Partage de Données dans les Systèmes Collaboratifs : De la Synchronisation à la Protection des Données

Imine Abdessamad

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Partage de Données dans les Systèmes Collaboratifs

De la Synchronisation à la Protection des Données

THÈSE

présentée et soutenue publiquement le 9 Décembre 2016

pour l'obtention d'une

Habilitation de l'Université de Lorraine

(mention informatique)

par

Abdessamad Imine

Composition du jury

Rapporteurs :	Angela Bonifati	Professeur, Université Claude Bernard, Lyon 1
	Frédéric Cuppens	Professeur, Télécom Bretagne
	Mohamed Mosbah	Professeur, Institut de Polytechnique de Bordeaux
Examinateurs :	Sihem Amer-Yahia	Directrice de Recherche CNRS, LIG de Grenoble
	Achour Mostefaoui	Professeur, Université de Nantes
	Dominique Méry	Professeur, Université de Lorraine, Nancy
	Michaël Rusinowitch	Directeur de Recherche INRIA, Nancy

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

Mis en page avec la classe thesul.

Remerciements

Je tiens à remercier très vivement les Professeurs Angela Bonifati, Frédéric Cuppens et Mohamed Mosbah pour avoir répondu positivement à ma demande d'être rapporteur. Je joins à ces remerciements les Professeurs Sihem Amer-Yahia, Achour Mostefaoui, Dominique Méry et Michaël Rusinowitch. Je les remercie chaleureusement du grand honneur qu'ils me font pour faire partie de mon jury.

Il m'est agréable de remercier amicalement mon parrain, Michaël Rusinowitch, pour m'avoir transmis le goût de la recherche. Le travail d'équipe que nous avons entrepris ensemble a été pour moi une expérience très enrichissante et qui continuera de l'être.

J'adresse un merci à mes collègues du projet INRIA Pesto, Christophe, Jannik, Laurent, Mathieu, Steve, Véronique et Vincent, ainsi que tous mes collègues du département informatique de l'IUT Charlemagne.

Un grand merci également à mes collègues de l'Ecole Polytechnique de Montréal au Canada : Hanifa Boucheneb et Aurel Randolph.

Ce travail est le fruit d'une collaboration. Aussi, je tiens à présenter mes sincères remerciements à mes anciens thésards qui ont contribué aux résultats présentés dans ce manuscrit : Asma Cherif, Houari Mahfoud et Bao-Thien Hoang.

Je dédie cette thèse à mon frère Mohamed Abou El-Kacem.

Sommaire

Introduction Générale

1 Contexte de la recherche

Ce document est une synthèse des recherches que j'ai effectuées depuis à peu près neuf ans. Une partie de ces recherches est la continuité des travaux que j'avais menés dans ma thèse de doctorat soutenue en décembre 2006. Quant à l'autre partie, elle a démarré après mon recrutement, en septembre 2007, en tant que maître des conférences à l'Université de Lorraine et chercheur au projet INRIA CASSIS [1] au LORIA.

Les travaux hérités de ma thèse de doctorat portent sur la conception formelle d'algorithmes de réplication optimiste appliqués aux systèmes collaboratifs (tels que les éditeurs collaboratifs) où la cohérence des objets partagés représente une propriété critique. L'obtention des financements de thèses et la participation à des projets de recherche, comme le projet ANR Streams (Solutions for peer-To-peer REAl-tiMe Social web), m'ont permis d'aborder des thèmes de sécurité autour des systèmes collaboratifs, telles que la proposition de nouveaux modèles de contrôle d'accès et la conception de protocoles distribués conciliant la vie privée et la collaboration dans les réseaux sociaux.

Depuis mon recrutement à l'Université de Lorraine, j'ai eu l'occasion de superviser six thèses. Trois d'entre elles ont été soutenues, dont les contributions sont brièvement décrites dans le présent document, et qui concernent les doctorants suivants :

— Asma Cherif a défendu sa thèse en octobre 2012. Elle a travaillé sur la combinaison de la collaboration et le contrôle d'accès dans les systèmes collaboratifs. Ce travail de thèse a permis de proposer un modèle générique basé sur l'approche de réplication optimiste de l'objet partagé ainsi que sa politique de contrôle d'accès.

— Houari Mahfoud a soutenu sa thèse en février 2014. Il a traité des problèmes concernant la gestion des restrictions d'accès pour des données XML partagées par plusieurs utilisateurs. Il a défini un nouveau modèle d'accès s'appuyant sur le standard XPath et la réécriture des requêtes.

— Bao-Thien Hoang a présenté publiquement sa thèse en février 2015. Il a abordé le problème de sondage dans les réseaux sociaux décentralisés où la vie privée et la réputation du votant sont cruciales. Il a proposé des protocoles de sondage basés sur le partage de secret sans cryptographie.

J'ai également supervisé un certain nombre d'étudiants en master et d'étudiants en stage [2] sur divers sujets liés aux systèmes collaboratifs telles que la conception des protocoles de sécurité et la vérification formelle des protocoles de synchronisation basés sur la réplication optimiste.

1. Ce projet INRIA est reconduit depuis janvier 2016 sous le nom de PESTO.
2. Les résultats obtenus lors de la visite de stage au LORIA d'Aurel Randolph, doctorant de l'Ecole Polytechnique de Montréal au Canada, sont brièvement décrits dans ce mémoire.

2 Principales contributions

Les nouvelles technologies de l'information et de la communication, notamment Internet, ont démocratisé le travail collaboratif assisté par ordinateur. Des systèmes informatiques permettent de fédérer plusieurs personnes, dispersées dans le temps, l'espace et à travers les organisations, pour réaliser un projet commun. Ainsi, les systèmes collaboratifs mettent à disposition un ensemble d'objets partagés (comme les documents textuels et/ou graphiques) pouvant à tout moment être visibles et accessibles par les collaborateurs. Par exemple, le travail collaboratif peut être appliqué pour : (i) l'édition simultanée d'un même article scientifique par plusieurs chercheurs ; (ii) la manipulation d'une base de données partagée par une équipe de producteurs (accès en écriture) et de consommateurs (accès en lecture) ; (iii) l'exploitation des données sociales (textes, images, commentaires, recommendations, etc.) dans les plateformes de réseautage social par une communauté d'utilisateurs dispersés géographiquement. Néanmoins, les besoins identifiés autour de la gestion des données partagées constituent toujours des verrous scientifiques importants. En effet, les utilisateurs requièrent toujours une collaboration sans contraintes qui leur permet des accès rapides, cohérents et sécurisés aux données partagées.

L'importance des systèmes collaboratifs a considérablement augmenté au cours des dernières années. La majorité de nouvelles applications sont conçues de manière distribuée pour répondre aux besoins du travail collaboratif. Ainsi, pour avoir une disponibilité permanente des données et un temps de réponse court, les objets partagés sont répliqués sur chaque site du collaborateur. Cette réplication nécessite des procédures de synchronisation pour le maintien de cohérence des différentes copies. La conception de telles procédures n'est pas une tâche facile même pour des objets simples tels que les documents textes et les arbres XML ordonnés. D'où la nécessité d'utiliser un cadre formel pour concevoir des solutions correctes.

La sécurité des données partagées joue un rôle croissant dans la confiance accordée aux systèmes collaboratifs par les utilisateurs. A ce titre, un des problèmes qui constitue un véritable défi est comment établir un équilibre entre la disponibilité et le contrôle d'accès aux données partagées. En effet, les interactions dans un groupe de travail visent à rendre les objets partagés disponibles à tous les membres de ce groupe, alors que le contrôle d'accès permet de restreindre cette disponibilité à certains membres. Dans le cas où les données partagées sont répliquées, les systèmes collaboratifs requièrent un temps court pour les mises à jour des copies. Cependant, valider chaque mise à jour locale par une autorisation émanant d'un site distant (hébergé sur un serveur) va certainement dégrader les performances du système. En outre, même dans un système collaboratif centralisé où les utilisateurs collaborent via un espace commun (par exemple, une base de données, SourceForge, etc.), la structure des données partagées (tels que les arbres et les graphes) rend problématique la conception de protocoles corrects de contrôle d'accès.

Par ailleurs, l'utilisation des systèmes collaboratifs peut aider les utilisateurs à prendre conjointement des décisions pour faire face à des problèmes complexes ou traiter de sujets d'actualité (telles que les activités en ligne sur les réseaux sociaux). Cette décision collective est souvent basée sur des sondages organisés entre plusieurs personnes (dispersées géographiquement) dont les résultats permettent d'opter vers le choix majoritaire. Pour une prise de décision juste et sécurisée, il est crucial de concevoir des protocoles de sondage efficaces, permettant la préservation de la vie privée des votants et limitant l'impact des comportements malhonnêtes sur la décision finale.

Dans ce qui suit, nous résumons quelques contributions de nos travaux de recherche autour des systèmes collaboratifs.

Dans un premier temps, nous nous sommes intéressés aux problèmes de la cohérence des objets partagés et synchronisés par une technique de réplication optimiste, appelée Transfor-

mation Opérationnelle (TO). En utilisant une méthode de synthèse de contrôleur basée sur les automates de jeu, nous montrons l'impossibilité de trouver une fonction de transformation pour assurer la cohérence des objets linéaires (tels que le texte, l'arbre ordonné XML, etc.) altérés par de simples opérations d'insertion et de suppression [RBIQ15, RBIQ12]. L'extension de l'opération d'insertion par une méta-donnée nous a permis de proposer une nouvelle fonction de transformation dont la propriété de convergence a été vérifiée par une technique de model-checking [RBIQ15, BIN10, BI09]. En outre, nous étudions formellement la combinaison des approches TO et d'annulation des opérations. En modélisant cette combinaison comme un problème de satisfaction de contraintes (CSP), nous établissons des conditions nécessaires et suffisantes pour qu'une fonction de transformation préserve la cohérence [CI16, CI15, Che12].

Dans un deuxième temps, nous traitons du problème de contrôle d'accès dans les systèmes collaboratifs pour étudier la composition de la gestion des autorisations (la partie administrative) et l'activité de collaboration (la partie applicative). Nous proposons un modèle générique basé sur une réplication de l'objet partagé ainsi que sa politique de contrôle d'accès [CIR14, CIR11, ICR09, Che12]. Le choix d'une approche optimiste pour la partie administrative confère une souplesse à la collaboration mais engendre des violations temporaires de la politique. Nous annulons donc l'effet des modifications illégales pour asseoir partout la même politique [ICR09, CI09]. De plus, en employant une technique symbolique de model-checking borné, nous spécifions formellement l'empilement de notre contrôle d'accès à un système collaboratif. L'analyse a permis de conclure que le contrôle d'accès est uniformément appliqué sur toutes les copies de l'objet et préserve la cohérence. Cette analyse nous a également permis de valider certains choix conceptuels de notre modèle [RIBQ13, RIBQ14].

Dans un troisième temps, nous étudions le problème d'accès à des informations confidentielles contenues dans des documents XML partagés par un nombre important d'utilisateurs ayant des rôles très variés. Étant donnée l'existence des spécifications de sécurité définissant les droits d'accès des utilisateurs, nous utilisons la réécriture des requêtes qui transforme une requête XPath, formulée par un utilisateur sur des données autorisées par sa propre spécification, en une autre requête sur le document XML originel. Cette réécriture évite de matérialiser des vues pour tous les utilisateurs. Plus précisément, au lieu d'utiliser un langage non-standard et coûteux tel que "Regular XPath", nous montrons que la réécriture des requêtes est toujours possible pour des vues de sécurité arbitraires (récursives ou non) en utilisant uniquement la puissance expressive de la norme standard XPath [MI15, MI12d, MI12a, MI12b, MI12c]. Enfin, nous avons développé le système SVMAX, qui met en œuvre nos solutions pour un support efficace à la sécurisation de l'accès et des mises-à-jour sur des vues XML [MIR13, Mah14].

Dans un quatrième temps, nous abordons le problème de sondage dans les réseaux sociaux décentralisés où le caractère secret des informations sélectionnées et la réputation de l'utilisateur sont très critiques. En effet, les utilisateurs désirent préserver la confidentialité de leurs votes et dissimuler, le cas échéant, leurs comportements non conformes au bon déroulement du scrutin. Nous avons mis au point un protocole de sondage basé sur le partage de secret et ne nécessitant aucune infrastructure cryptographique [HI12, BT15]. Notre protocole est asynchrone et s'appuie sur le graphe social pour établir une collaboration entre les utilisateurs [HI13]. Pour permettre une diffusion efficace de messages à travers les liens sociaux, nous avons recouru à une propriété basée sur le tri topologique des graphes sociaux [HI15]. Ensuite, nous définissons les conditions nécessaires et suffisantes pour que le vote soit sécurisé et précis en agrégeant les votes initiaux avec la présence d'utilisateurs malhonnêtes, qui tentent de biaiser le résultat final et divulguer les votes des participants honnêtes.

3 Organisation du document

Comme ce document présente une synthèse de mes travaux de recherche et une présentation des perspectives que ces travaux ouvrent, toutes les parties des contributions que je rapporte ne sont pas données avec le même niveau de détail. Tout d'abord aucune preuve n'est détaillée. Le détail des preuves peut être trouvé dans les articles publiés et les rapports de thèse. Dans la mesure du possible, j'ai essayé de présenter mon travail avec une vue plus au moins générale, même parfois au prix de donner seulement une très brève présentation pour certains résultats. Tel est le prix à payer pour rendre ce document relativement concis.

Ce document est organisé en quatre chapitres.

Dans le premier chapitre, nous étudions l'existence des fonctions de transformation pour synchroniser des objets partagés ayant une structure linéaire (tels que la liste, les chaînes de caractères, etc.) ainsi que le problème d'annulabilté dans les systèmes collaboratifs. Tout d'abord, nous donnons une présentation générale de l'approche des transformées opérationnelles. Ensuite, en utilisant une méthode de synthèse de contrôleur basée sur les automates de jeu, nous montrons l'impossibilité de trouver une fonction de transformation pour assurer la cohérence des objets linéaires modifiés par de simples opérations d'insertion et de suppression et nous proposons une nouvelle fonction correcte basée sur une signature étendue de l'opération d'insertion. Enfin, nous étudions formellement la combinaison des approches de réplication optimiste et d'annulation des opérations.

Dans le deuxième chapitre, nous présentons un modèle de contrôle d'accès bien adapté aux systèmes collaboratifs. Le contexte étant celui d'un système distribué avec réplication des données partagées, le modèle de gestion des autorisations permet également la réplication de la politique d'accés. Premièrement, nous présentons les composants de notre modèle de coordination. Ensuite, nous décrivons les couches de notre modèle générique de contrôle d'accès ainsi que le protocole optimiste pour le contrôle de concurrence sur les différentes copies de la politique. En utilisant une technique symbolique de model-checking, nous montrons enfin comment spécifier formellement l'empilement de notre modèle d'accès à un système collaboratif ainsi qu'analyser la préservation de la propriété de cohérence.

Dans le troisième chapitre, nous montrons qu'il est toujours possible de réécrire des requêtes XPath sur des schémas arbitraires (récursifs ou non) en utilisant uniquement la puissance expressive de la norme standard XPath. Nous proposons un language pour spécifier les politiques de contrôle d'accès XML ainsi qu'un algorithme efficace pour faire appliquer de telles politiques. Tout d'abord, nous donnons des notions préliminaires sur XML ainsi que les fragments XPath et nous présentons la problématique sur la réécriture des requêtes. Ensuite, nous décrivons formellement notre modèle de contrôle d'accès. Enfin, nous donnons une brève description sur les caractéristiques de notre système SVMAX ainsi que les mesures de performances.

Dans le quatrième chapitre, nous proposons la conception d'un protocole simple de sondage décentralisé basé sur le partage de secret et ne nécessitant ni autorité centrale ni cryptographie. Premièrement, nous décrivons notre modèle de sondage ainsi qu'une famille de graphe sociaux. Deuxièmement, nous présentons notre protocole de sondage et nous analysons sa correction sans et avec la présence des comportements malhonnêtes. Enfin, nous discutons l'impact des problèmes de défaillance des nœuds et la perte des message sur le comportement de notre protocole. Nous présentons également quelques exemples de graphes sociaux sur lesquels nous pourrons déployer notre protocole.

Enfin, nous concluons ce mémoire en résumant nos contributions de recherche et en indiquant quelques perspectives.

Publications autour de mes travaux

[BI09] H. Boucheneb and A. Imine. On model-checking optimistic replication algorithms. In *FMOODS/FORTE*, pages 73–89, 2009.

[BIN10] H. Boucheneb, A. Imine, and M. Najem. Symbolic model-checking of optimistic replication algorithms. In *IFM*, pages 89–104, 2010.

[BT15] Hoang Bao Thien. *On the Polling Problem for Decentralized Social Networks*. Theses, INRIA Nancy ; LORIA - Université de Lorraine, February 2015.

[Che12] Asma Cherif. *Access Control Models for Collaborative Applications*. Theses, Université Nancy 2, November 2012.

[CI09] Asma Cherif and Abdessamad Imine. Undo-based access control for distributed collaborative editors. In *Cooperative Design, Visualization, and Engineering, 6th International Conference, CDVE 2009, Luxembourg, Luxembourg, September 20-23, 2009. Proceedings*, pages 101–108, 2009.

[CI15] Asma Cherif and Abdessamad Imine. A constraint-based approach for generating transformation patterns. In *Proceedings 14th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2015, Madrid, Spain, 5th September 2015.*, pages 48–62, 2015.

[CI16] Asma Cherif and Abdessamad Imine. Using CSP for coordinating undo-based collaborative applications. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 1928–1935, 2016.

[CIR11] Asma Cherif, Abdessamad Imine, and Michaël Rusinowitch. Optimistic access control for distributed collaborative editors. In *SAC*, pages 861–868, 2011.

[CIR14] Asma Cherif, Abdessamad Imine, and Michaël Rusinowitch. Practical access control management for distributed collaborative editors. *Pervasive and Mobile Computing*, 15 :62–86, 2014.

[HI12] Bao-Thien Hoang and Abdessamad Imine. On the Polling Problem for Social Networks. In *OPODIS*, pages 46–60, 2012.

[HI13] Bao-Thien Hoang and Abdessamad Imine. On constrained adding friends in social networks. In *SocInfo*, pages 467–477, 2013.

[HI15] Bao-Thien Hoang and Abdessamad Imine. Efficient and decentralized polling protocol for general social networks. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, pages 171–186, 2015.

[ICR09] Abdessamad Imine, Asma Cherif, and Michaël Rusinowitch. A Flexible Access Control Model for Distributed Collaborative Editors. In Willem Jonker and Milan Petkovic, editors, *Secure Data Management*, volume 5776 of *Lecture Notes in Computer Science*, pages 89–106, 2009.

[Mah14] Houari Mahfoud. *Efficient Access Control to XML Data : Querying and Updating Problems*. Theses, Université de Lorraine, February 2014.

[MI12a] Houari Mahfoud and Abdessamad Imine. A general approach for securely updating xml data. In *Proceedings of the 15th International Workshop on the Web and Databases (WebDB 2012)*, pages 55–60, 2012.

[MI12b] Houari Mahfoud and Abdessamad Imine. On securely manipulating xml data. In *Foundations and Practice of Security - 5th International Symposium (FPS 2012), Revised Selected Papers*, volume 7743 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2012.

[MI12c] Houari Mahfoud and Abdessamad Imine. On securely manipulating xml data. In *Journées Bases de Données Avancées (BDA 2012)*, 2012.

[MI12d] Houari Mahfoud and Abdessamad Imine. Secure querying of recursive xml views : a standard xpath-based technique. In *Proceedings of the 21st World Wide Web Conference, WWW 2012 (Companion Volume)*, pages 575–576. ACM, 2012.

[MI15] Houari Mahfoud and Abdessamad Imine. Efficient querying of XML data through arbitrary security views. *Trans. Large-Scale Data- and Knowledge-Centered Systems*, 22 :75–114, 2015.

[MIR13] Houari Mahfoud, Abdessamad Imine, and Michaël Rusinowitch. Svmax : a system for secure and valid manipulation of xml data. In *Proceedings of the 17th International Database Engineering & Applications Symposium (IDEAS)*, pages 154–161. ACM, 2013.

[RBIQ12] Aurel Randolph, Hanifa Boucheneb, Abdessamad Imine, and Alejandro Quintero. On consistency of operational transformation approach. In *Proceedings 14th International Workshop on Verification of Infinite-State Systems, Infinity 2012, Paris, France, 27th August 2012.*, pages 45–59, 2012.

[RBIQ15] Aurel Randolph, Hanifa Boucheneb, Abdessamad Imine, and Alejandro Quintero. On synthesizing a consistent operational transformation approach. *IEEE Trans. Computers*, 64(4) :1074–1089, 2015.

[RIBQ13] Aurel Randolph, Abdessamad Imine, Hanifa Boucheneb, and Alejandro Quintero. Specification and verification using alloy of optimistic access control for distributed collaborative editors. In *Formal Methods for Industrial Critical Systems - 18th International Workshop, FMICS 2013, Madrid, Spain, September 23-24, 2013. Proceedings*, pages 184–198, 2013.

[RIBQ14] Aurel Randolph, Abdessamad Imine, Hanifa Boucheneb, and Alejandro Quintero. Spécification et analyse d'un protocole de contrôle d'accès optimiste pour éditeurs collaboratifs répartis. *Ingénierie des Systèmes d'Information*, 19(6) :9–32, 2014.

Chapitre 1

Synthèse de Fonctions de Transformation

Sommaire

1.1 Introduction

L'approche des Transformées Opérationnelles (TO) est basée sur une réplication optimiste qui permet à plusieurs utilisateurs (ou sites) de générer des opérations pour modifier simultanément une ou plusieurs données partagées et ensuite synchroniser leurs répliques divergentes afin d'obtenir une vue identique [34, 120]. Les opérations de chaque site sont d'abord exécutés sur la réplique locale et ensuite propagées vers d'autres sites pour être intégrées. TO se présente sous forme d'algorithmes transformant les opérations reçues (provenant d'autres sites) pour tenir compte de celles qui ont pu entre temps modifier l'état de l'objet. Elle utilise les propriétés sémantiques des opérations pour garantir la convergence (ou le même contenu) des copies. Plus précisément, elle exploite la transformation des opérations pour construire une histoire (ou la liste des opérations) de chaque copie de l'objet. Les histoires des différentes copies ne sont pas

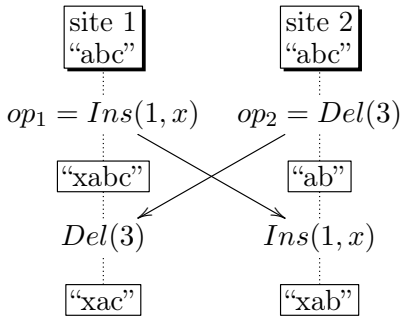


FIGURE 1.1 – Incorrecte intégration.

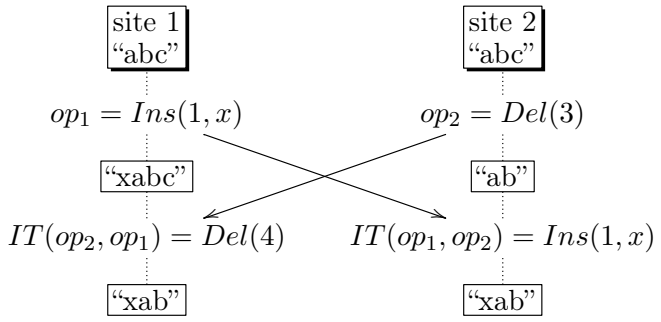


FIGURE 1.2 – Intégration avec transformation.

identiques mais elles sont équivalentes (elles aboutissent au même état final) puisque l'ordre d'exécution des opérations peut être différent d'une histoire à une autre.

A titre d'exemple, considérons deux sites (ou utilisateurs) 1 et 2 partageant un document dont la valeur initiale est "*abc*" (voir la figure 1.1). Le site 1 génère et exécute l'opération $op_1 = Ins(1, x)$ dont l'effet consiste à insérer le caractère x à la position 1 ; il produit la chaîne "*xabc*". En parallèle, le site 2 génère et exécute l'opération $op_2 = Del(3)$ dont l'effet est de supprimer le caractère c qui se trouve à la position 3. La chaîne produite est "*ab*". Lorsque op_2 arrive sur le site 1, elle sera exécutée sur l'état "*xabc*" et, au lieu de détruire le caractère c, elle va effacer le caractère b pour obtenir l'état "*xac*". Une fois arrivée sur le site 2, op_1 sera appliquée sur l'état "*ab*" pour le modifier en "*xab*". Nous remarquons que les deux sites divergent comme ils n'ont pas le même état. Pour éviter cette situation d'inconsistance (ou incohérence), l'approche TO est utilisée. Comme illustrée dans la figure 1.2, une fonction IT (transformation incluant les effets des autres opérations) est définie pour intégrer correctement les opérations. Ainsi, sur le site 1, op_2 est transformée pour inclure l'effet de op_1 en incrémentant sa position afin d'effacer le caractère c. Quant à op_1, elle reste inchangée après transformation sur le site 2.

Comparée à d'autres techniques optimistes, TO possède les avantages suivants : (i) Elle supporte un travail collaboratif sans contraintes. En effet, elle n'impose pas un ordre total sur les opérations concurrentes. Aussi, les sites peuvent échanger leurs modifications dans n'importe quel ordre. (ii) Elle permet la transformation des opérations pour les exécuter dans un ordre arbitraire même si à l'origine ces opérations ne commutent pas. (iii) Elle produit un état de convergence sans perte de mises-à-jour. Plusieurs systèmes collaboratifs sont basés sur l'approche TO, tels que Joint Emacs [101], CoWord [122], CoPowerPoint [122], Google Wave [3] et Google Docs [4].

Chaque objet partagé doit avoir son propre algorithme de transformation dont l'écriture incombe aux développeurs des applications collaboratives. Ainsi, pour chaque paire d'opérations, le développeur doit préciser comment transformer ces opérations l'une par rapport à l'autre. Pour garantir la convergence des copies, un algorithme de transformation doit satisfaire deux conditions [95, 101] (voir la définition 1.1), qui sont difficiles (voire même impossibles) à vérifier à la main. Par ailleurs, l'annulation des opérations est une fonctionnalité indispensable dans les systèmes collaboratifs. Néanmoins, combiner l'annulation des opérations avec l'approche TO, tout en préservant la convergence, reste un problème ouvert.

Dans ce chapitre, nous étudions l'existence des fonctions IT correctes (qui assurent la convergence) pour des objets linéaires (tels que la liste, la chaînes de caractères, etc.) ainsi que le problème d'annulabilté en présence de TO. La section 1.2 donne une présentation générale de l'approche des transformées opérationnelles. Dans la section 1.3, en utilisant une méthode de

3. `http://www.waveprotocol.org/whitepapers/operational-transform`
4. `https://en.wikipedia.org/wiki/Google_Docs`

synthèse de contrôleur basée sur les automates de jeu, nous montrons l'impossibilité de trouver une fonction de transformation assurant la cohérence des objets linéaires altérés par de simples opérations d'insertion et de suppression et nous donnons une nouvelle fonction correcte basée sur l'extension de l'opération d'insertion par une méta-donnée. Dans la section 1.4, nous étudions formellement la combinaison des approches TO et d'annulation des opérations. En modélisant cette combinaison comme un problème de satisfaction de contraintes (CSP), nous établissons des conditions nécessaires et suffisantes pour qu'une fonction de transformation préserve la cohérence. Enfin, nous passons en revue l'état de l'art dans la section 1.5 et nous terminons le chapitre par une conclusion.

1.2 Approche des Transformées Opérationnelles

1.2.1 Notions préliminaires

Un objet est dit *collaboratif* s'il constitue une ressource de données partagée par plusieurs sites de telle sorte que chaque site puisse lire et/ou écrire sur sa propre copie. Chaque objet collaboratif est caractérisé par : (i) un type (par exemple, un texte, un arbre XML, un système de fichiers, etc.) qui définit un ensemble énumérable d'états, noté $\mathcal{S}$; (ii) un ensemble d'opérations primitives, noté par $\mathcal{O}$; et (iii) Une fonction de transition $Do : \mathcal{S} \times \mathcal{O} \to \mathcal{S}$.

Une opération générée localement sur site est dite *locale*. Par contre, elle est dite *distante* si elle provient d'un autre site. Une *histoire* (ou journal d'opérations) est une liste utilisée pour stocker toutes les opérations (locales et distantes) exécutées sur un site. La notation $H = [o_1; o_2; \ldots; o_n]$ désigne une histoire composée des opérations o_i (avec $i = 1, \ldots, n$). La notation $H = H_1 \bullet H_2$ représente l'histoire obtenue par concaténation de deux histoires H_1 et H_2. L'application d'une histoire H sur un état s est définie récursivement comme suit :

$$Do^*([], s) = s \text{ et}$$
$$Do^*(H \bullet [o], s) = Do(o, Do^*(H, s)).$$

où [] dénote l'histoire vide et o est une opération. Deux histoires H_1 et H_2 sont *équivalentes*, notée $H_1 \equiv H_2$, ssi $Do^*(H_1, s) = Do^*(H_2, s)$ pour tout état s.

