $TP1$ and $TP2$ the results of verification of properties TP1 and TP2, respectively. Function $verifyTP1()$ verifies whether or not the domain of P satisfies one of the conditions of equivalence given in Table 3.4 [4] accordingly to the types of operations o_{12} and o_{21}. For instance, if o_{12} and o_{21} are insert operations, $TP1$ is satisfied iff $P = P \cup \{p_{12} = p_1 \leq p_2 = p_{21} - 1\}$ or $P = P \cup \{p_{21} = p_2 \leq p_1 = p_{12} - 1\}$ or $([o_1] \equiv [o_2] \wedge [o_{12}] \equiv [o_{21}])$.
Function $verifyTP2()$ verifies whether or not operations o_{3121} and o_{3212} are identical, i.e., they are of the same type and have the same parameters. In case o_{3121} and o_{3212} are not of type Nop, then they have identical position parameter iff $P = P \cup \{p_{3121} = p_{3212}\}$, which means that $p_{3121} = p_{3212}$ holds in the whole domain of P.

For example, suppose that 3 insert operations are selected by the edge (s_0, s_1). In this case, 3 DBMs P, Nd and C over $\{p_1, p_2, p_3\}$, $\{nd_1, nd_2, nd_3\}$ and $\{c_1, c_2, c_3\}$ are created. The initial domain of each DBM is $\mathbb{N}^3$, $\mathbb{N}$ being the set of non negative integers. Suppose now that for the operational transformation of o_1 with regard to o_2, the selected k is 2. The

4. We suppose that if o_{12} (resp. o_{21}) is an insert operation then $c_{12} = c_1$ (resp. $c_{21} = c_2$).

operational transformation case of o_2 with regard to o_1 is then $kk = SymCase(2) = 3$. In this case, $FixCase(o_1, o_2, k)$ adds sets of constraints $\{p_1 = p_2\}$, $\{nd_1 = nd_2\}$ and $\{c_1 < c_2\}$ to P, Nd and C, respectively (see Table 3.2). The operational transformations of o_1 with regard to o_2 and o_2 with regard to o_1 (i.e., $IT(k, o_1, o_2, o_{12}), IT(kk, o_2, o_1, o_{21})$) create two insert operations o_{12} and o_{21}, add to P, Nd and C sets of constraints $\{p_{12} = p_1, p_{21} = p_2 + 1\}$, $\{nd_{12} = nd_1, nd_{21} = nd_2\}$ and $\{c_{12} = c_1, c_{21} = c_2\}$, respectively. Finally, $verifyTP1()$ states that the property TP1 is satisfied. Afterwards, the same procedure is repeated for computing the operational transformations needed to verify TP2 (i.e., $o_{3121} = o_{3212}$).

We have used the tool UPPAAL [Larsen *et al.* (1997)] to verify whether or not our model satisfies the safety properties $AG\ TP1$ and $AG\ TP2$. UPPAAL states that the first property is satisfied (i.e., the proposed IT satisfies TP1). It concludes however that the second property is not satisfied. The only counterexamples provided are scenarios where there are at least two insert operations $o_1 = Ins(p_1, c_1, nd_1)$ and $o_2 = Ins(p_2, c_2, nd_2)$ generated at the same state s.t. $p_1 \leq p_2 \wedge nd_1 > nd_2$ or $p_1 \geq p_2 \wedge nd_1 < nd_2$. According to Lemma 1 and Theorem 1 such scenarios are infeasible. When we exclude such infeasible scenarios, UPPAAL concludes that property TP2 is satisfied. Therefore, the proposed IT satisfies TP1 and TP2.

Table 3.4 Condition of equivalence of $[o_1; o_{21}]$ and $[o_2; o_{12}]$

Type of o_{12}	Type of o_{21}	$[o_1; o_{21}] \equiv [o_2; o_{12}]$	Type of o_{12}	Type of o_{21}	$[o_1; o_{21}] \equiv [o_2; o_{12}]$
Ins	*Ins / Del*	$p_{12} = p_1 \leq p_2 = p_{21} - 1$	*Ins / Del*	*Ins*	$p_{21} = p_2 \leq p_1 = p_{12} - 1$
Del	*Ins / Del*	$p_{12} = p_1 < p_2 = p_{21} + 1$	*Ins / Del*	*Del*	$p_{21} = p_2 < p_1 = p_{12} + 1$
-	-	$[o_1] \equiv [o_2] \wedge [o_{21}] \equiv [o_{12}]$			

3.4.5 Comparison

In this section, we give comparison between our IT function and the function given in [Oster *et al.* (2006a)]. It is well known that the main issue for satisfying TP2 is due to the semantics of the delete operation. Recall that this operation removes an element at a given position and decreases the length of the shared document. That is why, in [Oster *et al.* (2006a)], the authors tackled the TP2 problem by changing the semantics of the deletion operation. Indeed, operation $Del(p)$ does not remove the element at position p but makes it invisible. This new semantics leads the authors to manage two distinct states, called *view* and *data* models. The view model is seen by the user and contains only visible characters while the data model contains all characters: characters displayed in the view model and the hidden characters (called *tombstones*) resulting from delete operations. Any generated update operation involves two updates from view and data models, respectively. As illustrated in Figure 3.19, consider the operation $Ins(3, y)$ related to the view model. The corresponding

position in the data model is processed by finding out the location of the 3-th visible character, after skipping each possible hidden character encountered. The search process relies on 3 visible characters. Hence, the corresponding operation related to the data model is $Ins(5, y)$. The hidden characters h and n located at position 1 and 4, respectively in the data model, are skipped.

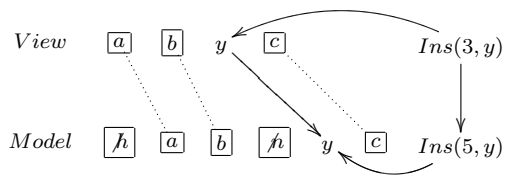

Figure 3.19 View and Model states (Figure taken from [Oster *et al.* (2006a)]).

Table 3.5 Complexity comparison.

Operation	TTF's solution [Oster *et al.* (2006a)]	Our solution
Insert	$O(m)$ where m is the length of the data model. The size of the data model will never decrease	$O(n)$ where n is the length of the local history
Delete	$O(m)$ where m is the length of the data model	O(1)

Although the authors presented a correct set of transformation functions, the length of the sequence is always increased (but never decreased), as the new semantics of deletion operation has no effect on the document length. In addition, their solution needs some extra procedures to manage two distinct states due to the hidden characters, namely: the view (the state seen by the user) and the model (the persistent state). Unlike this situation, our proposed transformation function considers the natural semantics of deletion operation as suggested by Ellis et *al.* in [Ellis et Gibbs (1989)] and our deletion operation removes physically the character and reduces the document length.

Our transformation function is based on the parameter *nd* filled with the number of deleted symbols to maintain consistency without any particular data model besides. As for complexities at generation time, a comparison is given in Table 3.5. Even though, the overall complexity is linear for both approaches. Unlike TTF [Oster *et al.* (2006a)], it should be noted that for our approach, the complexity of delete operation is always constant. Note also that the parameter *nd* is computed at the generation step thanks to Theorem 1, there is no need to recompute it when it is received by a remote site.

In Wu et al. [Wu et Pui (2009)], the authors have combined CRDT [Preguiça *et al.* (2009), Weiss *et al.* (2010)] with TTF [Oster *et al.* (2006a)]. Their approach is based on a data structure called Partial Persistent Sequences (PPS), where each item of the shared object has a unique position identifier, i.e., a rational number, called position stamp. It is

then always possible to allocate a new position stamp to any item to be inserted into the object. As in TTF [Oster *et al.* (2006a)], when an item is removed from the object, it is hidden and its position stamp is mapped to an empty item. To prevent situations where different users compute the same position concurrently, Wu et *al.* have proposed in [Wu *et al.* (2010)] to allocate in advance, to each user, sub-ranges of position identifiers. The range between two position identifiers is partitioned into sub-ranges, dividing the distance by the number of users. Between two given position stamps, a unique sub-range is assigned to each user. Even if the management of the position identifiers is improved, it remained non-optimal due to waste of the identifier space such as some ranges could be reserved but never used. In addition, the identifier space will increase rapidly. To deal with the rebalancing issue, Wu et *al.* have proposed in [Wu *et al.* (2010)] to remove the position identifiers of deleted items followed by a computation of new position stamps values for the non-removed items. As any CRDT, this approach does not need to satisfy the property TP2, however, its correctness depends on the uniqueness of position identifiers. The authors do not show clearly how to achieve the uniqueness of position identifiers.

3.5 Conclusion

In this paper, we have investigated, using the controller synthesis technique, whether or not there exist IT functions which ensure data convergence, based on the operational transformation framework. In this framework, an IT function ensures convergence iff it satisfies two properties TP1 and TP2. We have shown that there are some IT functions for the basic signatures of insert and delete operations, which satisfy TP1 but there is no IT function, which satisfies TP2. We have then identified two pairwise scenarios, which prevent to get a consistent function IT (with no one-to-many relation). These scenarios were useful to find an appropriate extra parameter for the insert operation and to define, based on this extra parameter, a new IT. The role of the extra parameter is to record the number of deleted symbols before the inserting position. This parameter is computed when the operation is generated and then updated whenever it is transformed against a delete operation with smaller position.

The consistency of our IT function is formally proved by means of a symbolic model checking technique, where all parameters of the update operations are handled symbolically using difference bound matrices. Initially, the verification of properties TP1 and TP2 is performed without imposing any relationship between parameters of the update operations (i.e., an over-approximation of the effective model). It concludes that TP1 is satisfied but TP2 is not satisfied. The counterexamples provided for property TP2 are scenarios where there

are at least two insert operations $o_1 = Ins(p_1, c_1, nd_1)$ and $o_2 = Ins(p_2, c_2, nd_2)$ generated at the same state s.t. $p_1 \leq p_2 \wedge nd_1 > nd_2$ or $p_1 \geq p_2 \wedge nd_1 < nd_2$. According to Lemma 1 and Theorem 1 such scenarios are infeasible. When we exclude such infeasible scenarios, the verification process concludes that property TP2 is satisfied. Therefore, the proposed IT satisfies both TP1 and TP2.

In the near future, we will apply our IT to a specific system as peer-to-peer collaborative editors and study performance issue. We also plan to investigate the existence of consistent IT functions, in case of shared object with non linear structure, using the controller synthesis and model checking techniques.

english

CHAPITRE 4

ARTICLE 2 : Specification and Verification using Alloy of Optimistic Access Control for Distributed Collaborative Editors

Aurel Randolph [§] , Abdessamad Imine [¶], Hanifa Boucheneb[§], and Alejandro Quintero[§]
otherlanguageenglish

Abstract

Distributed Collaborative Editors are interactive systems where several and dispersed users edit concurrently shared documents. Generally, these systems rely on data replication and use safe coordination protocol which ensures data consistency even though the users's updates are executed in any order on different copies. Controlling access in such systems is a challenging problem, as they need dynamic access changes and low latency access to shared documents. In [Imine *et al.* (2009)], a flexible access control protocol is proposed ; it is based on replicating the shared document and its authorization policy at the local memory of each user. To deal with latency and dynamic access changes, an optimistic access control technique is used where enforcement of authorizations is retroactive. However, verifying whether the combination of access control and coordination protocols preserves the data consistency is a hard task since it requires examining a large number of situations. In this paper, we specify this access control protocol in the first-order relational logic with Alloy, and we verify that it preserves the correctness of the system on which it is deployed in such a way that the access control policy is enforced identically at all participating user sites and, accordingly, the data consistency remains still maintained.

Keywords Access control policies, distributed collaborative editors, data consistency, formal specification, formal verification, Alloy.

This work is supported by grant number 138732 awarded by the Fonds de Recherche du Québec - Nature et Technologies (FQRNT-Équipe).

§. École Polytechnique de Montréal, Montréal, Canada. Email : {aurel.randolph, hanifa.boucheneb, alejandro.quintero}@polymtl.ca

¶. Lorraine University and INRIA Nancy-Grand-Est, France. Email : abdessamad.imine@loria.fr

4.1 Introduction

Distributed Collaborative Editors (DCE) enable several and dispersed users to form a group for editing simultaneously shared documents, such as articles, wiki pages and program source code (e.g. Google Docs). To achieve data availability, each user owns a local copy of the shared documents. Thus, the collaboration is performed as follows : each user site's updates are locally executed in a non blocking manner and then are propagated to the other sites in order to be executed on remote copies. Although being distributed applications, DCE are specific in the sense they must consider human factors. Moreover, they are characterized by the following features : (i) High local responsiveness : the system has to be as responsive as its single-user editors [Ellis et Gibbs (1989), Sun *et al.* (1998), Sun *et al.* (2006)] ; (ii) High concurrency : the users must be able to concurrently and freely modify any part of the shared document at any time [Ellis et Gibbs (1989), Sun *et al.* (1998)] ; (iii) Consistency : the users must eventually see a converged view of all copies [Ellis et Gibbs (1989), Sun *et al.* (1998)] ; (iv) Scalability : a group must be dynamic in the sense that users may join or leave the group at any time. Due to data replication and arbitrary exchange of updates, consistency preservation is one of the most critical properties in DCE. Accordingly, each DCE is endowed with *Coordination Protocol* (CP) to maintain globally consistent state.

Balancing the computing goals of collaboration and access control to shared information is a challenging problem in DCE [Tolone *et al.* (2005)]. Indeed, interaction in collaborative editors is aimed at making shared document available to all who need it, whereas access control seeks to ensure this availability only to users with proper authorization. To preserve the above cited DCE's features and avoid a central authority, a *flexible Access Control Protocol* (ACP) is proposed in [Imine *et al.* (2009)] where all updates will be checked at each user site without resorting to a central authority. In this model, a user will own two copies : the shared document and its authorization policies. This replication allows for high availability since when users want to read or update the shared document, this manipulation will be granted or denied by controlling only the local copy of the authorization policies. Due to the out-of-order execution of the shared document's updates and the authorization policy's updates, an optimistic approach is used that tolerates momentary violation of access rights but then ensures the copies to be restored in valid states (by undoing invalid document's updates) w.r.t the stabilized access control policy.

To ensure a safe access control in DCE (i.e. permitting legal updates and rejecting illegal updates on the shared document), a protocol stack is built by integrating an ACP on the top of any CP based on data replication and update logging [Imine *et al.* (2009)]. If we combine a correct CP (i.e. satisfying separately the consistency property) with an ACP : can

we verify that the consistency property is preserved by the new protocol ? This verification turns out a hard task and unmanageably complicated. Indeed, it requires examining a large number of situations since the updates are performed in different orders on different copies of the shared document and the authorization policy. Consequently, the verification of the combination correctness must be assisted by an automatic checker tool.

Contributions. We propose here a model which specifies concisely the ACP and verify the consistency property of any DCE integrating an ACP on the top of a consistent CP. We use the first-order logic "à la Alloy" to describe symbolically ACP and its environment. This choice is motivated by the possibility to handle symbolically bounded and unbounded variables such as queues of messages, logs, number of sites, number of operations generated by each site, etc. The consistency property is also specified in Alloy language and verified by Alloy analyzer, using a SAT-based bounded model checking. This technique is established as a good alternative to the classical symbolic model checking using binary decision diagrams (BDDs), as it can often handle much larger systems, by searching for counterexamples of bounded length.

Outline. This paper is organized as follows : Section 4.2 presents the flexible access control protocol. Section 4.3 is devoted to the formal specification of ACP and its environment. Section 4.4 discusses related work. Finally, the conclusion is presented in Section 4.5.

4.2 Optimistic Access Control Protocol for DCE

Shared documents are objects whose state can be altered by a set of *cooperative operations* generated by sites. For instance, a shared text document is modified by operations such as inserting a new section, deleting an existing paragraph and replacing an old line by new one. In [Imine *et al.* (2009)], an access policy is described as an indexed list of authorization rules, where each rule is a quadruple $\langle S, O, R, \omega \rangle$ with (i) S is set of subjects (sites or users), (ii) O is a set of objects (e.g. paragraphs or chapters), (iii) is R a set of access rights (e.g. deleting or updating paragraphs) and (iv) $\omega \in \{-, +\}$. The sign "+" represents a right *attribution* and the sign "−" represents a right *revocation*.

The state of the policy object can be altered by a set of *administrative operations* such as adding and removing authorizations. Administrative operations are generated by the administrator, at any time, and aimed to manage dynamically the right access to the shared documents. These operations are next broadcast to other sites, in order to modify their local copies of the policy object. Thus, on each site, cooperative operations are granted or denied by using the local copy of the policy and applying the first-match semantics : when an operation o is generated, the system checks o against its authorizations one by one, starting from

the first authorization and stopping when it reaches the first authorization l that matches o. If no matching authorizations are found, o is rejected. Note that every local policy copy maintains a monotonically increasing version counter that is incremented by every administrative operation performed on this copy.

The collaboration happens in optimistic approach and modifications could be applied in different orders at different sites. The messages are assumed to be exchanged via secure and reliable communication network : each message sent is received by each others without alteration. The flow of messages exchanged during the collaboration is illustrated in Figure 4.1.

4.2.1 Generation of Local Cooperative Requests

Locally, each site can generate some cooperative operations. Each generated cooperative operation is first checked against the local policy. If the operation is revoked then it is said to be invalid and its execution is aborted. When the operation is granted, it is set to valid status in the case of administrator site and to tentative status otherwise. The operation is then performed immediately on the local copy of the shared document. A resulting cooperative request is generated and attached with the number version of the policy copy on which the operation is granted. This cooperative request is finally broadcast to other sites.

4.2.2 Reception of Remote Cooperative Requests

When a remote cooperative request is received, it is first stored in a dedicated queue before being extracted. The request is extracted if it is causally-ready, when its attached version number of policy is less or equal than the current version of the local policy and its precedent cooperative request have been already integrated to the local copy of the shared

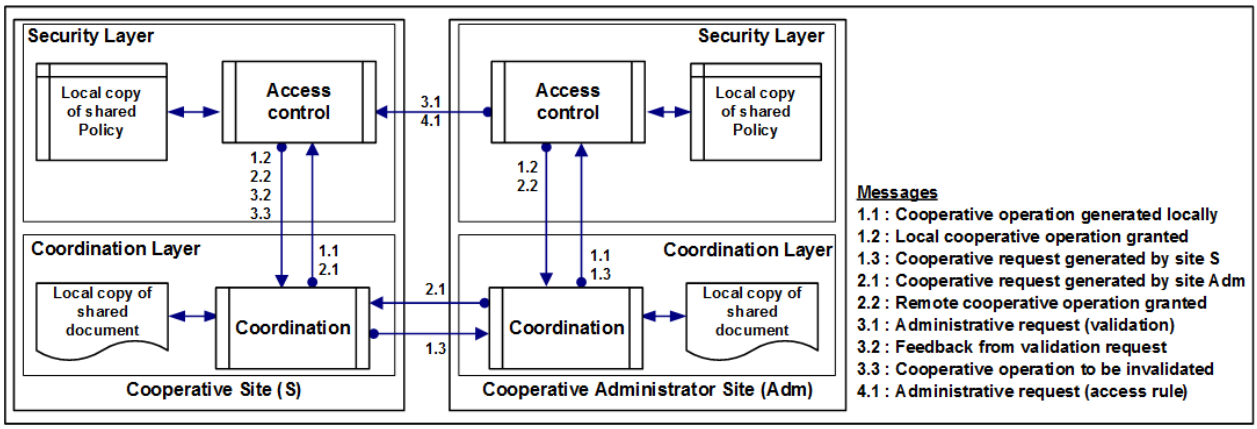

Figure 4.1 Flow of collaboration messages

document. This mechanism is setup to ensure that the access control protocol preserves the causality dependency with respect to precedent administrative requests and precedent cooperative requests.

After its extraction, the remote cooperative request is checked against the local administrative log to verify whether or not it is granted. If the request is granted, its status is set to valid, if the receiver is the administrator, otherwise, its status is tentative. If the receiver is the administrator then the policy version is incremented and a validation request is generated in order to broadcast it to other sites. The new version number is attached to the validation request before its broadcasting. Once, the cooperative operation is performed on the shared document with regard to the collaborative editor's procedures.

4.2.3 Generation of Administrative Operations

To manage the access control, the administrator produces some access rules called administrative operations. When an administrative operation is generated, the version number is incremented for the administrator's local policy, which is immediately updated by performing on, the generated administrative operation. Once, an administrative request with the corresponding new version number is generated and broadcast to other sites to enforce their own policy.