Durant l'exécution des opérations, il existe une relation de précédence qui doit être préservée par tous les sites. Soient deux opérations o_1 et o_2 générées respectivement sur les sites i et j. On dit que que o_1 *précède causalement* o_2 (ou o_2 *dépend causalement* de o_1), noté $o_1 \to o_2$, ssi : (i) $i = j$ et o_1 était générée avant o_2 ; ou, (ii) $i \neq j$ et l'exécution de o_1 sur le site j est arrivée avant la génération de o_2. Deux opérations o_1 et o_2 sont dites *concurrentes*, notée $o_1 \parallel o_2$, ssi ni $o_1 \to o_2$, ni $o_2 \to o_1$.

Pour déterminer les relations de causalité et concurrence, les *vecteurs d'horloges* sont communément utilisés depuis longtemps dans les éditeurs collaboratifs basés sur TO [34, 120]. Dans un système à n sites, un tel vecteur V possède n composantes. Pour un site j, chaque composante $V[i]$ compte le nombre d'opérations générées du site i qui ont été déjà exécutées sur le site j.

1.2.2 Critères de consistance

Un système collaboratif basé sur TO est *consistant* ssi il satisfait les propriétés suivantes :

1. *Préservation de la précédence causale.* Si $o_1 \to o_2$ alors o_1 est exécutée avant o_2 sur tous les sites.

2. *Convergence.* Lorsque tous les sites ont exécuté le même ensemble d'opérations, toutes les copies de l'objet collaboratif sont identiques.

Il est bien connu que l'utilisation des vecteurs d'horloges permet la préservation des la précédence causale. Lorsque tous les sites cessent de propager des opérations, le système arrivera un état où il n'y a plus de modifications sur l'objet collaboratif. Aussi, à cet état, les copies de l'objet collaboratif doivent être les mêmes. La préservation des relations de causalité est nécessaire mais pas suffisante pour garantir la convergence des copies. En effet, les opérations concurrentes peuvent être exécutées sur les différents sites dans un ordre quelconque. Pour aller vers cette convergence, les opérations concurrentes doivent être transformées entre elles moyennant une fonction de transformation IT qui doit satisfaire deux propriétés. Avant de présenter ces propriétés, nous définissons comment transformer une opération o par rapport à une histoire $H = [o_1; o_2; \ldots; o_n]$ comme suit :

$$IT^*(o, []) = o \text{ et}$$
$$IT^*(o, [o_1; o_2; \ldots; o_n]) = IT^*(IT(o, o_1), [o_2; \ldots; o_n]).$$

DÉFINITION 1.1 ***(Propriétés de convergence).*** Soit IT une fonction de transformation. Pour toutes les paires d'opérations concurrentes o_1, o_2 et o_3, IT est *correcte* (*consistante*) ssi les propriétés suivantes sont satisfaites :

— **Propriété TP1** : $[o_1; IT(o_2, o_1)] \equiv [o_2; IT(o_1, o_2)]$.

— **Propriété TP2** : $IT^*(o_3, [o_1; IT(o_2, o_1)]) = IT^*(o_3, [o_2; IT(o_1, o_2)])$. ■

La propriété **TP1** définit une *identité d'états*. Elle stipule que l'état produit en exécutant o_1 avant o_2 est le même que celui résultant de l'exécution de o_2 avant o_1. La propriété **TP2** définit une *identité d'opérations*. Elle stipule que le résultat de la transformation d'une opération par rapport à une séquence d'opérations concurrentes ne dépend pas de l'ordre selon lequel les opérations de cette séquence ont été transformées. Dans [101, 77], il a été montré que **TP1** et **TP2** sont suffisantes pour satisfaire la propriété de convergence pour un *nombre arbitraire d'opérations concurrentes* qui peuvent s'exécuter dans un *ordre arbitraire*.

1.2.3 Procédures d'intégration

Dans l'approche TO, chaque site est équipé de deux composants principaux : *le composant d'intégration* et *le composant de transformation*. Le premier composant se présente comme un algorithme qui est responsable de recevoir, diffuser et exécuter les opérations. Il est *indépendant* de la sémantique des objets collaboratifs. Plusieurs algorithmes d'intégration ont été proposés dans le domaine des éditeurs collaboratifs, comme dOPT [34], adOPTed [101], SOCT2 [118] et GOTO [120]. Le composant de transformation est un ensemble d'algorithmes qui est responsable de sérialiser par transformation deux opérations concurrentes. Tout algorithme de transformation est *spécifique* à la sémantique d'un objet collaboratif.

Chaque site génère séquentiellement les opérations et les stocke dans un journal d'opérations (ou histoire). Lorsqu'un site reçoit une opération distante o sur une histoire locale H, le composant d'intégration procède par les étapes suivantes :

1. Déterminer une histoire équivalente $H' = H_p \bullet H_c$ où (i) H_p contient les opérations qui précèdent causalement o, et (ii) H_c contient les opérations qui sont concurrentes à o.
2. Utiliser le composant de transformation pour produire o' qui est la forme transformée de o par rapport à H_c (c-à-d, $o' = IT^*(o, H_c)$).

3. Exécuter o' sur l'état courant du site et ajouter o' à l'histoire locale H.

Le composant d'intégration permet donc de construire l'histoire des opérations exécutées sur le site, tout en préservant la relation causale entre les opérations. Lorsque le système est au repos, les histoires des différents sites ne sont pas forcément identiques mais elles sont équivalentes (elles aboutissent au même état final) malgré que l'ordre d'exécution des opérations concurrentes peut être différent d'une histoire à une autre. Cette équivalence ne peut être assurée que si l'algorithme de transformation utilisé satisfait les propriétés **TP1** et **TP2**.

Par ailleurs, il existe une autre catégorie d'algorithmes d'intégration qui nécessitent seulement la propriété **TP1**, tels que GOT [121], SOCT4 [132], COT [123] et SPORC [41]. Ces algorithmes imposent un ordre total sur les opérations afin de maintenir le même chemin de transformation sur tous les sites. Bien que **TP1** est suffisante pour garantir la convergence des copies, ils reposent tous sur un serveur central pour établir un ordre global. Ce choix confère un degré de concurrence moins élevé et mène potentiellement vers un goulot d'étranglement.

1.3 Synchronisation des Structures de Données Linéaires

Dans cette section, nous nous intéressons à la synchronisation par transformation des objets linéaires (tels que la liste, la chaînes de caractères, etc.). Tout d'abord, nous allons donner un aperçu sur les problèmes de divergence sous-jacents aux solutions existantes. Ensuite, en utilisant une méthode de synthèse de contrôle, nous allons montrer l'inexistence d'une fonction de transformation basée sur de simples opérations d'insertion et de suppression. Enfin, nous allons proposer une nouvelle fonction de transformation correcte qui s'appuie sur l'extension de l'opération d'insertion par une méta-donnée.

Notons qu'un objet linéaire peut être perçu comme une séquence d'éléments d'un type de données $\mathcal{A}$ (*c-à-d*, un caractère, un mot, un paragraphe, un nœud XML, etc.). L'ensemble des opérations primitives qui modifient l'état de l'objet est comme suit :

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

où $Ins(p, c)$ ajoute l'élément c à la position p, $Del(p)$ supprime l'élément se trouvant à la position p et $Nop()$ est l'opération nulle qui laisse l'état de l'objet inchangé.

1.3.1 Problématique

Plusieurs algorithmes de transformation pour l'objet linéaire ont été proposés dans la littérature [34, 101, 121, 118, 62, 72]. Ces algorithmes diffèrent principalement par la gestion des situations de *conflit*. On est en présence d'un conflit dans le cas de deux insertions concurrentes à la même position. Il faut donc faire un choix pour déterminer quel élément ajouter avant l'autre. Cependant, il faut s'assurer que le même choix soit fait sur tous les sites. Les solutions proposées, dans la littérature, se basent sur l'extension de la signature de l'opération Ins par de nouveaux paramètres pour établir un ordre sur les insertions en conflit, comme par exemple : un *ordre de priorité* sur les opérations [34], un ordre total sur les *identifiants des sites* [101], et l'ordre *lexicographique* sur les caractères [118]. Mais aucun de ces algorithmes n'arrive à assurer la convergence [62, 63] : **TP1** ou **TP2**, ou toutes les deux ne sont pas vérifiées.

Pour illustrer une telle divergence, nous allons présenter l'algorithme de Suleiman et al. [118] qui enrichit l'opération d'insertion par deux paramètres av et ap. Ces paramètres mémorisent l'ensemble des opérations de suppression concurrentes à l'opération d'insertion. L'ensemble av contient les opérations de suppression qui ont détruit un caractère devant la position d'insertion.

Quant à l'ensemble ap, il contient les opérations qui ont effacé un caractère derrière la position d'insertion. Les paramètres av et ap seront modifiés durant les étapes de transformation (voir l'algorithme détaillé dans la figure 1.3).

Trois cas sont donc possibles pour résoudre le conflit occasionné par deux opérations concurrentes $op_1 = Ins(p, c_1, av_1, ap_1)$ et $op_2 = Ins(p, c_2, av_2, ap_2)$ ajoutant un caractère à la même position :

1. $(av_1 \cap ap_2) \neq \emptyset$: le caractère c_2 est inséré avant le caractère c_1,
2. $(ap_1 \cap av_2) \neq \emptyset$: le caractère c_2 est inséré après le caractère c_1,
3. $(av_1 \cap ap_2) = (ap_1 \cap av_2) = \emptyset$: dans ce cas un ordre lexicographique sur les caractères est utilisé pour choisir lequel des deux caractères (c_1 ou c_2) doit être inséré avant.

Il faut noter que lorsque les deux opérations insèrent le même caractère (*c-à-d*, $c_1 = c_2$) à la même position, l'une est exécutée et l'autre est ignorée en retournant l'opération nulle $Nop()$. En d'autres termes, un seul exemplaire des caractères ajoutés est gardé (similaire à la solution préconisée par Ellis et *al.* [34]). Par ailleurs, lors de la génération d'une opération d'insertion, les ensembles av et ap sont initialement vides.

$$\mathrm{IT}(Ins(p_1, c_1, av_1, ap_1), Ins(p_2, c_2, av_2, ap_2)) = \begin{cases} Ins(p_1, c_1, , av_1, ap_1) & \text{si } \mathrm{p_1} < \mathrm{p_2} \vee \\ & (p_1 = p_2 \wedge ap_1 \cap av_2 \neq \emptyset) \vee \\ & (p_1 = p_2 \wedge ap_1 \cap av_2 = av_1 \cap ap_2 = \emptyset \\ & \wedge c_1 > c_2) \\ Ins(p_1 + 1, c_1, av_1, ap_1) & \text{si } \mathrm{p_1} > \mathrm{p_2} \vee \\ & (p_1 = p_2 \wedge av_1 \cap ap_2 \neq \emptyset) \vee \\ & (p_1 = p_2 \wedge ap_1 \cap av_2 = av_1 \cap ap_2 = \emptyset \\ & \wedge c_1 < c_2) \\ Nop() & \text{sinon} \end{cases}$$

$$\mathrm{IT}(Ins(p_1, c_1, av_1, ap_1), Del(p_2)) = \begin{cases} Ins(p_1, c_1, av_1, ap_1 \cup \{Del(p_2)\}) & \text{si } \mathrm{p_1} \leq \mathrm{p_2} \\ Ins(p_1 - 1, c_1, av_1 \cup \{Del(p_2)\}, ap_1) & \text{sinon} \end{cases}$$

$$\mathrm{IT}(Del(p_1), Ins(p_2, c_2, av_2, ap_2)) = \begin{cases} Del(p_1) & \text{si } \mathrm{p_1} < \mathrm{p_2} \\ Del(p_1 + 1) & \text{sinon} \end{cases}$$

$$\mathrm{IT}(Del(p_1), Del(p_2)) = \begin{cases} Del(p_1) & \text{si } \mathrm{p_1} < \mathrm{p_2} \\ Del(p_1 - 1) & \text{si } \mathrm{p_1} > \mathrm{p_2} \\ Nop() & \text{sinon} \end{cases}$$

FIGURE 1.3 – Algorithme de Suleiman et *al.* [118].

Malheureusement, la solution de Suleiman ne marche pas dans tous les cas. En effet elle ne satisfait ni **TP1** [97] ni **TP2** [62, 13]. Le contre-exemple de **TP1** est donné par le couple d'opérations $(o'_1 = Ins(2, f, \{o_3\}, \{o_5\}), o'_2 = Ins(2, c, \{o_5\}, \{o_3\}))$, comme illustré à la figure 1.4. Le scénario donné sur cette figure consiste en quatre utilisateurs u_1, u_2, u_3 et u_4 sur différents sites. Les utilisateurs u_1, u_2 et u_3 ont généré et exécuté respectivement les séquences $S_1 = [o_1 = Ins(3, f, \emptyset, \emptyset)]$, $S_2 = [o_2 = Ins(2, c, \emptyset, \emptyset)]$ et $S_3 = [o_3 = Del(2); o_4 = Ins(2, e, \emptyset, \emptyset); o_5 = Del(2)]$. Ensuite, u_3 reçoit successivement les opérations o_1 et o_2. A son tour, u_4 reçoit dans l'ordre les opérations de S_3, o_2 et o_1. La fonction IT de Suleiman ne parvient pas donc à assurer la convergence (et plus spécifiquement la propriété **TP1**) pour $o'_1 = IT^*(o_1, S_3) = Ins(3, f, \{o_3\}, \{o_5\})$ et $o'_2 = IT^*(o_2, S_3) = Ins(2, c, \{o_5\}, \{o_3\})$.

1.3.2 Synthèse de fonctions de transformation

Dans [63, 12, 13], les auteurs ont utilisé un prouveur automatique de théorèmes ou une technique de model-checking pour vérifier la consistance d'un algorithme de transformation. Ils

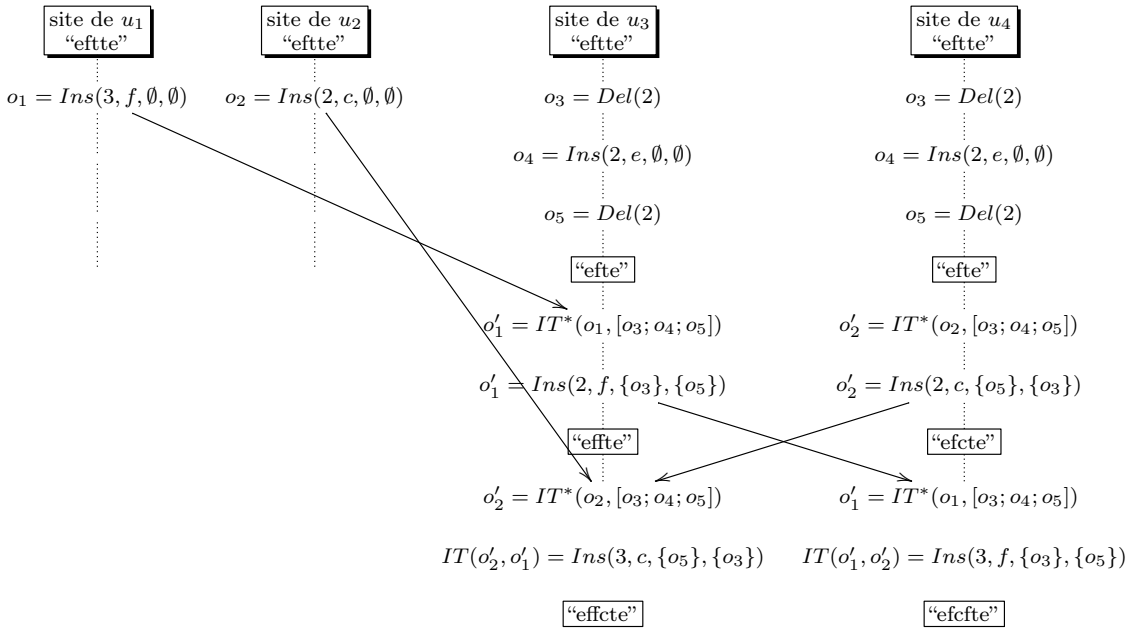


FIGURE 1.4 – Situation de divergence.

ont montré que tous algorithmes de transformation, proposés dans la littérature, divergent et un contre-exemple est fourni pour chaque algorithme. De ce fait, une question se pose : existe-t-il un algorithme consistant basé sur Ins et Del ?

Les techniques basées sur le model-checking et la démonstration automatique des théorèmes permettent de vérifier si un système donné vérifie certaines propriétés. Mais elles ne sont pas appropriées pour montrer l'existence d'un système satisfaisant des propriétés données. Par contre, les techniques de synthèse de contrôleurs permettent de s'attaquer à des problèmes plus généraux, comme vérifier si un système peut être modifié (ou forcé) afin de satisfaire des propriétés d'intérêt. Dans un tel cadre, le système comporte, en général, des transitions contrôlables et incontrôlables. L'objectif du contrôle est de trouver si une stratégie existe pour forcer les propriétés escomptées, en choisissant de manière appropriée les actions contrôlables à exécuter, quelque que soient les transitions incontrôlables qui sont franchies.

Nous nous sommes basés sur le principe de synthèse de contrôleurs pour concevoir une fonction de transformation qui satisfait les propriétés **TP1** et **TP2**. L'idée est de chercher une telle fonction dans l'ensemble des fonctions autorisées. Il y a quatre transformations autorisées pour chaque opération o : (i) incrémenter sa position, (ii) décrémenter sa position, (iii) la transformer en une opération nulle (c-à-d $Nop()$), et (iv) la laisser inchangée. Tout d'abord, nous examinons l'ensemble des fonctions autorisées pour trouver celles vérifiant **TP1**. Ensuite, nous cherchons parmi celles satisfaisant **TP1** les fonctions qui vérifient également **TP2**.

Pour ces investigations, nous utilisons le formalisme des automates de jeu à la UPPAAL [15]. Un automate de jeu est un automate avec deux types de transitions : contrôlable et incontrôlable. Chaque transition a une location de départ, une location d'arrivée et elle est annotée avec des sélections, des gardes (ou conditions) et des blocs d'actions. Les sélections permettent d'affecter, de manière non-déterministe, à un identificateur donné toutes les valeurs d'un intervalle donné. Les autres annotations (ou types d'étiquettes) font aussi partie de cette affectation. Un état est défini par la location actuelle et les valeurs actuelles de toutes les variables. Une transition est activée dans un état ssi l'emplacement actuel est la source de la transition et la garde est évaluée

à vrai. Le franchissement de la transition consiste à atteindre son emplacement de destination et d'exécuter de manière atomique son bloc d'actions pour changer l'état du système. Pour forcer certaines propriétés, les transitions activées qui sont contrôlables peuvent être retardées ou tout simplement ignorées. Cependant, les transitions incontrôlables ne peuvent être ni retardées ni ignorées.

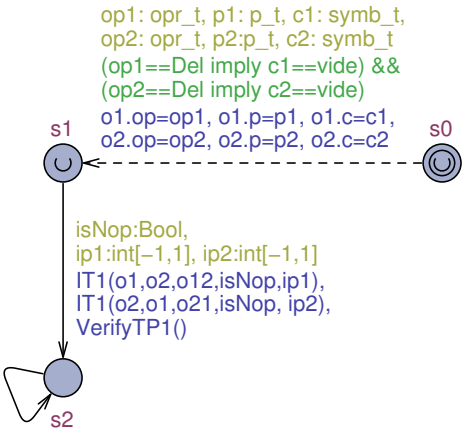


FIGURE 1.5 – Synthèse pour **TP1** [97].

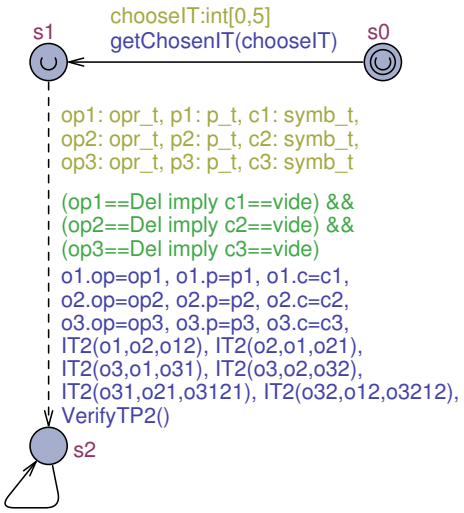


FIGURE 1.6 – Synthèse d'une fonction consistante [97].

Recherche de fonctions de transformation satisfaisant TP1. Une fonction IT satisfait **TP1** ssi pour toute paire d'opérations concurrentes o_1 et o_2, l'équivalence $[o_1; IT(o_2; o_1)] \equiv [o_2; IT(o_1, o_2)]$ est vraie. Pour vérifier s'il y a des fonctions de transformation qui satisfont la propriété **TP1**, nous avons représenté dans l'automate de jeu, illustré dans la figure 1.5, la génération d'opérations o_1 et o_2, le calcul de $IT(o_1, o_2)$ et $IT(o_2, o_1)$, et la vérification de $[o_1; IT(o_2; o_1)] \equiv [o_2; IT(o_1, o_2)]$. La génération des opérations est spécifiée par l'incontrôlable transition (s_0, s_1), puisque nous n'avons aucun contrôle sur les types d'opérations générées par les utilisateurs. Les transformations opérationnelles et la vérification de **TP1** sont représentées par la transition contrôlable (s_1, s_2).

Le modèle commence par sélectionner deux opérations o_1 et o_2. Le domaine de l'exploitation est fixé de manière à couvrir tous les cas de transformations. Ensuite, le modèle choisit deux transformations à appliquer à o_1 (resp. o_2) par rapport à o_2 (resp. o_1) et les applique en invoquant la fonction $IT1$. Notons que la fonction $IT1(o_1, o_2, o_{12}, IsNop, IP_1)$ retourne o_{12}, le résultat de la transformation de o_1 par rapport à o_2. Si o_1 est nulle (c-à-d, $IsNop = true$) alors $o_{12} = Nop()$, sinon la transformation de o_1 consiste à mettre à jour le paramètre position ($o_{12}.p = o_1.p + ip_1$). Cela signifie que 4 possibilités sont offertes pour transformer une opération o_1 par rapport à une autre opération o_2 : $Nop()$, décrémenter, maintenir ou incrémenter la position de o_1. Enfin, le modèle vérifie si **TP1** est satisfaite. Quelque soient les opérations o_1 et o_2 générées par la transition incontrôlable, la synthèse de contrôles vise à forcer la propriété **TP1** en choisissant de manière appropriée les transformations opérationnelles.

Nous avons utilisé l'outil *Uppaal-Tiga* [15] pour vérifier l'existence de certaines fonctions de transformation qui satisfont **TP1**. L'objectif de contrôle de sûreté pour **TP1** est la formule CTL $AG\ TP1$, où $TP1$ est définie dans le modèle comme une variable booléenne dont la valeur est vraie lorsque la propriété **TP1** est satisfaite. La fonction *VerifyTP1* met la variable booléenne

$TP1$ à faux si $[o_1; IT(o_2, o_1)] \not\equiv [o_2; IT(o_1, o_2)]$. L'outil *Uppaal-Tiga* conclut que la propriété est satisfaite, ce qui signifie qu'il y a, au moins, une stratégie pour forcer la propriété **TP1**. Nous reportons dans la table 1.1 les différentes fonctions de transformation (satisfaisant **TP1**) produites par *Uppaal-Tiga*.

Notons que même si certaines transformations satisfont **TP1**, elles sont inacceptables du point de vue sémantique. Par exemple, si $p_1 = p_2$, les transformations $IT(Del(p_1), Del(p_2)) = Del(p_1-1)$, $IT(Del(p_1), Del(p_2)) = Del(p_1)$ et $IT(Del(p_1), Del(p_2)) = Del(p_1+1)$ signifient que si deux utilisateurs génèrent simultanément la même opération de suppression, deux symboles seront supprimés dans chaque site, ce qui est inacceptable du point de vue sémantique. La seule transformation opérationnelle qui est logique pour ce cas est $IT(Del(p_1), Del(p_2)) = Nop()$. Cela signifie que seul le symbole à la position p_1 est supprimé dans chaque site. Après l'élimination de ces transformations incohérentes, il reste 2 possibilités pour $IT(Ins(p_1, c_1), Ins(p_2, c2))$ (avec $p_1 = p_2, c_1 \neq c_2$) et 3 possibilités pour $IT(Ins(p_1, c_1), Ins(p_2, c2))$ (avec $p_1 = p_2, c_1 = c_2$). Par conséquent, nous pouvons extraire 6 fonctions IT qui satisfont **TP1**. Ces fonctions diffèrent dans la manière dont les opérations conflictuelles sont gérées.

TABLE 1.1 – Fonctions de transformation synthétisées par *Uppaal-Tiga* pour **TP1** [97].

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

Recherche de fonctions de transformation satisfaisant TP1 et TP2. Pour vérifier s'il y a des fonctions qui satisfont les propriétés **TP1** et **TP2**, nous avons utilisé l'automate de jeu représenté à la figure 1.6. Ce modèle commence par la sélection d'une fonction vérifiant **TP1** (la gamme de *chooseIT* correspond aux 6 fonctions satisfaisant **TP1**, trouvées précédemment). Ensuite, il sélectionne trois opérations o_1, o_2 et o_3, et effectue les transformations nécessaires pour vérifier **TP2**. La fonction $IT2(o_1, o_2, o_{12})$ applique la fonction IT sélectionnée pour o_1 par rapport à o_2 et renvoie le résultat de cette transformation en o_{12}. Enfin, le modèle appelle la fonction $VerifyTP2$. L'objectif de contrôle est spécifié par la formule CTL $AG\ TP2$, où $TP2$ est une variable booléenne dont la valeur est vraie lorsque **TP2** est satisfaite. Cette variable est mise à faux par la fonction $VerifyTP2$ si $IT(IT(o_3, o_1), IT(o_2, o_1)) \neq IT(IT(o_3, o_2), IT(o_1, o_2))$.

Uppaal-Tiga conclut que la propriété $AG\ TP2$ ne peut être forcée, ce qui signifie qu'il n'y a pas de stratégie pour forcer la propriété **TP2**. En d'autres termes, il n'y a pas de fonction

IT, basée sur des paramètres classiques des opérations de suppression et insertion, qui satisfait à la fois **TP1** et **TP2**. Pourquoi une telle fonction n'existe pas ? Notre analyse a conduit à l'identification de deux scénarios symboliques qui empêchent l'obtention de cette fonction. Nous reportons dans les figures 1.7 et 1.8 respectivement ces deux séquences nommées scénarios 1 et 2. Pour vérifier **TP2** dans le scénario 1, les transformations effectuées sont :

$o_{21} = IT(o_2, o_1) = IT(Ins(p_1, c_2), o_1) = Ins(p_1, c_2)$,
$o_{12} = IT(o_1, o_2) = IT(Del(p_1), Ins(p_1, c_2)) = Del(p_1 + 1)$,
$o_{31} = IT(o_3, o_1) = Ins(p_1, c_3)$,
$o_{32} = IT(o_3, o_2) = Ins(p_1 + 2, c_3)$,
$IT(o_{32}, o_{12}) = IT(Ins(p_1 + 2, c_3), Del(p_1 + 1)) = Ins(p_1 + 1, c_3)$ et
$IT(o_{31}, o_{21}) = IT(Ins(p_1, c_3), Ins(p_1, c_2))$.

Pour la dernière transformation, nous avons différentes possibilités (voir le tableau 1.1). Pour satisfaire **TP2**, nous devons choisir $IT(Ins(p_1, c_3), Ins(p_1, c_2)) = Ins(p_1 + 1, c_3)$.

Pour le scénario 2, les transformations effectuées sont :
$o_{21} = IT(o_2, o_1) = Ins(p_1, c_2)$,
$o_{12} = IT(o_1, o_2) = Del(p_1)$,
$o_{31} = IT(o_3, o_1) = Ins(p_1, c_3)$,
$o_{32} = IT(o_3, o_2) = Ins(p_1, c_3)$,
$IT(o_{32}, o_{12}) = IT(Ins(p_1, c_3), Del(p_1)) = Ins(p_1, c_3)$ et
$IT(o_{31}, o_{21}) = IT(Ins(p_1, c_3), Ins(p_1, c_2))$.

Pour la dernière transformation opérationnelle, nous devons utiliser $IT(Ins(p_1, c_3), Ins(p_1, c_2)) = Ins(p_1, c_3)$ afin de satisfaire **TP2**.

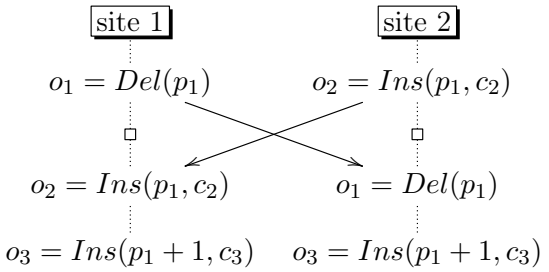


FIGURE 1.7 – Scénario 1.

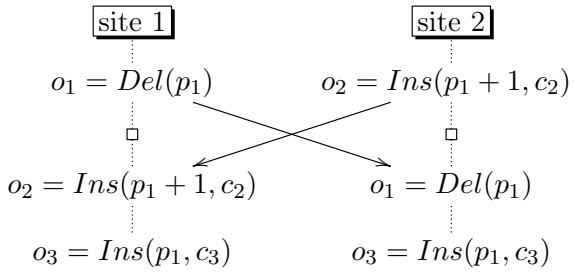


FIGURE 1.8 – Scénario 2.

Par conséquent, une fonction de transformation consistante, si elle existe, doit avoir des paramètres supplémentaires dans les signatures des opérations. Comme vu dans la section précédente, la fonction transformation de Suleiman et al. [118] utilise une extension de la signature de l'opération d'insertion par des ensembles contenant les suppressions avant et après la position d'insertion. Malheureusement, tous les paramètres supplémentaires proposées dans les fonctions de transformation, connues dans la littérature, sont insuffisantes ou inadaptées pour assurer la convergence [63, 12, 13]. Toutes ces tentatives infructueuses montrent que la conception d'une fonction consistante est une tâche difficile.

La principale différence entre les scénarios 1 et 2 (voir les figures 1.7 et 1.8) réside dans la position du symbole supprimé relativement aux symboles insérés par o_2 et o_3. Dans le scénario

1, le symbole supprimé est avant la position d'insertion de o_3, alors que, dans le scénario 2, il est avant la position d'insertion de o_2. L'extension de la signature de l'opération d'insertion avec le nombre de symboles effacés avant sa position (comme troisième paramètre) permet la définition des transformations appropriées pour que les deux scénarios 1 et 2 satisfassent **TP1** et **TP2** (voir les figures 1.9 et 1.10). En effet, dans le scénario 1, le symbole de $o_{31} = IT(o_3, o_1) = Ins(p_1, c_3, 1)$ doit précéder celui de $o_{21} = Ins(p_1, c_2, 0)$, comme o_{31} a le plus grand nombre de symboles supprimés avant sa position ($1 > 0$). Dans le scénario 2, $o_{21} = IT(o_2, o_1) = Ins(p_1, c_2, 1)$ a le plus grand nombre de symboles supprimés avant sa position. Son symbole devrait être inséré après celui de $o_{31} = Ins(p_1, c_3, 0)$. L'idée de notre nouvelle fonction IT est d'utiliser ce paramètre supplémentaire pour gérer les opérations conflictuelles. Dans ce qui suit, nous proposons sur la base de cette idée une nouvelle fonction de transformation et nous montrons formellement sa consistance.

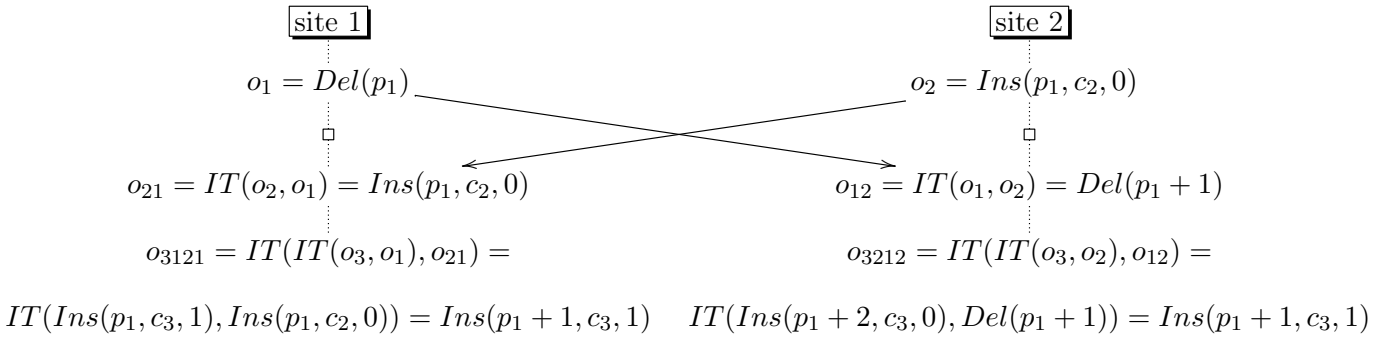


FIGURE 1.9 – Application de notre IT au scénario 1.

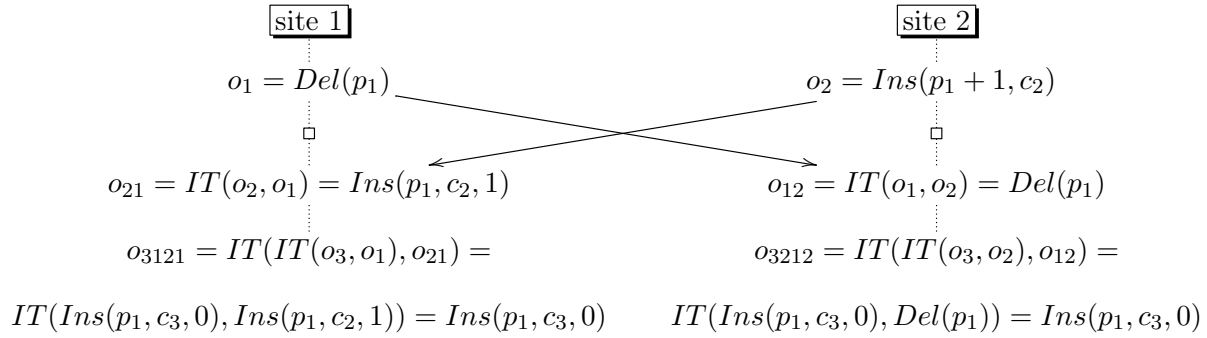


FIGURE 1.10 – Application de notre IT au scénario 2.

1.3.3 Proposition d'une nouvelle fonction de transformation

Extension de la signature de l'insertion. Nous proposons d'ajouter un nouveau paramètre, nommé *nd*, à la signature de l'opération d'insertion. Ce paramètre supplémentaire est rempli avec le nombre de symboles effacés avant la position d'insertion. Quand une opération d'insertion $o = Ins(p, c, nd)$ (p est la position d'insertion et c est l'élément à insérer) est générée, la valeur de son paramètre *nd* est calculée par rapport au contexte (c-à-d, l'histoire des opérations) de génération de o. Par la suite, ce paramètre est incrémenté à chaque fois qu'il est transformé par rapport à une opération d'effacement dont la position est avant sa position d'insertion.

DÉFINITION 1.2 Soit S une séquence d'opérations exécutées sur le document partagé avant de générer une opération d'insertion $o = Ins(p, c, nd)$. Le nombre de symboles supprimés avant p par la séquence S, noté par $ND(S, p)$, est défini récursivement comme suit :

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

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

A titre d'exemple, nous allons calculer le nombre de symboles supprimés avant la position 2, par la séquence $[Del(2); Ins(2, a); Ins(1, b)]$ (c-à-d $ND([Del(2); Ins(2, a); Ins(1, b)], 2)$). Ce nombre est égal au nombre de symboles effacés par $[Del(2); Ins(2, a)]$ avant la position 1, parce que $Ins(1, b)$ déplacera le symbole de la position 1 à la position 2. Par conséquent, $ND([Del(2); Ins(2, a); Ins(1, b)], 2) = ND([Del(2); Ins(2, a)], 1)$. Le nombre de symboles supprimés par $[Del(2); Ins(2, a)]$ avant la position 1 est égal à $ND([Del(2)], 1)$, puisque $Ins(2, a)$ n'a pas d'incidence sur les symboles avant la position 2. Enfin, comme $1 < 2$, il en résulte que $ND([Del(2)], 1) = ND([], 1) = 0$. Par conséquent, il n'y a pas de symbole effacé par $[Del(2); Ins(2, a); Ins(1, b)]$ avant la position 2. Notons que pour la séquence $[Ins(2, a), Del(3); Ins(1, b)]$, qui est équivalente à la précédente, nous trouverons le même nombre de symboles effacés avant la position 2.

Dans l'approche TO, les opérations sont intégrées dans les différents sites dans différents ordres. Pour assurer la convergence, notre procédure pour calculer ND doit donner le même résultat sur tous les sites. Plus formellement, $ND(S, p) = ND(S', p)$ pour toute position p et toute paire de séquences équivalentes S et S'. Dans ce qui suit, nous allons présenter notre nouvelle fonction de transformation et établir des propriétés intéressantes de ND qui seront utiles pour montrer la consistance (satisfaction de **TP1** et **TP2**) de notre fonction.

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

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

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

FIGURE 1.11 – Notre fonction de transformation [97].

Fonction de transformation basée sur ND. Dans notre fonction de transformation (voir la figure 1.11), les signatures des opérations d'insertion et de suppression sont respectivement $Ins(p, c, nd)$ et $Del(p)$, où p est une position, c est un symbole et nd est le nombre de symboles effacés avant la position p. Les situations conflictuelles entre deux insertions concurrentes sont gérées à l'aide de leurs paramètres nd. Plus précisément, lors de la transformation d'une paire d'opérations d'insertion ayant la même position, leurs paramètres nd sont d'abord comparées

afin de récupérer la relation entre leurs positions lors de la phase de génération. Si leurs *nd* sont égaux, alors leurs symboles sont comparés pour résoudre le conflit.

Nous établissons, dans le lemme 1.1 et le théorème 1.1, des relations entre les positions des opérations d'insertion et leurs paramètres *nd*. Nous avons d'abord supposé, dans le lemme 1.1, que les opérations sont générées après la même séquence d'opérations. Ensuite, nous considérons, dans le théorème 1.1, le cas où les opérations sont générées après des séquences différentes mais équivalentes (à savoir, elles sont constituées de la même série d'opérations originales exécutées, après intégration, dans des ordres différents). Intuitivement, ces relations signifient que pour toute paire d'opérations d'insertion générées sur le même état, la relation d'ordre de leurs positions est la même que la relation d'ordre de leur paramètres supplémentaires *nd*. Les preuves du lemme 1.1 et le théorème 1.1 sont données dans [97].

LEMME 1.1 Soient $o_1 = Ins(p_1, c_1, nd_1)$ et $o_2 = Ins(p_2, c_2, nd_2)$ deux opérations concurrentes générées après la même séquence d'opérations S : si $p_1 < p_2$ alors $nd_1 \leq nd_2$. ■

THÉORÈME 1.1 Soient $o_1 = Ins(p_1, c_1, nd_1)$ et $o_2 = Ins(p_2, c_2, nd_2)$ deux opérations concurrentes générées après deux séquences différentes mais équivalentes S_1 et S_2. Les assertions suivantes sont vraies : (i) si $p_1 = p_2$ alors $nd_1 = nd_2$; (ii) si $p_1 < p_2$ alors $nd_1 \leq nd_2$. ■

Plus concrètement, les relations entre les positions d'insertion et *nd* établies ci-dessus assurent une certaine cohérence à la procédure de calcul de ND donnée à la définition 1.2. En effet, toutes les opérations d'insertion avec la même position, générées sur le même état sur différents sites, auront le même paramètre *nd*, même si cet état a été atteint par des séquences différentes mais équivalentes. En outre, si la position d'insertion p_1 d'une opération o_1 est sur la gauche (ou avant) de la position d'insertion p_2 d'une autre opération o_2, alors le nombre de symboles effacés avant p_1 est inférieur ou égal au nombre de symboles supprimés avant p_2 (c-à-d $nd_1 \leq nd_2$). Notons que $nd_1 = nd_2$ dans le cas où il n'y a pas de symbole supprimé entre p_1 et p_2 , y compris la position p_1. Ce paramètre supplémentaire semble donc être approprié pour gérer les opérations en conflit.

Le théorème 1.1 est très important dans le sens où il nous permet d'éviter de recalculer le paramètre *nd* d'une opération d'insertion dans les sites distants. Il suffit de calculer ce paramètre lors de la génération de l'opération d'insertion et ensuite de l'inclure dans l'opération à diffuser vers tous les autres sites.

Vérification de la consistance. Pour prouver la consistance, il suffit de montrer que notre fonction satisfait à la fois les propriétés **TP1** et **TP2**. Pour ce faire, nous avons utilisé une technique de model-checking symbolique, où une abstraction de l'objet linéaire est définie et les opérations primitives (Ins et Del) sont manipulés symboliquement par l'intermédiaire des "*Difference Bound Matrices*" (DBMs). Dans notre contexte, DBM sont utilisées pour coder les ensembles de contraintes de la forme $p_i - p_j \leq c$, où p_i, p_j sont des variables entières et c est un nombre entier constant. Du point de vue pratique, une DBM est une matrice carrée P indexé par les variables. Chaque entrée P_{ij} représente la contrainte atomique $p_i - p_j \leq P_{ij}$. S'il n'y a pas de limite supérieure sur $p_i - p_j$ avec $i \neq j$, P_{ij} est fixée à ∞. L'entrée P_{ii} est fixée à 0. Les contraintes $p_i - p_j = c$ et $p_i - p_j \geq c$ sont considérées comme des abréviations des contraintes atomiques. Dans ce qui suit, nous utilisons de manière interchangeable les contraintes atomiques et leurs abréviations.

Le modèle représenté à la figure 1.12 est utilisé pour vérifier si notre fonction IT est consistante. Le modèle commence par sélectionner les types de 3 mises-à-jour des opérations o_1, o_2 et o_3 et l'initialisation des DBMs de leurs paramètres (les DBMs initiales ne contiennent pas de

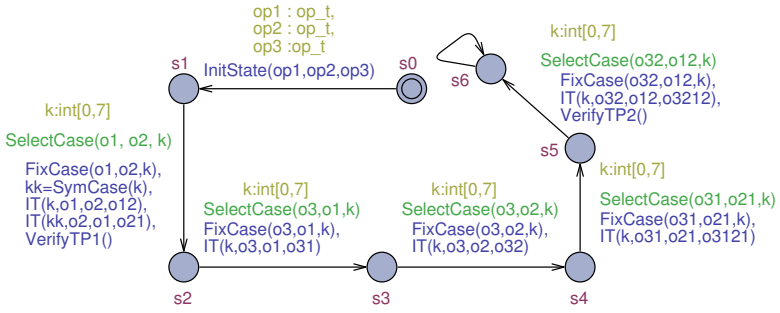


FIGURE 1.12 – Automate utilisé pour vérifier **TP1** et **TP2** [97].

contraintes). Tel est le rôle de la transition (s_0, s_1). Ensuite, le modèle sélectionne successivement pour chaque transformation opérationnelle, un cas de transformation possible parmi les 8 cas indiqués dans la table 1.2, applique le cas de transformation, et vérifie **TP1** ou **TP2** dès que les transformations nécessaires sont calculées.

A titre d'exemple, la transition (s_1, s_2) sélectionne un cas de transformation k pour o_1 par rapport à o_2 ($SelectCase(o_1, o_2, k)$), fixe le cas sélectionné ($FixCase(o_1, o_2, k)$) et l'applique pour transformer o_1 par rapport à o_2. Le cas symétrique $kk = SymCase(k)$ est utilisé pour transformer o_2 à l'égard de o_1. Après ces transformations ($IT(k, o_1, o_2, o_{12})$ et $IT(kk, o_2, o_1, o_{21})$), la propriété **TP1** est vérifiée. Concrètement, la fonction $FixCase(o_i, o_j, k)$ ajoute aux DBMs les contraintes correspondant au cas de transformation k de o_i par rapport à o_j. La fonction $IT(k, o_i, o_j, o_{ij})$ applique le cas de transformation k pour transformer o_i à l'égard de o_j et renvoie le résultat de la transformation en o_{ij}. Les fonctions $verifyTP1()$ et $verifyTP2()$ affectent respectivement dans les variables booléennes $TP1$ et $TP2$ les résultats de la vérification des propriétés **TP1** et **TP2**.

Supposons que trois opérations d'insertion sont choisies par la transition (s_0, s_1). Dans ce cas, trois DBMs P, Nd et C sur $\{p_1, p_2, p_3\}$, $\{nd_1, nd_2, nd_3\}$ et $\{c_1, c_2, c_3\}$ sont créées. Le domaine initial de chaque DBM est $\mathbb{N}^3$, $\mathbb{N}$ étant l'ensemble des entiers naturels. Supposons maintenant que pour la transformation opérationnelle de o_1 par rapport à o_2, le k sélectionné est 2. Le cas de transformation opérationnelle de o_2 à l'égard de o_1 est alors $kk = SymCase(2) = 3$. Dans ce cas, $FixCase(o_1, o_2, k)$ ajoute les ensembles de contraintes $\{p_1 = p_2\}$, $\{nd_1 = nd_2\}$ et $\{c_1 < c_2\}$ à P, Nd et C, respectivement (voir la table 1.2). Les transformations opérationnelles de o_1 à l'égard de o_2 et o_2 à l'égard de o_1 (c-à-d $IT(k, o_1, o_2, o_{12}), IT(kk, o_2, o_1, o_{21})$) résultent en deux opérations d'insertion o_{12} et o_{21}, ajoutent à P, Nd et C les ensembles de contraintes $\{p_{12} = p_1, p_{21} = p_2 + 1$, $\{nd_{12} = nd_1, nd_{21} = nd_2\}$ et $\{c_{12} = c_1, c_{21} = c_2\}$, respectivement. Enfin, $verifyTP1()$ indique que la propriété **TP1** est satisfaite. Par la suite, la même procédure est répétée pour le calcul des transformations opérationnelles nécessaires pour vérifier **TP2** (à savoir, $o_{3121} = o_{3212}$).

Nous avons utilisé l'outil UPPAAL [70] pour vérifier si notre modèle satisfait les propriétés de sûreté $AG\ TP1$ et $AG\ TP2$. L'outil UPPAAL indique que la première propriété est satisfaite (à savoir, notre fonction vérifie **TP1**). Cependant, il conclut que la seconde propriété n'est pas satisfaite. Les seules contre-exemples fournis sont des scénarios où il y a au moins deux opérations d'insertion $o_1 = Ins(p_1, c_1, nd_1)$ et $o_2 = Ins(p_2, c_2, nd_2)$ avec les relations suivantes : $[(p_1 \leq p_2) \wedge (nd_1 > nd_2)]$ ou $[(p_1 \geq p_2) \wedge (nd_1 < nd_2)]$. Selon le lemme 1.1 et le théorème 1.1 de tels scénarios sont infaisables. Lorsque l'on exclut ces scénarios inaccessibles, UPPAAL conclut que la propriété **TP2** est satisfaite. Par conséquent, notre fonction de transformation est consistante.

TABLE 1.2 – Cas de transformation pour $IT(o_1, o_2)$ et $IT(o_2, o_1)$[97].

k	Cas possibles pour $IT(o_1, o_2)$ et $IT(o_2, o_1)$
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	$(o_1 = Del(p_1) \vee o_2 = Del(p_2)) \wedge p_1 = p_2$

1.4 Annulation des opérations

1.4.1 Principe

La possibilité d'annuler des opérations effectuées par un utilisateur est une fonctionnalité très utile qui permet, par exemple, de défaire l'effet des modifications erronées. Plus précisément, en utilisant le mécanisme d'*annulation sélective* (c-à-d, choisir une opération à défaire par plusieurs opérations), il est possible de restaurer un état convergent précédent sans être obligé de refaire tout le travail effectué sur un objet partagé. Ce mécanisme consiste à réarranger les opérations dans l'histoire en utilisant l'approche TO. Ainsi, il est primordial de stocker toutes les opérations exécutées dans une histoire pour accomplir un processus d'annulation. En outre, toutes les opérations doivent être annulables (ou inversibles). Pour cela, nous supposons que chaque opération op a un inverse noté $\overline{op}$. Comme proposé dans [95, 119], pour annuler sélectivement l'opération op_i à partir d'une histoire donnée $L = [op_1; op_2; \ldots; op_i; \ldots; op_n]$, nous procédons par les étapes suivantes, comme illustré dans la figure 1.13 :

(1) Localiser op_i dans L et la marquer comme opération annulée : op_i^* ;
(2) Générer $\overline{op_i}$;
(3) Calculer $\overline{op}' = IT^*(\overline{op_i}, [op_{i+1}; \ldots; op_n])$ pour intégrer l'effet de la séquence $[op_{i+1}; \ldots; op_n]$ dans $\overline{op_i}$
(4) Exclure [5] l'effet de op_i en incluant l'effet de $\overline{op_i}$ dans la séquence $[op_{i+1}; \ldots; op_n]$ (*c-à-d* la séquence suivant op_i^*). Le résultat est donc $[op'_{i+1}; \ldots; op'_n]$;
(5) Exécuter $\overline{op}'$.

Enfin, la séquence $L \bullet \overline{op_i}'$ doit être équivalente à L' pour que le processus d'annulation soit correct.

Propriétés d'annulation. Pour formaliser la correction du processus d'annulation basé sur la transformation [95, 119, 123, 42], trois propriétés **IP1**, **IP2** et **IP3**, ont été proposées dans la littérature. Dans ce qui suit, nous considérons des fonctions de transformation IT consistantes (ou correctes), c-à-d qui vérifient **TP1** et **TP2**.

5. L'exclusion de l'effet de op_i à partir de la séquence $[op_{i+1}; \ldots; op_n]$ peut se faire comme suit :
$op \leftarrow \overline{op_i}$
Pour j de $i+1$ à n **faire**
$op'_j \leftarrow IT(op_j, op)$
$op \leftarrow IT(op, op_j)$
Fin Pour

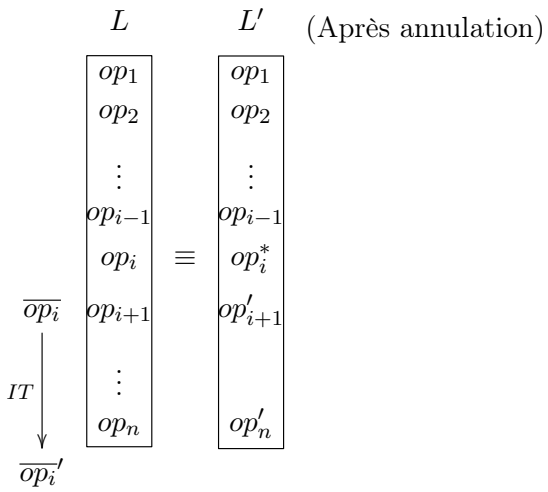


FIGURE 1.13 – Principe d'annulation.

DÉFINITION 1.3 (PROPRIÉTÉ (**IP1**)) Etant données une opération op et son inverse $\overline{op}$, alors : $[op; \overline{op}] \equiv \emptyset$. ■

La propriété **IP1** signifie que la séquence $[op; \overline{op}]$ a un effet nul sur l'état de l'objet.

DÉFINITION 1.4 (PROPRIÉTÉ (**IP2**)) Etant donnée une fonction de transformation IT. Pour toutes opérations op_1 et op_2 : $IT^*(op_1, [op_2; \overline{op_2}]) = op_1$. ■

Comme la séquence $[op_2; \overline{op_2}]$ n'a pas d'effet, la propriété **IP2** signifie que transformer op_1 contre op_2 et son inverse $\overline{op_2}$ doit produire la même opération (c-à-d, op_1).

DÉFINITION 1.5 (PROPRIÉTÉ (**IP3**)) Etant donnée une fonction de transformation IT. Pour toutes opérations op_1 et op_2 produisant les deux séquences équivalentes $[op_1; op'_2] \equiv [op_2; op'_1]$, avec $op'_1 = IT(op_1, op_2)$ et $op'_2 = IT(op_2, op_1)$, nous avons : $IT(\overline{op_1}, op'_2) = \overline{IT(op_1, op_2)}$. ■

La propriété **IP3** signifie que l'annulation de op_1 dans $[op_1; op'_2]$ doit être la même que l'annulation de sa forme transformée op'_1 dans $[op_2; op'_1]$.

1.4.2 Problématique

La violation de l'une des trois propriétés précédentes conduit à des situations de divergence, appelées communément des *puzzles*. Cela est dû au fait que, même si les fonctions de transformation considérées sont correctes (*c-à-d*, elles satisfont les propriétés **TP1** et **TP2**) elles ne sont pas suffisantes pour préserver la consistance de l'objet partagé lors de l'annulation des opérations. L'identification des puzzles est une tâche majeure qui contribuera à la conception de solutions d'annulation correctes. Tous les puzzles connus dans la littérature sont dus à la violation de **IP2** ou **IP3** par des fonctions de transformation. Pour mieux illustrer les propriétés d'annulation, nous présentons les exemples suivants :

EXEMPLE 1.1 Considérons un registre binaire partagé ayant deux opérations primitives pour modifier l'état d'un bit de 0 à 1 (et vice versa) : (i) *Up* pour activer le registre ; (ii) *Down* pour désactiver le registre. Nous pouvons proposer une fonction de transformation correcte IT_{bin}

(vérifiant **TP1** et **TP2**) pour ce registre, à savoir : $IT_{bin}(Up, x) = Up$ pour toute opération x, $IT_{bin}(Down, Up) = Up$ et $IT_{bin}(Down, Down) = Down$.

Si nous considérons que Up est l'inverse de $Down$ (et inversement), la propriété **IP1** n'est donc pas vérifiée car $Do^*([Down; Up], 0) \neq 0$. En utilisant IT_{bin}, la propriété **IP2** n'est pas aussi satisfaite comme $IT^*_{bin}(Down, [Down, Up]) = Up \neq Down$. Pour illustrer la violation de **IP3**, nous présentons dans la figure1.14 comment annuler l'opération op_1. Initialement, deux utilisateurs partagent un registre ayant la valeur 0. Les deux sites modifient simultanément leurs copies et échangent les opérations $op_1 = Up$ et $op_2 = Down$ pour converger vers l'état final 1. Supposons maintenant que l'opération op_1 est annulée dans les deux sites. Sur le site 1, l'annulation de op_1 se fait comme suit : (i) générer l'inverse de op_1, $\overline{op_1} = Down$; (ii) transformer $\overline{op_1}$ par rapport à op'_2 pour donner $IT(Up, Down) = Up$. Ainsi, l'état final après avoir annulé op_1 est 1 au niveau du site 1. Cependant, sur le site 2, l'exécution de $\overline{op'_1} = Down$ sur l'état 1 produit 0. Par conséquent, les deux copies divergent en raison de la violation de **IP3** puisque $IT(\overline{op_1}, IT(op_2, op_1)) \neq \overline{IT(op_1, op_2)}$. ■

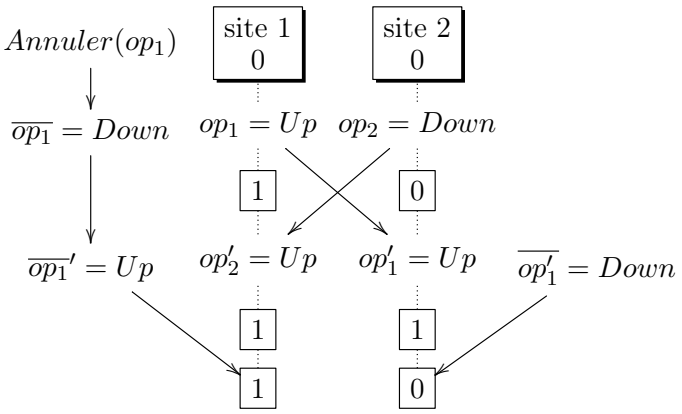


FIGURE 1.14 – Puzzle montrant la violation de **IP3**.

EXEMPLE 1.2 Considérons un registre d'entier modifié par deux opérations Inc et Dec qui incrémente et décrémente respectivement l'état du registre telle que l'une est l'inverse de l'autre. Une fonction de transformation correcte est définie comme $IT(op_i, op_j) = op_i$ pour toutes les opérations op_i, op_j dans $\{Inc, Dec\}$. Notons que l'opération Inc commute avec Dec. Il est clair que la propriété **IP1** est satisfaite puisque nous avons $[Inc; Dec] \equiv [Dec; Inc] \equiv \emptyset$.

Par ailleurs, la propriété **IP2** est satisfaite. En effet, $IT^*(op_1, [op_2; \overline{op_2}]) = op_1$ pour toutes les opérations op_1 et op_2. D'autre part, $IT(\overline{op_1}, IT(op_2, op_1)) = IT(\overline{op_1}, op_2) = \overline{op_1}$. Puisque $\overline{IT(op_1, op_2)} = \overline{op_1}$, on en déduit que $IT(\overline{op_1}, IT(op_2, op_1)) = \overline{IT(op_1, op_2)}$ pour toutes les opérations op_1, op_2 dans $\{Inc(), Dec()\}$. Par conséquent, **IP3** est vérifiée. Dans la figure 1.15, nous illustrons comment **IP3** est conservée lors de l'annulation de l'opération Inc générée simultanément à Dec. ■

Par conséquent, il est facile de montrer par des contre-exemples que l'annulation n'est pas toujours correcte, même si la fonction de transformation est consistante. La question qui se pose ici est de savoir comment définir une fonction de transformation consistante qui satisfait les propriétés d'annulation. Pour répondre à cette question, nous proposons, dans ce qui suit, de

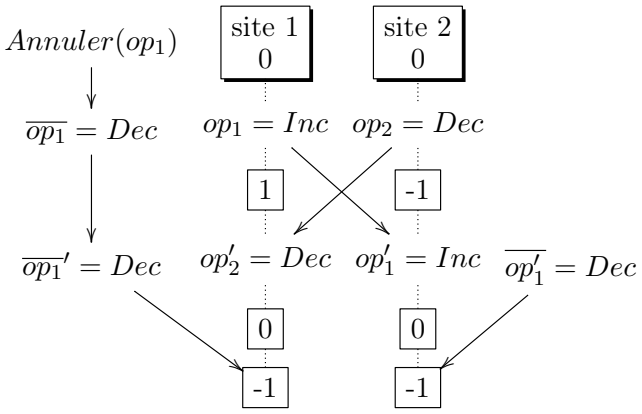


FIGURE 1.15 – Préservation de **IP3**.

formaliser le problème d'annulation comme un problème de satisfaction de contraintes (CSP) [6].