4.2.4 Reception of Remote Administrative Requests

There exists two kinds of remote administrative request : validation request and access rule based request. Each received remote administrative request is first stored in a dedicated queue then, extracted when it is causally-ready. The administrative request is said to be causally-ready if the value of its attached policy version number is the next value of the version number of the local policy (the difference is one) and in case of validation request, the corresponding cooperative operation is already executed on this site. Each extracted access rule based request, is performed on the local policy. Thereafter, if the access rule is restrictive, then all tentative cooperative operations, locally generated or received, which are concerned by the rule with regard to the rights, are undone. For the extracted validation request, the status of the corresponding cooperative operation is updated from tentative to valid. At the end of the treatment of the administrative request, the version number of the local policy is incremented.

4.2.5 Verification Issues

The DCE consists of several sites. Each of them maintains the shared objects and its access right policy, by generating, exchanging and performing some cooperative and administrative operations. As the numbers of sites, cooperatives operations and administrative operations are arbitrary, the queues of cooperative requests and administrative requests are unbounded. The system is then infinite and parameterisable by the number of sites, the number of cooperative operations to be generated by each site, and the number of administrative operations to be generated by the administrator. On each site, the shared objects are modified with respect to the local access right policy. Meanwhile, the local policy is enforced by taking into account the administrative operations generated and broadcast by the administrator. Thus, if the policy is not enforced identically at all sites, it can result in the security hole on the shared objects by permitting illegal modifications or rejecting legal modifications. In addition, this situation can lead to data inconsistency for the collaborative edition such as the document can diverge at the end of the collaboration.

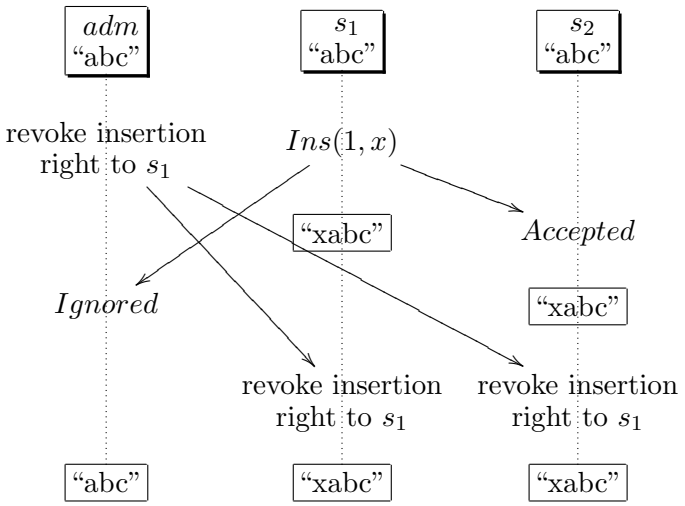

Figure 4.2 Divergence caused by introducing administrative operations

For instance, consider a group composed of an administrator *adm* and two sites S_1 and S_2. Initially, the three sites have the same shared document "abc" and the same policy object where S_1 is authorized to insert characters (see Figure 4.2). Suppose that *adm* revokes the insertion right of S_1 and sends this administrative operation to S_1 and S_2 so that enforce their local policy copies. Concurrently S_1 executes a cooperative operation $Ins(1, x)$ to derive the state "xabc" as it is granted by its local policy. When *adm* receives the S_1's operation, it will be ignored (as it is not granted by the *adm*'s local policy) and then the final state still remain "abc". As S_2 receives the S_1's insert operation before its revocation, he gets the

state "xabc" that will be unchanged even after having executed the revocation operation. We are in presence of data inconsistency (the state of *adm* is different from the state of S_1 and S_2) even though the policy object is same in all sites. In fact, the new policy object is not uniformly enforced among all sites because of the out-of-order execution of administrative and cooperative operations. Thus, security holes may be created. For instance some sites can accept cooperative operations that are illegal with respect to the new policy (e.g. sites S_1 and S_2).

To solve this problem, the ACP applies the principles of optimistic security [Povey (2000)] in such a way that the enforcement of the new policy may be retroactive with respect to concurrent cooperative operations. In this case, only illegal operations are undone. For instance, $Ins(1, x)$ as shown at 4.2 should be undone at S_1 and S_2 after the execution of the revocation.

It appears important to verify that the ACP preserves the correctness of the collaborative editing system on which it is deployed with regards to the security issues and data consistency. For this purpose, the sets of legal (valid) cooperative operations must be identical at all sites, when all generated and received cooperative and administrative operations are performed on each site (stable state). Performing such a verification is tricky and hard to do manually. So, the system must be automatically checked using formal methods.

4.3 Specification and Verification

Several model checking techniques have been proposed in the literature. These techniques can be classified into *explicit state model checker* and *Symbolic model checker*. In explicit state model checker [Holzmann (2004)], states, sets and relations are explicitly represented, whereas, in symbolic model checker, they are implicitly represented using boolean logic formulas. The category of symbolic model checker can be subdivided into *BDD-based model checkers* [Cimatti *et al.* (2000)] and *SAT-based bounded model checkers* [Schaeffer-Filho *et al.* (2009)]. BDD-based model checking allows to prove by considering the whole state space of the model that some property is satisfied but it does not scale well in practice. SAT-based bounded model checking is considered as a good alternative to BDD-based model checking. It is more appropriate to find bugs in infinite systems. Its basic idea is to search for a counterexample in traces whose length is bounded by some integer k [Frappier *et al.* (2010)]. If no bug is found then k is increased until either a bug is found or the computer resource limits are reached. We propose to use a SAT-based bounded model checker of the tool suite of Alloy [1], to verify that ACP preserves consistency of DCE.

1. MIT Sofware Design Group, Alloy : A language and Tool for Relational Models, [Online]. Available : `http://alloy.mit.edu/alloy/`, (Accessed : 5 May 2013)

4.3.1 Alloy

Alloy [1] , is a SAT-based bounded model-checker whose specification language and analyzer are inspired by Z notation [Woodcock et Davies (1996)] and SMV (Symbolic Model Verifier) [2]. The Alloy model consists of signatures, facts, functions and predicates denoted *sig*, *fact*, *fun* and *pred*, respectively. Signatures describe the sets and relations used to specify the system to be verified. Facts represent the constraints of the system that are always assumed to hold. The expected properties of the system are expressed as assertions (constraints) denoted *assert*. The Alloy analyzer is an automatic constraint solver which operates as an instance finder for the specified model that form counterexamples to the assertions. To find such an instance, Alloy proceeds by an exhaustive search over restricted scopes defined by the user [Schaeffer-Filho *et al.* (2009)]. The scope means the maximum number of occurrences assigned to each object of the model, but also means the maximum length of the execution traces. The principle of searching instance is based on the small scope hypothesis which states that an invalid assertion should have a small counterexample [Jackson (2006)]. The instance found is reported as counterexample and is guaranteed to be valid. Unfortunately, the failure of finding an instance should not be confused with its absence. Alloy is then useful to specify infinite models and find bugs. The use of signatures and fields like object-oriented programming classes increase its expressiveness [Pai *et al.* (2011)]. Moreover, Alloy analyzer allows to use several SAT-solvers like SAT4J [Le Berre et Parrain (2010)], zChaff [3], MiniSAT [4], Kodkod [Torlak et Dennis (2006)].

4.3.2 Formal Specification of ACP

The underlying access control model of ACP considers a set of subjects defined as users (or sites) including the administrator, a set of objects denoting a part of or the whole shared document, and a set of access rights. A policy is defined as a function that maps a set of subjects and a set of objects to a set of access rights. On each site, the policy is indexed with a version number which is incremented during the collaboration. In addition to that fundamental components of the model, we have some operations generated and exchanged in the system. There are cooperative operations with three kinds of status (tentative, valid, invalid), access rules and validation requests as administrative requests. For simplification

1. MIT Sofware Design Group, Alloy : A language and Tool for Relational Models,
Available : `http://alloy.mit.edu/alloy/`, (Accessed : 5 May 2013)
2. Canergie Mellon University, The SMV System,
Available : `http://www.cs.cmu.edu/~modelcheck/smv.html`, (Accessed : 5 May 2013)
3. Boolean Satis ?ability Research Group at Princeton, zChaff,
Available : `http://www.princeton.edu/~chaff/zchaff.html`, (Accessed : 5 May 2013)
4. Eén,Niklas and Sörensson, Niklas, The SMV System,
Available : `http://minisat.se/`, (Accessed : 5 May 2013)

purpose, we consider that the set of objects represents the whole document. Then, it is not necessarily to model the document. Our specification of the fundamental components of the model is shown at **Snippet** 1. From line 1 to 3, we represent the subject. We declare an abstract signature to represent the generic subject. It is extended to have cooperative site, which is also extended to represent an administrative site. To manage the number of policy, we create the signature *SiteVersion*.

The allowed cooperative operations performed on the shared document could be inserting, deleting, updating, etc. This set of operation types is described with the signature *oper_t* at **Snippet** 2. At the same snippet, cooperative operations are represented from line 2 to 10. Some constraints assumed always to hold are added from line 7. They express that the sender of the operation must not receive it back, the operation has one type and is attached to a version number of policy. To undo an operation we consider a new linked operation. It is specified in our model with the signature *UndoneOp*. The status of cooperative operation is represented by *OpStatus* with the three declinations.

```
1 abstract sig Site {}
2 sig CoopSite extends Site {}
3 one sig AdmSite extends Site {}
4 sig SiteVersion{}
```

Snippet 1: Specification of subjects

```
1  sig oper_t{}//identify a type of operation : insert, delete, etc.
2  sig Coop {
3      from : lone Site, // Site that generates and sends the cooperative operation
4      to : set Site, // Intended recipient(s) of a cooperative operation
5      type :lone oper_t,//Type of operation
6      vers : lone SiteVersion  // Version of local policy which granted the operation
7  }{(from !=none) implies {
8      no from & to and to=Site-from and type !=none and vers !=none and
9      # to > 1 //Allow to have at least 2 sites in the system
10 } }
11 sig UndoneOp{
12     owner : lone Site,
13     linkedOp : lone Coop
14 } abstract sig OpStatus {}
15 one sig tentative, invalid, valid extends OpStatus {}
```

Snippet 2: Specification of cooperative operations

To deal with authorizations, we define a signature *Authorization* extended to have *Plus* for right attribution and *Minus* for right revocation. The object representing the administrative request is abstracted and called *AdReq*. It is extended in validation request and access rule, denoted *Val* and *Rule*, respectively. In the relation *Rule*, rights are of the same type of

cooperative operations (*oper_t*). For instance we could have the right of inserting. Rights and authorization (field *signe*) are mapped to the *subject* field to describe an access rule as shown at **Snippet** 3.

```
1  abstract sig Authorization {}
2  one sig Plus, Minus extends Authorization {}
3  abstract sig AdReq
4      source : lone AdmSite,  // Intended recipient(s) of an administrative request
5      dest : set Site,  //Version of policy when generating the administrative request
6      vers : lone SiteVersion
7  }{(source !=none) implies{
8      no source&dest and dest=Site-source and vers !=none and
9      #dest>1  //Allow to have at least 2 receivers (sites) in the system
10 }}
11 sig Rule extends AdReq{
12     subject : some Site,
13     right : some oper_t,
14     signe : one Authorization
15 } sig Val extends AdReq{op : lone Coop}{ op.from != source }
```

Snippet 3: Specification of administrative requests

In addition to these core elements of the model, we define a global state of the system which consists of the state of each site. We called it *SiteState*. The state of each site is represented by the last number version of its policy, the sending and receiving cooperative operations, the sending and receiving administrative operations (only effective for the administrator), the snapshot view of some queues, administrative and cooperative operations which are causally-ready at the state. The corresponding signature is described as shown at **Snippet** 4. For the transition system, we create a linear ordering over states by using the Alloy ordering utility module (*open util/ordering[SiteState] as sitesstates*). This module is also used to manage the version number of policy (*open util/ordering[SiteVersion] as versOrder*).

The dynamics of the system is specified using *facts*. **Snippet** 5 descibes the generation of cooperative and administrative requests and their reception. We assume that when an operation is sent by a site, it is received by others at the next state of the system. When the cooperative and administrative requests are received, they are stored in appropriate queue and are extracted when there are causally-ready. To define the causally-ready expressions we use functions *FcausallySeq* and *QcausallySeq* for cooperative and administrative requests, res–pectively. These functions are presented at **Snippet** 6. Once extracted, there are processed.

Snippets 7 and 8 present the processing of the causally-ready cooperative operation and administrative request by a non-administrative site, respectively. The processing of a causally-ready cooperative request by the administrative site is shown at **snippet** 9.

Several complementary constraints are defined to control the dynamics of the system.

```
sig SiteState {
    versions : Site − > one SiteVersion, // Version
    CoopStatus : Site − > Coop− > OpStatus,  //Status of all cooperative operations
    sentCoop : Site − > lone Coop, // Cooperative operations sent in this state.
    sentAdReq : AdmSite − > lone AdReq, // Administrative requests sent.
    ReceivedCoop : Site − > set Coop,// Cooperative operations received.
    ReceivedAdReq : Site − > set AdReq, // Administrative requests received.
    F : Site − > (seq Coop),// Received cooperative operation's queue.
    Q : Site − > (seq AdReq),// Received administrative request's queue.
    H : (Site − > set Coop)+(Site − > set UndoneOp), //Cooperative log
    L : Site − > (seq Rule), // Administrative log.
    Vr : Site − > (seq Val), // Validation request log.
    CoopCausallyReady : Site − > lone Coop, // Operations which are causally-ready.
    AdCausallyReady : Site − > lone AdReq //Causally-ready administrative requests.
}
```

Snippet 4: Specification of the state of the system

```
fact GenerateCooperativeRequest{
    all pre : SiteState-sitesstates/last, S :Site, o :pre.sentCoop[S] |
        let post = sitesstates/next[pre] | {
            o.vers = pre.versions[S] //Set the site version to the operation
            o in post.H[S] //Add the operation to the owner's H
            {(S in CoopSite) and (o in tentative[post.CoopStatus[S]])} or
            {(S in AdmSite) and (o in valid[post.CoopStatus[S]])} //Set the status
            all Sj :Site-S| o in post.ReceivedCoop[Sj] //Broadcast
}}

            fact GenerateAdministrativeRequest{
                all pre : SiteState-sitesstates/last, S :AdmSite, a :pre.sentAdReq[S]|
                    let post=sitesstates/next[pre], v=pre.versions[S], vv=versOrder/next[v]|{
                        vv in post.versions[S] //Incrementation of the version for the next
                        SiteState
                        a.vers = vv
                        {(a in Rule) and (post.L[S]= add[pre.L[S], a])} or //Update the policy
                        {(a in Val) and (post.Vr[S]= add[pre.Vr[S], a])}
                        all Sj :Site-S| a in post.ReceivedAdReq[Sj] //Broadcast
            }}

                        fact RequestReception{
                            all post : SiteState-sitesstates/first| let pre = sitesstates/prev[post]|{
                                all S :Site, o :post.ReceivedCoop[S]|{
                                    o not in elems[pre.F[S]] and o in elems[post.F[S]]
                                    all oj :
                                    elems[pre.F[S]]|lastIdxOf[pre.F[S],oj]<lastIdxOf[post.F[S],o]}
                                all S :Site, a :post.ReceivedAdReq[S]|{
                                    a not in elems[pre.Q[S]] and a in elems[post.Q[S]]
                                    all aj :
                                    elems[pre.Q[S]]|lastIdxOf[pre.Q[S],aj]<lastIdxOf[post.Q[S],a]}}
}
```

Snippet 5: Specification of the generation and reception of requests

```
fun FcausallySeq[st : SiteState, S : Site] : lone Coop {
    { o : (elems[st.F[S]]- OpStatus[st.CoopStatus[S]])| {
        versOrder/lte [o.vers, st.versions[S]]
        all oj : (elems[st.F[S]]- OpStatus[st.CoopStatus[S]]-o)| {
            versOrder/lte [oj.vers, st.versions[S]] implies
                lastIdxOf [st.F[S],oj]>lastIdxOf [st.F[S],o] }} }
}

fun QcausallySeq[st : SiteState, S : Site] : lone AdReq {
    { ad : (elems[st.Q[S]] -elems[st.L[S]]-elems[st.Vr[S]])| {
        ad.vers= versOrder/next [st.versions[S]]
        all adj : (elems[st.Q[S]] -elems[st.L[S]]-elems[st.Vr[S]]-ad)| {
            (adj.vers= versOrder/next [st.versions[S]]) implies
                lastIdxOf [st.Q[S],adj] > lastIdxOf [st.Q[S],ad] }}}
}
```

Snippet 6: Specification of causally-ready expressions

These constraints are not explicitly expressed in the protocol but are necessary from our point of view to prevent unacceptable instances or counterexamples. For instance, at the beginning of the collaboration considered as the initial state, all queues are empty at each site. The complete specification is given in https ://sites.google.com/site/laboratoireverifom.

```
fact ReceiveAdminRequest{
    all st : SiteState-sitesstates/first-sitesstates/last,
    S :st.AdCausallyReady.AdReq, ad :st.AdCausallyReady[S]|
        let post=sitesstates/next[st]|{ ad in Val implies {
                ad.op in valid[post.CoopStatus[S]]
                post.Vr[S]= add[st.Vr[S], ad]}
            else {
                post.L[S]= add[st.L[S], ad] //Add to the policy
                ((ad.signe in Minus) and (S in CoopSite)) implies {
                    all oi :st.H[S]&
                    (tentative[st.CoopStatus[S]]-invalid[st.CoopStatus[S]]-
                    valid[st.CoopStatus[S]])|{(oi.type in ad.right and
                        oi.from in ad.subject) implies{
                            oi in invalid[post.CoopStatus[S]]
                            let uo=UndoneOp|{ uo.linkedOp=oi and
                                uo.owner=S and uo in post.H[S]} }}}}
        S->(versOrder/next[ st.versions[S]]) in post.versions }
}
```

Snippet 7: Processing of causally-ready administrative request by a site

4.3.3 Specification of Consistency Property

We deal with an access control model for DCE. Hence, property considered is related to safe and consistent collaboration. Firstly we consider the fundamental concept of stable state of the system. We define it as follows in definition 4.1.

```
fact ReceiveCoopRequestOrdSite{
    all st : (SiteState - sitesstates/first - sitesstates/last),
    o : st.CoopCausallyReady[CoopSite], S :(st.CoopCausallyReady.o)&CoopSite|
        let post= sitesstates/next[st]|{ (o.from in AdmSite) implies {
                o in valid[post.CoopStatus[S]] and o in post.H[S]}
            else{ o.from in CoopSite and
                ((some r : elems[st.L[S]] | {
                    versOrder/lt[o.vers, r.vers] and o.from in r.subject and
                    r.signe in Minus and o.type in r.right and
                    all rj : (elems[st.L[S]]-r) |{{versOrder/lt[o.vers, rj.vers] and
                        o.from in r.subject and o.type in rj.right} implies
                            lastIdxOf [st.L[S],rj] < lastIdxOf [st.L[S],r] }})
                or (no r : elems[st.L[S]] | {
                    versOrder/lt[o.vers, r.vers] and o.from in r.subject and
                    r.signe in Minus and o.type in r.right }))
                implies o in invalid[post.CoopStatus[S]]
                else {o in tentative[post.CoopStatus[S]] and o in post.H[S]}}}
}
```

Snippet 8: Processing of causally-ready cooperative request by a site

```
fact fact ReceiveCoopRequestAdm{
    all st : (SiteState - sitesstates/first - sitesstates/last),
    o : st.CoopCausallyReady[AdmSite], S : st.CoopCausallyReady.o&AdmSite|
        let post= sitesstates/next[st]|{(o.from in AdmSite) implies {
            o in valid[post.CoopStatus[S]] and o in post.H[S] }
            else{((some r : elems[st.L[S]] | {
                    versOrder/lt[o.vers, r.vers] and o.from in r.subject and
                    r.signe in Minus and o.type in r.right and
                    all rj : (elems[st.L[S]]-r) |{{versOrder/lt[o.vers, rj.vers] and
                        o.from in r.subject and o.type in rj.right}

                    implies lastIdxOf [st.L[S],rj] < lastIdxOf [st.L[S],r]}})
                or (no r : elems[st.L[S]] | {

                    versOrder/lt[o.vers, r.vers] and o.from in r.subject and
                    r.signe in Minus and o.type in r.right }))
                implies o in invalid[post.CoopStatus[S]]
                else {
                    o in valid[post.CoopStatus[S]]
                    let vad=Val |{
                        vad.op =o and vad.source=S and
                        vad.vers=versOrder/next[ st.versions[S]] and
                        (versOrder/next[st.versions[S]]) in post.versions[S]
                        and vad in st.sentAdReq[S]}
                        o in post.H[S]}}}
}
```

Snippet 9: Processing of causally-ready cooperative request by the administrator

Définition 4.1 ***(Stable state)*** *The system is said to be in a stable state iff each site completes the processing of all generated and received operations.*

Note that, according to the definition 4.1, it is possible to have several stable states during the collaboration. The consequence of the definition 4.1 is that at each stable state of the system, all sites have the same number version of policy. Recall that when an operation is generated, it is immediately sent to others. The broadcasting is considered as part of the generation process unless the operation is denied. Also we have assumed that the communication is message lossless and when an operation is sent by a site, it is received by others at the next state of the system.