1.4.3 Modélisation basée sur CSP

Le problème de l'annulabilité consiste à étudier l'existence de fonctions de transformation consistantes satisfaisant les propriétés d'annulation. Dans cette section, nous donnons d'abord une définition formelle des objets collaboratifs, et ensuite nous formulons notre problème d'annulation comme un CSP.

Object collaboratifs. Nous supposons qu'il y a plusieurs sites (ou utilisateurs) qui collaborent sur le même objet partagé qui est répliqué sur chaque site. Chaque site met à jour sa copie locale, puis diffuse ses modifications à d'autres sites. Avant qu'elles ne soient exécutées localement, des mises à jour distantes sont transformées (moyennant une fonction de transformation) par rapport aux opérations concurrentes contenues dans l'histoire locale du site récepteur afin d'intégrer leurs effets. Nous définissons formellement un objet collaboratif comme suit :

DÉFINITION 1.6 (OBJET COLLABORATIF CONSISTANT) *Objet Collaboratif Consistant* (OCC) est un triplet $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ avec :

- $\mathcal{S}t$ est un ensemble dénombrable d'*états* (ou l'espace d'états) de l'objet.
- $\mathcal{O}p$ est un ensemble dénombrable d'*opérations primitives* exécutées par l'utilisateur pour *modifier* l'état de l'objet, telle que chaque opération op a un *inverse unique* et *distinct* $\overline{op}$ dans $\mathcal{O}p$ de telle sorte que la séquence $[op; \overline{op}]$ a un effet nul.
- $IT : \mathcal{O}p \times \mathcal{O}p \to \mathcal{O}p$ est une fonction de transformation consistante (c-à-d, IT satisfait les propriétés **TP1** et **TP2**).

Un OCC est d'ordre n, noté n-OCC, si la taille de $\mathcal{O}p$ est n. ■

D'après la définition 1.6, un OCC n'a pas d'opérations nulles (*c-à-d*, il n'y a pas op dans $\mathcal{O}p$ tel que $Do(op, st) = st$ pour tout état st). En outre, chaque opération a un inverse unique et distinct (à savoir $op \neq \overline{op}$) et doit satisfaire la propriété **IP1** (voir la définition 1.3). Nous excluons également les opérations ayant le même inverse (c-à-d, $op = \overline{op}$), car elles ne sont pas intéressantes en pratique. Comme chaque opération a son inverse, la taille d'un OCC est toujours paire.

6. En anglais, Constraint Satisfaction Problem.

Notons que l'exemple 1.1 n'est pas un OCC puisque **IP1** n'est pas satisfaite alors qu'il est facile de montrer que l'exemple 1.2 est un OCC.

Etant donnée que **IP1** est supposée vraie par définition, un objet collaboratif consistant est dit annulable si sa fonction de transformation vérifie les propriétés d'annulation **IP2** et **IP3**.

Définition 1.7 (Annulabilité) Un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ est *annulable* ssi sa fonction de transformation IT satisfait les propriétés **IP2** et **IP3**. ■

Dans la suite, tous les objets utilisés sont des objets collaboratifs consistants (voir la définition 1.6).

Définition du CSP. Étant donné un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, notre problème d'annulabilité consiste à trouver toutes les fonctions de transformation satisfaisant les propriétés d'annulation. Toutefois, cette tâche se révèle être un problème combinatoire. C'est pourquoi, nous proposons de formaliser le problème d'annulabilité comme un CSP. En effet, les CSPs [131] sont des problèmes mathématiques définis comme un ensemble d'objets dont l'état doit satisfaire un certain nombre de contraintes. Les CSPs sont résolus dans un délai raisonnable grâce à la combinaison d'heuristiques et méthodes de recherche combinatoires. Formellement, un CSP est défini comme suit :

Définition 1.8 (CSP) Un CSP est défini comme un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, où :

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ est un ensemble des variables du problème ;
- $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ est un ensemble des domaines (valeurs) pour chaque variable.
- $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ est un ensemble de contraintes. Les contraintes peuvent être définies comme (i) contraintes arithmétiques telles que $=, \neq, <, \leq, >, \geq$; (ii) contraintes logiques telles que la disjonction, l'implication, etc.

Une évaluation des variables est une fonction allant de l'ensemble des variables à l'ensemble des domaines. Une solution est une évaluation qui satisfait toutes les contraintes à partir de $\mathcal{C}$. ■

Le problème d'annulabilité peut être représenté par une matrice carrée où les lignes et les colonnes sont les opérations et leur intersection contient le résultat de la transformation. Dans ce qui suit, nous donnons les ingrédients de notre modèle de CSP.

1. L'ensemble des variables. $\mathcal{X}$ est l'ensemble des différentes valeurs possibles prises par les opérations à transformer. Formellement, étant donné un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ tels que $\mathcal{O}p = \{op_1, \ldots, op_n\}$, alors $\mathcal{X} = \{IT(op_i, op_j) \mid op_i, op_j \in \mathcal{O}p\}$. Par exemple, si $\mathcal{O}p = \{op_1, op_2\}$, alors $\mathcal{X} = \{IT(op_1, op_1), IT(op_1, op_2), IT(op_2, op_1), IT(op_2, op_2)\}$. Par conséquent, un n-OCC a un ensemble de variables $\mathcal{X}$ de taille n^2.

2. Le domaine. L'ensemble des valeurs de chacune des variables (*c-à-d* le résultat de la transformation produite par IT pour un couple d'opérations) constitue le domaine. Dans notre cas, nous avons $\mathcal{D} = \mathcal{O}p$. Pour simplifier notre modèle, nous considérons $\mathbb{N}$ comme le domaine des opérations. Nous représentons la fonction de transformation IT de n-OCC par une matrice carrée de taille n^2 de telle sorte que les opérations correspondent aux indices des lignes et des colonnes. L'intersection de la ligne i avec la colonne j est l'évaluation de la fonction de transformation $IT(i, j)$. Cette représentation a n^{n^2} différentes affectations possibles dans l'espace de recherche qui est trop grand.

3. L'ensemble des contraintes. Les contraintes sont les conditions qui doivent être satisfaites par la fonction IT afin qu'une évaluation soit réduite à vrai. Ainsi, les contraintes sont **TP2**, **IP2**

et **IP3**. Nous excluons les propriétés **TP1** et **IP1** comme elles expriment une équivalence de séquences qui ne peut pas être exprimée par des relations mathématiques entre les différentes variables de $\mathcal{X}$. Pour simplifier notre problème CSP, ces deux dernières propriétés seront donc supposées satisfaites.

Pratiquement, pour notre problème CSP, il est possible de trouver des solutions inutiles tout en vérifiant la convergence et les propriétés d'annulation. Par exemple, une fonction de transformation correcte peut simplement annuler l'effet de l'opération distance ou celle de l'opération locale contre laquelle elle se transforme, comme nous allons le détailler dans l'exemple 1.3. Par conséquent, après la synchronisation de toutes les opérations, tous les utilisateurs vont perdre leurs mises à jour qui est loin d'être l'objectif de l'approche TO. Pour illustrer une telle situation, considérons l'exemple suivant :

Exemple 1.3 Soient op_1 et op_2 deux opérations sur des matrices carrées $M_{2,2}$ d'ordre 2, telles que :

$$\begin{array}{rccc} op_1: & M_{2,2} & \rightarrow & M_{2,2} \\ & A & \mapsto & 2 \times A^t \end{array} \qquad \begin{array}{rccc} op_2: & M_{2,2} & \rightarrow & M_{2,2} \\ & A & \mapsto & 2 \times \begin{pmatrix} a_{0,1} & a_{0,0} \\ a_{1,0} & a_{1,1} \end{pmatrix} \end{array}$$

où A^t est la transposée de la matrice A. Considérons l'ensemble des opérations $\mathcal{O}p = \{op_1, op_2, op_3, op_4\}$ où op_3 et op_4 sont les inverses, respectivement, de op_2 et op_1 (et vice versa). Pour toute opération op dans $\mathcal{O}p$, une fonction de transformation consistante peut être définie sur $\mathcal{O}p$ comme suit : (i) $IT(op_1, op) = op_2$; (ii) $IT(op_2, op) = op_1$; (iii) $IT(op_3, op) = op_4$; et (iv) $IT(op_4, op) = op_3$; . Selon la propriété **TP1**, les équivalences suivantes doivent être satisfaites :

$$[op_2; op_2] \equiv [op_1; op_1] \tag{1.1}$$
$$[\overline{op_2}; op_1] \equiv [op_2; \overline{op_1}] \tag{1.2}$$
$$[\overline{op_1}; op_2] \equiv [op_1; \overline{op_2}] \tag{1.3}$$
$$[\overline{op_1}; \overline{op_1}] \equiv [\overline{op_2}; \overline{op_2}] \tag{1.4}$$

La fonction IT ci-dessus satisfait chacune de ces équivalences. Néanmoins, une telle transformation n'est pas pratique, car elle annule simplement l'effet de l'opération effectuée lors de la réception d'une opération distante. Par exemple $IT(op_3, op_2) = op_4 = \overline{op_2}$. ■

Par conséquent, nous proposons d'améliorer notre modèle de CSP avec les contraintes **C1** (voir la définition 1.9) et **C2** (voir la définition 1.10) afin d'éviter des évaluations indésirables de IT qui cachent l'avantage de l'approche TO (c-à-d inclure l'effet des opérations concurrentes).

La propriété **C1** interdit la transformation d'une opération en son inverse :

Définition 1.9 (Propriété **C1**) Soit un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$. Pour toutes opérations op_i et op_j de $\mathcal{O}p$, $IT(op_i, op_j) \neq \overline{op_i}$. ■

En ce qui concerne la propriété **C2**, elle écarte toutes les fonctions IT qui transforment une opération distante en l'inverse d'une opération locale.

Définition 1.10 (Propriété **C2**) Soit un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$. Pour toutes opérations op_i et op_j de $\mathcal{O}p$, si $op_j \neq \overline{op_i}$ alors $IT(op_i, op_j) \neq \overline{op_j}$. ■

En conséquence, l'ensemble final des contraintes est $\mathcal{C} = \{\mathbf{TP2}, \mathbf{IP2}, \mathbf{IP3}, \mathbf{C1}, \mathbf{C2}\}$.

1.4.4 Analyse des modèles de transformation

Pour obtenir tous les résultats expérimentaux du problème d'annulabilité (c-à-d, calculer toutes les évaluations de IT par rapport à notre modèle de CSP) dans un délai raisonnable, nous avons mis en place un prototype java basé sur le solveur Choco [16]. Ce solveur est une bibliothèque (open source) java, qui permet de décrire des problèmes combinatoires sous la forme de problèmes de satisfaction de contraintes et les résout avec des techniques de programmation par contraintes.

Comme nous représentons la fonction de transformation par une matrice carrée, il est possible d'avoir des solutions symétriques. Pour fournir un nombre minimal et utile de solutions, nous avons implémenté un module qui élimine toutes les solutions symétriques.

Dans cette section, nous présentons comment devrait être la fonction de transformation de telle sorte que la propriété de l'annulabilité soit correctement gérée. En particulier, nous étudions si la commutativité des opérations est nécessaire et suffisante pour l'annulabilité. Notre question découle de l'observation faite à partir des exemples 1.2 et 1.1 : le registre entier est annulable et ses opérations sont commutatives, mais le registre binaire ne l'est pas et ses opérations ne commutent pas. Les preuves des lemmes et théorèmes énoncés ci-dessous sont données dans [17, 18].

La propriété de commutativité. Nous définissons formellement la commutativité comme suit :

Définition 1.11 (Commutativité) Deux opérations op_1 et op_2 commutent ssi $[op_1; op_2] \equiv [op_2; op_1]$. ■

Dans ce qui suit, nous disons qu'un ensemble d'opérations $\mathcal{O}p$ est commutative si toutes ses opérations sont commutatives deux à deux.

Théorème 1.2 Etant donné un OCC $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$. Pour toutes opérations op_1, op_2 dans $\mathcal{O}p$, op_1 commute avec op_2 ssi $IT(op_1, op_2) = op_1$. ■

Une question subsidiaire peut être posée : comment définir une fonction de transformation de telle sorte que l'objet collaboratif est annulable et si la commutativité est nécessaire ou non pour obtenir la propriété d'annulabilité ?

Tout d'abord, nous montrons que la commutativité est suffisante pour l'annulabilité. En d'autres termes, nous montrons que pour tout objet $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, si $IT(op_i, op_j) = op_i$ pour toutes les opérations op_i et op_j de $\mathcal{O}p$, alors C est annulable (voir le lemme 1.2).

Lemme 1.2 (La commutativité implique l'annulabilité) Etant donné un objet $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, si $\mathcal{O}p$ commutative alors C annulable. ■

Dans ce qui suit, nous discutons des solutions fournies par notre prototype pour des OCC d'ordres 2, 4 et 6. Ensuite, nous allons vérifier si leurs opérations commutent ou non.

OCC d'ordre 2. Pour discuter des évaluations correctes de la fonction IT dans le cas d'un 2-OCC, nous considérons $\mathcal{O}p = \{op_1, op_2\}$ telles que $\overline{op_1} = op_2$. Lors de l'exécution de l'ensemble des contraintes, une seule solution a été fournie par notre solveur (voir la figure 1.16).

Cette sortie montre qu'un annulable OCC d'ordre 2 nécessite une fonction de transformation de vérifier $IT(op_i, op_j) = op_i$ pour toutes les opérations op_i et op_j de $\mathcal{O}p$ (c-à-d, chaque opération est transformée en elle-même). Ainsi $\mathcal{O}p$ commute comme indiqué par le théorème 1.2. En conséquence, la commutativité est nécessaire pour annuler correctement les opérations concurrentes dans le cas de 2-OCC.

IT	op_1	op_2
op_1	op_1	op_1
op_2	op_2	op_2

FIGURE 1.16 – Sortie pour 2-OCC.

OCC d'ordre 4. La figure 1.17 montre la sortie de notre solveur dans le cas de 4-OCC. Pour cette expérience, nous avons considéré un ensemble de quatre opérations $\mathcal{O}p = \{op_1, op_2, op_3, op_4\}$ telles que $\overline{op_1} = op_4$ et $\overline{op_2} = op_3$ (et vice versa). Similairement à 2-OCCO, la commutativité est nécessaire pour atteindre l'annulabilité puisque chaque opération est transformée en elle-même (voir le théorème 1.2).

IT	op_1	op_2	op_3	op_4
op_1	op_1	op_1	op_1	op_1
op_2	op_2	op_2	op_2	op_2
op_3	op_3	op_3	op_3	op_3
op_4	op_4	op_4	op_4	op_4

FIGURE 1.17 – Sortie pour 4-OCC.

OCC d'ordre 6. Nous discutons ici les fonctions de transformation fournis par notre solveur pour un OCC d'ordre 6. Nous avons considéré $\mathcal{O}p = \{op_1, op_2, op_3, op_4, op_5, op_6\}$ telles que $\overline{op_1} = op_6$, $\overline{op_2} = op_5$ et $\overline{op_3} = op_4$. La figure 1.18 montre que trois solutions sont possibles pour atteindre l'annulabilité.

(1) Solution 1

IT	op_1	op_2	op_3	op_4	op_5	op_6
op_1	op_1	op_4	op_1	op_1	op_4	op_1
op_2	op_2	op_2	op_2	op_2	op_2	op_2
op_3	op_3	op_6	op_3	op_3	op_6	op_3
op_4	op_4	op_1	op_4	op_4	op_1	op_4
op_5	op_5	op_5	op_5	op_5	op_5	op_5
op_6	op_6	op_3	op_6	op_6	op_3	op_6

(2) Solution 2

IT	op_1	op_2	op_3	op_4	op_5	op_6
op_1	op_1	op_4	op_1	op_1	op_3	op_1
op_2	op_2	op_2	op_2	op_2	op_2	op_2
op_3	op_3	op_1	op_3	op_3	op_6	op_3
op_4	op_4	op_6	op_4	op_4	op_1	op_4
op_5	op_5	op_5	op_5	op_5	op_5	op_5
op_6	op_6	op_3	op_6	op_6	op_4	op_6

(3) Solution 3

IT	op_1	op_2	op_3	op_4	op_5	op_6
op_1	op_1	op_1	op_1	op_1	op_1	op_1
op_2	op_2	op_2	op_2	op_2	op_2	op_2
op_3	op_3	op_3	op_3	op_3	op_3	op_3
op_4	op_4	op_4	op_4	op_4	op_4	op_4
op_5	op_5	op_5	op_5	op_5	op_5	op_5
op_6	op_6	op_6	op_6	op_6	op_6	op_6

FIGURE 1.18 – Sortie pour 6-OCC.

Ainsi, un annulable 6-OCC n'est pas nécessairement commutatif. En effet, parmi les trois solutions proposées par notre solveur seule la dernière commute selon le théorème 1.2. Cependant, une observation très importante qui peut être faite est : 4 opérations de l'ensemble des opérations sont transformées au moins 4 fois en elles-mêmes et 2 autres sont toujours transformés en elles-mêmes. L'analyse des deux solutions 1 et 2 montre chaque solution est formée par un OCC commutatif d'ordre 4 et un autre OCC commutatif d'ordre 2 telle que la transformation inter-OCCs (transformation entre 4-OCC et 2-OCC) ne commute pas.

Discussion. Notre précédente étude prouve que la commutativité est étroitement liée à l'annulabilité alors que l'approche TO a été proposée pour aller au-delà de la commutativité. En effet,

les OCCs d'ordres 2 et 4 sont annulables ssi leurs ensembles d'opérations commutent comme indiqué dans le théorème suivant :

THÉORÈME 1.3 La commutativité est nécessaire et suffisante pour l'annulabilité des CCOs d'ordre $n \leq 4$.

Cependant, la commutativité est suffisante mais pas nécessaire pour atteindre l'annulabilité dans le cas de OCC d'ordre $n \geq 6$. En effet, notre solveur fournit trois fonctions de transformation correctes qui sont annulables où une seule est commutative selon le théorème 1.2. Les deux autres ne commutent pas, mais se composent de deux sous-ensembles d'opérations commutatifs. En conséquence, un 6-OCC est formé par deux intra-commutatifs sous-OCC tels que la transformation entre les deux CCO ne commute pas. L'analyse de la sortie présentée dans la figure 1.18 montre que l'ensemble des opérations $\mathcal{O}p$ de tout annulable 6-OCC est :

1. soit commutatif, *c-à-d* $IT(op_i, op_j) = op_i$ pour toutes les opérations op_i, op_j de $\mathcal{O}p$, ou
2. soit l'union de deux sous-ensembles $\mathcal{O}p_1$ et $\mathcal{O}p_2$ tels que : $\mathcal{O}p_1$ et $\mathcal{O}p_2$ sont commutatifs de tailles 2 et 4 respectivement (*c-à-d* $IT(op_i, op_j) = op_i$, pour toutes opérations $(op_i, op_j) \in \mathcal{O}p_x \times \mathcal{O}p_x$, $x \in \{1, 2\}$). La transformation inter OCCs est résumée comme suit :
   (a) pour toute paire d'opérations $(op_i, op_j) \in \mathcal{O}p_1 \times \mathcal{O}p_2$, $IT(op_i, op_j) = op_i$
   (b) pour toute paire d'opérations $(op_i, op_j) \in \mathcal{O}p_2 \times \mathcal{O}p_1$,
      i. soit $IT(op_i, op_j) = IT(op_i, \overline{op_j})$;
      ii. ou $IT(op_i, op_j) = \overline{IT(op_i, \overline{op_j})}$

Les solutions produites par notre solveur dans le cas de 8-OCCs valident l'observation précédente et suivent les modèles trouvés ci-dessus.

En outre, nos expériences fournissent un petit nombre de solutions qui simplifie grandement l'étude du problème de l'annulabilité. En effet, notre ensemble de contraintes réduit considérablement le nombre d'évaluations correctes pour transformer des opérations. Par exemple, un 6-CCO génère normalement 6^{6^2} fonctions de transformation alors que nous obtenons seulement 3 modèles.

Pour résumer, ce travail prouve qu'il n'y a qu'un seul moyen possible de transformer des opérations pour des OCCs d'ordre 2 et 4 pour assurer qu'ils soient annulables. Cette solution unique consiste à transformer chaque opération en elle-même ; ainsi la commutativité est nécessaire et suffisante pour obtenir l'annulabilité. Dans le cas contraire, la commutativité est seulement suffisante. En outre, un annulable OCC d'ordre $n \geq 6$ est l'union de deux intra-commutatifs sous-CCO.

1.5 Etat de l'Art

Plusieurs travaux existent sur l'approche des transformées opérationnelles. Nous scindons ces travaux en trois catégories.

Verification des fonctions de transformation. Pour prouver la consistance, il suffit de montrer que la fonction de transformation satisfait à la fois les propriétés **TP1** et **TP2**. Dans [63], les auteurs ont proposé un cadre formel pour la modélisation et la vérification des fonctions de transformation en utilisant des spécifications algébriques. Pour vérifier **TP1** et **TP2**, ils ont utilisé un prouveur automatique de théorèmes. Cependant, cette approche comporte quelques lacunes : (i) le modèle du système est correct mais incomplet w.r.t. **TP1** et **TP2** (à savoir,

il ne garantit pas que la violation de ces propriétés est vraiment possible) ; (ii) il n'y a pas d'indications pour comprendre les contre-exemples (lorsque les propriétés ne sont pas vérifiées) ; (iii) elle exige une certaine interaction (par l'injection de nouveaux lemmes) pour effectuer la vérification. Dans [13], les auteurs ont contourné ces lacunes et proposé une technique de model-checking symbolique basée sur le concept des "Difference Bound Matrices" (DBMs) pour vérifier si une fonction de transformation est consistante. La vérification est effectuée automatiquement et symboliquement sans effectuer des copies différentes de l'objet partagé et d'exécuter explicitement les mises à jour. En outre, contrairement à [63], l'approche proposée dans [13] fournit des contre-exemples symboliques que l'on peut facilement dérouler.

Fonctions de transformations pour les objets linéaires. Il est bien connu que la plupart des situations de divergence sont dues à la violation de la propriété **TP2** [63]. Ceci est causé par la sémantique l'opération de suppression (*Del*). Rappelons que cette opération supprime un élément à une position donnée et diminue la longueur du document partagé. Dans [92], les auteurs ont proposé une nouvelle fonction de transformation en changeant la sémantique de l'opération de suppression. En effet, l'opération $Del(p)$ ne supprime pas l'élément à la position p, mais la rend invisible. Cette nouvelle sémantique conduit à gérer deux états distincts, appelés modèles de *vue* et *données*. Bien que leur fonction de transformation est correcte, la longueur du document s'accroit toujours (et ne diminue jamais), comme l'opération de suppression n'a aucun effet sur la longueur du document. En outre, leur solution a besoin de quelques procédures supplémentaires pour gérer les deux états distincts en raison des caractères cachés. Contrairement à leur solution, notre fonction de transformation est consistante et considère la sémantique naturelle de l'opération de suppression, comme suggéré par Ellis et *al.* dans [34], qui permet une gestion simple et efficace de la taille du document.

L'annulation des opérations. Plusieurs travaux ont proposé une fonction d'annulation pour les éditeurs de collaboratifs. La majorité de ces solutions sont basées sur l'utilisation du journal afin de stocker les opérations exécutées et de récupérer les états antérieurs. *Swap then undo* [95] a été la première méthode pour l'annulation sélective. Elle consiste à placer l'opération sélectionnée à la fin de l'histoire par permutation et exécution de son inverse. Malheureusement, cette solution ne permet pas d'annuler toute opération car il n'est pas toujours possible d'échanger des opérations dans le journal à cause de certaines opérations qui pourraient être conflictuelles. *Undo/Redo* [100] a été proposée pour résoudre le problème des conflits. Elle consiste à défaire toutes les opérations dans l'ordre chronologique inverse. Cependant, elle est coûteuse car elle nécessite d'effectuer de nombreuses étapes et ne permet pas d'annuler une opération dans tous les cas. *UNO* [133] consiste à générer une nouvelle opération ayant l'effet inverse de l'opération à annuler. Bien que la complexité de l'annulation soit linéaire, cette solution est compatible uniquement avec des applications basées sur la fonction de transformation proposée dans [92] où les caractères du document ne sont pas effectivement supprimés. Les deux travaux *ANYUNDO-X* [119] et *COT* [123] supportent la synchronisation et l'annulation dans les éditeurs collaboratifs. Cependant, ils ont tous les deux une complexité exponentielle du fait qu'ils évitent la satisfaction des propriétés d'annulation **IP2** et **IP3**. Enfin, l'algorithme *ABTU* [114] propose une solution d'annulation basée sur l'algorithme de transformation ABT [74]. Même si l'algorithme proposé a une complexité linéaire, il ne permet pas d'annuler toute opération puisque l'annulation pourrait être rejetée dans certains cas. Les auteurs supposent que leur algorithme assure la convergence. Cependant, cet algorithme diverge dans certains cas [17].

1.6 Conclusion

Dans ce chapitre, nous avons donné une vue générale du modèle des transformées opérationnelles, qui est utilisé pour sérialiser par transformation des opérations concurrentes. Chaque objet collaboratif doit posséder sa propre fonction de transformation. Pour garantir la cohérence des copies, cette fonction doit satisfaire deux propriétés **TP1** et **TP2**. Aussi, la vérification de telles propriétés (et plus particulièrement **TP2**) demeure toujours un problème ouvert pour les objets linéaires (tels que le texte, l'arbre ordonné XML, etc.).

En utilisant une méthode de synthèse de contrôleur basée sur les automates de jeu, nous avons montré l'impossibilité de trouver une fonction de transformation pour assurer la cohérence des objets linéaires altérés par de simples opérations d'insertion et de suppression. L'extension de l'opération d'insertion par une méta-donnée nous a permis de proposer une nouvelle fonction de transformation dont la propriété de convergence a été vérifiée par une technique de model-checking. En outre, nous avons étudié formellement la combinaison des approches TO et d'annulation des opérations. En modélisant cette combinaison comme un problème de satisfaction de contraintes (CSP), nous avons établi des conditions nécessaires et suffisantes pour qu'une fonction de transformation préserve la cohérence, à savoir : étant donné n le nombre d'opérations pour modifier un objet collaboratif, la commutativité est nécessaire et suffisante pour $n \leq 4$, sinon elle est seulement suffisante.

Chapitre 2

Contrôle d'Accès Optimiste pour des Systèmes Collaboratifs

Sommaire

2.1 Introduction

L'importance des systèmes collaboratifs a considérablement augmenté au cours des dernières années. La majorité de nouvelles applications sont conçues de manière distribué pour répondre aux besoins du travail collaboratif. Ainsi, pour avoir une disponibilité permanente des données et un temps de réponse court, les objets partagés (tels que les documents textes, les systèmes de fichiers, etc.) sont répliquées sur chaque site du collaborateur. Cette réplication nécessite des procédures de synchronisation pour le maintien de cohérence des différentes copies.

La sécurité des données partagées joue un rôle croissant dans la confiance accordée aux systèmes collaboratifs par les utilisateurs. A ce titre, un des problèmes qui constitue un véritable défi est comment établir un équilibre entre la disponibilité (par réplication) et le contrôle d'accès aux données partagées. En effet, les interactions dans un groupe de travail visent à rendre les objets partagés disponibles à tous les membres de ce groupe, alors que le contrôle d'accès permet de restreindre cette disponibilité à certains membres. Comme les données partagées sont répliquées,

les systèmes collaboratifs requièrent un temps court pour les mises à jour des copies. Cependant, avaliser chaque mise à jour locale par une autorisation émanant d'un site distant (hébergé sur un serveur) va certainement dégrader les performances du système. Il serait d'ailleurs inapproprié de garder un contrôle centralisé des accès.

Pour minimiser les temps d'attentes dus à la vérification des droits d'accès, nous proposons de répliquer la politique d'accès. Ainsi, chaque utilisateur maintient localement deux copies : l'objet partagé ainsi que sa politique contenant les règles d'autorisation. Les mises à jour de l'objet partagé et les mises à jour de la politique sont appliquées dans des ordres arbitraires sur les différents sites utilisateurs. L'absence de coordination sûre entre ces différentes mises à jour peut provoquer des trous de sécurité (à savoir, permettre des mises à jour illégales ou de rejeter les mises à jour légales). En utilisant le concept de *sécurité optimiste*, nous proposons une approche qui tolère la violation momentanée des droits d'accès mais assure la restauration des copies vers des états valides par rapport à la politique courante.