Once the stable state is reached by the system, it is possible to verify if the protocol preserves its correctness. The property of interest relies on data consistency. The *data consistency property* allows to know if the protocol is enforced identically at all sites in context of dynamic changes of access rules. The goal is to prevent any security hole with regards to the granted and denied operations and the convergence of the shared document. The data consistency property is satisfied by the model iff for all stable state of the system, for all two disjoint sites, the sets of valid cooperative operations are the same. The data consistency property is specified using assertion (see **Snippet** 10).

As explained in Section 4.3.1, the scope indicates the maximal number of occurrences of each signature used during the searching, but also the maximal length of the execution trace. In our context, the scope denotes the maximum number of states (*SiteState*), sites, cooperative operations, administrative operations, etc. Considering that the collaboration held with concurrency, at each state, each site could do an action according to an operation (generate, send, received, process). To analyze the data consistency property, several checking scenarios are used by specifying different scopes. The results state "*No counterexample found. Assertion may be valid*". Note that the SAT-solver used is SAT4J.

```
1 assert ValidPreservation1{
2     all st :Stable[ ]|{ all disj Si, Sj :Site|{
3         valid[st.CoopStatus[Si]] in valid[st.CoopStatus[Sj]]
4         valid[st.CoopStatus[Sj]] in valid[st.CoopStatus[Si]]}}
5 }
```

Snippet 10: Data Consistency property

4.4 Related Work

Several access control models have been proposed in the literature for collaborative environment. An overview of access control models, including their principles, advantages and potential shortcomings, is available in [Tolone *et al.* (2005)]. Among these models there are Role-Based Access Control (RBAC) [Sandhu *et al.* (1996)] and its variants [Joshi *et al.*

(2004), Piromruen et Joshi (2005), Lee et Luedemann (2007)], which deal with a decentralized authorization. Their drawback is that they do not allow dynamic reassignment of roles. There is very little recent contributions on replicating authorization policies in the literature [Samarati *et al.* (1996), Xin et Ray (2007)]. These contributions deal with database systems and are not so flexible to support dynamic changes of authorization policies. For instance, Xin T. et al. used in [Xin et Ray (2007)], an extension of two-phase locking protocol as concurrency control technique to update policies. This technique is not suitable in the context of DCE.

Alloy has been used in several case study applications in security, access control and security policies domains with regards to model, framework or protocol. In [Hu et Ahn (2008)], Hu H. et al. presented a case study on verification for the NIST/ANSI standard model for Role Based Access Control (RBAC). The authors verified only one property related to the role deleting. The Alloy analyzer allowed them to conclude that the functional definition of *DeleteRole* function proposed by the NIST/ANSI standard for hierarchical RBAC misses a step for removing inheritance. In [Pai *et al.* (2011)], authors confirmed the known security vulnerability in oAuth, an open authentication protocol for the Web, using Alloy. Samuel A. et al. [Samuel *et al.* (2007)] specified the Generalized Spatio-Temporal Role Based Access Control (GST-RBAC) model using Alloy. The detection of some conflicts by the analyzer helped them to refine their proposed framework. Similarly, in [Toahchoodee *et al.* (2009)], the authors applied Alloy to specify and verify a spatio-temporal access control correctness. The analyzer showed possible conflict permissions assignment to the same role.

4.5 Conclusion

In this paper, we have presented a case study on formal specification and verification of a flexible access control protocol running on the top of a DCE. The purpose is to verify that data consistency of the DCE is preserved by the protocol. Proving data consistency of such systems is very challenging as they are infinite with a high degree of concurrency. To deal with these limitations, we have proposed an abstract model and used Alloy, a SAT-based bounded model-checker. We have shown that ACP preserves the correctness for several scopes. Alloy specification language, based on the first-order relational logic, is very appropriate to describe infinite systems. However, bounded model-checker techniques are known to be useful to find bugs but less appropriate to prove the absence of bugs. In the future, we plan to investigate the existence of a finite abstract model, which preserves data consistency. english

CHAPITRE 5

ARTICLE 3 : On Consistency Preservation with Optimistic Access Control for Distributed Collaborative Editors

Aurel Randolph ** , Hanifa Boucheneb**, Abdessamad Imine ††, and Alejandro Quintero**

Abstract

Distributed Collaborative Editors are interactive systems where several and dispersed users edit concurrently some shared documents. Generally, these systems rely on data replication and use safe coordination protocol which ensures data consistency even though the users's updates are executed in any order on different copies. Controlling access in such systems is a challenging problem, as they need dynamic access changes and low latency access to shared documents. To cope with this situation, a flexible access control protocol is proposed in [Imine *et al.* (2009)], based on replicating the shared document and its authorization policy at the local memory of each user. To deal with latency and dynamic access changes, an optimistic access control technique is used, allowing retroactive enforcement of authorizations. However, verifying whether the setup of access control over coordination protocols preserves the data consistency is a hard task, since it requires examining an unbounded number of situations, arising from the infinite nature of the system. In this paper, we propose a finite abstract model and show that it preserves the consistency. We use a symbolic model-checking tool Uppaal, to specify with automata, the behavior of the abstract model and the consistency requirement. Finally, we verify if it preserves the consistency of the system on which it is deployed in such a way that the access control policy is enforced identically at all participating user sites and, accordingly, the data consistency remains still maintained.

**. A. Randolph, H. Boucheneb and A. Quintero are with the Department of Computer and Software Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Québec, Canada, H3C 3A7. Email : {aurel.randolph, hanifa.boucheneb, alejandro.quintero}@polymtl.ca

††. A. Imine is with INRIA Grand-Est and Nancy-Université, France. Email : abdessamad.imine@loria.fr

Keywords Access control policies, distributed collaborative editors, Formal methods, Model checking, infinite system, data consistency, Correctness proofs, Symbolic execution, Uppaal.

5.1 Introduction

Distributed Collaborative Editors (DCE) enable several and dispersed users to form a group for editing simultaneously shared documents, such as articles, wiki pages and program source code. Google Drive is an example of such a system but in a centralized architecture. To achieve data availability, each user has a local copy of the shared document. Updates generated by each user are locally executed in a non blocking manner and then broadcast to the other sites in order to be executed on their copies. Although being distributed applications, DCE are specific in the sense they must consider human factors. Moreover, they are characterized by the following features: (i) High local responsiveness: the system has to be as responsive as its single-user editors [Ellis et Gibbs (1989),Sun *et al.* (1998),Sun *et al.* (2006)]; (ii) High concurrency: the users must be able to concurrently and freely modify any part of the shared document at any time [Ellis et Gibbs (1989), Sun *et al.* (1998)]; (iii) Scalability: a group must be dynamic in the sense that users may join or leave the group at any time; (iv) Consistency: the users must eventually see a converged view of all copies [Ellis et Gibbs (1989), Sun *et al.* (1998)]. Consistency preservation is one of the most critical properties in DCE. It stems on data replication and arbitrary exchange of updates. Accordingly, each DCE is endowed with *Coordination Protocol* (CP) to maintain globally consistent state.

Balancing the computing goals of collaboration and access control to shared information is a challenging problem in DCE [Tolone *et al.* (2005)]. Indeed, interaction in collaborative editors is aimed at making shared document available to all who need it, whereas access control seeks to ensure this availability only to users with proper authorization. To preserve the above cited DCE's features and avoid a central authority, a *flexible Access Control Protocol* (ACP) is proposed in [Imine *et al.* (2009)] where all updates will be checked at each user site without resorting to a central authority. In this model, a user will own two copies: the shared document and its authorization policies. This replication allows for high availability since when users want to read or update the shared document, this manipulation will be granted or denied by controlling only the local copy of the authorization policies. The execution order of updates on the shared document or authorization policy are arbitrary. An optimistic approach is used to tolerate momentary violation of access rights and afterwards, ensure the copies to be restored in valid states (by invalidating denied updates) w.r.t the stabilized access control policy.

To ensure a safe access control in DCE (i.e. permitting legal updates and rejecting illegal updates on the shared document), a protocol stack is built by integrating an ACP on the top of any CP based on data replication and update logging [Imine *et al.* (2009)]. If we combine a correct CP (i.e. satisfying separately the consistency property) with an ACP, can we verify that the consistency property is preserved by the new protocol ? This verification turns out a hard task and unmanageably complicated. Indeed, it requires examining an infinite number of situations since the updates are performed in different orders on different copies (arbitrary number) of the shared document and the authorization policy. Consequently, the verification of the combination correctness must be assisted by an automatic checker tool.

Contributions. We propose here a finite abstract model which is equivalent with regard to the requirement, to a collaborative system which uses ACP. We specify the abstract model with model-checking technique in order to verify the preservation of consistency property of any DCE integrating an ACP on the top of a consistent CP. For this purpose, we use a symbolic model-checking tool Uppaal [1], to specify the behavior of the abstract model and the consistency requirement. This choice is motivated by the possibility to handle symbolically queues of messages, logs, variables such as number of sites, number of operations generated by each site, etc. The consistency property is specified in CTL language and verified by Uppaal model-checker.

Outline. This paper is organized as follows: the flexible access control protocol is presented in Section 5.2 and formalized in Section 5.3. Section 5.4 is devoted to the investigation of a finite abstract model preserving the consistency property. Section 5.5 deals with the formal specification and verification of ACP. Section 5.6 discusses related work. Finally, the conclusion is presented in Section 5.7.

5.2 Optimistic Access Control Protocol for DCE

Shared documents are objects whose state can be altered by a set of *cooperative operations* generated by sites. For instance, a shared text document is modified by operations such as inserting a new section, deleting an existing paragraph and replacing an old line by a new one. In [Imine *et al.* (2009)], an access policy is described as an indexed list of authorization rules, where each rule is a quadruple $\langle S, O, R, \omega \rangle$ with (i) S is set of subjects (sites or users), (ii) O is a set of objects (e.g. paragraphs or chapters), (iii) R is a set of access rights (e.g. deleting or updating paragraphs) and (iv) $\omega \in \{-, +\}$. The sign "+" represents a right *attribution* and the sign "−" represents a right *revocation*.

Each site consists of two layers: coordination layer and security layer. The coordination

1. http://www.uppaal.com

layer is in charge of editing the shared document. The role of the security layer is to manage the access control policy. To achieve this objective, the security layer edits a local copy of a shared policy by producing some *administrative operations* with regards to the access control policy. Then, the state of the shared policy object is altered by a set of administrative operations such as adding and removing authorizations. Administrative operations are generated by the administrator, at any time, and aimed to manage dynamically the right access to the shared documents. These operations are next broadcast to other sites, in order to modify their local copies of the policy object. Thus, on each site, the cooperative operations are granted or denied by using the local copy of the policy and applying the first-match semantics: when an operation o is generated, the system checks o against its authorizations one by one, starting from the last authorization and stopping when it reaches the first authorization l that matches o. If no matching authorizations are found, o is rejected. Note that every local policy copy maintains a monotonically increasing version counter that is incremented by every administrative operation performed on this copy.

The collaboration happens in optimistic approach and modifications could be applied in different orders at different sites. The messages are assumed to be exchanged via secure and reliable communication network: each message sent is received by each others without alteration. The flow of messages exchanged during the collaboration is illustrated in Figure 5.1.

5.2.1 Generation of Local Cooperative Requests

Locally, each site generates some cooperative operations. The processing of the generated cooperative operation is shown at Figure 5.2. Each generated cooperative operation is firstly checked against the local policy. If the operation is not compliant with the local policy, it is said to be invalid and its execution is aborted. At the administrator site, when the operation is granted, its status is set to valid. At a non-administrator site, the status is set to tentative if it is granted. The operation is then performed immediately on the local

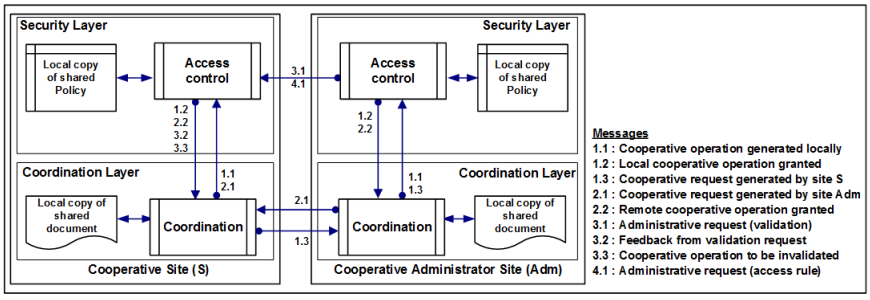

Figure 5.1 Flow of collaboration messages

copy of the shared document. A resulting cooperative request is generated and attached with the number version of the policy copy on which the operation is granted. This cooperative request is finally broadcast to other sites.

5.2.2 Reception of Remote Cooperative Requests

When a remote cooperative request is received, it is firstly stored in a dedicated queue before being extracted. The request is extracted if it is causally-ready. Thus, its attached version number of policy is less or equal than the current version of the local policy and its precedent cooperative request have been already integrated to the local copy of the shared document. This mechanism is setup to ensure that the access control protocol preserves the causality dependency with respect to the precedent administrative requests and precedent cooperative requests.

After its extraction, the remote cooperative request is checked against the local administrative log, to verify whether or not it is granted. When the remote cooperative request is granted, its status is set to valid if the receiver is administrator and to tentative, otherwise. If the receiver is administrator, the number version of its policy is incremented and a validation request is generated. The new version number is attached to the validation request which is broadcast to other sites. Afterwards, the corresponding cooperative operation is performed locally on the administrator's copy with regard to the collaborative editor's procedures. The processing of a remote cooperative operation is presented at Figure 5.3.

5.2.3 Generation of Administrative Operations

To manage and control the access, the administrator produces some access rules called administrative operations. When an administrative operation is generated, the version number of the administrator's local policy is incremented. Then, the policy is updated by performing on, the generated administrative operation. A corresponding administrative request with the latest version number of the policy is generated and broadcast to other sites to enforce their policy.

5.2.4 Reception of Remote Administrative Requests

There exist two kinds of remote administrative requests: validation request and access rule based request. Each received remote administrative request is first stored in a dedicated queue then, extracted when it is causally-ready. The administrative request is said to be causally-ready if the value of its attached policy version number is the next value of the version number of the local policy (the difference is one) and in case of a validation request,

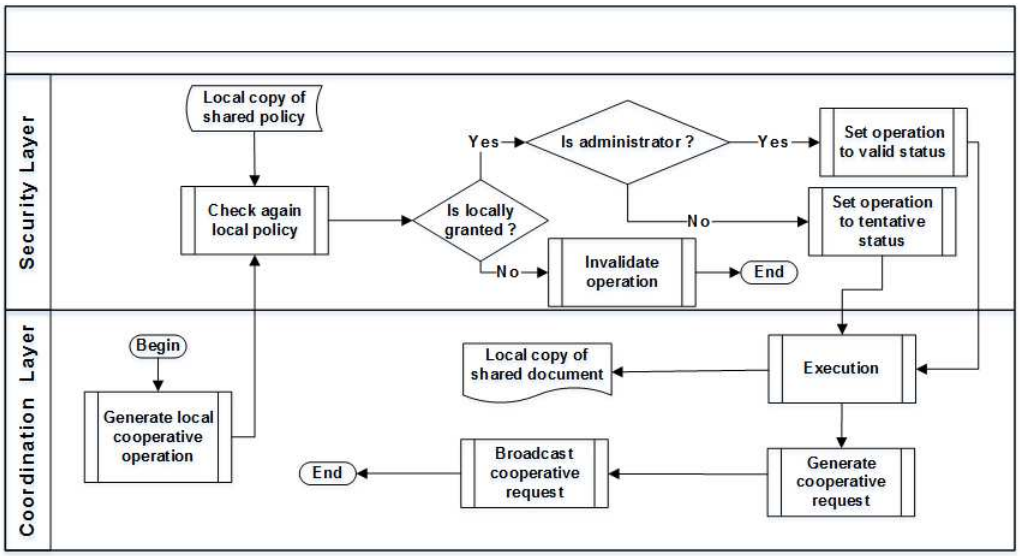

Figure 5.2 Processing of a cooperative operation at generation time.

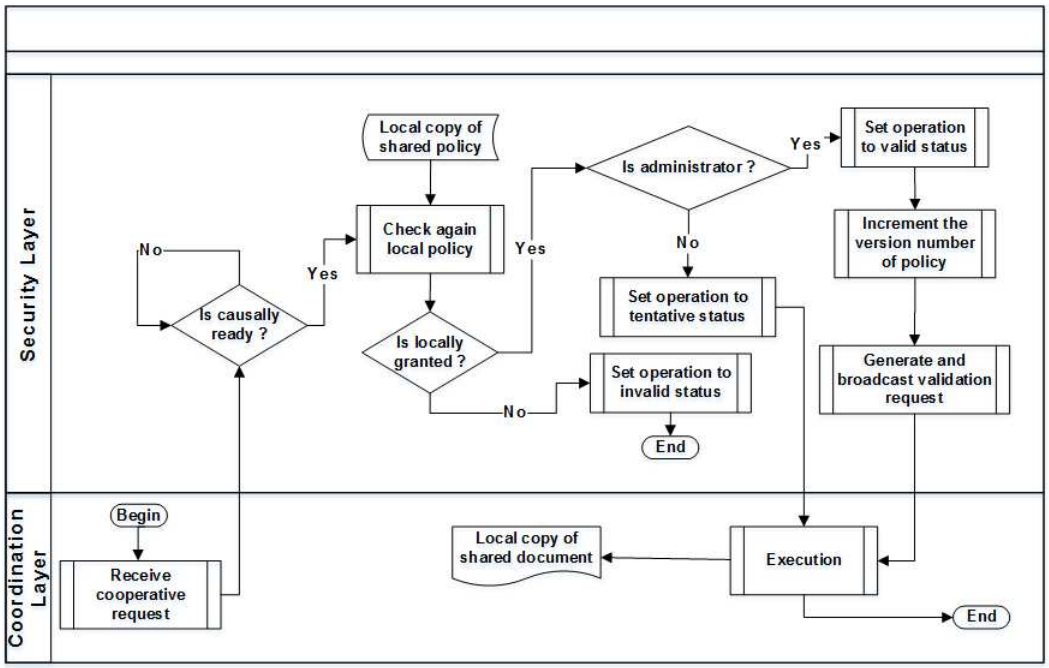

Figure 5.3 Processing of a received cooperative operation.

the corresponding cooperative operation is already executed on this site. Each extracted access rule based request, is performed on the local policy. Thereafter, if the access rule is restrictive, then all tentative cooperative operations, locally generated or received, which are concerned by the rule with regard to the rights, are undone. For the extracted validation request, the status of the corresponding cooperative operation is updated from tentative to valid. At the end of the treatment of the administrative request, the version number of the local policy is incremented. The processing of administrative request is shown at Figure 5.4.

5.2.5 Verification Issues

A Distributed Collaborative Editor consists of several sites. Each of them maintains the shared objects and its access right policy, by generating, exchanging and performing some cooperative and administrative operations. As the numbers of sites, cooperatives operations and administrative operations are arbitrary, the queues of cooperative requests and administrative requests are unbounded. The system is then infinite and parameterisable by the number of sites, the number of cooperative operations to be generated by each site, and the number of administrative operations to be generated by the administrator. On each site, the shared objects are modified with respect to the local access right policy. Meanwhile, the local policy is enforced by taking into account the administrative operations generated and broadcast by the administrator. Thus, if the policy is not enforced identically at all sites, it can result in the security hole on the shared objects by permitting illegal modifications or rejecting legal modifications. In addition, this situation can lead to data inconsistency for the collaborative edition such as the document can diverge at the end of the collaboration.