Dans ce chapitre, nous présentons un modèle de contrôle d'accès bien adapté aux systèmes collaboratifs [60]. La sécurisation de la collaboration est faite via une architecture en couche. Le protocole de contrôle d'accès est déployé au dessus du protocole de coordination. Le contexte étant celui d'un système distribué avec réplication des données partagées, le modèle de contrôle d'accès permet également la réplication de la politique de contrôle d'accés. Ainsi, en plus de la copie de l'objet partagé, les utilisateurs diposent localement d'une copie de la politique de d'accès, qu'ils aient un profil administrateur ou non. Dès lors que l'administrateur prend l'initiative d'une opération administrative, il l'envoie aux autres utilisateurs pour une mise à jour de leurs politiques locales respectives, tout en exécutant la règle d'accès correspondante sur sa propre politique. Le choix d'un modèle de contrôle d'accès distribué avec réplication de la politique de contrôle d'accès permet de bâtir une politique locale qui sert à effectuer sur place la vérification des modifications locales. Un tel choix augmente la réactivité du système car l'utilisateur non-administrateur peut exécuter une opération approuvée par sa copie de la politique.

Ce chapitre est composé comme suit : la section 2.2 présente les composants de notre modèle de coordination. Dans la section 2.3, nous décrivons les couches de notre modèle générique de contrôle d'accès ainsi que le protocole optimiste pour contrôler la concurrence sur les différentes copies de la politique. En utilisant une technique symbolique de model-checking, nous montrons dans la section 2.4 comment spécifier formellement l'empilement de notre modèle d'accès à un système collaboratif ainsi qu'analyser la préservation de la propriété de cohérence. Enfin, nous passons en revue l'état de l'art dans la section 2.5 et nous terminons le chapitre par une conclusion.

2.2 Modèle de Coordination

2.2.1 Données partagées

Objet partagé. Nous considérons les objets collaboratifs partagés par de nombreux utilisateurs afin d'effectuer une tâche commune. Les opérations modifiant l'objet partagé sont appelées *opérations coopératives*. Par exemple, un objet partagé peut être un document qui est une suite de caractères, de mots ou de paragraphes. Pratiquement, les éditeurs collaboratifs manipulent un tel document et le modifient moyennant deux opérations coopératives : (i) $Ins(p, e)$ où p est la position d'insertion et e l'élément (c-à-d caractère, mot, paragraphe, etc.) à ajouter à la position p. (ii) $Del(p, e)$ qui supprime l'élément e à la position p. Les opérations coopératives sont générées localement par chaque site, puis diffusées à d'autres collaborateurs.

Nous considérons que chaque site exécute et stocke des opérations coopératives dans un journal appelé *journal coopératif* (voir le chapitre 1). Notre objectif est d'étendre l'objet collaboratif avec une couche de sécurité pour contrôler l'accès aux différentes copies manipulées par différents collaborateurs.

Politique partagée. Nous considérons un modèle de contrôle d'accès basé sur des *politiques d'autorisation* et utilisant trois ensembles pour spécifier ces politiques, à savoir :

1. S est l'ensemble des *sujets*. Un sujet peut être un utilisateur ou un groupe d'utilisateurs.
2. O est l'ensemble des *objets*. Il peut s'agir de tout l'objet partagé, d'un ou plusieurs éléments constitutifs de cet objet partagé.
3. R est l'ensemble des *droits d'accès*. Chaque droit est associé à une opération que le sujet peut effectuer sur l'objet partagé. Par exemple, nous pouvons considérer le droit d'ajouter un nouvel élément (i), de supprimer un élément (d) et de mettre à jour un élément (u).

Une politique d'autorisation spécifie donc les opérations qu'un utilisateur peut exécuter sur un objet partagé.

Définition 2.1 (Politique) Une *politique* $P : 2^S \times 2^O \rightarrow 2^R \times \{+, -\}$ associe un ensemble de sujets et un ensemble de d'objets à un ensemble de droits signés. Le signe "+" représente l'*attribution* d'un droit et le signe "−" représente la *révocation* d'un droit. ■

Nous représentons une politique P comme une liste indexée d'autorisations. Chaque autorisation $l_i \in P$ est un quadruplet $\langle S_i, O_i, R_i, \omega_i \rangle$ où $S_i \subseteq S$, $O_i \subseteq O$, $R_i \subseteq R$ et $\omega_i \in \{-, +\}$. Par exemple, l'autorisation $l = \langle *, *, i, + \rangle$ attribue un droit positif pour insérer de nouveaux objets pour tous les utilisateurs (Le symbole $*$ désigne une quantification universelle).

Une autorisation est dite *positive* (resp. *négative*) lorsque $\omega = +$ (resp. $\omega = -$). Les autorisations négatives sont utilisées pour accélérer le processus de vérification. Nous utilisons une sémantique basée sur la première correspondance (ou en anglais "first match") : quand une opération coopérative est générée, le système vérifie sa politique d'accès en commençant à partir de la première autorisation et il s'arrête lorsqu'il trouve la première autorisation l qui correspond à l'opération coopérative. Si aucune correspondance n'est trouvée, alors l'opération est rejetée.

Nous supposons que l'utilisateur qui attribue des autorisations est en mesure d'effectuer des *opérations administratives*. Une opération administrative consiste tout simplement à modifier la politique en ajoutant ou en supprimant des autorisations.

Définition 2.2 (Opérations administratives) L'état d'une politique est représenté par un triplet $\langle P, S, O \rangle$ où P est la liste des autorisations. L'administrateur peut modifier l'état de la politique en utilisant l'ensemble suivant des *opérations administratives* :

— $AddAuth(p, l)$ pour insérer une autorisation l à la position p ;

— $DelAuth(p, l)$ pour enlever une autorisation l se trouvant à la position p ;

Une opération administrative a est dite *restrictive* ssi $a = AddAuth(p, l)$ et l est négative ou $a = DelAuth(p, l)$ et l est positive. ■

La politique peut être considérée comme un objet partagé par le groupe d'utilisateurs. Elle est donc *répliquée* chez chaque utilisateur afin d'éviter des latences élevées dues à la vérification de la validité d'une opération coopérative par rapport à la politique. La réplication vise à améliorer la disponibilité de l'objet partagé ainsi que sa politique.

2.2.2 Modèle de Collaboration

Nous considérons le modèle suivant de collaboration :

1. Lorsqu'un utilisateur génère une opération coopérative sur la copie locale de l'objet partagé, cette opération sera validée ou rejetée en la contrôlant uniquement par rapport à la copie locale de l'objet de la politique.
2. Une fois accordées et exécutées, les opérations coopératives sont ensuite diffusées à d'autres utilisateurs qui doivent à leur tour vérifier si ces opérations distantes sont autorisées par la politique locale avant de les exécuter.
3. Lorsqu'un administrateur modifie sa copie de la politique locale en ajoutant ou supprimant des autorisations, il envoie ses modifications aux autres utilisateurs afin de mettre à jour les autres copies de la politique.

Nous supposons que les messages sont envoyés via un réseau de communication sécurisé et fiable, et les utilisateurs sont identifiés et authentifiés afin d'associer correctement l'accès à ces utilisateurs.

Un autre aspect important de la politique de contrôle d'accès est de savoir si le modèle est *obligatoire* ou *discrétionnaire*. En effet, deux approches sont possibles lors de la construction de la couche de sécurité basé sur le modèle présenté ci-dessus : *mono-administrateur* et *multi-administrateurs*. Ainsi, il est préférable pour notre modèle de contrôle d'accès d'être *flexible* et *facilement extensible* du mono au multi-administrateurs afin de satisfaire une grande variété d'applications. Chaque approche a des avantages et des inconvénients.

Mono-administrateur. De nombreuses applications distribuées ne nécessitent qu'un seul administrateur dans le groupe, telles que la gestion en ligne des examens et des environnements coopératives des programmation où le superviseur de l'application dispose d'une pleine autorité sur le groupe de collaboration. Cette forme de contrôle est utile pour maintenir l'attention du groupe sur le travail à fournir, en particulier lorsque l'on a des délais à respecter [75]. Dans de telles applications, l'administrateur se charge de la création et la supervision des groupes.

Le principal avantage d'un tel modèle est sa simplicité. En effet, il permet une mise à jour facile de la politique comme seul l'administrateur change les droits d'accès. Cependant, d'un point de vue de sécurité, il est plus difficile de contrôler tous les utilisateurs, particulièrement dans le cas des groupes volumineux de collaboration. En outre, l'absence du site de l'administrateur en raison de pannes matérielles ou logicielles pourrait laisser la politique dans un état statique.

Multi-administrateurs. Il existe plusieurs applications collaboratives qui nécessitent une approche *discrétionnaire* en ce sens que le propriétaire d'un objet est le seul a avoir le droit d'attribuer des autorisations à d'autres utilisateurs sur cet objet. Sinon, l'opération administrative est rejetée. Par exemple, les agendas partagés sont des applications collaboratives où une politique multi-administrateurs serait appropriée [2]. En effet, dans le contexte professionnel, une secrétaire peut avoir besoin de voir instantanément l'activité de son responsable en vue d'organiser ses réunions. Il serait plus pratique si le secrétaire était en mesure de mettre à jour les informations relatives à une réunion dans le calendrier du responsable. Il est donc intéressant de donner au responsable la possibilité d'accorder le droit de modification à la secrétaire sur tout ou une partie de son calendrier partagé.

Ainsi dans une approche discrétionnaire, chaque utilisateur u dispose d'une politique P_u sur ses propres objets, qui évolue indépendamment des autres politiques. Chaque administrateur peut donc spécifier des droits d'accès à d'autres utilisateurs sur ses objets.

Pour identifier la politique de l'administrateur d'un objet partagé o, qui est modifié par l'opération coopérative c, nous considérons l'action primitive $Administrator(c)$ qui retourne le propriétaire de l'objet o.

Même si notre modèle de contrôle d'accès est simple, nous montrerons dans la suite que l'application de la politique pose un certain nombre de problèmes subtiles.

2.3 Modèle Générique de Contrôle d'Accès

2.3.1 Architecture en couches

Nous présentons ici l'architecture générique de notre modèle de contrôle d'accès qui peut être intégrée à toute application collaborative utilisant une histoire pour stocker toutes les modifications. Pour bien illustrer notre modèle, nous définissons l'application collaborative comme une structure possédant trois couches différentes (voir la figure 2.1) : (i) la couche de coordination pour gérer les opérations locales et les opérations distantes (CL) ; (ii) la couche du contrôle d'accès (ACL) ; (iii) l'interface générique (GI) qui relie les couches CL et ACL.

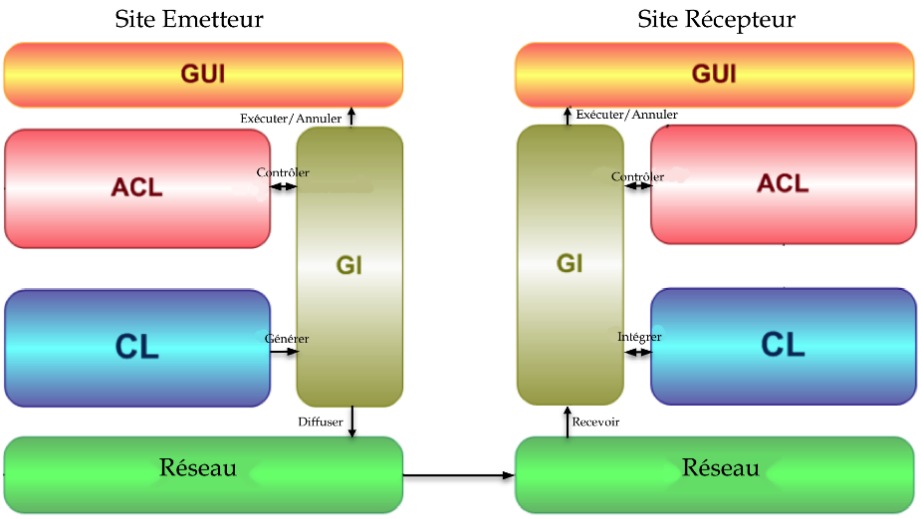


FIGURE 2.1 – Différentes couches de notre modèle de contrôle d'accès.

La couche de coordination CL (par exemple OPTIC [60] et COT [123]) représente le protocole chargé de synchroniser les mises à jour concurrentes. Cette couche est considérée comme une boîte noire et est indépendante des autres couches. Chaque site exécute d'abord les modifications générées localement, puis les diffuse aux autres utilisateurs. Le site récepteur capte des opérations distantes du réseau et les intègre localement par sa CL.

Pour construire une couche de contrôle d'accès générique (ACL) et la combiner à une CL donnée tout en préservant la convergence (même contenu des copies), nous avons besoin de bien définir l'interface générique (GI) qui relie les deux couches. Pour ce faire, nous utilisons l'ensemble suivant d'opérations qui gèrent les connexions entre les couches ACL et CL : (1) $Check(c)$ contrôle si l'opération coopérative c communiquée à GI est autorisée par ACL ; (2) $Apply(c)$ lance l'exécution de l'opération c sur l'état réel de l'objet partagé ; (3) $Undo(c)$ pour annuler toute

opération coopérative c du journal qui est en *conflit*[7] avec l'opération administrative restrictive a reçue par GI. Nous décrivons ci-dessous le flux d'interactions entre les trois couches :

1. Lorsqu'une opération coopérative c est reçue du réseau, elle est tout d'abord re-dirigée vers GI au lieu de la CL afin d'être vérifiée par rapport à la politique en exécutant $Check(c)$.
2. Si c est autorisée par ACL, elle est alors prise en charge par $Apply(c)$ de l'interface GI pour l'exécuter sur l'état local de l'objet partagé. Sinon, l'opération est rejetée de ACL, puis ignorée par CL.
3. Lorsqu'une opération administrative distance a est reçue, elle est communiquée à GI pour être appliquée dans ACL et modifier la copie locale de la politique.
4. S'il y a des opérations coopératives concurrentes et en conflit avec a, la couche générique GI traitera l'opération $Undo(c)$ afin d'annuler toutes les opérations coopératives c qui ne sont plus légales après la réception de a. Cette étape est cruciale car elle permet de forcer la convergence des copies en rétablissant l'état correct vu par l'administrateur au moment de la génération de l'opération administrative a.
5. Quand une opération coopérative c est localement générée, elle d'abord exécutée puis re-dirigée vers GI pour contrôler si elle est autorisée par ACL. Si elle n'est pas valide par rapport à la politique locale, elle est annulée avec $Undo(c)$ [8]. Sinon, l'effet reste valable sur l'état de l'objet partagé. Enfin, l'opération c sera diffusée à travers le réseau.

2.3.2 Protocole de Collaboration

La dynamique de la collaboration est réalisée et maintenue par les interactions entre le protocole de coordination (CL) et le protocole de contrôle d'accès (ACL) via des tâches de génération, de contrôle, de réception et d'exécution d'opérations coopératives et administratives.

Génération d'opérations administratives. Nous supposons que seul un administrateur peut générer des opérations administratives. Dès la génération, le protocole d'accès associe à l'opération administrative, un numéro de version croissant, lui permettant ainsi de maintenir une causalité entre les différentes opérations administratives. Une opération administrative et son numéro de version associé permettent de constituer une requête administrative (définition 2.3) qui sera aussitôt appliquée à la politique d'accès de l'administrateur, pour ajouter une nouvelle règle d'accès. Au terme de la mise à jour de la politique, la requête administrative est propagée aux autres utilisateurs pour une mise en conformité de leurs politiques d'accès respectives. En ce qui concerne le processus de génération d'une règle d'accès, les différentes étapes du processus sont présentées à la figure 2.2.

Définition 2.3 Une *requête administrative* est un triplet (id, o, v) où : (i) id est l'identifiant de l'administrateur ; (ii) o est l'opération administrative et (iii) v est le numéro de version de la politique d'accès. ■

Génération d'opérations coopératives. Une opération coopérative peut être générée par n'importe quel utilisateur participant à la collaboration, y compris l'administrateur. Lorsqu'elle est générée, l'opération coopérative est immédiatement soumise à une vérification préliminaire de conformité par rapport à la politique locale.

7. Une opération coopérative c est en conflit avec une opération administrative a lorsque c et a sont concurrentes, $Administrator(c)$ est l'émetteur de a et c est révoquée par a.

8. Nous supposons que l'utilisateur est conscient de ses actes pour limiter la répétition du mécanisme faire/défaire.

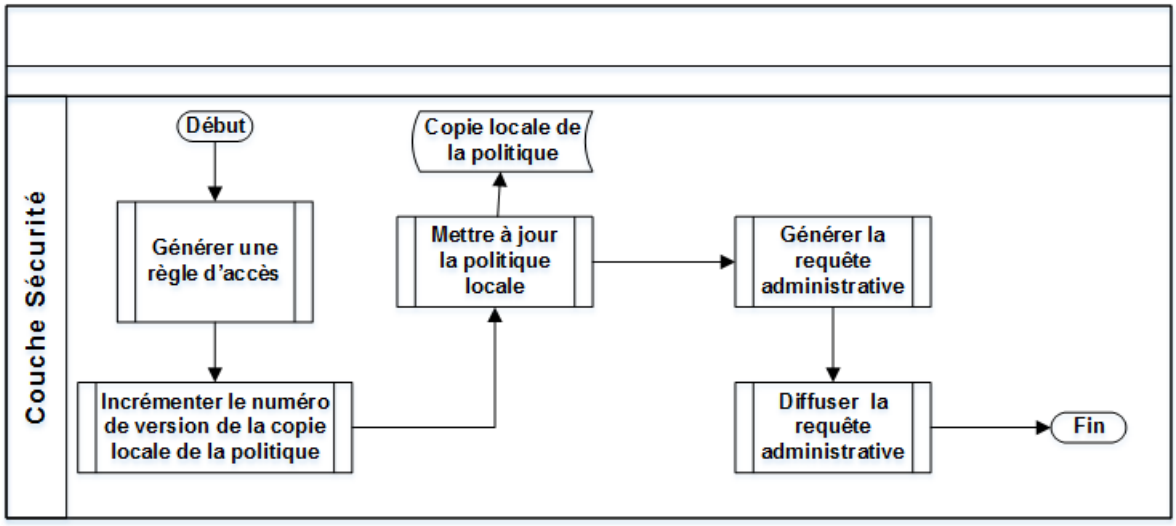


FIGURE 2.2 – Processus de génération et diffusion d'une règle d'accès [98].

Le protocole gère pour chaque opération coopérative un statut qui permet de savoir lors de l'évolution du système, si l'opération est approuvée ou non. Ainsi une opération révoquée a un statut dit "*invalide*". Une opération locale approuvée par la politique d'accès d'un utilisateur qui n'est pas administrateur, se voit affecter un statut appelé "*tentative*" pour exprimer le caractère temporaire de l'approbation. Dans le cas d'un administrateur, un statut appelé "*valid*" est affecté à l'opération locale pour exprimer le caractère définitif de l'approbation. De plus, le numéro de version courant de la politique locale d'accès est attaché à l'opération coopérative approuvée localement pour ainsi constituer une requête coopérative (voir définition 2.4). Celle-ci est envoyée aux autres utilisateurs.

En conséquence, une opération révoquée localement n'est pas connue des autres utilisateurs. Une opération diffusée aux autres utilisateurs (statut "*tentative*" ou "*valide*") est par la suite exécutée localement par le protocole de coordination, tandis qu'une opération révoquée (statut "*invalide*") ne peut être exécutée. Tous les traitements, de la génération de l'opération à son exécution ou non, en passant par le contrôle de conformité préliminaire, l'attribution de statut et l'association de numéro de version, sont considérés comme une transaction et constituent une et une seule opération au niveau du système. Les traitements réalisés par le système lors d'une telle transaction sont présentés à la figure 2.3.

DÉFINITION 2.4 Un requête coopérative q est un tuple (c, r, a, o, v, f) où : (i) c identifie le site émetteur de la requête ; (ii) r est un numéro sériel d'opérations coopératives ; (iii) a est l'identifiant de la requête cooperative précédente ; (iv) o est l'opération coopérative concernée par la requête ; (v) v est le numéro de version de la politique à la génération et (vi) f est le statut de l'opération o. L'identifiant d'une requête cooperative est la concaténtion de l'identifiant du site ($q.c$) et du numéro sériel de l'opération coopérative ($q.r$). La composante a prend la valeur nulle si la requête ne dépend d'aucune autre. ■

Le contrôle de conformité préliminaire est utilisé pour s'assurer que l'opération est cohérente par rapport à la politique locale au moment de sa génération. Il a pour avantage de contribuer à la réactivité du système, contrairement au cas où il aurait fallu attendre l'approbation d'un l'administrateur distant avant d'exécuter ou non une opération coopérative donnée. Le numéro

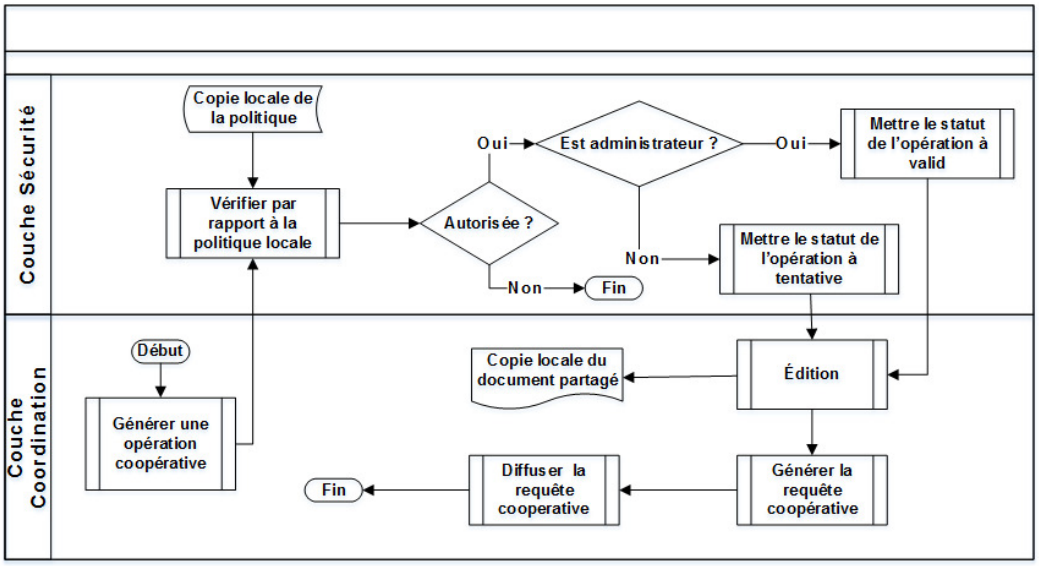


FIGURE 2.3 – Transaction correspondant à la génération d'opération coopérative [98].

de version de la politique locale d'accès associé à l'opération coopérative approuvée permet de maintenir une causalité entre opérations coopératives et administratives, en ce sens qu'il est possible de situer l'opération coopérative par rapport à la règle administrative qui a servi à l'approuver.

Pour illustrer la nécessité d'avoir recours au maintien de la causalité entre opérations coopératives et administratives, considérons un système collaboratif composé de trois utilisateurs dont un administrateur tel que présenté à la figure 2.4. Les utilisateurs sont nommés adm, s_1 et s_2 ; adm étant l'administrateur. Admettons que la copie locale du document partagé contienne au début de la collaboration le texte "abc" et que le droit de suppression sur le document soit initialement accordé à s_2. Supposons à présent que adm produit une règle d'accès interdisant la suppression à s_2 pendant que ce dernier génère en concurrence l'opération $Del(1, a)$ ayant pour effet local, l'obtention du texte "bc". À la réception de l'opération de suppression par l'administrateur, celle-ci sera ignorée car jugée non conforme à la politique locale. Si l'utilisateur s_1 reçoit la révocation du droit de suppression de s_2 et l'applique avant de recevoir l'opération de suppression, cette dernière sera ignorée alors que la réception de la révocation du droit de suppression à s_2 n'a aucun effet sur l'opération. L'exécution des deux opérations permet d'obtenir sur chacun des sites des documents contenant "abc", "abc" et "bc", respectivement. En conclusion, si les numéros de versions avaient été attachés aux opérations, il aurait été possible pour les utilisateurs de se rendre compte que l'opération de l'utilisateur s_2 correspond à une version de la politique locale antérieure à celle de l'opération de l'administrateur.

Réception d'opérations administratives. À la réception d'une opération administrative par un site, elle est systématiquement sauvegardée dans une queue. Elle y sera extraite seulement quand elle est est *causalement prête* : c'est-à-dire le numéro de version de l'opération est le successeur du numéro de version de la politique d'accès locale. Une fois extraite, elle est ajoutée à un journal administratif et appliquée localement à la politique d'accès. Toutes les opérations précédemment exécutées et qui ont encore un statut "*tentative*" sont ensuite passées en revue, seulement si l'opération administrative indique une révocation. Ce faisant, toute opération de

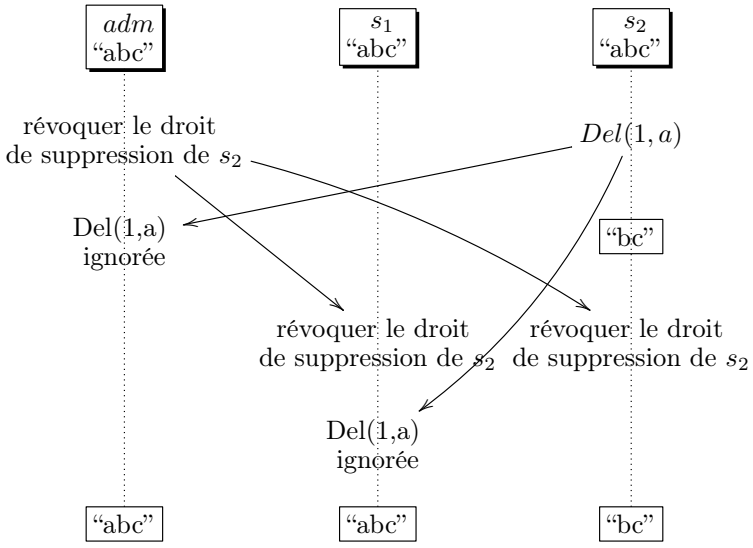


FIGURE 2.4 – Divergence causée par l'absence de lien de causalité entre opérations coopérative et administrative.

statut "*tentative*" concernée par la nouvelle règle d'accès est annulée. L'opération annulée prend désormais un statut appelé "*invalide*", pour indiquer qu'elle n'est pas autorisée. Pour finir, le numéro de version de la politique est incrémenté.

La causalité établie entre opérations administratives par l'intermédiaire du numéro de version permet de s'assurer que les règles d'accès sont ajoutées à la politique dans le même ordre sur tous les sites. La possibilté offerte par le protocole, qui consiste à permettre à une opération administrative reçue et causalement prête, d'annuler une opération coopérative ayant un statut "*tentative*", donne au protocole de coordination l'opportunité d'annuler une opération précédemment exécutée afin de se conformer à la politique d'accès. La politique d'accès est de ce fait rétroactive. Une telle stratégie est dite *optimiste* car elle permet aux copies de l'objet partagé de diverger, puis de converger par la suite, au regard du respect de la politique d'accès [94] et de la concurrence des opérations. Elle est bien adaptée aux applications réparties car elle évite de maintenir un ordre global entre les opérations coopératives et administratives considérées conjointement. La mise en œuvre d'une telle stratégie est bien évidemment favorisée par la gestion d'un numéro de version pour la politique d'accès et d'un statut pour les opérations coopératives.

En considérant à nouveau le scénario illustré à la figure 2.4, la mise en œuvre de la politique optimiste appuyée par la gestion des statuts, conduit à une exécution cohérente. En effet, à l'exécution de l'opération $Del(1, a)$ par s_2, un statut "*tentative*" lui est affecté. La réception par la suite de l'ordre de révocation du droit de suppresion et son application sur le site s_2 entraîne l'annulation de l'opération $Del(1, a)$ telle qu'indiquée à la figure 2.5.

Réception d'opérations coopératives. À la réception d'une opération coopérative, elle est systématiquement sauvegardée dans une queue dédiée. Une fois qu'elle est causalement prête, elle est extraite de la queue des opérations coopératives. Une opération coopérative est causalement prête si son numéro de version attaché est plus petit ou égal au numéro de version courant de la politique d'accès locale et si l'opération coopérative qui la précède est déjà prise en compte sur ce site. Après son extraction, l'opération coopérative est soumise à une vérification de conformité par rapport à la politique d'accès, grâce au journal administratif. En cas d'échec, l'opération

prend le statut "*invalide*" et ainsi prend fin son traitement sur le site. Si la vérification de l'opération coopérative reçue est concluante sur un site qui n'est pas administrateur, elle garde son statut d'origine (celui obtenu à la génération) tout en étant exécutée par le protocole de coordination. Dans le cas où la vérification de l'opération coopérative reçue est concluante sur un site administrateur, l'opération prend le statut "*valide*". Il s'en suit une incrémentation du numéro de version de la politique d'accès et la production d'une requête administrative dite de *validation*. La requête de validation est envoyée aux autres sites dans le but de les informer de la validité de l'opération coopérative en question. Le nouveau numéro de version est attaché à la requête de validation avant sa diffusion. Pour finir, le protocole de coordination de l'administrateur exécute localement l'opération autorisée.