For instance, consider a group composed of an administrator *adm* and two sites S_1 and S_2. Initially, the three sites have the same shared document containing the text *bcd* and the same policy object where S_1 is authorized to insert characters (see Figure 5.5). Suppose that *adm* revokes the insertion right of S_1 and sends this administrative operation to S_1 and S_2 to enforce their local policy copies. Concurrently S_1 executes a cooperative operation $Ins(1, a)$ to derive the state "abcd" as it is granted by its local policy. When *adm* receives the S_1's operation, it will be ignored (as it is not granted by the *adm*'s local policy) and then the final state still remain *bcd*. As S_2 receives the S_1's insert operation before its revocation, he gets the state *abcd* that will be unchanged even after having executed the revocation operation. We are in presence of data inconsistency (the state of *adm* is different from the state of S_1 and S_2) even though the policy object is the same in all sites. In fact, the new policy object is not uniformly enforced among all sites because of the out-of-order execution of administrative and cooperative operations. Thus, security holes may be created. For instance some sites

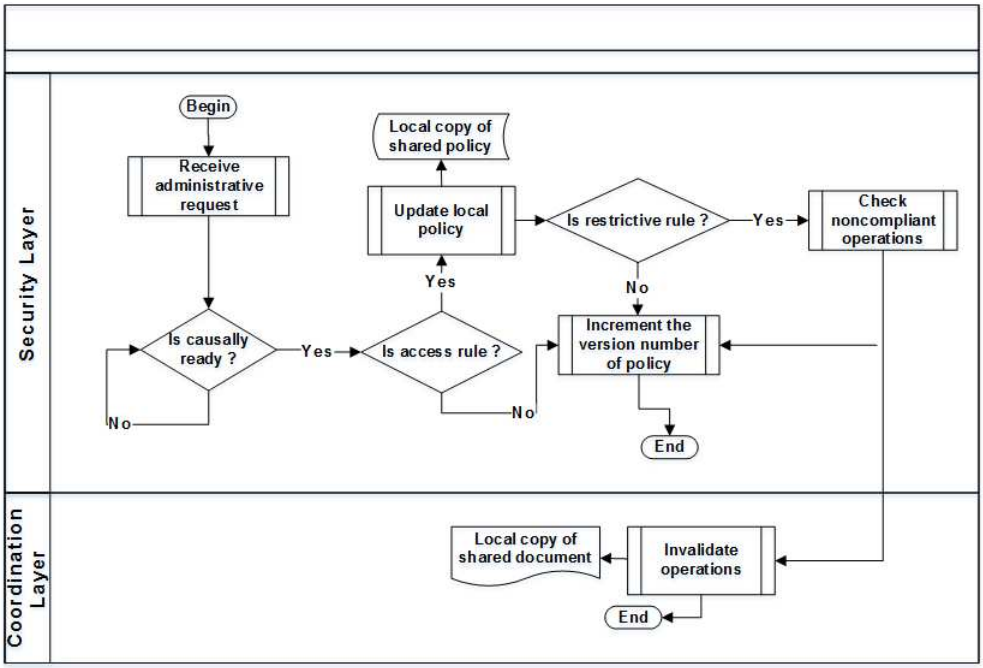

Figure 5.4 Processing of received administrative request.

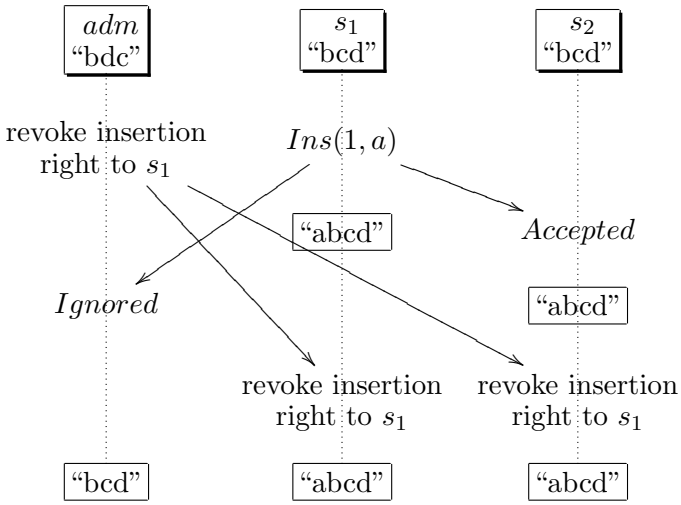

Figure 5.5 Divergence caused by introducing administrative operations

can accept cooperative operations that are illegal with respect to the new policy (e.g. sites S_1 and S_2).

To solve this problem, the ACP applies the principles of optimistic security [Povey (2000)] in such a way that the enforcement of the new policy may be retroactive with respect to concurrent cooperative operations. In this case, only illegal operations are undone. For instance, $Ins(1, a)$ as shown at Figure 5.5 should be undone at S_1 and S_2 after the execution of the revocation.

It appears important to verify that the ACP preserves the correctness of the collaborative editing system on which it is deployed with regards to the security issues and data consistency. For this purpose, the sets of legal (valid) cooperative operations must be identical at all sites, when all generated and received cooperative and administrative operations are performed on each site (stable state). Performing such a verification is tricky and hard to do manually. So, the system must be automatically checked using formal methods. However, the automatic checking tools could lead to severe state explosion, time consumption, result in a lack of memory and finally aborted, or could not cover all the state space needed for verification. For instance, previously, we experiment the verification issue with explicit model checking technique under Uppaal. The exploration does not scale over 4 sites. It results in a lack of memory and finally aborted. In addition, in [Randolph *et al.* (2013)], Alloy [2] is used to verify whether or not ACP preserves the correctness of DCE. Alloy Analyzer works by reduction to SAT, which technique is suitable for infinite systems. However, the analysis does not either cover the space state of the system. For this purpose, we investigate the existence of a finite abstract model which is equivalent to our infinite system with regards to the consistency property.

5.3 Definitions

In this section we give several formal definitions related to the collaboration and ACP. We are particularly interested in the evolution of the status of cooperative operations at each a site. Let us denote :

- $\mathcal{O}$, the set of cooperative operations.
- $\mathcal{A}$, the set of administrative operations.
- $POLICY$, the set of policies.
- $SITE$, the set of sites.

2. MIT Sofware Design Group, Alloy : A language and Tool for Relational Models. Retrieved April 4, 2014 from `http://alloy.mit.edu/alloy/`

Définition 5.1 *[**Model of the System**]. The system consists of an arbitrary number n of cooperative sites including one administrator. It is defined by a model $M_{\langle n,1\rangle} = (SITE, \mathcal{O}, \mathcal{A}, ACP)$ with ACP the access control protocol.*

Each site $S_i \in SITE$ has a local copy of the shared policy denoted P_i, with $P_i \subseteq POLICY$. The version number attached to the local copy of the shared policy is denoted v_i. Its value increases during the collaboration process. P_i is empty at the beginning of the collaboration such as v_i is initialized to 0. The administrator is denoted S_1.

During the collaboration, the system evolves as operations are processed at each site. These changes are traduced by different states which succeed each other.

Définition 5.2 *[**Local and Global State of the system**]. The local state of the system at site $S_i \in SITE$ is defined by its current number version of the policy, the administrative log and the set of cooperative operations available (generated and received) at this site. We denote $STATE_i$, the set of local states of the system at S_i.*

We define a relation $<$ over $STATE_i$ such as for all $st_{\alpha i}$, $st_{\beta i}$ in $STATE_i$, $st_{\alpha i} < st_{\beta i}$ if and only if $st_{\beta i}$ could be reached from $st_{\alpha i}$, after the execution of a sequence of actions of site S_i. $st_{\beta i}$ is the successor of $st_{\alpha i}$ if it could be reached from $st_{\alpha i}$, after the execution of a single action.

The global state of the system is a tuple of local states of all sites. We denote $STATE$, the set of global states of the system.

$$STATE = \{st, st = (st_1..., st_i, ..., st_n) \text{ with } st_i \in STATE_i, n = |SITE|\} \tag{5.1}$$

We define a relation $<$ over $STATE$ such as for all st_α, st_β in $STATE$, $st_\alpha < st_\beta$ if and only if there exists at least one site S_i such as the component $st_{\beta i}$ of st_β could be reached from the component $st_{\alpha i}$ of st_α, after the execution of a sequence of actions of site S_i, with $st_{\alpha i} \in STATE_i$ and $st_{\beta i} \in STATE_i$.

When the system evolves, each cooperative operation gets different status at each site. The changes of status are the result of applying ACP to the collaboration process.

Définition 5.3 *[**Status of a coopreative operation**]. At any site, a cooperative operation could have (i) a tentative status iff the operation is not validated yet, (ii) an invalid status iff it is denied or unauthorized, (iii) a valid status iff it is validated.*

We denote $STATUS$, the set of status of a cooperative operation.

$$STATUS = \{tentative, invalid, valid\} \tag{5.2}$$

Définition 5.4 *[**Transition Relationships**]. The transition relationships δ between local states of the system at site $S_i \in SITE$ is a relation from the cartesian product $STATE_i \times Act$ to $STATE_i$, where Act is the set of actions related to the execution process of the protocol at S_i.*

$$\delta : STATE_i \times Act \rightarrow STATE_i \tag{5.3}$$

We associate the operator $\rightsquigarrow$ with the transition relation δ to indicate the change of status of an operation as defined in the statement 5.4.

$$\forall st_{\alpha i}, st_{\beta i} \in STATE_i, st_{\alpha i} \rightsquigarrow st_{\beta i} \Longleftrightarrow st_{\alpha i} < st_{\beta i} \wedge \exists a \in Act, \delta(st_{\alpha i}, a) = st_{\beta i} \tag{5.4}$$

Définition 5.5 *[**Causality**]. The causality relationship between operations is expressed by the Causally-ready function (denoted cReady). It maps a received cooperative operation or a received administrative request, a site S_i and a state of the system, to a boolean value which is true if the received cooperative operation or the received administrative request is causally-ready at S_i in the state st and false otherwise.*

$$cReady : (\mathcal{O} \cup \mathcal{A}) \times SITE \times STATE \rightarrow \{true, false\} \tag{5.5}$$

Recall that a received cooperative operation is causally-ready iff its attached version number of policy is less or equal than the current version number of the local policy and its precedent cooperative operation have been already integrated to the local copy of the shared document. The received administrative request is causally-ready if the value of its attached version number of policy is one unit greater than the current version number of the local policy and in case of validation request, the corresponding cooperative operation is already executed on the site.

The causally-ready notion is applied to remote cooperative and administrative requests. According to the access control protocol, when a cooperative request is causally-ready, the remote-checking process is executed. The objective of this process is to verify if the received operation is compliant with the local administrative log. To deel with the result of this stage, we use a boolean function as presented in Definition 5.6.

Définition 5.6 *[**Remote-Checking**]. The remote-checking against administrative log denoted checkR() is a function that maps a cooperative operation, a site and a state of the system, to a boolean value which is the result of checking the considered operation against the administrative log. The function returns true iff the operation is not denied by the remote-checking process, and false otherwise.*

$$checkR : \mathcal{O} \times SITE \times STATE \rightarrow \{true, false\} \tag{5.6}$$

For instance, with the cooperative operation o at the site S_i, when the system is in the state st, $checkR(o, S_i, st) = true$ means that the operation is granted and $checkR(o, S_i, st) = false$ states that the operation is denied.

Similarly to a remote cooperative operation, a remote administrative operation based on rule is used in a checking process. Thus the administrative operation is used to verify whether or not some cooperative operations with the *tentative* status are compliant with the related access rule. The role of the function $checkT$ is to return the result of this processing, as indicated in Definition 5.7.

Définition 5.7 *[**Checking of Tentative Operation**]. The checking at a site S_i (denoted $checkT$) of a restrictive received administrative request against a cooperative operation which is in the status tentative, is a function that maps a cooperative operation, a site, an administrative operation (request) and a state of the system, to a boolean value which is the result of checking the operation against the restrictive received administrative request. The function returns true iff the operation is not denied by the access rule corresponding to the administrative request, and false otherwise.*

$$checkT : \mathcal{O} \times SITE \times \mathcal{A} \times STATE \rightarrow \{true, false\} \tag{5.7}$$

A remote administrative operation which is not based on rule, is related to the validation of a cooperative operation. According to the protocol, when this administrative operation is causally-ready, it is performed in order to change the status of the related cooperative operation. To know whether or not the validation request is already performed for an operation at a site, we define a function denoted $perfV$ as stated at Definition 5.8.

Définition 5.8 *[**Validation Request Performed**]. Let us consider a cooperative operation o, a site S_i and a global state st of the system. The function $perfV$ maps a cooperative operation, a site and a global state of the system, to a boolean value. It returns true iff the validation request related to the cooperative operation o is already performed at S_i when the system is in the state st and false otherwise.*

$$perfV : \mathcal{O} \times SITE \times STATE \rightarrow \{true, false\} \tag{5.8}$$

As we are interested in the evolution of the status of cooperative operation at each site, it appears necessary to have the information about the status. This task is completed by a function denoted val (see Definition 5.9).

Définition 5.9 *[**Operation Valuation**]. The operation valuation denoted val is a function that maps a cooperative operation, a site and a global state of the system to the status of the operation at this site when the system is in the considered state. It returns the status of the operation for the parameters indicated.*

$$val : \mathcal{O} \times SITE \times STATE \rightarrow STATUS \tag{5.9}$$

Définition 5.10 ***[Stable State]****. The system is said to be in a stable state iff each site completes the processing of all generated and received operations. The subset of the stable states of the system is denoted $STABLE$. We have : $STABLE \subset STATE$.*

Définition 5.11 ***[Satisfiability of the Data Consistency Property]****. The data consistency property φ is satisfied at a stable state st of the system iff the valuation of each processed cooperative operation is the same at each site.*

$$\forall st \in STABLE, st \models \varphi \quad \Leftrightarrow \quad \forall o \in \mathcal{O}, \forall S_i, S_j \in SITE, val(o, S_i, st) = val(o, S_j, st) \tag{5.10}$$

5.4 A finite abstract model preserving consistency property of ACP

As the consistency property means that the decision taken for an operation is the same at each site, we will focus on a model based on the status of operations. According to ACP, the status of an operation can change as expressed in Property 1.

Property 1 *The possible evolutions of the status of a cooperative operation at any site are :*

$$tentative \rightsquigarrow invalid \mid valid \tag{5.11}$$

This property states that at any site, the status *tentative* of any cooperative operation (local or remote) could be changed in *invalid* or *valid*. The status *valid* and *invalid* are final as they are maintained and do not change.

5.4.1 Generation and Reception of a Cooperative Operation

The cooperative operations which are denied locally at generation time are not considered as they are not known by other users and consequently have no impact on the consistency property. So, we consider only the cooperative operations locally granted at generation time and broadcast to other sites. The properties related to an operation at the generation or the reception time are as follows.

Property 2 *At any non administrator site, the status of any cooperative operation locally granted, at generation time, is tentative at this site.*

Property 3 *At the administrator site, the status of any cooperative operation locally granted, at generation time, is valid at this site.*

Property 4 *The status of any remote cooperative operation at reception time is the same as at its generation time.*

The properties 2 and 3 infer that initially, any cooperative operation locally generated holds either tentative or valid status. The property 4 states that the initial status of any remote cooperative operation does not change during the message sending process.

5.4.2 Execution Process

To investigate a finite abstract model preserving consistency property of ACP, we propose to examine the execution process cases and their resulting status for any considered cooperative operation. Let o be any cooperative operation and S_i be a site. According to the ACP, the different processing scenarios of the cooperative operation o at site S_i are:

1. Scenario 1 : o is a local operation at S_i (the owner is S_i) and S_i is the administrator.
2. Scenario 2 : o is a local operation at S_i (the owner is S_i) and S_i is not the administrator.
3. Scenario 3 : o is received by S_i and S_i is the administrator. The owner of o is not the administrator.
4. Scenario 4 : o is received by S_i. S_i is not the administrator but the administrator is the owner of o.
5. Scenario 5 : o is received by S_i. S_i is not the administrator and the administrator is not the owner of o.

Based on these five scenarios, we point out several properties.

Property 5 *Let o be a local cooperative operation at the administrator site S_1, st_α be a global state of the system at the generation time of o.*

$$\forall st_\beta \in STATE - \{st_\alpha\}, st_\beta > st_\alpha \wedge val(o, S_1, st_\beta) = valid \tag{5.12}$$

This property follows from the properties 1 and 3. Indeed, the status of any cooperative operation granted at generation time by the administrator site is *valid* which is a final status. Then for any state the status of this kind of operation is maintained to valid at the administrator site.

Property 6 *Let o be a local cooperative operation at S_i which is not the administrator ($i \neq 1$), st_α be the global state of the system when o is granted and broadcast by S_i, ar be an access*

rule based administrative request, $vr(o)$ a validation request related to the operation o.

$$val(o, S_i, st_\alpha) = tentative \tag{5.13}$$

$$\forall st_\beta, st_\gamma \in STATE, st_\beta < st_\gamma \wedge val(o, S_i, st_\beta) = tentative \wedge checkT(o, S_i, ar, st_\beta) = false \Rightarrow$$
$$val(o, S_i, st_\gamma) = invalid \tag{5.14}$$

$$\forall st_\beta, st_\gamma \in STATE, st_\beta < st_\gamma \wedge val(o, S_i, st_\beta) = tentative \wedge checkT(o, S_i, ar, st_\beta) = true \Rightarrow$$
$$val(o, S_i, st_\gamma) = tentative \tag{5.15}$$

$$\forall st_\beta, st_\gamma \in STATE, st_\beta < st_\gamma \wedge val(o, S_i, st_\beta) = tentative \wedge perfV(o, S_i, st_\beta) = true \Rightarrow$$
$$val(o, S_i, st_\gamma) = valid \tag{5.16}$$

The statement (5.13) follows from the Property 2. When broadcast by the local site, the status of the cooperative operation is *tentative*. The statement (5.14) states that any local cooperative operation with the status *tentative* is invalidated by any administrative request which denies the execution of this cooperative operation. It involves a new status for the operation such as *invalid*. The statement (5.15) states that any local cooperative operation with the status *tentative* maintains this status if any administrative request does not deny its execution. The statement (5.16) states that any local cooperative operation with the status *tentative* which validation request is processed locally is set to be valid. It involves a new status for the operation such as *valid*.

Property 7 *Let o be any remote cooperative operation at site S_1 which is the administrator.*

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_1, st_\alpha) = tentative \wedge checkR(o, S_1, st_\alpha) = true \Rightarrow$$
$$val(o, S_1, st_\beta) = valid \tag{5.17}$$

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_1, st_\alpha) = tentative \wedge checkR(o, S_1, st_\alpha) = false \Rightarrow$$
$$val(o, S_1, st_\beta) = invalid \tag{5.18}$$

The property states that any remote cooperative operation which is granted at administrator site using the check-remote function is said to be valid. The operation is said to be invalid if it is pointed out as denied by the check-remote function.

Property 8 *Let o be a remote operation at site S_i ($i \neq 1$) such as the owner is the administrator.*

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = valid \wedge cReady(o, S_i, st_\alpha) = true \Rightarrow$$
$$val(o, S_i, st_\beta) = valid \tag{5.19}$$

The property states that when any remote cooperative operation which owner is the administrator is processed, its status is set to *valid*. With regards to the protocol, if o is

causally-ready at S_i, it is not checked against the administrative log before being processed as its owner is the administrator. Then, it maintains its valid status.

Property 9 *Let o be a remote operation at a non-administrator site S_i such as its owner is not the administrator, ar be an access rule based administrative request, $vr(o)$ a validation request related to the operation o.*

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = tentative \wedge checkR(o, S_i, st_\alpha) = false \Rightarrow val(o, S_i, st_\beta) = invalid \quad (5.20)$$

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = tentative \wedge checkR(o, S_i, st_\alpha) = true \Rightarrow val(o, S_i, st_\beta) = tentative \quad (5.21)$$

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = tentative \wedge checkT(o, S_i, ar, st_\alpha) = false \Rightarrow val(o, S_i, st_\beta) = invalid \quad (5.22)$$

$$\forall st_\alpha, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = tentative \wedge checkT(o, S_i, ar, st_\alpha) = true \Rightarrow val(o, S_i, st_\beta) = tentative \quad (5.23)$$

$$\forall st_{i\alpha}, st_\beta \in STATE, st_\alpha < st_\beta \wedge val(o, S_i, st_\alpha) = tentative \wedge perfV(o, S_i, st_\alpha) = true \Rightarrow val(o, S_i, st_\beta) = valid \quad (5.24)$$

The statement (5.20) indicates that any remote cooperative operation checked against the administrative log and revealed to be denied is said to be invalid. On the contrary when revealed to be not denied, the operation hold the status *tentative* as pointed out by the statement (5.21). The statement (5.22) states that any remote cooperative operation with the status *tentative* is invalidated by any administrative request which denies the execution of this cooperative operation. It involves a new status for the operation such as *invalid*. The statement (5.23) states that any remote cooperative operation with the status *tentative* maintains this status if any administrative request does not deny its execution. The statement (5.24) states that when the validation request of any remote cooperative operation with the status *tentative* is processed locally, this operation is set to be valid. It involves a new status for the operation such as *valid*.

Lemma 2 *At each stable state of the system, at each site, the status of any cooperative operation (local or remote) is invalid or valid.*

Proof 3 *In a stable state, all operations are processed (see Definition 5.10). As, no processing is pending, no operation has a tentative status, the operations hold one of the status invalid or valid. According to the property 1, invalid and valid are final.*

5.4.3 Analyzis of the Execution Process

The behaviour of each site consists of the execution of several functions devoted to : verify whether or not some requests are causally-ready (*cReady*), check remote cooperative operations (*checkR*), and check tentative operations using received administrative requests (*checkT*), respectively. This list of functions does not take into account the collaboration tasks such as generate and execute cooperative operation which are done by the Coordination Layer. However this list could be completed with the performing of validation requests when they are causally-ready and the undoing of cooperative operation requests. The execution of these functions has the effect to change the status of the operation as pointed out with the properties 5, 6, 7, 8 and 9 and Lemma 2. The diagram shown at Figure 5.6 presents the evolution of the status of the cooperative operation at any site with regards to its features : administrator or not, owner or not.

Let us consider a cooperative operation owned by the administrator. This case is implemented by the scenario 1 of the execution process. Examining the list of scenarios presented at Section 5.4.2 and the diagram shown at Figure 5.6, the scenario 1 must be associated with the scenario 4. Indeed, when the owner of the operation is the administrator, the operation is remote at other sites.

Lemma 3 *In a stable state, any operation owned by the administrator has the same status at all other sites.*

Proof 4 *Let us consider a collaborative system with an arbitrary number of sites with one administrator. Let us also consider one operation and assume that its owner is the administrator. The operation is remote at other sites. Based on the properties 5 and 8 illustrated by the edges labelled scenario 1 and scenario 4 at Figure 5.6, the only one final status for the remote cooperative operation is valid as soon as its processing is completed. Thus, from the state following the processing of the remote cooperative operation, to the end the collaboration, it holds the status valid as at the owner's (administrator) site. Particularly, in any followed stable state of the system, the remote operation has a valid status as at the owner's site.*

Lemma 4 *Let us consider a collaborative system with an arbitrary number n of sites including one administrator. The number of sites needed to cover the execution process when the cooperative operation is generated by the administrator is two.*

Proof 5 *According to Lemma 3, non-administrator sites have similar behavior with regards to any operation generated by the administrator. Thus, the received operation holds a valid status after being processed. To cover the Scenario 1 in which the operation o is generated by*

the administrator, one non-administrator site will be sufficient. Thus, we have a system of two sites.

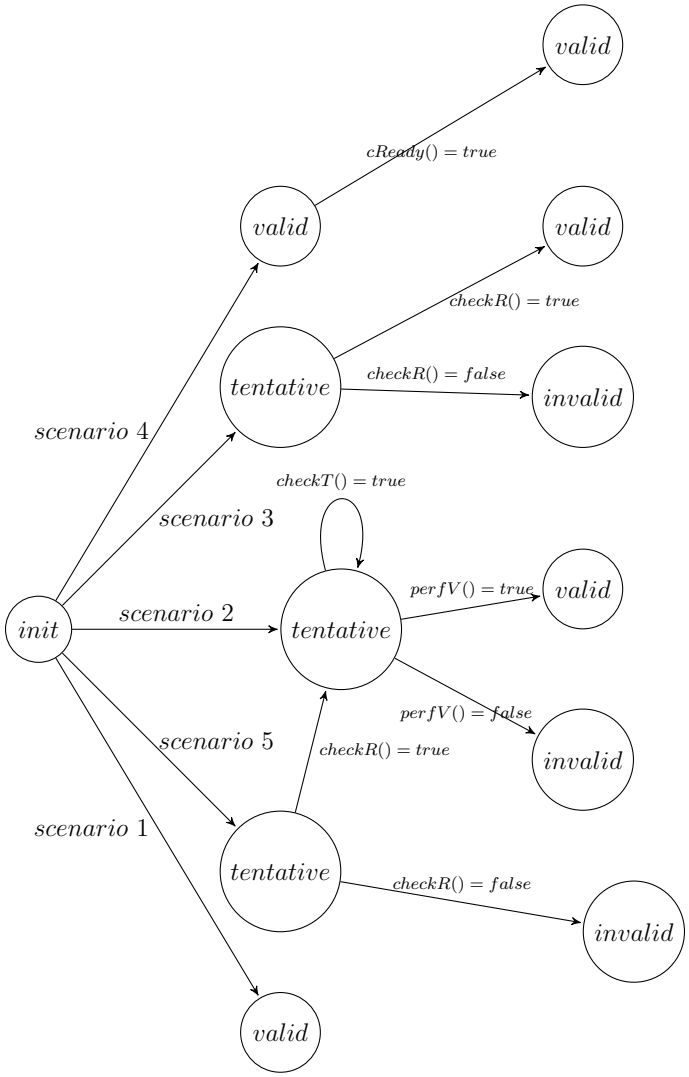

Figure 5.6 Execution process at any site.

Let us consider now a cooperative operation generated by a non-administrator. This case is implemented by the scenario 2 of the execution process. Examining the list of scenarios presented at Section 5.4.2 and the diagram shown at Figure 5.6, the scenario 2 must be associated with the scenarios 3 and 5. Indeed, when the owner of the operation is not administrator, the operation is remote at other sites including at administrator site.

Lemma 5 *Let us consider a cooperative operation generated by a non-administrative site. The number of sites needed to cover the execution process when the operation is not generated*

by the administrator is three.

Proof 6 *Let us consider a system which consists of an arbitrary number n of sites, and assume that a cooperative operation is generated by a non-administrative site. The system could be split in three (03) subsets : the owner of the operation as a non-administrator, the administrator, and the other $(n-2)$ non-administrator sites. These 3 subsets implement the scenario 2, scenario 3 and scenario 5 of the execution process, respectively. One site is sufficient to represent all the $(n-2)$ sites of the third subset as they have similar behavior with regards to any received operation (Property 9). Then to cover this scenario three sites will be sufficient.*

Theorem 2 *Let us consider a collaborative system with an arbitrary number n of sites. To cover all the execution process cases, the system could be sufficiently abstracted to a new system of three sites including the administrator.*

Proof 7 *(i) With the Lemma 4, two sites are sufficient to cover the execution process when the administrator generates a cooperative operation (scenario 1 and scenario 4). (ii) With the Lemma 5, three sites are sufficent to cover the execution process when a non-administrator generates a cooperative operation (scenarios 2, 4 and 5). This latter case (ii) can be considered as extension of the former case (i) such as it contains the administrator and at least one non-administrator. Then to cover the five scenarios considered at Section 5.4.2, we consider the maximum number figured out (between 2 and 3).*

We establish in Theorem 2 that the system could be sufficiently abstracted to a new system of three sites including the administrator. It appears now important to establish that the infinite system and the abstract model are equivalent with regards to the consistsency property. This is done with the Theorem 3 and its proof.

Theorem 3 *Let $M_{\langle n,1\rangle}$ be a model of a collaborative system which uses ACP and consists of an arbitrary number n of sites including one administrator, $M'_{\langle 3,1\rangle}$ be the corresponding abstract model which consists of 3 sites including the administrator and φ be the consistency property. The abstract model $M'_{\langle 3,1\rangle}$ satisfies the consistency property iff the collaborative model $M_{\langle n,1\rangle}$ satisfies the consistency property.*

$$M'_{\langle 3,1\rangle} \models \varphi \quad \Leftrightarrow \quad M_{\langle n,1\rangle} = (SITE, \mathcal{O}, \mathcal{A}, ACP) \models \varphi \tag{5.25}$$

Proof 8 *Assume that $SITE = \{S_1, S_2, S_3, ..., S_n\}$ with S_1 the administrator and let $M'_{\langle 3,1\rangle}$ consists of $\{S_1, S_i, S_j\}$with $i \neq j$.*

$M_{\langle n,1\rangle} \models \varphi \Rightarrow M'_{\langle 3,1\rangle} \models \varphi$ is trivial when setting $n = 3$ and considering that $M_{\langle 3,1\rangle} \models \varphi \Rightarrow M'_{\langle 3,1\rangle} \models \varphi$. Assume that $M'_{\langle 3,1\rangle} \models \varphi$. The system $\langle S_1, S_2, S_3\rangle \models \varphi$ like any system $\langle S_1, S_i, S_j\rangle$ with $i \neq j$, $i \geq 4$and $j \geq 4$. While adding S_4 to the system $\langle S_1, S_2, S_3\rangle$, S_4 could be associated with S_2 (respectively S_3) such as, when S_2 (respectively S_3) owned an operation, the behavior of S_3 (respectively S_2) and S_4 is similar with regards to the status of the operation. Also, when S_4 owned an operation, the behavior of S_2 and S_3 is similar with regards to the status of the operation. Then $\langle S_1, S_2, S_3\rangle \models \varphi \Rightarrow \langle S_1, S_2, S_3, S_4\rangle \models \varphi$. Iteratively, the site S_n could be added to $\langle S_1, S_2, S_3, ..., S_{n-1}\rangle$ such as $\langle S_1, S_2, S_3, ..., S_{n-1}\rangle \models \varphi \Rightarrow \langle S_1, S_2, S_3, ..., S_{n-1}, S_n\rangle \models \varphi$, from which $M'_{\langle 3,1\rangle} \models \varphi \Rightarrow M_{\langle n,1\rangle} \models \varphi$.

5.5 Model Checking of the Abstract Model

Several model checking techniques have been proposed in the literature. These techniques can be classified into *explicit state model checker* and *Symbolic model checker*. In explicit state model checker [Holzmann (2004)], states, sets and relations are explicitly represented, whereas, in symbolic model checker, they are implicitly represented using boolean logic formulas. The category of symbolic model checker can be subdivided into *SAT-based bounded model checkers* [Schaeffer-Filho *et al.* (2009)] and *BDD-based model checkers* [Cimatti *et al.* (2000)]. In SAT-based bounded model checking, the basic idea is to search for a counterexample in traces whose length is bounded by some integer k [Frappier *et al.* (2010)]. If no bug is found then k is increased until either a bug is found or the computer resource limits are reached. BDD-based model checking allows to prove by considering the whole state space of the model that some property is satisfied but it does not scale well in practice. However, when the state space is not too big, the BDD-based approach could be appropriate. The Collaborative System of n arbitrary number of sites using ACP over the CP is reduced previously to an equivalent system of 4 sites. This reduced system allows significant gain in both space and time for verification. A BDD-based model checker is then the most suitable to verify the ACP. We propose to use the tool suite of Uppaal [1] to verify that ACP preserves consistency of DCE.

5.5.1 Uppaal

UPPAAL is a tool suite for symbolic model-checking generally used for validation and verification of real-time systems but also used for non real-time systems. In addition to its symbolic model-checker, Uppaal offers a graphical editor for system descriptions and a graphical simulator. The description model is a set of timed automata [Alur et Dill (1994)] (or

1. http://www.uppaal.com

simple automata) extended with binary channels, broadcast channels, C-like types, variables and functions. The simulator allows in addition, to get and replay, step by step, counterexamples obtained by its symbolic modelchecker. The model-checker is based on a forward on-the-fly method, allows to compute over 5 millions of states.

5.5.2 Description of the system

To formally describe the system using the UPPAAL tool, three models are defined. The first model specifies the behavior of a cooperative operation at each site. The second model represents an administrative request (rule-based or validation request). The third model defines the behavior of ACP . The system described does not take into account CP as well as it performs operations allowed by ACP. Thus, only the final status of operations are required for analysis. The sites are represented by their identifers.

In a distributed collaborative editing system, sites (users) communicate via a network. The network is abstracted and not explicitly represented. This is done by setting global variables which store informations related to cooperative operations and administrative requests, instead of using and managing queues of messages.

Sites

The sites are represented by their identifers. The number of sites is declared as in statement 5.26 and the type of the identifiers of sites is declared as established in statement 5.27.

$$const\ int\ NbSites = 3; \tag{5.26}$$

$$typedef\ int\ [0, NbSites - 1]\ id_site; \tag{5.27}$$

Evolution of Cooperative Operation

The Uppaal process called *Op* traduces the evolution of the operation, pointing out the succession of its status and different events which occured during ACP processing. The process behavior of cooperative operation is depicted by the automaton shown in Figure 5.7. As the operation is executed at each site, the parameter of the process is the site identifier named *id*. The site identifier is attributed as indicated in statement 5.28. Then, an operation *op* is represented at any site by *op[id]*.

$$const\ id_site\ id \tag{5.28}$$

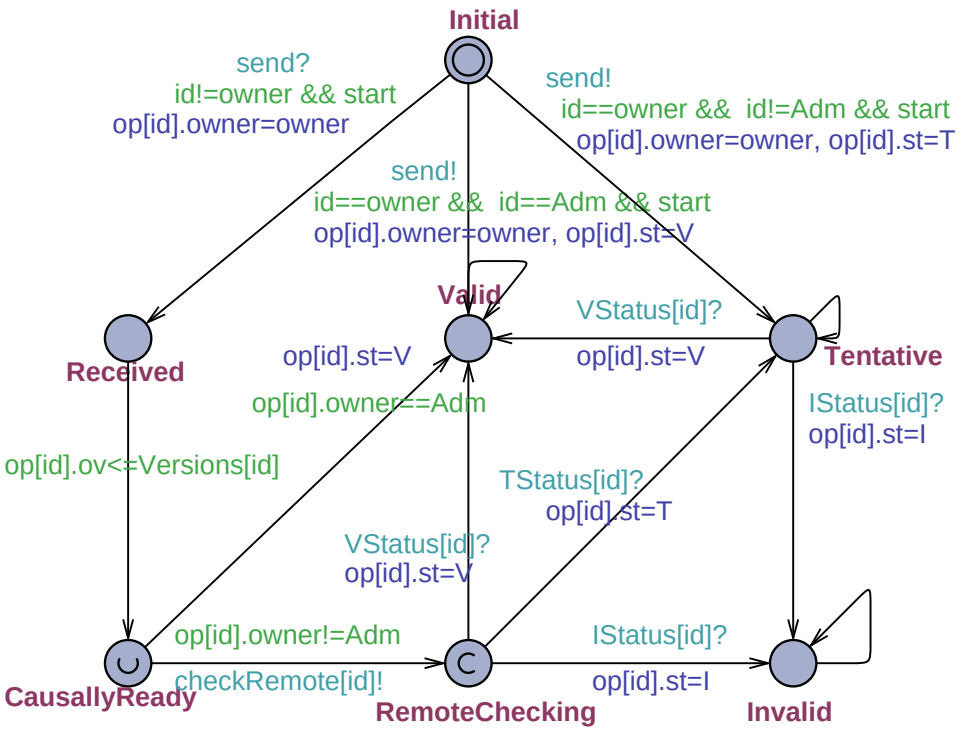

Figure 5.7 The model of cooperative operation.

In the model of the operation, starting by the location named *Initial*, a broadcast channel (called *send*) is used to broadcast the operation from the sender to other sites. The next location is *Valid*, if the owner is the administrator and the status of the operation is set to *valid* (indicated by *op[id].st=V*). When the owner is not the administrator, the next location is called *Tentative* as well as the operation's status is set to *tentative* (indicated by *op[id].st=T*). Otherwise, the next location is *Received* and the status is maintained *tentative*. When the operation is causally-ready at the remote site, the automaton move to the location *CausallyReady*. From this location the operation could access to *Valid* location with *valid* status if its owner is administrator or start-off the remote-checking procedure. In case of success, the result of the execution of this procedure is to set the status either to *valid* or *tentative* by moving to location *Valid* or *Tentative*, respectively, according to whether the remote site is administrator or not. When the remote-checking procedure denies the operation, the next location is *Invalid* and the corresponding status is *invalid* (indicated by *op[id].st=I*). Binary channels *VStatus*, *IStatus*, and *TStatus* are used jointly with ACP to synchronise status changes and corresponding location.

Evolution of Adminstrative Request

The process model devoted to administrative request is called *AdReq* and shown at Figure 5.8. This model depicted just the evolution of administrative request at non-administrator site. Indeed, there is no particular evolution at administrator site for administrative request except the starting-off of policy update by rule-based administrative request. A global variable reprensenting administrative request is then used by administrator.

The process has two parameters : the remote site identifier (*id*) and administrative request identifier (*r_id*). To show that the request is available at remote site, it firstly stays in the location *Received*. When it is causally-ready in the model, the request moves to the location *CausallyReady*. With regards to the protocol, the behavior of an administrative request after being causall-ready, depends on its type. Indeed, a rule-based administrative request is used to check cooperative operation with *tentative* status and the policy is updated locally. Also, validation-based administrative request triggers the update of the status of the corresponding cooperative operation, from *tentative* to *valid*. To execute such procedures, the administrative request sets up a synchronization with ACP using binary channel (*TentativeChecking[id]!* or *UpdateOperation[id]!*, respectively). The final location is then *Applied* to indicate that the administrative request is peformed completly.

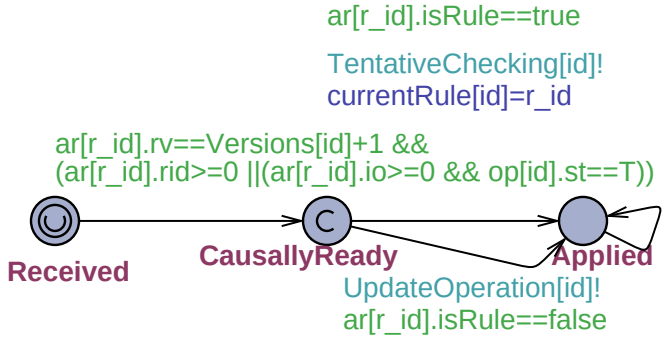

Figure 5.8 The model of administrative request.

ACP

The main process of the system is called *ACP* and depicted by the automaton in Figure 5.9. This process is executed by each site (administrator or non-administrator). The only one parameter of the process is the site identifier named *id*. According to the protocol, ACP is responsible to generate and broadcast administrative requests (only by administrator), receive and perform administrative request (only by non-administrator) and check received

cooperative operation against policy (or administrative log) .

From the *Initial* location, to generate a rule-based adminsitrative request, the model uses the function *RuleReq()*. The generation of the request is completed by the increasing of the number version of the policy (*Versions[id]++*) and its update by adding the new rule (function *updatePolicy()*). To perform a rule-based administrative request at a remote site (non-administrator), the model calls the function *checkAgainstRule()* which results on checking all cooperative operations with *tentative* status at the current site. The execution of this function starts as soon as the automaton receives the binary synchronized message *TentativeChecking[id]?* from *AdReq* automaton, with the update of the number version of the policy, followed by adding the corresponding rule to the policy.

To generate a validation-based adminsitrative request, the model uses the function *validationReq()*, preceded by the increasing of the number version of the policy (*Versions[id]++*). To perform a validation-based adminsitrative request at remote site, the binary synchronized channel *UpdateOperation* is used. Note that the broadcast of a new validation-based administrative request is abstract by setting a global variable as for a new rule-based administrative request.

To check received cooperative operations against policy (or administrative log), the binary channel *checkRemote[id]* is used followed by the execution of the function *checkAgainstPolicy()*.

5.5.3 Consistency Property and Verification of the system

To make arbitrary the choice of site as owner of cooperative operation, we add an automaton as shown in Figure 5.10. In addition, this automaton helps to indicate, if the executions of the cooperative operations are performed by ACP at each site. In this case, the process stays in the location *Final*.

$$system\ Supervisor,\ Op,\ AdReq,\ ACP; \tag{5.29}$$

Using UPPAAL, the definition of the system is given by the declaration shown at statement 5.29. This declaration means that the system consists of Supuervisor, Cooperative operations, Administrative requests and ACP.