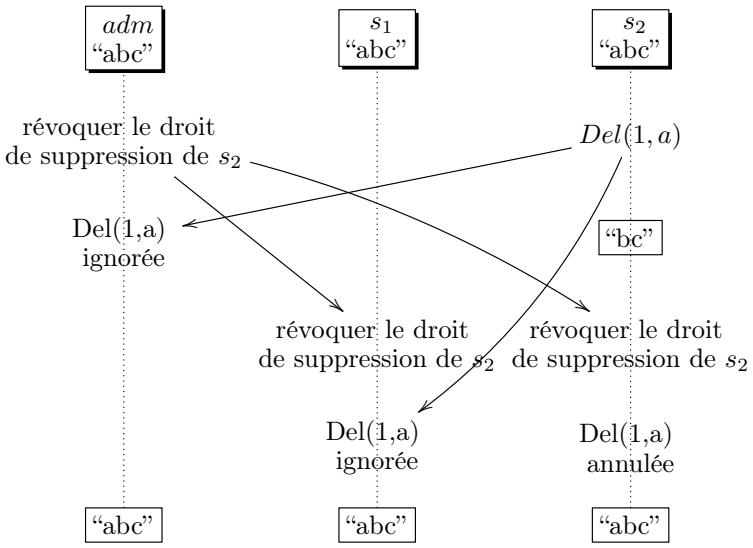


FIGURE 2.5 – Exemple de l'application de la retroactivité du protocole.

Réception de requêtes de validation. Une requête de validation reçue est traitée comme une requête administrative portant sur une règle d'accès. Elle utilise la même queue et est soumise à la même condition de causalité avant son extraction (voir la génération des opérations administratives). Une fois que la requête de validation est extraite de la queue administrative, le statut de l'opération coopérative sur laquelle elle porte est mis à jour. Son statut devient "*valide*". Le numéro de version de la politique est incrémenté.

2.3.3 Implémentation et Evaluation

Nous avons développé un prototype en java, P2PAgenda [9], sur la base du protocole de coordination OPTIC [60]. Notre prototype permet aux utilisateurs de modifier simultanément un agenda partagé, ainsi que les droits d'accès afin de protéger une partie de l'agenda. Ainsi, chaque utilisateur dispose d'une liste de règles d'accès qu'il peut mettre à jour en fonction de sa politique.

Des expérimentations ont été effectuées sur la plate-forme distribuée Grid5000 [10] afin de déterminer le temps de réponse nécessaire pour voir l'effet d'une opération coopérative dans les sites distants après l'avoir contrôlée par rapport à la politique.

9. `http://www.loria.fr/~imine/tools/p2pAgenda/p2pAgenda.htm`.
10. `http://www.grid5000.fr`.

Temps de traitement pour le contrôle des requêtes locale et distante. Pour valider notre approche lorsque différentes tailles de politiques sont manipulées, nous avons mesuré les performances de notre prototype sur des requêtes coopératives et des règles d'accès générées de manière aléatoire. La politique générée était simple et non optimale (c-à-d, elle contient des règles redondantes). La figure 2.6 montre le temps d'exécution nécessaire pour contrôler les requêtes locales et distantes.

Ainsi, nous montrons que notre méthode peut ainsi manipuler de très larges politiques et journaux et elle produit un délai allant de 1.18 à 16 ms, en fonction de la taille de la politique ou le journal administratif. Ces résultats sont encourageants en comparaison avec le temps de réponse requis pour les applications collaboratives (c-à-d, 100 ms [73]). En effet, les résultats de l'expérience montrent que le temps nécessaire pour vérifier une opération est faible (environ 14 ms pour une opération distante et 16 ms pour une opération locale avec une politique propriétaire contenant 10 000 autorisations). Ce bon résultat est obtenu grâce à l'utilisation des techniques de hachage pour avoir un accès direct à nos objets dans chaque autorisation.

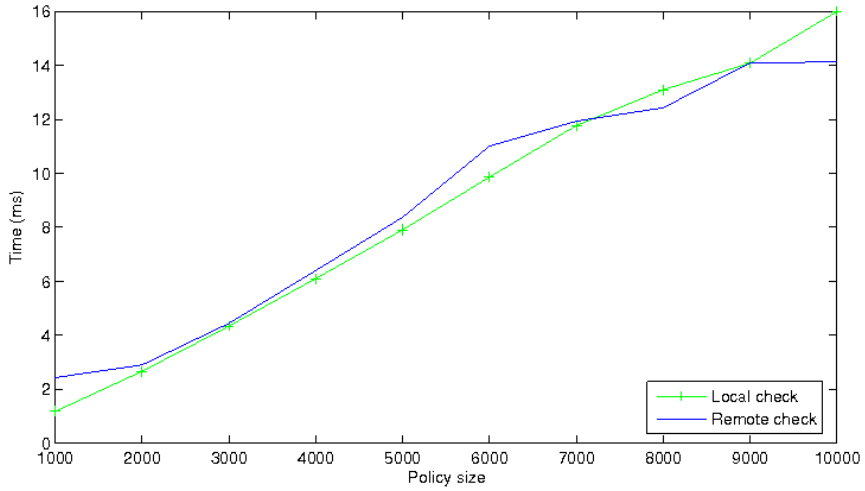


FIGURE 2.6 – Temps pour le contrôle des requêtes locale et distante.

Pour accélérer plus les traitements, il serait judicieux d'éliminer la redondance des règles dans les politiques pour optimiser le contrôle local. De plus, les journaux administratifs pourraient être taillés ou supprimés grâce au mécanisme de ramasse-miettes [19].

Variation du temps de réponse par rapport au nombre des pairs. Pour valider notre configuration expérimentale, nous avons également mesuré le temps de réponse de notre couche de sécurité pour différentes tailles de groupes d'utilisateurs. La deuxième expérience montre le temps de réponse pour l'application d'un journal coopératif contenant 150 000 opérations et une politique formée par de nombreuses politiques propriétaires où chacune contient 5000 autorisations. Le temps de réponse correspond au temps requis par une opération pour être vue sur un site distant. Dans la figure 2.7, deux cas sont présentés dans la courbe : la premier concerne le cas où la politique reste la même (à savoir le contexte ne change pas dans les sites de l'expéditeur et le récepteur), et le second concerne le cas où la politique change durant l'échange de mises à jour entre les utilisateurs.

Nous pouvons tirer plusieurs conclusions à partir de la figure 2.7. Tout d'abord, nous pouvons évidemment voir que pour un journal coopératif contenant 150 000 opérations et une politique

contenant 80 politiques propriétaires où chacune contient 5000 autorisations, le temps de réponse est inférieur à 100 ms. Deuxièmement, le nombre de pairs ne dégrade pas les performances de notre système puisque le seul paramètre qui a un impact sur le temps de réponse lorsque la taille du groupe croît est θ (θ correspond à la limite maximale nécessaire pour qu'un message traverse le réseau de tout expéditeur à son récepteur) qui dépend uniquement de la configuration du réseau. Enfin, nous remarquons que la politique de sécurité prend moins de temps de traitement lorsque le contexte reste le même ; ce qui est logique puisque le contrôle ne produit pas de traitements supplémentaires du fait que la politique ne change pas.

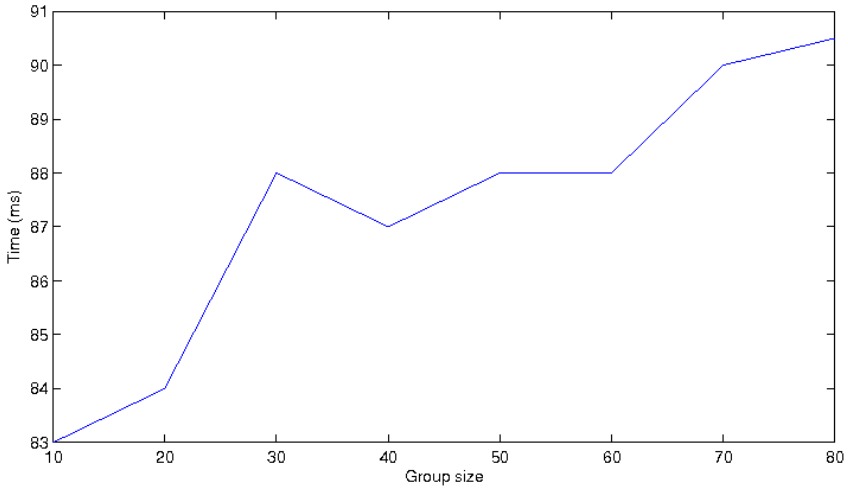


FIGURE 2.7 – Temps de réponse.

Traitement induit par le contrôle d'accès. Pour évaluer l'efficacité de notre couche de contrôle d'accès, nous comparons le temps d'exécution requis par la couche de coordination, le contrôle d'accès et la communication entre les pairs. Sur la figure 2.8, "Integration" correspond au temps requis par la couche de coordination pour exécuter une requête coopérative. "Access Control" correspond au temps d'exécution nécessaire pour évaluer le droit d'accès pour une opération coopérative par rapport à la politique ou le journal administratif. Pour mesurer l'impact de la politique de contrôle d'accès, nous calculons le coût relatif du contrôle d'accès. De toute évidence, le coût relatif du contrôle d'accès est d'environ 21% du coût total. Ces mesures sont prometteuses et démontrent l'applicabilité de la solution.

2.4 Modélisation Formelle du Protocole de Contrôle d'Accès

2.4.1 Propriété fonctionnelle du contrôle d'accès

Dans notre modèle, la réplication est observée à deux niveaux. Le premier niveau se réfère à la réplication de l'objet partagé (par exemple document ou agenda) dès lors qu'un utilisateur rejoint le groupe collaboratif : il obtient une réplique de l'objet partagé. Le second niveau est celui des opérations coopératives. Une fois générées sur un site, les opérations coopératives doivent être prises en compte sur tous les autres sites participant à la collaboration. Ce dernier niveau de réplication permet de maintenir les répliques de l'objet partagé. Les applications collaboratives doivent satisfaire la propriété critique qu'est (i) la cohérence, en plus d'augmenter ou améliorer (ii) la disponibilité des répliques d'objets partagés, (iii) la réactivité locale du systéme et (iv) la

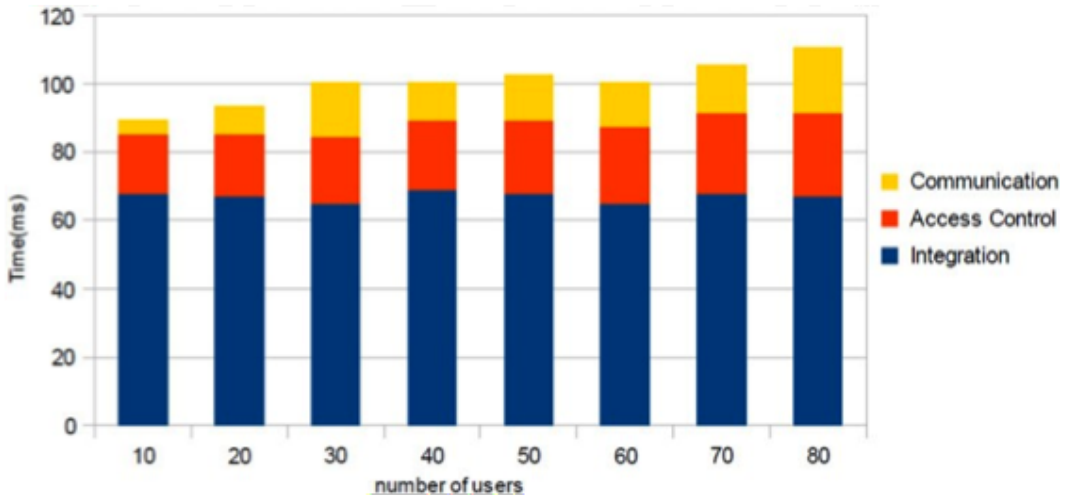


FIGURE 2.8 – Temps de réponse.

performance du système. Lorsqu'une politique de sécurité doit être déployée dans un tel système, elle doit être conçue de manière à préserver les propriétés du système.

Ainsi, si le système collaboratif permet initialement de garantir la cohérence des répliques d'objets à travers son protocole de coordination, la politique de sécurité mise en oeuvre dans le système ne doit pas affecter la garantie de cohérence qu'il offre. Considérons le contexte d'un protocole de coordination au dessus duquel est déployé le protocole de contrôle d'accès étudié dans [61, 19]. Ce protocole de contrôle d'accès ne couvre pas la réplique initiale de l'objet partagé. Une abstraction peut donc être faite de la propriété d'amélioration de la disponibilité. La réactivité locale du système et la performance peuvent être considérées comme des propriétés d'ordre opérationnel car caratérisées par des contraintes opérationnelles telle que la qualité du réseau de communication. La préservation de la cohérence est la principale propriété qui soit fonctionnelle car liée à la conception du protocole de contrôle d'accès mis en œuvre.

La préservation de la propriété de cohérence du système par le protocole de contrôle d'accès suppose que le protocole de coordination doit exécuter sur chaque site les mêmes opérations, même si ce n'est pas dans le même ordre. Il n'est donc pas tolérable que sur un site donné le protocole de contrôle d'accès autorise une opération coopérative qui devrait être refusée ou refuse une opération coopérative qui devrait être autorisée. Autrement dit, sur deux sites distincts, le protocole de contrôle d'accès ne doit pas prendre des décisions différentes par rapport à une même opération.

Vérifier que le protocole ne contient pas de faille portant atteinte à la cohérence du système est une tâche complexe en raison de la nature infinie du système. En effet, le nombre d'utilisateurs, le nombre d'opérations coopératives par utilisateur et le nombre d'opérations administratives sont arbitraires. Notons aussi que l'ordre d'exécution entre les deux types d'opérations (coopérative et administrative) est également arbitraire. La vérification en utilisant des outils basés sur des modèles explicites conduit inéluctablement à une situation d'explosion combinatoire. L'approche des modèles symboliques permet de résorber tant soit peu la difficulté. Etant bien adapté pour modéliser les systèmes infinis, nous proposons d'utiliser l'outil Alloy [88] pour spécifier le protocole de contrôle d'accès que nous avons décrit dans la section précédente, et vérifier qu'il n'affecte pas la propriété de cohérence sous-jacente aux systèmes collaboratifs. Alloy permet d'utiliser la logique de premier ordre et les notions de la théorie des ensembles pour décrire les systèmes.

2.4.2 Spécification du Protocole

La modélisation du protocole de contrôle d'accès a pour but de produire une description qui représente symboliquement le comportement du protocole. A cet effet, les fondements théoriques (voir la section 2.2) ainsi que le dynamisme qui découle de la mise en œuvre du protocole (voir la section 2.3) doivent être pris en compte. Grâce à son langage de spécification inspiré de la notation Z [134], Alloy offre l'abstraction nécessaire pour accomplir cette tâche. La spécification d'un modèle "à la Alloy" consiste à définir des ensembles et des relations appelés signatures (notés *sig*) pour décrire sous forme de relations, le comportement d'un système. Les contraintes du système qui sont toujours satisfaites sont définies grâce à des faits (notés $fact$). Des fonctions (notées fun) et prédicats (notés *pred*) peuvent également être utilisés pour compléter la description proprement dite du système ou compléter l'expression des contraintes, ou encore les propriétés attendues du système. Ces dernières représentent les contraintes à satisfaire par le système et exprimées sous forme d'assertions (notées *assert*). La quantification est gérée grâce à plusieurs éléments : *one* indique l'unicité, *lone* indique "au plus un", *none* indique l'inexistence, *some* signifie "il existe au moins un" (quantificateur existentiel ∃), *all* signifie "tout" et permet de traduire l'universalité (quantificateur universel ∀).

Objets de base du protocole. Dans cette section, nous présentons la spécification des objets de base qui permettront plus tard de décrire la dynamique du protocole. Il s'agit entre autres des sites (utilisateurs), des opérations coopératives et administratives, ainsi que leurs types respectifs, les requêtes administratives découlant de règles d'accès ou de validation d'opérations coopératives.

```
[1]  abstract sig Site {}
[2]  sig CoopSite extends Site {}
[3]  one sig AdmSite extends CoopSite{}
[4]  sig SiteVersion{}
[5]  sig oper_t{}                                   // Type d'opérations : insertion, suppression, etc.
[6]  sig Coop {
[7]      from : lone Site,                                  // Site ayant généré l'opération
[8]      type :lone oper_t,                                 // Type d'opération
[9]      vers : lone SiteVersion                    // Numéro de version de la politique locale
[10]     }{(from!=none) implies {                   // Indique que si l'opération a un auteur, alors
[11]         type!=none                                     // elle a un type
[12]         vers!=none }                                   // et une version.
[13] }
[14] abstract sig Authorization {}
[15] one sig Plus, Minus extends Authorization {}
[16] abstract sig AdReq{                                    // Requête administrative
[17]     source : lone AdmSite,          // Indique l'administrateur ayant généré la requête administrative
[18]     vers : lone SiteVersion                    // Numéro de version de la politique locale
[19]     }{source!=none implies vers!=none      // Indique que si la règle a un auteur, alors elle a une version
[20] }
[21] sig Rule extends AdReq{                                // Règle d'accès
[22]     subject : some Site,
[23]     right : some oper_t,
[24]     signe : one Authorization
[25] }
[26] sig Val extends AdReq{op : lone Coop}{ op.from!= source }          // Ordre de validation
```

Spécification 1. Objets de base

Dans notre modèle, un site est d'abord décrit comme un ensemble générique ou signature abstraite (*Site {}*). De cette définition, nous dérivons grâce à la notion d'héritage (extension sous Alloy), les deux catérogies de sites à savoir administrateur (*AdmSite{}*) et non-administrateur (*CoopSite{}*). Puisque chaque site gère un compteur de numéro de version de la politique d'accès, la signature (*SiteVersion{}*) a été définie pour l'implémenter. Pour prendre en compte les

divers types d'opérations coopératives (par exemple, les opérations Ins et Del) que supporte le système colaboratif, nous avons défini une signature ($oper\_t\{\}$). Les opérations coopératives proprement dites sont décrites (signature $Coop\{\}$) par : le site générateur de l'opération, le type de l'opération et le numéro de version de la politique au moment de la génération. En se référant aux spécifications du protocole, on distingue deux types de requêtes administratives : celles découlant des règles d'accès et celles relatives à la validation d'opérations coopératives par l'administrateur. Une requête administrative a donc été définie de manière générique ($AdReq\{\}$) et sert à encapsuler soit une règle d'accès soit un ordre de validation d'opération coopértive ($Rule\{\}$ et $Val\{\}$). La spécification d'une règle d'accès s'appuie formellement sur la définition 2.1. Cependant, pour des raisons de simplicité et sans perte de généralité, nous considérons que toutes les règles se rapportent à un seul objet. A cet effet, il n'est plus nécessaire de laisser l'objet O_i dans le quadruplet $\langle S_i, O_i, R_i, w_i \rangle$. Ainsi nous ne gardons que l'ensemble des utilisateurs, les droits d'accès et la nature des droits accordés. La modélisation à la Alloy des objets de bases est donnée dans la spécification 1.

Etat du système. Le numéro de version de la politique, le statut de chaque opération sur chaque site, les queues et histoires locales respectives permettent d'obtenir l'état local d'un site. La transition entre deux états se fait par l'exécution des actions suivantes : générer et envoyer une opération coopérative, recevoir une opération coopérative, traiter une opération coopérative causalement prête, générer et envoyer une requête administrative, recevoir une requête administrative, et traiter une requête administrative causalement prête. Certaines de ces actions peuvent êtres combinées lors du franchissement d'une transition. L'état global du système est défini comme l'ensemble des états locaux des sites participants. Le système passe alors d'un état global à un autre, dès que l'un au moins des sites exécute l'une des actions citées ci-dessus. Le statut d'une opération est modélisé d'abord par une signature générique ($OpStatus$), puis elle est déclinée en trois signatures disjointes représentant les statuts "$tentative$", "$invalide$", "$valide$". L'état global du système est modélisé dans la spécification 2. Il est à noter que cette spécification n'indique pas l'état initial du système. Dans l'état initial, toutes les queues et histoires sont vides, le numéro de version de la politique est à 0. L'État initial est défini dans la spécification 3.

```
[1]  abstract sig OpStatus {}
[2]  one sig tentative, invalide, valid extends OpStatus {}
[3]  sig SiteState {
[4]      versions : Site ->one SiteVersion,                  // Association Site et son numéro de version de politique
[5]      CoopStatus : Site -> Coop-> OpStatus,                          // Association site, opération, Statut
[6]      sentCoop : Site -> lone Coop,                  // Ensemble des opérations coopératives générées et envoyées
[7]      sentAdReq : AdmSite -> lone AdReq,     // Ensemble des opérations administratives générées et envoyées
[8]      F : Site -> (seq Coop),                          // Queue des opérations coopératives pour chaque site
[9]      Q : Site -> (seq AdReq),                       // Queue des opérations administratives pour chaque site
[10]     H : Site -> set Coop,                              // Histoire des opérations coopératives exécutées
[11]     L : Site -> (seq Rule),                                                  // Log administratif par site
[12]     Vr : Site -> (seq Val),                          // Requêtes de validation traités pour chaque site
[13]     CoopCausallyReady : Site -> lone Coop,          // Opérations coopératives causalement prêtes par site
[14]     AdCausallyReady : Site -> lone AdReq           // Requêtes administratives causalement prêtes par site
[15] }
```

Spécification 2 État du système

Transitions entre états du système La dynamique du système se traduit par un changement successif d'états. Nous définissons alors un ensemble de transitions E, en considérant l'ensemble $SiteState$ des états du système (spécification 2) et Act, un ensemble fini d'actions exécutées lors

```
[1]  fact InitialStateConstraints {
[2]      all S :Site | let st=sitesstates/first| versOrder/first in st.versions[S]
[3]      no sitesstates/first.CoopStatus
[4]      no sitesstates/first.F
[5]      no sitesstates/first.Q
[6]      no sitesstates/first.L
[7]      no sitesstates/first.Vr
[8]      no sitesstates/first.H
[9]      no sitesstates/first.CoopCausallyReady
[10]     no sitesstates/first. AdCausallyReady
[11]     no sitesstates/first.sentAdReq[AdmSite] & Val
[12]     some sitesstates/first.sentCoop
[13]     no sitesstates/last.sentCoop
[14]     no sitesstates/last.sentAdReq
[15]     no sitesstates/prev[sitesstates/last].sentCoop
[16]     no sitesstates/prev[sitesstates/last].sentAdReq
[17] }
```

Spécification 3 État initial du système

du franchissement des transitions.

$$E \subseteq SiteState \times Act \times SiteState \tag{2.1}$$

$$\begin{aligned} Act = \{&GenerateCooperativeRequest(),\\ &GenerateAdministrativeRequest(),\\ &ReceiveCoopRequestOrdSite(), ReceiveCoopRequestAdm(),\\ &ReceiveAdminRequest()\} \end{aligned} \tag{2.2}$$

Tout en faisant changer l'état du système, l'exécution des actions a pour effet de modifier les statuts des opérations, les numéros de version des politiques locales, les queues et les histoires. Selon le protocole, ces actions sont : générer et envoyer une requête administrative, recevoir une requête administrative, générer et envoyer une opération coopérative, recevoir une opération coopérative, traiter une opération coopérative ou une requête administrative causalement prête. Notre modèle les spécifie sous forme de contraintes qui sont toujours satisfaites.

2.4.3 Analyse du Protocole

Cette section est consacrée à l'analyse du protocole de contrôle d'accès, au regard de la propriété de cohérence et de certains choix conceptuels. Nous avons choisi l'analyseur Alloy pour effectuer cette tâche. L'analyseur Alloy 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 un solveur SAT. Parmi ces solveurs on distingue : Kodkod [130], MiniSAT [33], SAT4J [71], zChaff [11]. Dans notre cas, le solveur SAT4J a été utilisé, pour bénéficier d'une optimisation de la réduction lors de l'exploration.

Analyse de la préservation de cohérence. La satisfaction de la propriété de cohérence par un système collaboratif fait référence à un état stable du système (définition 2.5).

Définition 2.5 Le système est dit dans un état stable si et seulement si toutes les opérations générées ont été exécutées sur tous les sites. ■

De cette définition, les deux proprétés équivalentes suivantes peuvent être dégagées.

PROPRIÉTÉ 2.1 Dans un état stable du système, aucune opération coopérative exécutée n'a le statut *tentative*. ■

PROPRIÉTÉ 2.2 Dans un état stable du système, chaque opération coopérative exécutée a soit le statut *invalide*, soit le statut *valid*. ■

Le système satisfait la propriété de cohérence si et seulement si pour tout état stable, les copies des objets partagés sont identiques sur tous les sites. En considérant un système avec contrôle d'accès, la préservation de la propriété de cohérence peut être formulée telle que présentée dans la définition 2.6.

DÉFINITION 2.6 Le protocole préserve la propriété de cohérence du système si et seulement si dans tout état stable du système, chaque opération coopérative exécutée a le même statut sur tous les sites. ■

En exploitant la propriété 2.2 elle peut être reformulée uniquement en fonction des statuts finaux des opérations (définition 2.7).

DÉFINITION 2.7 Le protocole préserve la propriété de cohérence du système si et seulement si dans tout état stable du système, chaque opération coopérative exécutée a soit le statut *invalide* sur tous les sites, soit le statut *valid* sur tous les sites. ■

Dans le cadre de l'analyse de la préservation de la propriété de cohérence du système, nous considérons uniquement les opérations coopératives générées, envoyées, exécutées et ayant un statut *valid*. En effet, une opération ayant un statut *invalide* sur un site n'a pas été exécutée sur ce site ou bien, a été exécutée puis déclarée invalide et sujette à annulation. Les opérations invalides sont considérées comme non avenues. L'analyse de la préservation de la propriété de cohérence du système se base alors sur la comparaison de l'ensemble des opérations valides (statut *valid*) des sites respectifs. Ainsi, deux sites quelconques doivent avoir le même ensemble d'opérations valides (spécification 4).

```
[1] assert ValidPreservation{
[2]     all st :Stable[]|{
[3]         all disj Si, Sj :Site|{
[4]             valid[st.CoopStatus[Si]] in valid[st.CoopStatus[Sj]]
[5]             valid[st.CoopStatus[Sj]] in valid[st.CoopStatus[Si]] } }
[6] }
```

Spécification 4 Préservation de la cohérence du système

Alloy étant un outil de vérification bornée sur modèle (*bounded model checker*), l'analyse a été faite en spécifiant diverses bornes. La plus grande valeur de borne qui a été atteinte est 13 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. Pour toutes les bornes considérées, les résultats de l'analyse indiquent une absence de contre-exemple "*No counterexample found. Assertion may be valid.*".

Analyse des choix conceptuels. En plus de s'assurer de la préservation de la propriété de cohérence du système par le protocole, il était important de motiver certains choix conceptuels du protocole. Il s'agit des choix (i) de maintenir une relation de causalité entre les opérations coopératives et administratives ; et (ii) de produire des requêtes de validation d'opérations coopératives.

1) Relation de causalité entre les opérations coopératives et administratives :
La rétroactivité du protocole n'est pas suffisante pour conduire à une collaboration cohérente avec un déploiement uniforme de la politique d'accès sur tous les sites. Lorsque la génération d'une opération coopérative se fait en concurrence avec l'émission d'une règle d'accès restrictive qui rend l'opération illégale, il est évident que ces deux opérations ne seront pas nécessairement reçues dans le même ordre sur tous les sites. Si en plus, peu de temps après, l'administrateur rétablit au site concerné, le droit d'exécuter le même type d'opération, alors il faut désormais considérer que la nouvelle opération peut même être reçue par un site avant les autres. De façon générale, six ordres de réception des trois opérations sont envisageables en faisant une permutation. L'un de ces cas est illustré à la figure 2.9. Dans cet exemple, le site s_1 reçoit la révocation de la suppression par s_2, suivi de l'autorisation de suppression et ensuite l'opération de suppression. La rétroactivité du protocole a permis au site s_2 d'annuler son opération de suppression et ainsi d'être cohérente avec l'administrateur. Cependant sur le site s_1, vu que le parcours de la politique se fait en recherchant la première règle d'accès qui concerne l'opération coopérative manipulée, la suppression sera acceptée. Ce qui conduirait à une incohérence. Pour faire face à de telles situations, une causalité entre opérations coopératives et administratives est proposée dans le protocole. La relation de causalité entre opérations coopératives et administratives est possible grâce à l'ajout d'un numéro de version de la politique d'accès à l'opération coopérative à sa génération. Quand une requête coopérative ou administrative est reçue sur le site distant, elle reste dans la queue correspondante tant qu'elle n'est pas causalement prête. Pour la requête coopérative, être prête causalement signifie que son numéro de version de politique attaché est plus petit ou égal au numéro de la version courante de la politique locale et que la requête coopérative qui la précède est déjà prise en compte sur ce site. Par contre, pour une requête administrative, être prête causalement signifie que son numéro de version de politique attaché est le suivant (dans l'ordre croissant) du numéro de la version courante de la politique locale. Pour évaluer l'effet de ces dispositions du protocole, un mutant de la spécification du protocole a été défini. Dans cette version de la spécification, les numéros de version n'ont pas été attachés aux requêtes et la condition de causalité qui impose aux requêtes une certaine attente dans la queue n'a pas été prise en compte. Le mutant permet de trouver le scénario de la figure 2.9.

2) Validation d'opérations coopératives :
Considérons comme illustré à la figure 2.10, que le site s_1 génère une opération de suppression ($Del(1, a)$) reçue et exécutée par l'administrateur *adm* en tant qu'opération valide. Si peu de temps après *adm* émet une révocation du droit de suppression à s_1, il se pourrait que s_2 reçoive l'opération administrative de révocation avant l'opération coopérative de suppression. L'exécution de l'opération administrative par s_2 est conforme aux mesures de rétroactivité du protocole et de causalité entre requêtes coopéeratives et administratives. Ainsi, à la réception de l'opération coopérative elle sera ignorée car la règle de révocation empêchera son exécution. Cette situation est source d'incohérence. Le choix conceptuel opéré pour y faire face est basé sur le statut *tentative* des opérations coopératives qui ne sont pas générées par l'administrateur lui-même. Il s'agit pour l'administrateur de générer une nouvelle opération administrative appelée requête de validation. Elle confirme la validation d'une opération coopérative légale. La génération de requête de validation comme celle de la requête de règle d'accès, entraîne une incrémentation du numéro de version de la politique d'accès. Le nouveau numéro lui est attaché et une fois sur les autres sites, elle est soumise aux même conditions de causalité que les autres requêtes administratives. Ce choix conceptuel aurait permis à *adm* de produire la requête de validation avant la requête de révocation. Les opérations administratives étant totalement ordonnées par numéro croissant de version de politque d'accès, la causalité impose qu'elles soient exécutées

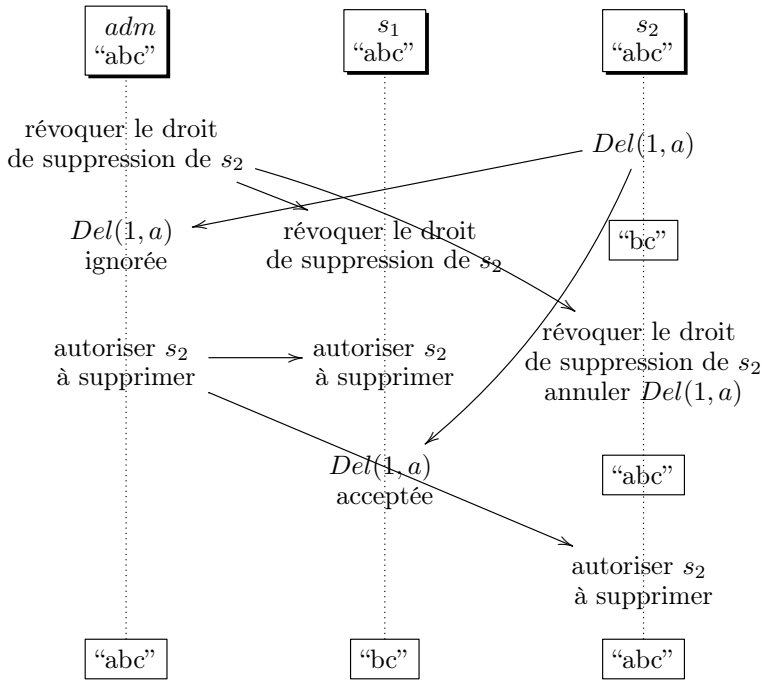


FIGURE 2.9 – Nécessité d'une causalité entre les opérations coopératives et administratives.

dans l'ordre. L'opération de révocation aurait alors été retardée sur le site s_2 jusqu'à ce que la requête de validation soit exécutée. Ainsi l'incohérence ne serait plus observée. Un mutant de la spécification du protocole a été défini, où la gestion de la validation a été omise, et produit le scénario présenté à la figure 2.10.

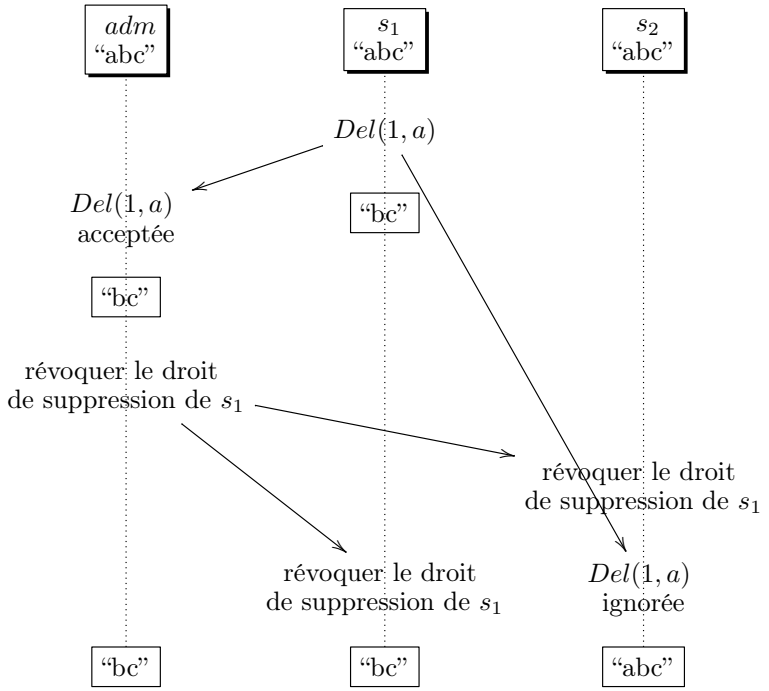


FIGURE 2.10 – Nécessité d'une validation des opérations coopératives.

2.5 Etat de l'Art

La conception d'un modèle de contrôle d'accès qui tient compte des spécificités d'un travail collaboratif réparti est sujette à plusieurs exigences [129], à savoir : 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. En se basant sur ces exigences, Tolone et *al.* ont déterminé dans [129], des critères qui leur ont permis de faire une étude comparative entre plusieurs modèles de contrôle d'accès. L'étude a porté sur la matrice d'accès (*Access Matrix Model*) [109], le contrôle d'accès basé sur les rôles (*Role-Based Access Control*, RBAC) [108], le contrôle d'accès basé sur les tâches (*Task-Based Access Control*, TBAC) [127, 110], le contrôle d'accès basé sur les équipes (*Team-Based Access Control*, TMAC) [126], le contrôle d'accès basé conjointement sur les informations de contexte et les équipes (*Context-Based Team-Based Access Control*, C-TMAC) [46], le contrôle d'accès spatial (*Spatial Access Control*, SAC) [14], le contrôle d'accès sensible au contexte (*Context-Aware Access Control*) [23]. Tolone et *al.* ont alors mis en lumière les forces et faiblesses de chacun d'eux au regard du contexte complexe de l'édition collaborative répartie. De cette étude, on peut retenir que le TMAC est le plus adapté, en ce qui concerne une édition collborative 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 [3] 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 [65] consacré à un milieu médical. STRAC est en fait une variante de C-TMAC qui considère la situation médicale comme contexte déterminant le modèle.

Malheureusement, nous n'avons pas encore trouvé d'étude consacrée à la vérification de la préservation de la propriété de cohérence par TMAC et ses dérivées, si ces derniers devraient être déployés dans un cadre comme celui-ci. Aussi, bien qu'il existe plusieurs études telles que [1, 128, 59, 107], relatives à la satisfaction de certaines propriétés par RBAC et quelques-unes de ses variantes, très peu de travaux sont consacrés à un contexte comme celui des systèmes collaboratifs répartis ou à un contexte similiaire comme le *workflow*. Dans [111], les auteurs ont procédé à la vérification formelle des processus *workflow* collaboratifs en utilisant l'outil SPIN [58]. Ces travaux portent sur un cadre collaboratif et abordent la satisfaction d'une propriété de cohérence, comme la plupart des travaux de vérification de *workflow*. Cependant, [111] ne couvre pas des questions de sécurité.

2.6 Conclusion

Dans ce chapitre, nous avons proposé un modèle générique pour le contrôle d'accès bien adapté aux systèmes collaboratifs. Il est basé sur la réplication optimiste de l'objet partagé ainsi que sa politique d'autorisation. Nous avons présenté le protocole de contrôle d'accès qui peut être déployé au dessus d'un protocole de coordination utilisé par tout système collaboratif. Pour valider notre modèle, nous avons développé un agenda collaboratif en java et utilisant un contrôle d'accès optimiste. Nos résultats expérimentaux ont été réalisés sur la plate-forme GRID'5000 distribuée pour démontrer l'efficacité de notre modèle. Le coût relatif du contrôle d'accès est d'environ 21% du coût total. Ces mesures sont prometteuses et démontrent l'applica-

bilité de la solution. Ce travail démontre que les solutions de sécurité répliquées donnent lieu à des perspectives de recherche intéressantes. En outre, en employant une technique symbolique de model-checking borné, nous avons spécifié formellement l'empilement de notre contrôle d'accès à un système collaboratif. L'analyse d'une telle spécification nous a montré que le contrôle d'accès est uniformément appliqué sur tous les sites et préserve la cohérence. Elle a également permis de valider certains choix conceptuels de notre modèle.

Chapitre 3

Contrôle d'Accès pour des Données XML partagées

Sommaire

3.1 Introduction

XML (eXtensible Markup Language) est une recommendation W3C pour présenter les données dans un format qui peut être traité facilement et échangé sur de multiples plateformes. Un document XML représente non seulement l'information de base, mais aussi les relations entre les données sous la forme d'une structure hiérarchique. En outre, il peut être consulté et/ou mis à jour sans avoir besoin d'une définition statique du schéma. Plusieurs solutions basées sur XML sont proposées pour la représentation des données dans de nombreux systèmes de base de données (tels que Oracle 11g et IBM DB2) qui peuvent être exploitées et partagées par un nombre important d'utilisateurs. Aussi, les opérations sur des données XML doivent être faciles, rapides et surtout à l'abri des accès non autorisés. En effet, plusieurs organisations (principalement médicales et commerciales) manipulent des informations sensibles qui devraient être sélectivement exposées à différentes classes d'utilisateurs ayant des privilèges d'accès très variés. L'usage des données médicales ERH (en anglais, Electronic Health Record) est un exemple typique, où toutes les données des patients sont stockées dans une base de données centralisée qui peuvent être exploitées par différents personnels des services de santé : les infirmières, les médecins, les

pharmaciens, les compagnies d'assurance, etc. Pour des données XML, il peut y avoir plusieurs groupes d'utilisateurs qui veulent interroger les mêmes données. Pour ces groupes d'utilisateurs, des privilèges d'accès différents peuvent être imposées, en spécifiant quelles parties des données sont accessibles aux utilisateurs. Le problème de l'accès sécurisé aux données XML est d'appliquer ces privilèges. De par sa structure hiérarchique et les relations entre les données, ce problème d'accès sélectif nécessite des solutions spécifiques.

Différents modèles ont été proposés pour le contrôle d'accès à des données XML. Etant donné un document XML T conforme à un schéma D, une spécification d'accès S est définie qui régit les informations inaccessibles à partir de D. A partir de S, une vue de schéma D_v est dérivée et ensuite mise à disposition de l'utilisateur pour décrire seulement ses parties accessibles. En outre, une vue de données virtuelle T_v est extraite qui affiche uniquement les parties accessibles de T. XPath est le langage le plus utilisé pour interroger ces vues. Pour chaque requête XPath Q posé sur T_v, le principe de *réécriture des requêtes* consiste à transformer Q en une autre requête Q' de telle sorte que l'évaluation de Q sur T_v donne le même résultat que l'évaluation de Q' sur le document originel T. Bien qu'un effort considérable a été fait pour la réécriture des requêtes sur des vues XML virtuelles, la plupart des algorithmes existants sont limités en ce sens qu'ils traitent seulement des schémas non récursifs. Pour surmonter cette limitation, un langage non-standard XPath régulier (en anglais, Regular XPath) a été proposé pour assurer la réécriture des requêtes sur des schémas récursifs (c-à-d, un élément est défini directement ou indirectement en fonction de lui même). Néanmoins, ce processus est très coûteux car les requêtes réécrites peuvent être de taille exponentielle. De plus, il n'y a pas de solution standard qui existe pour évaluer les requêtes régulières. Les résultats trouvés autour de XPath régulier restent au stade théorique et ils sont encore impraticables.

Dans ce chapitre, nous montrons qu'il est toujours possible de réécrire des requêtes XPath sur des schémas arbitraires (récursifs ou non) en utilisant uniquement la puissance expressive de la norme standard XPath. Nous proposons un language pour spécifier les politiques de contrôle d'accès XML ainsi qu'un algorithme efficace pour faire appliquer de telles politiques. Ce chapitre est composé comme suit : la section 3.2 donne des notions préliminaires sur XML ainsi que les fragments XPath et présente la problématique sur la réécriture des requêtes. Dans la section 3.3, nous décrivons formellement notre modèle de contrôle d'accès. La section 3.4 survole les caractéristiques de notre système SVMAX (Secure and Valid MAnipulation of XML) ainsi que les mesures de performances. Enfin, nous passons en revue l'état de l'art dans la section 3.5 et nous terminons le chapitre par une conclusion.

3.2 Manipulation des Vues XML

3.2.1 Notions préliminaires

Nous présentons les notions de base et les définitions qui sont utilisées dans ce chapitre.

Schéma XML. Nous définissons la notion du schéma XML basée sur le standard *Document Type Definition* (DTD) [103].

Définition 3.1 (**DTD**) Un schéma XML (ou une DTD) D est un triplet (Σ, P, *Root*), où Σ est un ensemble fini de *types d'éléments*, *Root* est un type particulier de Σ appelé le *type racine*, et P est une fonction définissant les types d'éléments tel que pour tout A dans Σ, $P(A)$ est une expression régulière α, représentant le *contenu* de A et définie comme suit :

$$\alpha \texttt{ := str} \mid \epsilon \mid B \mid \alpha\text{','}\alpha \mid \alpha\text{'|'}\alpha \mid \alpha\texttt{*} \mid \alpha+ \mid \alpha\,?$$

où `str` représente le type texte `PCDATA`, ϵ est le mot vide, B est un type d'élément dans Σ, α','α dénote la concaténation et la disjonction est notée par α'|'α. La règle de production pour A est notée par $A \rightarrow P(A)$. Pour tout type d'élément B dans $P(A)$, B (resp. A) est appelé un *type enfant* (resp. *type parent*) de A (resp. B). De plus, $P(A)$ peut être définie en utilisant '$*$' (l'ensemble ayant zéro ou plusieurs éléments), '+' (l'ensemble ayant un ou plusieurs éléments), et '?' (élément optionnel).

Une DTD D est dite *récursive* s'il existe un élément type A défini directement ou indirectement par lui-même. ■

Exemple 3.1 Considérons un *département* scolaire représenté par la DTD (Σ,P,*dept*) avec Σ={*dept*, *course*, *project*, *cname*, *takenBy*, *givenBy*, *students*, *scholarship*, *student*, *sname*, *mark*, *professor*, *pname*, *grade*, *type*, *private*, *public*, *descp*, *results*, *result*, *members*, *member*, *name*, *qualif*, *theoretical*, *experimental*, *sub-project*}. Les règles de production du schéma sont comme suit :

dept	$\rightarrow$	(*course*+, *project*$*$)
course	$\rightarrow$	(*cname*, *takenBy*, *givenBy*)
takenBy	$\rightarrow$	(*students*)
students	$\rightarrow$	(*scholarship*?, *student*+)
scholarship	$\rightarrow$	(*student*+)
student	$\rightarrow$	(*sname*, *mark*)
givenBy	$\rightarrow$	(*professor*+)
professor	$\rightarrow$	(*pname*, *grade*)
project	$\rightarrow$	(*type*, *descp*, *results*, *members*, *sub-project*)
type	$\rightarrow$	(*private* | *public*)
results	$\rightarrow$	(`str` | *result*)$*$
members	$\rightarrow$	(*member*+)
member	$\rightarrow$	(*name*, *qualif*, (*theoretical* | *experimental*)$*$)
sub-project	$\rightarrow$	(*project*$*$)

Un département (*dept*) possède une liste de cours (*course*) ainsi que zéro ou plusieurs projets (*project*). Un cours possède un nom (*cname*), une liste d'étudiants et une liste de professeurs représentées repsectivement par *takenBy* et *givenBy*. Un étudiant (*student*) a un nom (*sname*), un indentifiant (*mark*) et peut posséder une bourse d'études (*scholarship*). Un professeur (*professor*) est défini par un nom (*pname*) et un grade (*grade*). Un projet (*project*) est déterminé par son type (*type* qui peut être privé (*private*) ou public (*public*)), sa description (*descp*), certains résultats (*results*) et peut avoir zéro ou plusieurs projets (par l'intermédiaire de (*sub-project*)). Un membre (*member*) d'un projet donné est représenté par son nom (*name*), une qualification (*qualif* est étudiant, professeur, chercheur, etc.) et une liste de contributions (dont chaque contribution est soit théorique (*theoretical*) soit expérimentale (*experimental*)). Notons que l'élément (*results*) est un mélange de types (combinaison de valeurs textuelles et d'éléments (*result*)). En outre, les éléments *private* et *public* sont vides, tandis que les autres types d'élément (par exemple, *mark*, *result*) sont des éléments texte. ■

Documents XML. Nous modélisons un document XML comme un arbre ordonné. Soit Σ un ensemble fini d'étiquettes de nœuds (avec une étiquette spéciale `str`) et C un ensemble infini de valeurs de texte. Nous représentons nos documents XML avec une structure, appelée *Arbre XML*, défini comme suit :

DÉFINITION 3.2 (**ARBRE XML**) Un arbre XML T sur Σ est une structure (N, $root$, $R_{\downarrow}$, $R_{\rightarrow}$, λ, ν), où N est un ensemble de nœuds, $root \in N$ est la racine, $R_{\downarrow} \subseteq N \times N$ est la relation parent-enfant, $R_{\rightarrow} \subseteq N \times N$ est une relation de successeur sur les nœuds ordonnés ayant le même parent, $\lambda : N \rightarrow \Sigma$ est une fonction associant à chaque nœud une étiquette, et $\nu : N \rightarrow C$ est une fonction attribuant une valeur de texte à chaque nœud avec l'étiquette `str`. ■

Les relations $R_{\downarrow^*}$ et $R_{\rightarrow^*}$ représentent respectivement la fermeture transitive et réflexive de $R_{\downarrow}$ et $R_{\rightarrow}$. Nous utilisons $R_{\uparrow}$ et $R_{\leftarrow}$ pour désigner respectivement l'inverse de $R_{\downarrow}$ et $R_{\rightarrow}$. En outre, $R_{\uparrow^*}$ et $R_{\leftarrow^*}$ désignent respectivement l'inverse de $R_{\downarrow^*}$ et $R_{\rightarrow^*}$.

DÉFINITION 3.3 (**VALIDATION DES ARBRES XML** [86]) Un arbre XML $T = (N, r, R_{\downarrow}, R_{\rightarrow}, \lambda, \nu)$, défini sur l'ensemble Σ d'étiquettes de nœuds, est *conforme* (ou *valide*) à une DTD D=($Ele, P, root$) si les conditions suivantes sont satisfaites :

1. La racine de T est associée à la racine $root$ (c-à-d, $\lambda(r)$=$root$).
2. Chaque nœud dans T est marqué soit avec un type d'élément A dans Ele, ou avec `str`, appelé nœud de texte et $\Sigma = Ele \cup \{\texttt{str}\}$.
3. Pour chaque élément ayant k enfants ordonnés $n_1, ..., n_k$, le mot $\lambda(n_1), ..., \lambda(n_k)$ appartient au langage régulier défini par $P(A)$.
4. Chaque nœud de texte n (c-à-d, avec $\lambda(n) = \texttt{str}$) porte une chaîne $\nu(n)$ (c-à-d, `PCDATA`) et elle est la feuille de l'arbre. ■

Notons que les éléments de T sont les nœuds de N qui sont étiquetés avec Ele, tandis que les nœuds représentent les éléments et les nœuds texte (c-à-d, les nœuds marqués avec `str`). Par la suite, nous utilisons les termes de nœud et élément de manière interchangeable.

Nous appelons T une instance d'une DTD D si T est conforme à D. Nous notons $\mathcal{T}(D)$ l'ensemble de tous les arbres XML conformes à D. Par exemple, la figure 3.1 représente [11] un document XML conforme à la DTD de l'exemple 3.1.

Requêtes XPath. Nous définissons ici les différents fragments de XPath [9] qui sont utilisés dans le présent chapitre. Le premier fragment présenté permet de cibler des nœuds vers le bas (d'où l'appellation en anglais "Downward") dans un document XML [64].

DÉFINITION 3.4 (**FRAGMENT DOWNWARD DE XPATH**) Nous notons par $\mathcal{X}$ le fragment *downward* de XPath qui est défini comme suit :

$$
\begin{array}{l}
p \;:=\; \alpha::\eta \mid p[q]\cdots[q] \mid p/p \mid p \cup p \\
q \;:=\; p \mid p{=}c \mid q \wedge q \mid q \vee q \mid \neg\,(q) \\
\alpha \;:=\; \varepsilon \mid \downarrow \mid \downarrow^{+} \mid \downarrow^{*}
\end{array}
$$

où p représente une requête XPath, η est un test de nœud qui peut être un type d'élément, $*$ (qui correspond à tous les types), ou la fonction *text()* (qui teste si un nœud est un nœud texte). Le symbole c est une chaîne constante, et $\cup$, $\wedge$, $\vee$, $\neg$ désignent respectivement l'*union*, la *conjonction*, la *disjonction*, et la *négation* ; α représente les axes de XPath et peut être ε (*self*), $\downarrow$ (Child), $\downarrow^{+}$ (*descendant*), ou $\downarrow^{*}$ (*descendant-or-self*). Enfin, l'expression q est appelée un qualificatif, un *filtre* ou un *prédicat*. ■

11. Nous rappelons que les indices dans nos exemples d'arbres XML sont utilisés pour distinguer entre les éléments du même type, par exemple, $course_1$ et $course_2$. En outre, en raison de la limitation de l'espace, nous nous concentrons uniquement sur certains nœuds tandis que $\bigtriangleup$ désigne le reste.

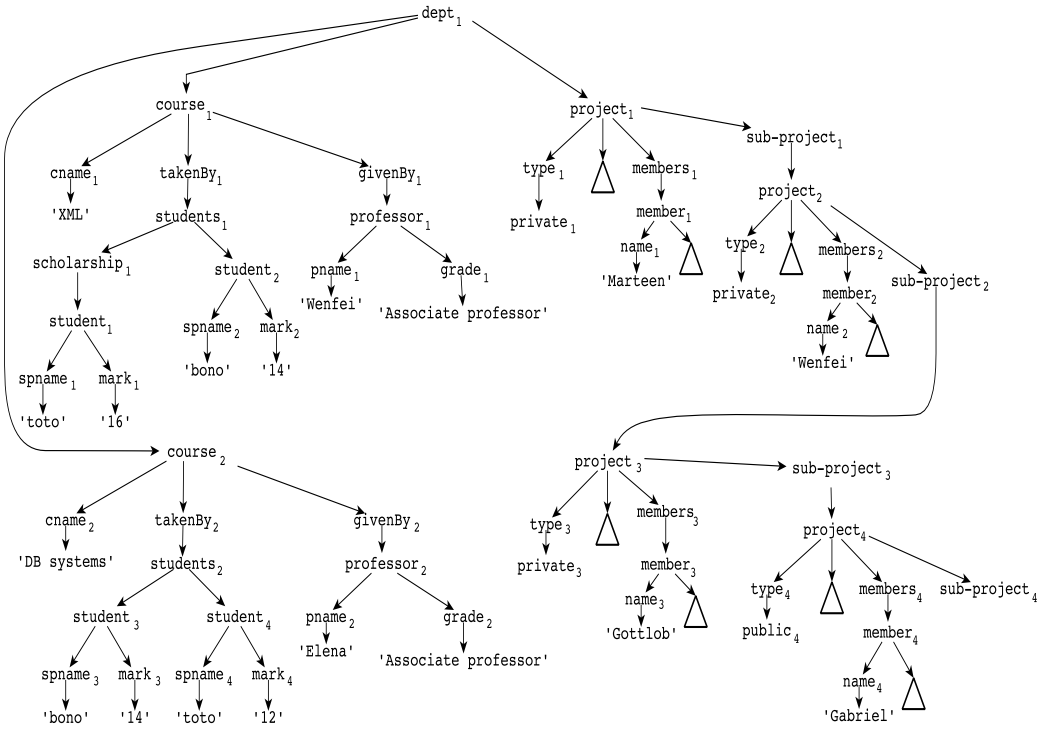


FIGURE 3.1 – Exemple de document XML *department*.

Un qualificatif q est dit *valide* à un nœud n, notée $n \vDash q$, si et seulement si l'une des conditions suivantes est remplie : (i) q est un prédicat atomique qui, lorsqu'il est évalué sur n, retourne au moins un nœud (c-à-d, il y a des nœuds accessibles à partir de n par l'intermédiaire de q) ; (ii) q est donné par α : : $text()$=c et il y a au moins un nœud, accessible selon l'axe α à partir de n, qui a un nœud texte avec la valeur c ; (iii) q est une expression booléenne et elle est évalué à vrai à n (par exemple, $n \vDash \neg(q)$ si et seulement si la requête q retourne l'ensemble vide sur n). Nous notons par $\mathcal{S}[\![Q]\!](T)$ l'ensemble des nœuds résultant de l'évaluation de la requête Q sur l'arbre XML T.

Dans ce qui suit, nous définissons des fragments plus expressifs de XPath qui sont utilisés pour surmonter la limitation de la réécriture des requêtes discutées plus tard dans la section 3.2.2.

DÉFINITION 3.5 (**FRAGMENT ÉTENDU**) Nous considérons un fragment étendu de $\mathcal{X}$, noté par $\mathcal{X}^{\Uparrow}_{[n,=]}$, et défini comme suit :

$$
\begin{array}{l}
p \;:=\; \alpha::\eta \;\mid\; p[q]\cdots[q] \;\mid\; p/p \;\mid\; p \cup p \;\mid\; p[1] \\
q \;:=\; p \;\mid\; p{=}c \;\mid\; q \wedge q \;\mid\; q \vee q \;\mid\; \neg\,(q) \;\mid\; p = \varepsilon::* \\
\alpha \;:=\; \varepsilon \;\mid\; \downarrow \;\mid\; \downarrow^{+} \;\mid\; \downarrow^{*} \;\mid\; \uparrow \;\mid\; \uparrow^{+} \;\mid\; \uparrow^{*}
\end{array}
$$

Le fragment $\mathcal{X}$ est enrichi par trois axes ascendants, à savoir $\uparrow$ (*parent*), $\uparrow^{+}$ (*ancestor*), et $\uparrow^{*}$ (*ancestor-or-self*), ainsi que les prédicats concernant la *position* du nœud et la *comparaison des nœuds*. ■

En général, le prédicat de la position, défini par $[k]$ ($k \in \mathbb{N}$), est utilisé pour renvoyer le k-ème nœud à partir d'un ensemble ordonné de nœuds [9]. Par exemple, puisque nous modélisons

des documents XML comme des arbres ordonnés, la requête $\downarrow :: *[2]$ à un nœud n retourne son second nœud enfant. La *comparaison de nœud* est utilisée pour vérifier l'identité de deux nœuds. Plus précisément, le prédicat $[p_1 = p_2]$ est valide à un nœud n seulement si l'évaluation des requêtes XPath à droite et à gauche retournent le même nœud unique. Notons que si p_1 et/ ou p_2 se rapportent à plus d'un seul nœud alors une erreur dynamique est déclenchée. A titre d'exemple, le prédicat $[\uparrow^* :: *[1] = \varepsilon :: *]$ est valable à tout nœud n puisque les requêtes $\uparrow^* :: *[1]$ et $\varepsilon :: *$ sont équivalentes et renvoient le même nœud unique sur tout nœud de contexte.

Pour notre travail, nous nous limitons à des usages très simples des prédicats relatifs à la position et la comparaison des nœuds, telles que, respectivement, les formes $[1]$ et $[p = \varepsilon :: *]$. Ces formes seront suffisantes pour résoudre le problème de la réécriture des requêtes XPath comme nous le verrons par la suite. En outre, sur la base de ces restrictions notre fragment de la définition 3.5 nécessite moins de temps d'évaluation par rapport au fragment global.

Nous résumons nos extensions de fragment $\mathcal{X}$ par les sous-ensembles suivants : $\mathcal{X}^{\Uparrow}$ ($\mathcal{X}$ avec des axes ascendants), $\mathcal{X}^{\Uparrow}_{[n]}$ ($\mathcal{X}^{\Uparrow}$ avec le prédicat de position), et $\mathcal{X}^{\Uparrow}_{[n,=]}$ ($\mathcal{X}^{\Uparrow}_{[n]}$ avec la comparaison des nœuds). Notons que nous utilisons le fragment $\mathcal{X}$ pour spécifier les politiques de sécurité, ainsi que formuler les requêtes des utilisateurs.

Requêtes Regular XPath. Nous parlons ici de l'extension des requêtes XPath avec l'*opérateur de fermeture transitive* " * ". Par exemple, la *fermeture transitive et réflexive* de la requête XPath $\downarrow :: A$, notée $(\downarrow :: A)$ *, est l'union infinie (où ϵ désigne la requête vide) : $\epsilon \cup \downarrow :: A \cup \downarrow :: A/\downarrow :: A \cup \downarrow :: A/\downarrow :: A/\downarrow :: A \cup \ldots$. La fermeture transitive est une opération naturelle et utile qui permet de définir des chemins récursifs, et plusieurs langages pour les données semi-structurées la supportent (par exemple, les requêtes SQL récursives [39, 66]). La principale préoccupation ici est que le standard XPath [22, 9] ne supporte pas la fermeture transitive, et donc des chemins récursifs arbitraires ne sont pas exprimable dans ce langage [124].