We consider (i) three sites including one administrator with regards to the abstract model established in Section 5.4, (ii) one cooperative operation owned by a site choosen arbitrarily between the three sites, (iii) two access rules related to the cooperative operation such as the first is a positive rule and the second a restrictive rule. The first rule is considered to be positive because in our scope, we are interested in cooperative operation iniatially granted

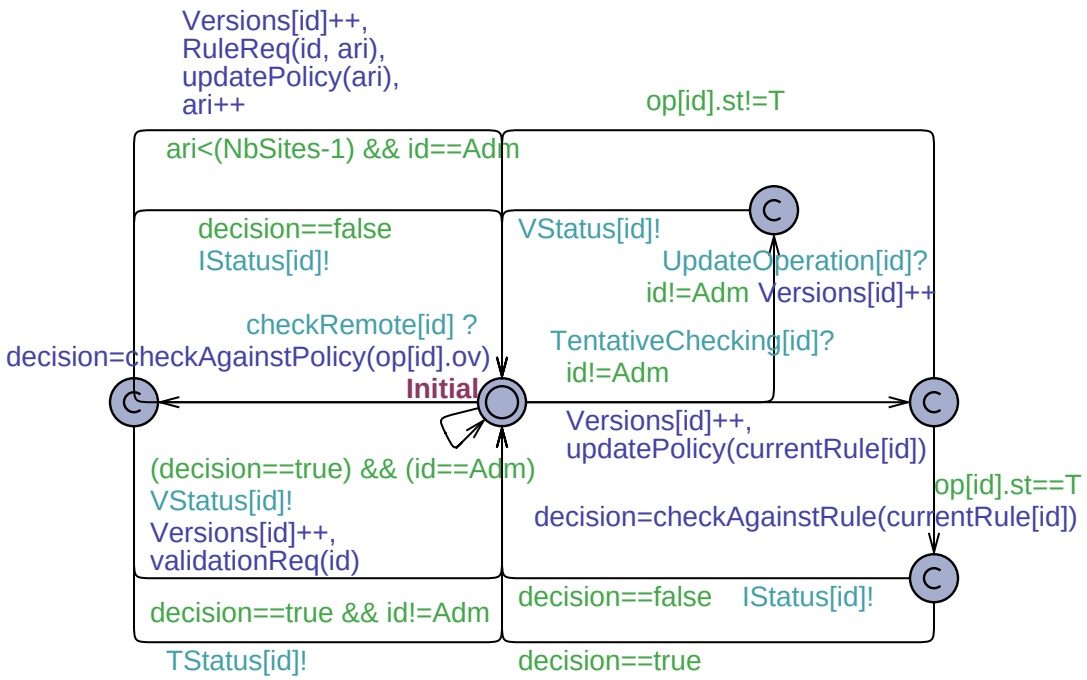

Figure 5.9 The model of ACP.

at generation time.

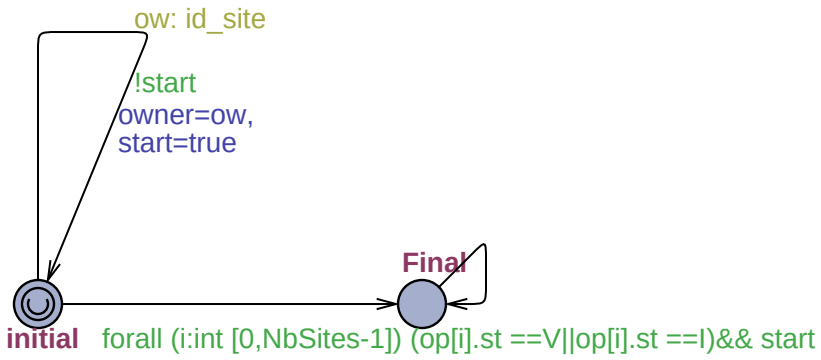

Figure 5.10 Automaton devoted to choose the owner of the cooperative operation.

To verify the consistency requirement, we are interested in checking the final status of every cooperative operation when the system is in a stable state (see Definition 5.10). The consistency requirement is specified by the CTL formula shown at staement 5.30. The formula means that if the cooperative operations and all administrative requests are processed at each site by ACP, the cooperative operations at remote sites have the same status as at the administrator site. After the verification, the Uppaal model-checker states : *Property is*

satisfied. Note that before verifying the main property, we check the absence of deadlocks ($A[]notDeadlock$). A state s of a model is in deadlock if and only if there is no action enabled in s nor in states reachable from s by time progression.

$$(Supervisor.Final \,\&\,\& forall(i : int[0,1]) forall(j : int[1,2]) \\ AdReq(i,j).Applied) - - > forall(i : int[0,1])(op[i].st == op[Adm].st) \quad (5.30)$$

5.6 Related Work

It is well known that analyze security properties of communication protocols in general, is a tricky task [Malik *et al.* (2013), Khan *et al.* (2005)]. Access control models and protocols [Jayaraman *et al.* (2013), Lee et Luedemann (2007), Tolone *et al.* (2005), Piromruen et Joshi (2005), Joshi *et al.* (2004), Sandhu *et al.* (1996)] do not escape this challenge, which is worsened in our particular distributed context, by the approach of replicating authorization policies [Samarati *et al.* (1996),Xin et Ray (2007)]. The main complexity relies on the infinite state of systems. Often, the analysis refers to formal methods, such as using automatic checking tools, [Randolph *et al.* (2013), Hu et Ahn (2008), Malik *et al.* (2013), Khan *et al.* (2005)] are examples. In [Randolph *et al.* (2013)], Alloy [2], a SAT-based bounded model-checker, is used to specify and verify if the data consistency of the DCE is preserved by the *flexible Access Control Protocol* [Imine *et al.* (2009)]. The verification has shown that ACP preserves the correctness. Although Alloy specification language, based on the first-order relational logic, is very appropriate to describe infinite systems, bounded model-checker techniques are known to be useful to find bugs but less appropriate to prove the absence of bugs. Thus, the study has covered just several restricted scopes, which does not allow to assert for large scopes. [Jayaraman *et al.* (2013)] presents the access control policy analysis tool Mohawk [3]. This tool uses techniques for abstraction refinement and bound estimation for bounded model checker. It is suitable to find bugs in access control policy based on Administrative Role-Based Access Control (ARBAC). These bugs are related to consistency or correctness of enterprise access-control policy systems. The Mohawk tool could not be used to verify the consistency preservation in DCE but the consistency of the access control policy. Khan et al. [Khan *et al.* (2005)] proposed a generic approach to verifying security protocols in the explicit state model-checker SPIN [Holzmann (2004)]. They used Promela modelling based on logic programming to deal with the potentially infinite *agents*. However,

2. MIT Sofware Design Group, Alloy : A language and Tool for Relational Models. Retrieved April 4, 2014 from `http://alloy.mit.edu/alloy/`

3. Mohawk: A tool for verifying access-control policies. Retrieved June 7, 2014 from `http://code.google.com/p/mohawk/`

the number of instances examined is restricted. In [Hu et Ahn (2008)], Hu H. et al. have used Alloy to verify the NIST/ANSI standard model for Role Based Access Control (RBAC), w.r.t the role deleting property. The Alloy analyzer allowed them to conclude that the functional definition of *DeleteRole* function proposed for hierarchical RBAC misses a step for removing inheritance. This work did not take into account all functional properties of RBAC model. In [Malik *et al.* (2013)], UPPAAL tool is used to verify the Inter Control Center Communications Protocol (ICCP), which is the protocol used among control centers for data exchange and control. In addition, to secure ICCP without modifying the protocol itself, the authors enable a communication checker, devoted to detect and create alerts on potential vulnerability exploitations in ICCP. The analyze of the communication checker by the Uppaal model-checker helped the authors to refine iteratively the design of the checking mechanisms.

5.7 Conclusion

In this paper, we have investigated the existence of a finite abstract model which is equivalent w.r.t. the consistency property, to an infinite collaborative system layered with ACP. The goal was to prove with the automatic checking using formal methods, that the system obtained by deploying the ACP over any consistent CP preserves the consistency property. The finite abstract model obtained consists of three cooperative sites including one administrator. For the verification, we have formally defined, using tool UPPAAL, a symbolic model, which take into account, the behavior and the consistency requirement of the system. The Uppaal model-checker concludes on the satisfaction of the consistency property and allow us to conclude that the flexible access control protocol preserves the consistency of a DCE on the top of which it runs. It is important to note that the symbolic model allows a significant gain in both space and time, as the state explosion problem is avoided. In the future, we plan to work on ACP with multpile administrators.

english

CHAPITRE 6

DISCUSSION GÉNÉRALE

Ce chapitre fait une syntèse de nos travaux et contributions. Il présente également une discussion sur la méthodologie suivie, ainsi qu'une analyse de nos résultats.

6.1 Synthèse des travaux

Les travaux présentés dans cette thèse ont débuté par une revue de littérature. Elle a été consacrée à la cohérence et au contrôle d'accès dans les systèmes d'édition collaborative. Une attention particulière a été portée aux spécificités d'une édition massivement répartie. Ainsi, le Chapitre 2 nous a permis d'explorer les différentes approches, modèles et algorithmes proposés pour assurer la cohérence dans les systèmes d'édition collaborative. Plusieurs modèles de contrôle d'accès conçus pour de pareils systèmes ont été également passés en revue. En étudiant ces divers travaux, nous avons tenté de délimiter la frontière des connaissances dans le domaine. Ainsi, les avancées réalisées avec les travaux existants ainsi que leurs limites ont été étudiées grâce à une revue détaillée de la littérature. Sur cette base, nos recherches ont donné lieu à des propositions consignées dans cinq articles dont trois font l'objet de chapitres de cette thèse.

Dans un premier volet, les recherches ont porté sur la synthèse d'un algorithme de transformation inclusive qui assure la convergence. La motivation vient du fait que la littérature a revélé la problématique de la cohérence. Nous avons commencé par explorer l'existence de mécanismes de réplication optimiste convergents. Les investigations ont été poussées plus loin pour découvrir les causes de divergence. L'identification des causes a inspiré la conception d'une nouvelle fonction de transformation inclusive. La preuve formelle a été apportée qu'elle garantit la convergence des documents. Ce volet a donné lieu à deux articles dont l'un est consigné dans la présente thèse, au Chapitre 3.

Le deuxième volet porte sur l'intégration d'un modèle de contrôle d'accès, aux systèmes d'édition collaborative massivement répartis. Nous considérons que le mécanisme de coordination (gestion de la concurrence et de la transformation des opérations) utilisé par le système, assure nativement la cohérence, à lui seul. D'un point de vue conceptuel, le protocole flexible de contrôle d'accès proposé pour les éditeurs collaboratifs répartis a été déployé en surcouche au protocole de synchronisation utilisé par le mécanisme de coordination. L'exercice a été de prouver que cette combinaison de protocoles ne compromet pas la cohérence. Cette véri-

fication est bornée et n'a permis que de garantir la préservation de la propriété jusqu'à une certaine borne. L'espace d'états du système n'a donc pas été totalement couvert. Ces travaux ont servi à l'écriture de deux articles dont l'un constitue le Chapitre 4 de la thèse.

Le troisème volet s'inscrit dans la poursuite des travaux du deuxième volet, en terme de preuve formelle. En effet, pour régler définitivement le problème de la vérification et aller au delà de la borne atteinte avec l'analyseur *Alloy*, la question s'est posée de savoir comment couvrir l'espace d'états de ce système infini. Il est connu que vérifier un système infini se heurte souvent à des problèmes d'explosion combinatoire et de défaut de ressources mémoire. Les réflexions ont alors été orientées dans le sens de la recherche d'un modèle fini sur lequel la propriété sera vérifiée. L'investigation d'une réduction qui préserve les propriétés du modèle de base, nous a conduit à proposer un modèle abstrait ayant un espace d'état fini. La preuve a ensuite été apportée formellement que ce dernier préserve effectivement les propriétés du système initial, au regard du traitement des opérations coopératives. Par la suite, il a été prouvé que le modèle fini préserve la cohérence. Ce volet a également servi à produire l'article présenté au Chapitre 5 de cette thèse.

6.2 Méthodologie

Dans le système considéré, la coordination est décentralisée. Chaque site a sa réplique de l'objet partagé. Un document textuel à structure linéaire est considéré comme l'objet partagé édité. Le système n'admet que deux opérations : insertion d'un caractère à une position donnée et suppression d'un caractère situé à une position donnée. Les opérations générées sur un site sont propagées aux autres sites pour être prises en compte. Nous ne considérons pas le mécanisme de dissémination à travers le réseau de communication exploité, des opérations générées localement sur un site. Nous ne faisons pas non plus une hypothèse sur le temps de propagation de ces opérations dans le réseau. Ainsi, toute opération envoyée à travers le réseau de communication est supposée toujours parvenir à destination, sans altération.

Pour mener à bien les investigations, la question de l'existence ou non de mécanismes de réplication optimiste convergents à été traduite en un problème de contrôle. Le formalisme des automates de jeu a été utilisé pour décrire le système. L'outil *Uppaal-Tiga* a permis de représenter le système et de vérifier la satisfaction de la propriété TP1. La vérification de la TP1 se traduit par la question : existe-t-il une stratégie gagnante pour la TP1 ? *Uppaal-Tiga* a conclu que la propriété est satisfaite, c'est-a-dire qu'il existe des ITs qui satisfont la TP1. Ces différents ITs ont été extraits grâce à l'outil *verifytga* de *Uppaal-Tiga*. Par la suite, ces ITs identifiés ont servi à formuler le problème de contrôle pour la TP2, grâce aux automates de jeu. Il peut être libellé comme suit : parmi les ITs qui satisfont la TP1,

existe-t-il au moins un qui satisfait la TP2 ? *Uppaal-Tiga* a conclu qu'aucun de ces ITs ne satisfait la TP2. Sur la base de ces résultats, la résolution du problème de contrôle a consisté à faire dans un premier temps les investigations pour déceler les cas de transformation qui posent problème. Ensuite l'analyse de ces cas a inspiré le choix d'une signature pour les opérations d'insertion. La nouvelle signature est obtenue en ajoutant un troisième paramètre à l'opération d'insertion. La sémantique associée au paramètre est le nombre de caractères supprimés avant la position indiquée. Nous avons établi et prouvé formellement, un lien entre ce nouveau paramètre et la position d'insertion. Un algorithme de détermination des valeurs du nouveau paramètre a été proposé puis prouvé exact. Il s'en suit la synthèse d'une nouvelle fonction de transformation inclusive. La preuve que l'IT proposé satisfait les propriétés TP1 et TP2 a été faite en utilisant la preuve symbolique. En effet, la technique de model-checking basée sur les matrices de bornes a été combinée avec le formalisme d'automate. L'outil *Uppaal* a servi de cadre pour la preuve. Une évaluation comparative portant sur la complexité a été faite par rapport à une solution proposée dans la littérature.

Pour ces travaux, le nombre de sites est un paramètre du système. Il est supposé constant. Ce choix vient du fait que la vérification de la TP1 nécessite deux opérations concurrentes générées par deux sites différents. Celle de la TP2 nécessite trois opérations concurrentes générées par trois sites différents. Le nombre de sites considéré comme paramètre du système est trois. Pour ce qui concerne les opérations concurrentes, chaque site a la possibilité de générer soit une opération d'insertion, soit une opération de suppression. La transformation d'une insertion ne peut pas donner une suppression. La transformation d'une suppression ne peut pas donner une insertion. Cependant, la transformation d'une opération concurrente par rapport à une autre, peut donnner lieu à une nouvelle opération dénommée $NoOp$. Elle signifie qu'aucune opération ne résulte de la transformation de l'opération considérée. Elle ne donne lieu à aucune exécution d'opération sur l'état courant. Il n'y a pas de changement d'état. Les propriétés TP1 et TP2 sont exprimées en langage CTL, sous la forme de propriétés de sûreté.

Pour sécuriser l'édition, l'option a été faite de combiner le mécanisme de coordination supposé nativement cohérent, avec un protocole de contrôle d'accès. La propriété de cohérence doit être préservée. Pour ce faire il faut choisir le protocole parmi ceux qui sont proposés dans la littérature, ou en concevoir un. Il faut par la suite prouver que le protocole convient. La conception d'un nouveau protocole n'est nécessaire que si tous les protocoles proposés dans la littérature sont prouvés inutilisables dans le contexte complexe des systèmes d'édition collaborative massivement répartis. Ou encore, si une nouvelle méthode intéressante pour des raisons de performance ou de facilité d'implémentation. Nous avons opté pour un protocole que nous jugeons adéquat pour atteindre nos objectifs. En effet, le protocole flexible de contrôle d'accès

proposé pour les éditeurs collaboratifs répartis permet de déployer une sécurité optimiste. Il autorise une rétroaction dans l'application des règles d'accès. De surcroît, son déploiement se fait par réplication au même titre que les opérations coopératives qui permettent d'éditer le document à sécuriser. à ces fonctionnalités s'ajoute la gestion de la préséance causale entre les opérations administratives et les opérations coopératives. Ce protocole a alors été déployé au dessus du protocole de synchronisation du mécanisme de coordination. Il faille dès lors, prouver que cette combinaison de protocoles ne compromet pas la cohérence. Pour ce faire, le système doit une fois encore être modélisé pour être formellement vérifié. La modélisation a été faite en utilisant l'outil *Alloy*. La propriété attendue a également été spécifiée puis vérifiée par l'analyseur *Alloy*, en association avec un solveur SAT. Cette vérification est faite pour un nombre maximal de treize sites. Notons que cette borne est importante du point de vue de la combinatoire engendrée car elle nécessite des ressources de calcul et de mémoire importantes. Cependant, l'espace d'états du système n'a pas été totalement couvert. La vérification faite avec *Alloy* permet de conclure une absence de divergence sur le domaine délimité par la borne, mais pas sur l'ensemble de l'espace d'états. L'analyseur *Alloy* est bien adapté pour la spécification des systèmes infinis. Il permet de passer de la spécification du système, qui est basée sur la logique de premier ordre et la théorie des ensembles, à une formulation sous la forme normale conjonctive. Une fois traduit sous cette forme, le système peut alors être résolu avec n'importe quel solveur SAT, mais seulement dans une approche de *model-checking* borné.

Pour affronter le problème de la vérification de la préservation de la cohérence, nous avons cherché un système fini équivalent. Nous avons dans un premier temps défini formellement toutes les proporiétés qui découlent de la description du protocole. Cette formalisation a débouché sur le recensement des différents scénarii d'exécution d'opérations d'édition de document. La trace d'exécution de ces scénarii, rapporté aux statuts finaux des opérations a servi à avoir un modèle abstrait du processus d'exécution. Ce dernier a inspiré la proposition du modèle abstrait représentant le système en étude. Enfin, pour vérifier si le nouveau modèle abstrait préserve la cohérence, le formalisme d'automate a encore été utilisé avec *Uppaal*.

6.3 Analyse des résultats

Pour conforter les travaux relatifs à la proposition d'un nouvel IT, nous avons procédé à une analyse comparative. Notre IT a été comparée en terme de complexité, à la solution TTF. Les comparaisons ont porté sur les opérations d'insertion et de suppression, au moment de leur génération. Les résultats sont satisfaisants pour les deux types d'opérations. Dans le cas d'une suppression la complexité est constante pour notre IT alors qu'elle est linéaire pour

la solution TTF. Pour une insertion, les complexités sont linéaires dans les deux cas.

Après avoir discuté de nos approches et résultats, nous concluons cette thèse.

CHAPITRE 7

CONCLUSION

Un système d'édition collaborative est un logiciel qui permet à un groupe d'utilisateurs d'éditer conjointement des objets partagés. Lorsque deux utilisateurs interagissent dans un contexte réparti, il n'est pas toujours possible d'établir un ordre (au sens de Lamport [Lamport (1978)]) entre leurs opérations respectives. Il résulte de cette situation de concurrence entre opérations, des défis majeurs. Il s'agit de l'obtention de documents cohérents et la gestion des accès. La grande taille du système et sa dynamique constituent des contraintes supplémentaires au contexte. Dans cette thèse, nous avons essayé d'aborder ces différents défis en les analysant, en proposant des solutions et finalement en prouvant que ces solutions sont satisfaisantes. Dans ce chapitre, nous présentons le sommaire des contributions apportées, suivi d'une critique de nos travaux à travers leurs limitations. Pour finir, nous énonçons les avenues sur lesquelles seront axés nos futurs travaux.

7.1 Sommaire des contributions de la thèse

Nos recherches ont donné lieu à plusieurs contributions entrant dans le cadre d'une édition collaborative cohérente et sécurisée. Elles nous ont permis d'atteindre nos objectifs et se résument comme suit.

- Existence de mécanismes de réplication optimiste convergents. L'approche de gestion de cohérence considéré est la transformée opérationnelle. Les objets manipulés sont les documents textuels linéaires. Nous avons prouvé que la position dans laquelle un caractère doit être supprimé est suffisante comme paramètre pour une opération de suppression. La preuve a été aussi donnée que le caractère et la position correspondante ne sont pas suffisantes comme paramètres pour une opération d'insertion. Grâce à nos travaux, nous avons conforté la possibilité d'avoir une édition cohérente. En effet, elle est possible, à condition d'isoler certaines transformations ou de trouver une bonne façon de les faire.
- Conception d'un algorithme de transformation inclusive devant garantir la convergence dans un contexte réparti de réplication optimiste. Cet objectif est atteint grâce à (i) la définition d'une nouvelle signature pour l'insertion, (ii) la précision d'un algorithme pour déterminer les valeurs du nouveau paramètre impliqué et (iii) la synthèse d'un nouvel IT. L'IT permet de réaliser la suppression avec une complexité constante. L'opé-

ration d'insertion est réalisée avec une complexité linéaire. La preuve a été apportée que l'IT satisfait les propriétés TP1 et TP2. En conséquence, l'IT proposé garantit la convergence. Deux avancées notables sont ainsi conjointement réalisées : obtention d'un IT convergent à moindre coût.

– Intégration du contrôle d'accès et préservation de la cohérence des systèmes d'édition collaborative. L'objectif de sécurisation d'une édition collaborative par le contrôle d'accès est réalisé par le déploiement d'un modèle flexible qui procède par réplication avec effet rétroactif. Les critères que sont la spécification de haut niveau des droits d'accès, la généricité et la flexibilité du modèle, l'aspect dynamique du modèle, le maintien des indicateurs de performance à des seuils acceptables. Ils justifient la préférence pour un contrôle d'accès optimiste qui maintient une relation de causalité entre les opérations administratives mais aussi entre les opérations administratives et les opérations coopératives. Des efforts de spécification du système ont été faits dans un premier temps dans une approche de *model-checking* bornée. Ceci a permis d'avoir une première idée de la préservation de la propriété de cohérence par le protocole, pour des instances non massives. Par la suite, une avancée a consisté à trouver un modèle fini équivalent au système en étude, au regard de la cohérence. La démarche formelle ébauche un cadre de réduction du système applicable pour de futures études sur un tel système ou un système similaire. Le modèle abstrait qui en découle a servi à prouver la préservation de la propriété de cohérence. Le protocole de contrôle d'accès est ainsi formellement prouvé fiable pour les systèmes d'édition collaborative répartis, même avec une contrainte impliquant un grand nombre d'utilisateurs.

7.2 Limitations des travaux

Nos travaux présentent quelques limitations. Elles concernent essentiellement la portée des travaux et des aspects de flexibilité.

– Dans le chapitre 3, la première limitation est relative au type des objets considérés. En effet, nous n'avons manipulé que des objets textuels ayant une structure linéaire. L'élargissement de la portée en prenant en compte d'autres d'objets bonifierait les travaux. Par exemple, vu l'accroissement de l'utilisation des documents au format XML, l'étude des documents semis-structurés permettrait de couvrir cette nature d'objets. La prise en compte de plusieurs natures d'objets amène à considérer plusieurs sémantiques, ce qui ajoute une complexité non négligeable. Les réflexions pourraient être orientées dans le sens d'une généricité ou dans celui de la paramétrisation de la cohérence en lui associant un seuil ou un degré. Par exemple, l'édition d'une image pourrait réquérir un

paramètre de cohérence de valeur plus faible que l'édition de texte. Le paramètre sera donc fixé, en se basant sur la sensibilité du contexte d'application.

- La deuxième limitation de ce travail est qu'il ne couvre que des opérations d'insertion et de suppression de caractères. Il est vrai que le niveau de granularité le plus bas pour un texte est le caractère. Il est également vrai que certaines opérations peuvent être composées à partir des deux opérations que sont l'insertion et la suppression. Par exemple, un remplacement de caractère peut être vu comme la composition d'une suppression par une insertion de caractère. Cependant une opération d'annulation nécessite une plus grande précaution. Elle ne doit pas simplement être vue comme une opération inverse. Elle appellerait des transformations que certains auteurs ont dénommé « transformations inverses ». La prise en compte de cette nature d'opération dans notre travail augmenterait son champ d'action.
- La limitation commune aux chapitres 4 et 5 relève du fonctionnemnt du système étudié. Le fait de ne pas considérer plusieurs administrateurs limite l'architecture du système et celle du protocole. La présence de plusieurs adminsitrateurs engendrerait des opérations administratives concurrentes. Dans ce cas, on aurait un «système de pleine concurrence». Il serait question de traiter de la concurrence (i) entre opérations coopératives et (ii) entre opérations coopératives et opérations administratives mais aussi (iii) entre opérations administratives. La complexité qui découle de la concurrence entre opérations administratives s'expliquerait par la possibilité d'avoir des règles d'accès contradictoires. Par exemple, supposons que le système contient deux administrateurs. L'un d'eux émet une règle d'accès qui autorise une opération coopérative pour un utilisateur donné. Si concurremment, le second émet une règle d'accès qui interdit la même opération à l'utilisateur, un conflit s'en suivra. Une piste pour mieux aborder cette limitation serait de redéfinir une nouvelle sémantique pour les opérations adminstratives. On pourra alors s'inspirer des travaux du chapitre 3 pour étudier dans un premier temps la cohérence des politiques locales de sécurité. Ensuite, les présents travaux des chapitres 4 et 5 seront exploités pour parachever l'étude.

7.3 Indication des travaux futurs

Les travaux futurs seront en partie consacrés à surmonter les limitations de cette thèse :

- à court terme, certaines natures d'objets, pourraient être isolément étudiées au regard de la problématique de cohérence. L'approche OT serait considérée avec une sémantique adaptée à chaque nature d'objets. Par la suite, le contrôle d'accès serait étudié dans une perspective multi-administrateurs.

- à moyen terme, la cohérence des politiques de contrôle d'accès ferait l'objet de nos recherches.
- à long terme, dans une perspective de généralisation, les travaux seront étendus pour traiter des objets génériques. Puis, la cohérence globale du système retiendrait notre attention. Elle couvrirait des objets de natures génériques, un contrôle d'accès multi-administrateurs basé sur des politiques cohérentes.

Au delà des limitations de la thèse, en mettant à profit la réalisation des avenues ci-dessus énumérées, nos futurs travaux seront consacrés aux pistes ci-après.

- élaboration d'un cadre général de réduction des systèmes d'édition collaborative, quelle que soit la sémantique de l'objet considéré. Ce cadre pourra inspirer d'autres recherches de réduction de systèmes infinis, autre que les systèmes collaboratifs.
- L'étude et l'amélioration des performances du protocole de contrôle d'accès. Il en sera ainsi pour l'algorithme de transformation inclusive proposée. Il est actuellement de complexité linéaire pour l'opération d'insertion. Nous n'excluons pas la possibilité de rechercher une nouvelle fonction d'IT dont la complexité serait constante ou tout au moins sub-linéaire, pour l'insertion.

RÉFÉRENCES

[Abdunabi *et al.* (2013)] ABDUNABI, R., RAY, I. et FRANCE, R. (2013). Specification and analysis of access control policies for mobile applications. *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*. ACM, New York, NY, USA, SACMAT '13, 173–184.

[Ahmed et Shirmohammadi (2006)] AHMED, D. T. et SHIRMOHAMMADI, S. (2006). A hybrid p2p protocol for real-time collaboration. *Enabling Technologies : Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*. IEEE, 73–78.

[Akbarinia *et al.* (2007)] AKBARINIA, R., PACITTI, E. et VALDURIEZ, P. (2007). Data currency in replicated dhts. *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, SIGMOD '07, 211–222.

[Alotaiby et Chen (2004)] ALOTAIBY, F. T. et CHEN, J. (2004). A model for team-based access control (tmac 2004). *Information Technology : Coding and Computing, International Conference on*. IEEE Computer Society, vol. 1, 450–450.

[Alur et Dill (1994)] ALUR, R. et DILL, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126, 183–235.

[Bakhshi et Gurov (2007)] BAKHSHI, R. et GUROV, D. (2007). Verification of peer-to-peer algorithms : A case study. *Electronic Notes in Theoretical Computer Science*, 181, 35 – 47. Combined Proceedings of the Second International Workshop on Coordination and Organization (CoOrg 2006) and the Second International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2006).

[Bernstein et Goodman (1981)] BERNSTEIN, P. A. et GOODMAN, N. (1981). Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR)*, 13, 185–221.

[Bernstein et Goodman (1983)] BERNSTEIN, P. A. et GOODMAN, N. (1983). Multiversion concurrency control : theory and algorithms. *ACM Trans. Database Syst.*, 8, 465–483.

[Bernstein *et al.* (1987)] BERNSTEIN, P. A., HADZILACOS, V. et GOODMAN, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Boucheneb et Imine (2009)] BOUCHENEB, H. et IMINE, A. (2009). On model-checking optimistic replication algorithms. *FMOODS/FORTE*. 73–89.

[Boucheneb *et al.* (2010)] BOUCHENEB, H., IMINE, A. et NAJEM, M. (2010). Symbolic model-checking of optimistic replication algorithms. *IFM*. 89–104.

[Bullock et Benford (1999)] BULLOCK, A. et BENFORD, S. (1999). An access control framework for multi-user collaborative environments. *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*. ACM, New York, NY, USA, GROUP '99, 140–149.

[Cart et Ferrié (2007)] CART, M. et FERRIÉ, J. (2007). Asynchronous reconciliation based on operational transformation for p2p collaborative environments. *Collaborative Computing : Networking, Applications and Worksharing, 2007. CollaborateCom 2007. International Conference on*. 127–138.

[Cassez *et al.* (2005)] CASSEZ, F., DAVID, A., FLEURY, E., LARSEN, K. G. et LIMEI, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. *CONCUR-LNCS*, 3653, 60–80.

[Cimatti *et al.* (2000)] CIMATTI, A., CLARKE, E., GIUNCHIGLIA, F. et ROVERI, M. (2000). NUSMV : A New Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer*, 2, 410–425.

[Covington *et al.* (2001)] COVINGTON, M. J., LONG, W., SRINIVASAN, S., DEV, A. K., AHAMAD, M. et ABOWD, G. D. (2001). Securing context-aware applications using environment roles. *Proceedings of the sixth ACM symposium on Access control models and technologies*. ACM, 10–20.

[Ellis et Gibbs (1989)] ELLIS, C. A. et GIBBS, S. J. (1989). Concurrency control in groupware systems. *SIGMOD Conference*. vol. 18, 399–407.

[Frappier *et al.* (2010)] FRAPPIER, M., FRAIKIN, B., CHOSSART, R., CHANE-YACK-FA, R. et OUENZAR, M. (2010). Comparison of Model Checking Tools for Information Systems. *Proceedings of the 12th international conference on Formal engineering methods and software engineering*. Springer-Verlag, Berlin, Heidelberg, ICFEM'10, 581–596.

[Georgiadis *et al.* (2001)] GEORGIADIS, C. K., MAVRIDIS, I., PANGALOS, G. et THOMAS, R. K. (2001). Flexible team-based access control using contexts. *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*. ACM, New York, NY, USA, SACMAT '01, 21–27.

[Holzmann (2004)] HOLZMANN, G. J. (2004). *The Spin Model Checker : Primer and Reference Manual*. Addison-Wesley.

[Hu et Ahn (2008)] HU, H. et AHN, G. (2008). Enabling Verification and Conformance Testing for Access Control Model. *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, New York, NY, USA, SACMAT '08, 195–204.

[Imine (2006)] IMINE, A. (2006). *Conception formelle d'algorithmes de réplication optimiste. Vers l'édition Collaborative dans les réseaux Pair-à-Pair*. Phd thesis, University of Henri Poincaré, Nancy, France.

[Imine (2008)] IMINE, A. (2008). Decentralized concurrency control for real-time collaborative editors. A. O. Djamal Benslimane, éditeur, *8th international conference on New technologies in distributed systems - NOTERE'2008*. ACM New York, NY, USA, Lyon, France, 313–321. ISBN : 978-1-59593-937-1.

[Imine (2009)] IMINE, A. (2009). Coordination model for real-time collaborative editors. *COORDINATION*. 225–246.

[Imine (2010)] IMINE, A. (2010). On Coordinating Collaborative Objects. M. R. Mousavi et G. Salaün, éditeurs, *9th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA)*. Open Publishing Association, Paris, France, vol. 30 de *Electronic Proceedings in Theoretical Computer Science*, 78–92.

[Imine *et al.* (2009)] IMINE, A., CHERIF, A. et RUSINOWITCH, M. (2009). A Flexible Access Control Model for Distributed Collaborative Editors. W. Jonker et M. Petkovic, éditeurs, *Secure Data Management*, Springer Berlin / Heidelberg, vol. 5776 de *Lecture Notes in Computer Science*. 89–106.

[Imine *et al.* (2003)] IMINE, A., MOLLI, P., OSTER, G. et RUSINOWITCH, M. (2003). Proving correctness of transformation functions in real-time groupware. *Proceedings of the Eighth Conference on European Conference on Computer Supported Cooperative Work*. Kluwer Academic Publishers, Norwell, MA, USA, ECSCW'03, 277–293.

[Imine *et al.* (2006)] IMINE, A., RUSINOWITCH, M., OSTER, G. et MOLLI, P. (2006). Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, 351, 167–183.

[Jackson (2006)] JACKSON, D. (2006). *Software Abstractions : Logic, Language, and Analysis*. The MIT Press.

[Jayaraman *et al.* (2013)] JAYARAMAN, K., TRIPUNITARA, M., GANESH, V., RINARD, M. et CHAPIN, S. (2013). Mohawk : Abstraction-refinement and bound-estimation for verifying access control policies. *ACM Trans. Inf. Syst. Secur.*, 15, 18 :1–18 :28.

[Joshi *et al.* (2004)] JOSHI, J. B. D., BHATTI, R., BERTINO, E. et GHAFOOR, A. (2004). Access-Control Language for Multidomain Environments. *IEEE Internet Computing*, 8, 40–50.

[Kawagoe et Kasai (2011)] KAWAGOE, K. et KASAI, K. (2011). Situation, team and role based access control. *Journal of Computer Science*, 7.

[Khan *et al.* (2005)] KHAN, A., MUKUND, M. et SURESH, S. (2005). Generic verification of security protocols. P. Godefroid, éditeur, *Model Checking Software*, Springer Berlin Heidelberg, vol. 3639 de *Lecture Notes in Computer Science*. 221–235.

[Kumawat et Khunteta (2010)] KUMAWAT, S. et KHUNTETA, A. (2010). A survey on operational transformation algorithms : Challenges, issues and achievements. *International Journal of Computer Applications*, 3, 30–38.

[Lamport (1978)] LAMPORT, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21, 558–565.

[Larsen *et al.* (1997)] LARSEN, K., PETTERSSON, P. et YI, W. (1997). Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1, 134–152.

[Le Berre et Parrain (2010)] LE BERRE, D. et PARRAIN, A. (2010). The Sat4j Library, Release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7, 59–64. System description.

[Lee et Luedemann (2007)] LEE, H. K. et LUEDEMANN, H. (2007). lightweight Decentralized Authorization Model for Inter-Domain Collaborations. *Proceedings of the 2007 ACM workshop on Secure web services*. ACM, New York, NY, USA, SWS '07, 83–89.

[Li et Li (2004)] LI, D. et LI, R. (2004). Preserving operation effects relation in group editors. *ACM conference on Computer supported cooperative work*. ACM, New York, NY, USA, CSCW '04, 457–466.

[Li et Li (2008a)] LI, D. et LI, R. (2008a). An approach to ensuring consistency in peer-to-peer real-time group editors. *Comput. Supported Coop. Work*, 17, 553–611.

[Li et Li (2008b)] LI, D. et LI, R. (2008b). An operational transformation algorithm and performance evaluation. *Computer Supported Cooperative Work (CSCW)*, 17, 469–508.

[Li et Li (2010)] LI, D. et LI, R. (2010). An admissibility-based operational transformation framework for collaborative editing systems. *Computer Supported Cooperative Work*, 19, 1–43.

[Malik *et al.* (2013)] MALIK, S., BERTHIER, R., BOBBA, R. B., CAMPBELL, R. H. et SANDERS, W. H. (2013). Formal design of communication checkers for iccp using uppaal. *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*. 486–491.

[Martin *et al.* (2010)] MARTIN, S., URSO, P. et WEISS, S. (2010). Scalable xml collaborative editing with undo. R. Meersman, T. Dillon et P. Herrero, éditeurs, *On the Move to Meaningful Internet Systems : OTM 2010*, Springer Berlin / Heidelberg, vol. 6426 de *Lecture Notes in Computer Science*. 507–514. 10.1007/978-3-642-16934-2_37.

[Molli *et al.* (2003a)] MOLLI, P., OSTER, G., SKAF-MOLLI, H. et IMINE, A. (2003a). Using the transformational approach to build a safe and generic data synchronizer. *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*. ACM Press, 212–220.

[Molli *et al.* (2003b)] MOLLI, P., OSTER, G., SKAF-MOLLI, H. et IMINE, A. (2003b). Using the transformational approach to build a safe and generic data synchronizer. *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*. ACM, New York, NY, USA, GROUP '03, 212–220.

[Oster *et al.* (2006a)] OSTER, G., MOLLI, P., URSO, P. et IMINE, A. (2006a). Tombstone transformation functions for ensuring consistency in collaborative editing systems. *International Conference on Collaborative Computing : Networking, Applications and Worksharing (CollaborateCom)*. 1 –10.

[Oster *et al.* (2006b)] OSTER, G., URSO, P., MOLLI, P. et IMINE, A. (2006b). Data consistency for p2p collaborative editing. *20th Conference on Computer Supported Cooperative Work*. ACM, New York, NY, USA, 259–268.

[Pai *et al.* (2011)] PAI, S., SHARMA, Y., KUMAR, S., PAI, R. et SINGH, S. (2011). Formal Verification of OAuth 2.0 Using Alloy Framework. *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*. 655–659.

[Piromruen et Joshi (2005)] PIROMRUEN, S. et JOSHI, J. B. D. (2005). An RBAC Framework for Time Constrained Secure Interoperation in Multi-Domain Environments. *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*. 36–45.

[Povey (2000)] POVEY, D. (2000). Optimistic security : a new access control paradigm. *Proceedings of the 1999 workshop on New security paradigms*. ACM, New York, NY, USA, NSPW '99, 40–45.

[Preguiça *et al.* (2009)] PREGUIÇA, N., MARQUES, J., SHAPIRO, M. et LETIA, M. (2009). A commutative replicated data type for cooperative editing. *29th IEEE International Conference on Distributed Computing Systems*. 395–403.

[Randolph *et al.* (2013)] RANDOLPH, A., IMINE, A., BOUCHENEB, H. et QUINTERO, A. (2013). Specification and verification using alloy of optimistic access control for distributed collaborative editors. C. Pecheur et M. Dierkes, éditeurs, *Formal Methods for Industrial Critical Systems*, Springer Berlin Heidelberg, vol. 8187 de *Lecture Notes in Computer Science*. 184–198.

[Ressel *et al.* (1996)] RESSEL, M., NITSCHE-RUHLAND, D. et GUNZENHAUSER, R. (1996). An integrating, transformation-oriented approach to concurrency control and undo in group editors. *ACM CSCW'96*. Boston, USA, 288–297.

[Saito et Shapiro (2005)] SAITO, Y. et SHAPIRO, M. (2005). Optimistic replication. *ACM Comput. Surv.*, 37, 42–81.

[Samarati *et al.* (1996)] SAMARATI, P., AMMANN, P. et JAJODIA, S. (1996). Maintaining Replicated Authorizations in Distributed Database Systems. *Data & knowledge engineering*, 18, 55–84.

[Samuel *et al.* (2007)] SAMUEL, A., GHAFOOR, A. et BERTINO, E. (2007). A framework for specification and verification of generalized spatio-temporal role based access control model. Rapport technique, Purdue University.

[Sandhu *et al.* (1996)] SANDHU, R., COYNE, E., FEINSTEIN, H. et YOUMAN, C. (1996). Role-Based Access Control Models. *Computer*, 29, 38–47.

[Sandhu et Samarati (1994)] SANDHU, R. et SAMARATI, P. (1994). Access control : principle and practice. *Communications Magazine, IEEE*, 32, 40–48.

[Schaeffer-Filho *et al.* (2009)] SCHAEFFER-FILHO, A., LUPU, E., SLOMAN, M. et EISENBACH, S. (2009). Verification of Policy-Based Self-Managed Cell Interactions Using Alloy. *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*. 37–40.

[Shapiro et Preguiça (2007)] SHAPIRO, M. et PREGUIÇA, N. (2007). Designing a commutative replicated data type. Rapport technique Report RR-6320, Institut National de la Recherche en Informatique et Automatique (INRIA).

[Suleiman *et al.* (1997)] SULEIMAN, M., CART, M. et FERRIÉ, J. (1997). Serialization of concurrent operations in a distributed collaborative environment. *ACM GROUP'97*. 435–445.

[Suleiman *et al.* (1998)] SULEIMAN, M., CART, M. et FERRIÉ, J. (1998). Concurrent operations in a distributed and mobile collaborative environment. *IEEE ICDE'98*. 36–45.

[Sun et Ellis (1998)] SUN, C. et ELLIS, C. (1998). Operational transformation in real-time group editors : issues, algorithms, and achievements. *ACM CSCW'98*. 59–68.

[Sun *et al.* (1998)] SUN, C., JIA, X., ZHANG, Y., YANG, Y. et CHEN, D. (1998). Achieving convergence, causality-preservation and intention-preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5, 63–108.

[Sun *et al.* (2006)] SUN, C., XIA, S., SUN, D., CHEN, D., SHEN, H. et CAI, W. (2006). Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13, 531–582.

[Sun et Sun (2009)] SUN, D. et SUN, C. (2009). Context-based operational transformation for distributed collaborative editing systems. *IEEE Trans. on Parallel and Distributed Systems*, 20, 1454–1470.

[Thomas et Sandhu (1994)] THOMAS, R. et SANDHU, R. (1994). Conceptual foundations for a model of task-based authorizations. *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings.* 66–79.

[Thomas (1997)] THOMAS, R. K. (1997). Team-based access control (tmac) : A primitive for applying role-based access controls in collaborative environments. *Proceedings of the Second ACM Workshop on Role-based Access Control.* ACM, New York, NY, USA, RBAC '97, 13–19.

[Thomas et Sandhu (1997)] THOMAS, R. K. et SANDHU, R. S. (1997). Task-based authorization controls (tbac) : A family of models for active and enterprise-oriented authorization management. *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI : Status and Prospects.* Citeseer, vol. 113, 166–181.

[Tlili *et al.* (2010)] TLILI, M., AKBARINIA, R., PACITTI, E. et VALDURIEZ, P. (2010). Scalable p2p reconciliation infrastructure for collaborative text editing. *Proceedings of the 2010 Second International Conference on Advances in Databases, Knowledge, and Data Applications.* IEEE Computer Society, Washington, DC, USA, DBKDA '10, 155–164.

[Tlili *et al.* (2008)] TLILI, M., DEDZOÉ, W. K., PACITTI, E., VALDURIEZ, P., AKBARINIA, R., DUBOST, L., DUMITRIU, S., LAURIÈRE, S., CANALS, G., MOLLI, P. et MAIRE, J. (2008). P2P logging and timestamping for xwiki. *Proceedings of the 8th International Conference on New Technologies in Distributed Systems.* ACM, New York, NY, USA, NOTERE '08, 25 :1–25 :4.

[Toahchoodee *et al.* (2009)] TOAHCHOODEE, M., RAY, I., ANASTASAKIS, K., GEORG, G. et BORDBAR, B. (2009). Ensuring Spatio-temporal Access Control for Real-world Applications. *Proceedings of the 14th ACM symposium on Access control models and technologies.* ACM, New York, NY, USA, SACMAT '09, 13–22.

[Tolone *et al.* (2005)] TOLONE, W., AHN, G.-J., PAI, T. et HONG, S.-P. (2005). Access Control in Collaborative Systems. *ACM Comput. Surv.*, 37, 29–41.

[Torlak et Dennis (2006)] TORLAK, E. et DENNIS, G. (2006). Kodkod for Alloy users.

[Vidot *et al.* (2000)] VIDOT, N., CART, M., FERRIÉ, J. et SULEIMAN, M. (2000). Copies Convergence in a Distributed Real-time Collaborative Environment. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*. ACM, New York, NY, USA, CSCW '00, 171–180.

[Weiss *et al.* (2009)] WEISS, S., URSO, P. et MOLLI, P. (2009). Logoot : A scalable optimistic replication algorithm for collaborative editing on p2p networks. *29th IEEE International Conference on Distributed Computing Systems*. 404 –412.

[Weiss *et al.* (2010)] WEISS, S., URSO, P. et MOLLI, P. (2010). Logoot-undo : Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21, 1162 –1174.

[Woodcock et Davies (1996)] WOODCOCK, J. et DAVIES, J. (1996). *Using Z : Specification, Refinement, and Proof*. Prentice Hall.

[Wu et Pui (2009)] WU, Q. et PUI, C. (2009). Consistency in real-time collaborative editing systems based on partial persistent sequences. Rapport technique, Georgia Institute of Technology.

[Wu *et al.* (2010)] WU, Q., PUI, C. et FERREIRA, J. A. E. F. (2010). A partial persistent data structure to support consistency in real-time collaborative editing. *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. 776–779.

[Xin et Ray (2007)] XIN, T. et RAY, I. (2007). A Lattice-Based Approach for Updating Access Control Policies in Real-time. *Inf. Syst.*, 32, 755–772.

ANNEXE A : DÉFINITIONS COMPLÉMENTAIRES

Définition .1 (Relation réflexive) . *Une relation $\mathcal{R}$ définie dans un ensemble E est réflexive si et seulement si, pour tout élément e appartenant à E, le couple (e, e) appartient à $\mathcal{R}$.*

$$\forall e \in E,\ e\mathcal{R}e \text{ est vrai} \tag{1}$$

Définition .2 (Relation antisymétrique) .. *Une relation $\mathcal{R}$ définie dans un ensemble E est antisymétrique si et seulement si, pour tout élément e_1, e_2 appartenant à $\mathcal{R}$, lorsque les couples (e_1, e_2) et (e_2, e_1) appartiennent à $\mathcal{R}$, alors e_1 et e_2 sont égaux.*

$$\forall e_1, e_2 \in \mathcal{R}, (e_1\mathcal{R}e_2) \wedge (e_2\mathcal{R}e_1) \Rightarrow (e_1 = e_3) \tag{2}$$

Définition .3 (Relation transitive) . *Une relation $\mathcal{R}$ définie dans un ensemble E est transitive si et seulement si, pour tout élément e_1, e_2, e_3 appartenant à $\mathcal{R}$, lorsque les couples (e_1, e_2) et (e_2, e_3) appartiennent à $\mathcal{R}$, alors le couple (e_1, e_3) appartient à $\mathcal{R}$.*

$$\forall e_1, e_2, e_3 \in \mathcal{R}, (e_1\mathcal{R}e_2) \wedge (e_2\mathcal{R}e_3) \Rightarrow (e_1\mathcal{R}e_3) \tag{3}$$

Définition .4 (Relation d'ordre) . *Une relation $\mathcal{R}$ dans un ensemble E est appelé une relation d'ordre partiel ou de préordre si et seulement si elle est **réflexive**, **antisymétrique** et **transitive**.*
Un ensemble E combiné à une relation d'ordre partiel $\mathcal{R}$ est appelé un ensemble partiellement ordonné et est noté $(S, \mathcal{R})$.

Définition .5 (éléments comparables) . *Les éléments e_1 et e_2 d'un ensemble partiellement ordonné $(S, \mathcal{R})$ sont dits comparables si et seulement si, l'un des couples (e_1, e_2) ou (e_2, e_3) appartient à $\mathcal{R}$.*

Définition .6 (Ordre total) . *Si $(S, \mathcal{R})$ est un ensemble partiellement ordonné et que tous les éléments de S sont deux à deux comparables, alors S est appelé ensemble totalement ordonné et $\mathcal{R}$ est un ordre total.*

Définition .7 (Histoire légale) . *Une histoire (séquence d'opérations) h est dite légale sur un état st si la précondition de chaque opération de h est satisfaite.*

Définition .8 (équivalence des histoires) . *Deux histoires h_1 et h_2 sont équivalentes pour tout état st (noté $h_1 \equiv_{st} h_2$), si et seulement si, h_1 et h_2 sont légales sur st, de même longueur et leurs exécutions sur l'état st conduisent à un même état.*

$$\begin{aligned} &(i)\ h_1 \text{ et } h_2 \text{ sont légales sur } st \\ &(ii) \qquad |h_1| = |h_2| \\ &(iii) \quad Do^*(h_1, st) = Do^*(h_2, st) \end{aligned} \tag{4}$$

avec Do^* la fonction qui retourne l'état obtenu en exécutant une séquence ou une histoire sur un état donné.

Définition .9 (Extension de IT) . *Nous définissons une extension à la fonction de transformation IT comme suit :*

$$\begin{aligned} IT^* : \mathcal{H} \times \mathcal{H} &\rightarrow \mathcal{H} \\ IT^*(h, []) &= h && (5) \\ IT^*([], h)] &= [] && (6) \\ IT^*(h_1, [h_2; h_3]) &= IT^*(IT^*(h_1, h_2), h_3) && (7) \\ IT^*([h_1; h_2], h_3) &= [IT^*(h_1, h_3); IT^*(h_2, IT^*(h_3, h_1))] && (8) \\ IT^*([o_1], [o_2]) &= [IT(o_1, o_2)] && (9) \end{aligned}$$

avec [] l'histoire vide, h, h_1, h_2, h_3 toutes histoires légales et o_1 et o_2 toutes opérations.
IT^* sert à transformer une histoire légale par rapport à une autre histoire légale. Ainsi, $IT^*(h_1, h_2)$ est l'histoire légale obtenue en transformant l'histoire légale h_1 par rapport à l'histoire légale h_2.
L'expression (5) signifie quela transformation d'une histoire légale h par rapport à une histoire légale vide produit h. L'expression (6) signifie que la transformation d'une histoire légale vide par rapport à une histoire légale h donne l'histoire légale vide. L'expression (7) signifie que la transformation d'une histoire légale h_1 par rapport à la séquence d'opérations obtenue en exécutant l'histoire légale h_3 après l'histoire légale h_2 se fait en deux étapes. h_1 est transformée par rapport à h_2 puis le résultat $IT^*(h_1, h_2)$ est ensuite transformé par rapport à h_3. L'expression (8) indique que la transformation de la séquence d'opérations obtenue en exécutant l'histoire légale h_2 après l'histoire légale h_1 par rapport à l'histoire légale h_3 donne une séquence composée de la transformation de h_1 par rapport à h_3 et de la transformation de h_2 par rapport $IT^*(h_3, h_1)$ qui est la transformation de h_3 par rapport à h_1. L'expression (9) définit que la transformation d'une séquence d'opération contenant la seule opération o_1

par rapport à une séquence d'opération contenant la seule opération o_2 donne une séquence contenant la seule opération $IT(o_1, o_2)$ qui est le résultat de la transformation de l'opération o_1 par rapport à o_2 en utilisant la fonction de transformation IT. Cette dernière expression indique le lien entre IT et IT^*.

Définition .10 (Opérations commutatives) *. Deux opérations o_1 et o_2 commutent si et seulement si, pour tout état st, les séquences d'exécution $[o_1; o_2]$ et $[o_2; o_1]$ conduisent toutes deux à des états corrects et sont équivalentes. Elles sont également dites commutatives.*

ANNEXE B : ALGORITHMES DE TRANSFORMATIONS INCLUSIVES

B.1. Algorithme dOPT [Ellis et Gibbs (1989)]

```
[1]  IT (Ins(p1, c1, pr1), Ins(p2, c2, pr2)) =
[2]      if (p1 < p2) then return Ins(p1, c1, pr1)
[3]      else if (p1 > p2) then return Ins(p1 + 1, c1, pr1)
[4]      else if (c1 == c2) then return Nop()
[5]      else if (pr1 > pr2) then return Ins(p1 + 1, c1, pr1)
[6]      else return Ins(p1, c1, pr1)
[7]      end if
[8]  IT (Ins(p1, c1, pr1),Del(p2, pr2)) =
[9]      if (p1 < p2) then return Ins(p1, c1, pr1)
[10]     else return Ins(p1 - 1, c1, pr1)
[11]     end if
[12] IT (Del(p1, pr1), Ins(p2, c2, pr2)) =
[13]     if (p1 < p2) then return Del(p1, pr1)
[14]     else return Del(p1 + 1, pr1)
[15]     end if
[16] IT (Del(p1, pr1),Del(p2, pr2)) =
[17]     if (p1 < p2) then return Del(p1, pr1)
[18]     else if (p1 > p2) then return Del(p1 - 1, pr1)
[19]     else return Nop()
[20]     end if
```

B.2. Algorithme adOPTed [Ressel *et al.* (1996)]

```
[1]  IT (Ins(p1, c1, u1), Ins(p2, c2, u2)) =
[2]      if (p1 < p2 or (p1 = p2 and u1 < u2)) then return Ins(p1, c1, u1)
[3]      else return Ins(p1 + 1, c1, u1)
[4]      end if
[5]  IT (Ins(p1, c1, u1),Del(p2, u2)) =
[6]      if (p1 ≤ p2) then return Ins(p1, c1, u1)
[7]      else return Ins(p1 - 1, c1, u1)
[8]      end if
[9]  IT (Del(p1, u1), Ins(p2, c2, u2)) =
[10]     if (p1 < p2) then return Del(p1, u1)
[11]     else return Del(p1 + 1, u1)
[12]     end if
[13] IT (Del(p1, u1),Del(p2, u2)) =
[14]     if (p1 < p2) then return Del(p1, u1)
[15]     else if (p1 > p2) then return Del(p1 - 1, u1)
[16]     else return Nop()
[17]     end if
```

B.3. Algorithme de Sun [Sun *et al.* (1998)]

```
[1]  IT (Ins(p1, s1, l1), Ins(p2, s2, l2)) =
[2]      if (p1 < p2) then return Ins(p1, s1, l1)
[3]      else return Ins(p1 + l2, s1, l1)
[4]      end if
[5]  IT (Ins(p1, s1, l1), Del(p2, l2)) =
[6]      if (p1 ≤ p2) then return Ins(p1, s1, l1)
[7]      else if (p1 > p2 + l2) then return Ins(p1 - l2, s1, l1)
[8]      else return Ins(p2, s1, l1)
[9]      end if
[10] IT (Del(p1, l1),Ins(p2, s2, l2)) =
[11]     if (p2 ≥ p1 + l1) then return Del(p1, l1)
[12]     else if (p1 ≥ p2) then return Del(p1 + l2, l1)
[13]     else return [Del(p1, p2 - p1) ; Del(p2 + l2, l1 - (p2 - p1))]
[14]     end if
[15] IT (Del(p1, l1),Del(p2, l2)) =
[16]     if (p2 ≥ p1 + l1) then return Del(p1, l1)
[17]     else if (p1 ≥ p2 + l2) then return Del(p1 - l2, l1)
[18]     else if (p2 ≤ p1 and p1 + l1 ≤ p2 + l2) then return Del(p1, 0)
[19]     else if (p2 ≤ p1 and p1 + l1 > p2 + l2) then return Del(p2, (p1 + l1) - (p2 + l2))
[20]     else if (p2 > p1 and p2 + l2 ≥ p1 + l1) then return Del(p1, p2 - p1)
[21]     else return Del(p1, l1 - l2)
[22]     end if
```

B.4. Algorithme SOCT2 [Suleiman *et al.* (1998)]

```
[1]  IT (Ins(p1, c1, av1, ap1), Ins(p2, c2, av2, ap2)) =
[2]      if (p1 < p2) then return Ins(p1, c1, av1, ap1)
[3]      else if (p1 > p2) then return Ins(p1 + 1, c1, av1, ap1)
[4]      else if (av1 ∩ ap2 ≠ ∅) then return Ins(p1 + 1, c1, av1, ap1)
[5]      else if (ap1 ∩ av2 ≠ ∅) then return Ins(p1, c1, av1, ap1)
[6]      else if (code(c1) > code(c2)) then return Ins(p1, c1, av1, ap1)
[7]      else if (code(c1) < code(c2)) then return Ins(p1 + 1, c1, av1, ap1)
[8]      else return Nop()
[9]      end if
[10] IT (Ins(p1, c1, av1, ap1),Del(p2)) =
[11]     if (p1 ≤ p2) then return Ins(p1, c1, av1, ap1∪ {Del(p2)})
[12]     else return Ins(p1 - 1, c1, av1∪ {Del(p2)}, ap1)
[13]     end if
[14] IT (Del(p1), Ins(p2, c2, av2, ap2)) =
[15]     if (p1 < p2) then return Del(p1)
[16]     else return Del(p1 + 1)
[17]     end if
[18] IT (Del(p1),Del(p2)) =
[19]     if (p1 < p2) then return Del(p1)
[20]     else if (p1 > p2) then return Del(p1 -1)
[21]     else return Nop()
[22]     end if
```

B.5. Algorithme d'Imine [Imine *et al.* (2003)]

```
[1]  IT (Ins(p1, o1, c1), Ins(p2, o2, c2)) =
[2]      if (p1 < p2) then return Ins(p1, o1, c1)
[3]      else if (p1 > p2) then return Ins(p1 + 1, o1, c1)
[4]      else if (o1 < o2) then return Ins(p1, o1, c1)
[5]      else if (o1 > o2) then return Ins(p1 + 1, o1, c1)
[6]      else if (code(c1) < code(c2)) then return Ins(p1, o1, c1)
[7]      else if (code(c1) > code(c2)) then return Ins(p1 + 1, o1, c1)
[8]      else return Nop()
[9]      end if
[10] IT (Ins(p1, o1, c1),Del(p2)) =
[11]     if (p1 ≤ p2) then return Ins(p1, o1, c1)
[12]     else return Ins(p1-1, o1, c1)
[13]     end if
[14] IT (Del(p1), Ins(p2, o2, c2)) =
[15]     if (p1 < p2) then return Del(p1)
[16]     else return Del(p1 + 1)
[17]     end if
[18] IT (Del(p1),Del(p2)) =
[19]     if (p1 < p2) then return Del(p1)
[20]     else if (p1 > p2) then return Del(p1-1)
[21]     else return Nop()
[22]     end if
```

B.6. Algorithme SDT [Li et Li (2004), Li et Li (2008a)]

```
[1]  Soient op1 = Ins(p1, c1, u1) et op2 = Ins(p2, c2, u2)
[2]  IT(op1, op2) =
[3]      si β(op1) < β(op2) alors retourner Ins(p1, c1, u1)
[4]      sinon si β(op1) > β(op2) alors etourner Ins(p1 + 1, c1, u1)
[5]      sinon si p1 < p2 alors retourner Ins(p1, c1, u1)
[6]      sinon si p1 > p2 alors retourner Ins(p1 + 1, c1, u1)
[7]      sinon si u1 < u2 alors retourner Ins(p1, c1, u1)
[8]      sinon retourner Ins(p1 + 1, c1, u1)
[9]      fin si
[10] IT(Ins(p1, c1, u1),Del(p2, u2)) =
[11]     si (p1 ≤ p2) alors retourner Ins(p1, c1, u1)
[12]     sinon retourner Ins(p1-1, c1, u1)
[13]     fin si
[14] IT(Del(p1, u1), Ins(p2, c2, u2)) =
[15]     si (p1 < p2) alors retourner Del(p1, u1)
[16]     sinon retourner Del(p1 + 1, u1)
[17]     fin si
[18] IT(Del(p1, u1),Del(p2, u2)) =
[19]     si (p1 < p2) alors retourner Del(p1, u1)
[20]     sinon si (p1 > p2) alors retourner Del(p1-1, u1)
[21]     sinon retourner Nop()
[22]     fin si
```