En dépit de ses avantages pratiques, aucun moteur XML ne prend en charge l'opérateur de fermeture transitive. Ceci a amené les chercheurs à définir certaines extensions du langage XPath afin de permettre la définition des expressions de chemins récursifs. Une étude utile est donnée dans [86] sur les propriétés théoriques de XPath 1.0 étendues avec des expressions régulières de chemin. Sur la base de leurs définitions, la classe de requêtes XPath régulières, notée $\mathcal{X}_{reg}$, est définie comme suit ($p*$ désigne une répétition infinie de la requête p) :

$$
\begin{array}{l}
p \;:=\; \alpha :: ntst \mid p* \mid p[q]\cdots[q] \mid p/p \mid p \cup p \\
q \;:=\; p \mid p{=}c \mid q \wedge q \mid q \vee q \mid \neg\,(q) \\
\alpha \;:=\; \varepsilon \mid \downarrow \mid \downarrow^{+} \mid \downarrow^{*}
\end{array}
$$

Sur la base de l'algorithme d'évaluation formelle des requêtes de $\mathcal{X}_{reg}$ décrit dans [38], qui repose sur les automates MFAs (*Mixed Finite state Automatas*), nous obtenons les résultats suivants :

PROPOSITION 3.1 Etant donnée une requête Q à partir de $\mathcal{X}_{reg}$ définie sur la DTD D, Q peut être traduite en un équivalent automate MFA M de taille au plus $O(|Q|.|D|)$ en un temps au plus de $O(|Q|^2.|D|^2)$. De plus, M peut être évalué sur toute instance T de D en temps et espace d'au plus de $O(|Q|^2.|D|^2 + |Q|.|D|.|T|)$. ■

3.2.2 Problématique

Nous présentons dans cette section notre problème de base, à savoir répondre à des requêtes XML posées sur des vues récursives de sécurité.

Vues XML de sécurité. La notion de *vue de sécurité*, introduite par [117], consiste à définir pour chaque groupe d'utilisateurs une vue pour le document XML sous-jacent qui affiche toutes (et seules) les parties autorisées pour ces utilisateurs. Fan et al. [36] ont raffiné cette notion en introduisant d'abord un langage pour spécifier les politiques de contrôle d'accès et un algorithme de réécriture pour faire appliquer ces politiques. Les vues de sécurité sont maintenant la base de la plupart des modèles de contrôle d'accès XML existants [36, 37, 38, 99, 28, 31, 68, 49, 76, 125].

Soit T un document XML qui est conforme à une DTD D. Ce document peut être interrogé simultanément par différents utilisateurs ayant différents privilèges d'accès. Une politique de contrôle d'accès, telle que définie dans [36], est une extension du document DTD D associant des *conditions d'accessibilité* aux types d'éléments de D. Ces conditions spécifient les éléments de T auxquels les utilisateurs ont accès. Plus précisément, une spécification d'accès est définie comme suit[36] :

Définition 3.6 (**Spécification des accès**) Une *spécification d'accès* S est le couple (D, $\boldsymbol{ann}$) composé d'une DTD D et d'une application partielle $\boldsymbol{ann}$ telles que, pour chaque règle de production $A \rightarrow P(A)$ et chaque type d'élément B dans $P(A)$, $\boldsymbol{ann}(A, B)$ est une annotation de la forme :

$$\boldsymbol{ann}(A, B) := Y \mid N \mid [Q]$$

où $[Q]$ est un prédicat XPath. La racine est annotée par défaut Y. ■

Intuitivement, la spécification des valeurs Y, N, et $[Q]$ indique que les enfants B des éléments A dans une instanciation de D sont respectivement *accessibles*, *inaccessibles*, ou *conditionnellement accessibles*. Si $\boldsymbol{ann}$ (A, B) n'est pas explicitement définie, B hérite de l'accessibilité de A. D'autre part, si $\boldsymbol{ann}(A, B)$ est explicitement définie alors B peut *outrepasser* l'accessibilité héritée de A.

Exemple 3.2 Nous considérons la DTD *département* de l'exemple 3.1 et nous définissons certains privilèges d'accès pour les professeurs. Supposons qu'un professeur, identifié par son nom $PNAME, peut accéder à toutes ses informations de cours, sauf l'information indiquant si un étudiant donné est titulaire d'une bourse d'études. La spécification d'accès, S=($dept$, $\boldsymbol{ann}$), correspondant à ces privilèges peut être spécifiée comme suit :

$$\boldsymbol{ann}(dept, course) = [\underbrace{\downarrow::givenBy/\downarrow::professor/\downarrow::pname = \$\text{PNAME}}_{Q_1}]$$

$$\boldsymbol{ann}(students, scholarship) = N$$

$$\boldsymbol{ann}(scholarship, student) = [\uparrow^+ :: course[Q_1]]$$

Ici $PNAME est traité comme un paramètre constant, à savoir quand une valeur concrète (par exemple, $Mahfoud$), est affectée à $PNAME, la spécification définit la politique de contrôle d'accès pour le professeur $Mahfoud$. Notons que $\boldsymbol{ann}(course, takenBy)$ n'est pas explicitement définie, ce qui signifie que, dans une instanciation de la DTD *département*, un élément $takenBy$ hérite de son accessibilité à partir de son élément parent $course$. Cette accessibilité est soit Y ou N selon l'évaluation du prédicat $[Q_1]$ au niveau de l'élément $course$; de même pour $cname$, $students$, $givenBy$ et ses descendants. L'annotation $\boldsymbol{ann}(students, scholarship)$=$N$ sur un élément de $scholarship$ outrepasse l'accessibilité héritée de son ancêtre $course$ pour rendre cet élément de $scholarship$ inaccessible. De plus, l'annotation $\boldsymbol{ann}(scholarship, student)$=$[\uparrow^+ :: course[Q_1]]$ l'emporte sur l'accessibilité N héritée de l'élément $scholarship$ et indique que $student$, qui sont

les enfants de *scholarship*, sont conditionnellement accessibles (c-à-d, ils sont accessibles si le professeur est autorisé à accéder à leur ancêtre *course*). ■

Les politiques de contrôle d'accès basées sur la spécification de la définition 3.6 sont appliquées à travers la dérivation d'une *vue de sécurité* [36]. Une vue de sécurité est une extension du document XML initial et la DTD qui : *1*) peut être automatiquement dérivée d'une spécification d'accès, *2*) affiche à l'utilisateur toutes (et uniquement) les parties accessibles du document XML, *3*) fournit à l'utilisateur un schéma de toutes ses données accessibles afin qu'il puisse formuler et optimiser ses requêtes, et *4*) permet une traduction sûre des requêtes des utilisateurs pour empêcher l'accès aux données sensibles [12]. Plus formellement, une vue de sécurité XML est définie comme suit [36] :

Définition 3.7 (**Vue de sécurité**) Etant donnée une spécification d'accès S=(D, $\boldsymbol{ann}$) définie sur une DTD D non récursive. Une *vue de sécurité* est un couple (D_v, σ) où D_v est une vue de D qui présente le schéma des données que l'utilisateur est autorisées à accéder, et σ est une fonction définie comme suit : $\sigma(A, B)$ est l'ensemble des expressions XPath qui, une fois, évaluées sur un élément A d'un arbre XML T de $\mathcal{T}(D)$ elles retournent tous ses descendants B. En d'autres termes, σ associe toute instance de D à une instance de D_v qui contient seulement des données accessibles. ■

La vue DTD D_v est donnée à l'utilisateur pour la formulation et l'optimisation de ses requêtes. Cependant, l'ensemble des expressions XPath définies par σ sont cachées à l'utilisateur et utilisées pour extraire pour tout arbre XML T de $\mathcal{T}(D)$, une vue T_v de T qui contient uniquement les nœuds autorisés et accessibles de T.

Exemple 3.3 Considérons la spécification d'accès S=($dept$, $\boldsymbol{ann}$) pour l'exemple 3.2. La DTD vue $dept_v$=(Σ_v, $dept$, P_v) extraite de la DTD *department* peut être calculée en éliminant le type d'élément *scholarship*, c-à-d, $\Sigma_v := \Sigma \setminus \{scholarship\}$, et en changeant la définition des types d'éléments *dept* et *students* comme suit :

$$\begin{array}{lcl} P_v(dept) & := & (course*, project*) \\ P_v(students) & := & (student+) \\ P_v(A) & := & P(A), \text{ pour tous les types d'éléments } A \text{ restants dans } \Sigma_v \end{array}$$

La fonction σ est définie sur les règles de production de $dept_v$ comme suit (voir l'exemple 3.2 pour la définition de $[Q_1]$) :

$$\begin{array}{lcl} dept & \rightarrow & P_v(dept) : \\ & & \sigma(dept, course) = \downarrow :: course[Q_1] \\ & & \sigma(dept, project) = \downarrow :: project \\ students & \rightarrow & P_v(students) : \\ & & \sigma(students, student) = \\ & & \downarrow :: student \cup \downarrow :: scholarship/\downarrow :: student[\uparrow^+ :: course[Q_1]] \\ A & \rightarrow & P_v(A) : \text{(pour chaque type d'élément restant } A \text{ dans } \Sigma_v) \\ & & \sigma(A, B) = \downarrow :: B \text{ (pour chaque type d'enfant } B \text{ dans } P_v(A)) \end{array}$$

En utilisant la vue de sécurité résultante V=($dept_v$, σ), la vue du document XML de la figure 3.1 est dérivée comme illustré à la figure 3.2. Cette vue montre les nœuds du XML originel qui sont seulement accessible par rapport à la spécification S=($dept$, $\boldsymbol{ann}$). Notons que tous les descendants de l'élément $project_1$ restent inchangés. ■

12. Cette traduction est nécessaire que si les vues des données sont virtuelles, à savoir non matérialisées.

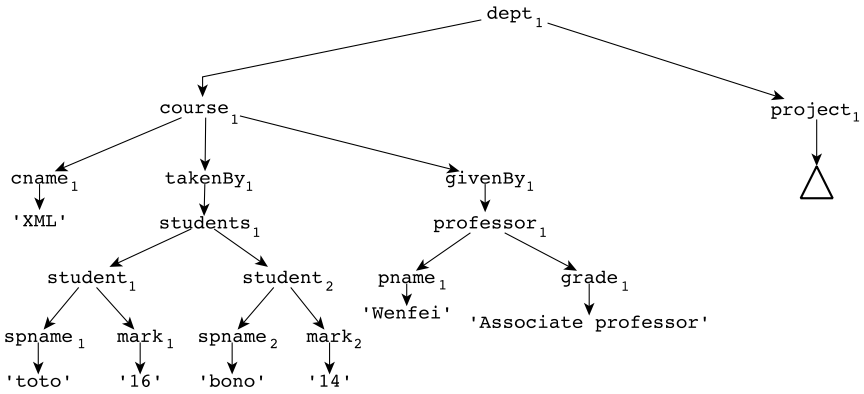


FIGURE 3.2 – Vue du document XML *dept* par rapport à la politique de l'exemple 3.2.

Étant donnée une vue de sécurité V=(D_v, σ) définie pour une spécification d'accès S=(D, ***ann***). Pour chaque instance T de D et sa vue T_v calculée en utilisant la fonction σ, on peut soit matérialiser T_v et évaluer les requêtes des utilisateurs directement sur elle [68, 40], ou garder T_v virtuelle pour des raisons de performance [38, 28, 49, 76]. En cas de vues virtuelles, le principe de la *réécriture des requêtes* est utilisé pour traduire chaque requête Q, définie dans D_v sur la vue virtuelle T_v, en une requête correcte Q^t définie dans D sur le document originel T telle que : l'évaluation de Q sur T_v renvoie le même ensemble de nœuds que l'évaluation de la requête réécrite Q^t sur T.

EXEMPLE 3.4 Considérons la requête $\downarrow :: dept/\downarrow :: course$ du professeur *Wenfei* définie sur la vue de la figure 3.2. En utilisant la vue de sécurité de l'exemple 3.3, cette requête peut être réécrite en $\downarrow :: dept/\sigma(dept, course)$, qui est égale à :

$$\downarrow :: dept/\downarrow :: course[\downarrow :: givenBy/\downarrow :: professor/\downarrow :: pname=\text{“}Wenfei\text{”}]$$

L'évaluation de cette requête sur le document XML originel de la figure 3.1 retourne seulement les éléments accessibles *course* (c-à-d, $course_1$). ■

Comme la plupart des approches existantes pour la sécurisation des données XML sont basées sur le modèle de vue de sécurité, nous discutons par la suite les principales limites de ce modèle.

Inconvénients des vues de sécurité. Nous étudions seulement le cas d'interrogation de données XML virtuelles [13]. Nous discutons, dans ce qui suit, les obstacles rencontrés lors de la manipulation des vues récursives [14] pour la réécriture des requêtes. Même si la réécriture de requêtes XPath est assez simple pour les vues de sécurité non récursives, certains obstacles peuvent être dus à des vues récursives qui rendent ce processus de réécriture impossible pour une certaine classe de requêtes XPath. Plus précisément, le processus de réécriture est basé sur la définition de la fonction σ qui, en cas de DTD récursives, ne peut pas être définie dans XPath comme nous allons le montrer dans l'exemple suivant.

EXEMPLE 3.5 Considérons la DTD *department* de l'exemple 3.1 et nous supposons qu'un personnel d'un département, identifié par son nom $PNAME, peut accéder à l'information de tout

13. Les problèmes liés à la manipulation des vues matérialisées XML [68, 40] sont en dehors du sujet d'intérêt de notre travail.

14. Une vue de sécurité est *récursive* si elle est définie sur une DTD récursive.

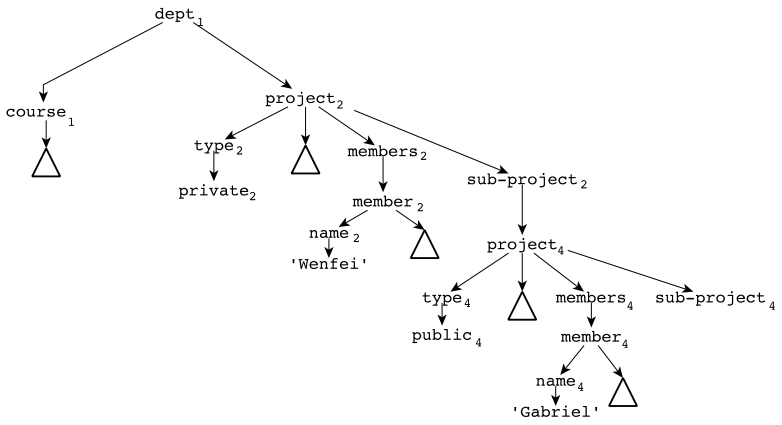


FIGURE 3.3 – Vue du document XML *dept* par rapport la politique de l'exemple 3.5.

projet dans lequel il est membre ainsi que toutes les informations concernant les projets publics. La spécification d'accès, S=(*dept*, ***ann***), correspondant à ces privilèges, est définie comme suit :

$$\textbf{\textit{ann}}(dept, project) = \textbf{\textit{ann}}(sub\text{-}project, project) = \underbrace{[\downarrow::type/\downarrow::public \vee \downarrow::members/\downarrow::member[\downarrow::name = \$\text{PNAME}]]}_{Q_2}$$

Notons que si le prédicat $[Q_2]$ est valide au niveau de l'élément *project* alors tous ses descendants héritent de cette accessibilité sauf les éléments *sub-project* qui pourraient l'outrepasser (cela dépend de l'évaluation de $[Q_2]$). Considérons le cas du professeur *Wenfei*. La vue du document XML, de la figure 3.1 est dérivée et illustrée dans la figure 3.3. Etant donné un élément accessible *dept*, il y a un ensemble infini de chemins qui relient cet élément à ces enfants accessible de type *project*. Plus précisément, $\sigma(dept, project)$ peut être définie en utilisant la fermeture transitive "$*$" avec : $\sigma(dept, project) = (\downarrow :: project[\neg(Q_2)]/\downarrow :: sub\text{-}project)*/\downarrow :: project[Q_2]$.

Le chemin récursif $(\downarrow :: project[\neg(Q_2)]/\downarrow :: sub\text{-}project)*$ est défini sur seulement des éléments inaccessibles. Ainsi, l'expression $\sigma(dept, project)$ doit extraire, sur chaque élément accessible de type *dept* dans les données d'origine, les descendants accessibles de type *project* qui apparaissent dans la vue des données comme les enfants immédiats de cet élément *dept*. En d'autres termes, un élément m de type *project* est affiché dans la vue de la figure 3.3 comme un enfant immédiat d'un élément n de type *dept* si et seulement si : m et n sont à la fois accessibles dans l'arbre originel de la figure 3.1, et soit m est un enfant immédiat de n ou séparé de n par seulement des éléments inaccessibles. Prenons le cas des éléments $dept_1$ et $project_2$ de l'arbre de la figure 3.1. Après avoir caché l'élément inaccessible $project_1$, $project_2$ apparaît dans la vue de la figure 3.3 comme enfant immédiat de $dept_1$. Le même principe est appliqué pour les éléments $project_2$ et $project_4$. ■

Les auteurs de [86] ont montré que l'opérateur "$*$" (connu en anglais comme "kleen star") ne peut pas être exprimé en XPath. Pour cette raison, la fonction σ de l'exemple 3.5 ne peut pas être définie dans le XPath standard qui rend le processus de réécriture des requêtes plus difficile (voire même impossible). Nous sommes principalement motivés par l'étude de la *fermeture* d'une classe importante de requêtes XPath (notée $\mathcal{X}$) pour la réécriture des requêtes, à savoir si toutes les requêtes de cette classe peuvent être réécrites sur des vues de sécurité arbitraires (récursives ou non). Nous définissons formellement la *propriété de fermeture* comme suit :

DÉFINITION 3.8 Un langage L pour interroger des données XML est *clos* pour la réécriture des requêtes s'il existe une fonction $\mathcal{R} : L \rightarrow L$ qui, pour toute spécification d'accès S=(D,***ann***) et pour toute vue D_v d'une DTD D, traduit chaque requête Q de L définie sur D_v en une autre requête $\mathcal{R}(Q)$ définie dans L sur D tel que : pour tout arbre $T \in \mathcal{T}(D)$ muni de sa vue virtuelle T_v, $\mathcal{S}[\![\mathcal{R}(Q)]\!](T)$=$\mathcal{S}[\![Q]\!](T_v)$. ■

Notons que Fan et al. [36] montrent que le fragment $\mathcal{X}$ (voir la définition 3.4) est fermé pour la réécriture des requêtes dans le cas de vues de sécurité non récursives. Toutefois, cette fermeture n'est pas possible en cas de récursivité, comme le montre le théorème suivant :

THÉORÈME 3.1 ([38, 64]) En cas de vues de sécurité récursives, le fragment XPath $\mathcal{X}$ n'est pas fermé pour la réécriture des requêtes. ■

Enfin, il faut souligner qu'aucune solution pratique n'existe pour répondre aux requêtes XML sur des vues de sécurité récursives. Certains résultats théoriques existent qui sont basés sur le langage XPath régulier ("Regular XPath") qui permet la définition des requêtes récursives. Selon [38, 49], le fragment $\mathcal{X}_{reg}$ de la section 3.2.1 est fermé pour la réécriture. Cependant, certains inconvénients majeurs sont à noter : il n'y a pas de solution standard qui existe pour évaluer les requêtes régulières. L'évaluation pour le fragment XPath régulier est plus coûteuse que le XPath standard. En plus, tous les systèmes de SGBD à base de XML supportent le langage standard de XPath. Les résultats trouvés autour de XPath régulier sont encore impraticables.

3.3 Modèle de Contrôle d'Accès

Nous présentons dans ce qui suit la DTD *hospital* qui correspond à de vraies données médicales gérées par un hôpital [115]. Cet exemple sera utilisé dans le reste de ce chapitre. Toutes les preuves des lemmes et théorèmes sont données dans [78, 82].

EXEMPLE 3.6 La DTD *hospital* (Σ,P,*hospital*) est définie avec les règles de production (les définitions des éléments de type `str` sont omises) suivantes :

hospital → (*department**)
department → (*name*, *patient**)
patient → (*pname*, *wardNo*, *parent*?, *sibling*?, *symptoms**, *intervention*)
parent → (*patient**)
sibling → (*patient**)
symptoms → (*symptom**)
intervention → (*doctor*, *treatment*)
doctor → (*dname*, *specialty*)

treatment → (*type*, *Tresult**, *diagnosis*)
diagnosis → (*Dresult**, *implies*?)
implies → (*treatment* | *intervention*)

La DTD de l'hôpital consiste en une liste de départements (*department*). Chaque département (défini par son nom (*name*)) possède une liste de patients (*patient*) résidant dans l'hôpital. Le patient est connu par son nom (*name*), son numéro (*wardNo*), et les antécédents médicaux de la famille au moyen des relations définies récursivement des parents (*parent*) et des frères et soeurs (*sibling*), ainsi qu'une liste de symptômes (*symptoms*). L'hospitalisation est marquée

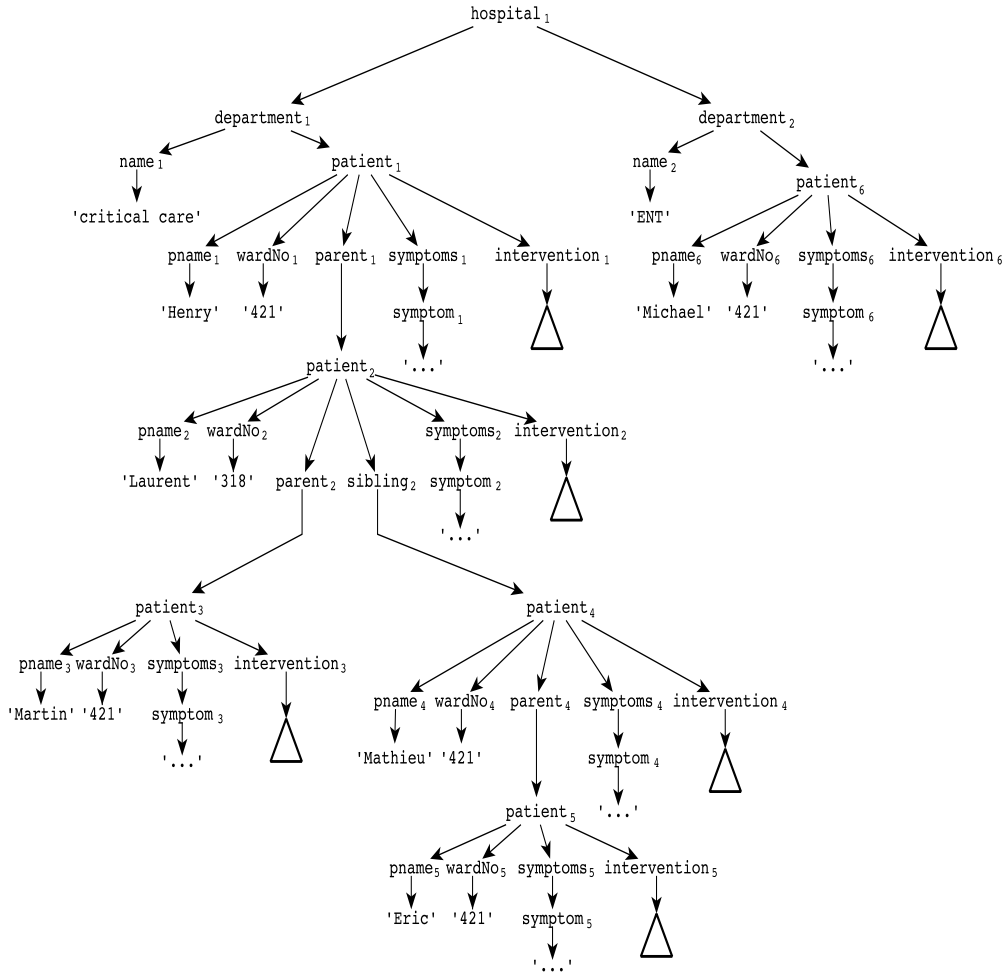


FIGURE 3.4 – Exemple des données de l'hôpital.

par l'intervention (*intervention*) d'un ou de plusieurs médecins en fonction de leur spécialité et de l'exigence de soins aux patients. Pour chaque intervention, l'hôpital maintient également le responsable (*doctor*) (représenté par son nom (*dname*) et sa spécialité (*speciality*)), et le traitement appliqué (*treatment*). Un traitement est décrit par son type (*type*), une liste de résultat (*TResult*), et il est suivi par une phase de diagnostic (*diagnostic*). Selon les résultats du diagnostic (*Dresult*), soit un autre traitement est prévu ou l'intervention d'un autre médecin/spécialiste/ expert est sollicitée [15]. Une instance de cette DTD de l'hôpital est donnée à la figure 3.4 (le contenu de certains textes sont abrégées par «...»). ■

3.3.1 Spécification de la politique d'accès

Spécification d'accès. Fan et al. [36] ont proposé le premier langage pour la spécification des politiques de contrôle d'accès XML par l'annotation des grammaires DTD. En outre, les auteurs

15. Selon [115], cela peut se produire lorsque le traitement nécessaire est en dehors du domaine d'expertise du médecin responsable actuel.

de [68] ont étudié la classification de ces politiques par rapport à l'annotation par défaut, l'héritage et l'outrepassement d'annotations. Dans ce travail, nous considérons le cas des politiques *top-down* où le nœud racine de l'arborescence XML est accessible par défaut et chaque nœud intermédiaire peut soit hériter l'annotation de son nœud parent ou l'outrepasser (voir la définition 3.6). Bien que les deux langages de spécification d'accès définis dans [36, 68] sont basés sur le même principe (c-à-d, annoter les types d'éléments de la DTD avec Y, N et $[Q]$), il y a une différence significative dans l'utilisation des annotations conditionnelles (c-à-d, des annotations de la forme $[Q]$). Nous considérons l'exemple suivant pour plus de détails :

EXEMPLE 3.7 Supposons qu'il y a deux annotations $\boldsymbol{ann}(A, B)=[\neg\ (\downarrow :: D)]$ et $\boldsymbol{ann}(C, D)=Y$ définies sur un simple arbre XML composé par seulement un chemin :

$$R \rightarrow A \rightarrow B \rightarrow C \rightarrow D$$

Notons que le prédicat $[\neg\ (\downarrow :: D)]$ est invalide au niveau du nœud d'élément B. Selon [36], tout sous-arbre enraciné par cet élément B est inaccessible et donc la seconde annotation qui concerne le nœud d'élément D ne prend pas effet. Selon [68], cependant, le nœud d'élément D remplace la valeur N héritée de son élément ancêtre B et devient accessible. ■

En général, soit n un nœud d'élément qui est concerné par une annotation de la forme $[Q]$. Pour les premiers travaux [36], si $n \not\models Q$, alors tout sous-arbre enraciné à n est inaccessible et aucune annotation définie sur les descendants de n ne peut prendre effet. Pour le deuxième travail [68], cependant, même si $n \not\models Q$, les descendants de n peuvent remplacer cette annotation pour devenir accessible.

Nous supposons que les deux définitions sont des applications pratiques utiles et peuvent nécessiter l'application des deux types d'annotations, même au sein du même scénario. Pour cette raison, nous présentons un langage de spécification d'accès raffiné et plus expressif dont les spécifications d'accès sont définies comme suit :

DÉFINITION 3.9 (**SPÉCIFICATION D'ACCÈS ÉTENDUE**) Nous définissons une *spécification d'accès* S comme une paire $(D, \boldsymbol{ann})$ composée d'une DTD D et une application partielle $\boldsymbol{ann}$ tels que : pour chaque règle de production $A \rightarrow P(A)$ et chaque type d'élément B dans $P(A)$, $\boldsymbol{ann}$ est une annotation de la forme :

$$\boldsymbol{ann}(A, B) := Y \mid N \mid [Q] \mid N_h \mid [Q]_h$$

où $[Q]$ est un prédicat XPath. Les annotations de la forme N_h et $[Q]_h$ sont appelées *downward clos*. Le type de racine de D est annoté de Y par défaut. ■

A partir de la définition 3.6, rappelons que les annotations de la forme Y, N et $[Q]$ indiquent respectivement qu'un élément de B, enfant d'un élément A, est *accessible*, *inaccessible*, ou *conditionnellement accessible*. Nous permettons l'outrepassement entre les annotations des trois formes précédentes. En d'autres termes, chaque élément concerné par une annotation de la forme Y, N, ou $[Q]$ l'emporte sur son annotation héritée si elle est définie avec l'un de ces trois formes. Les valeurs particulières de spécification N_h et $[Q]_h$ indiquent respectivement que l'outrepassement est *révoqué* ou *conditionnellement autorisé*. Plus précisément, soient $n_1, \ldots, n_l$ $(l \geq 2)$ des éléments de type, respectivement, $A_1, \ldots, A_l$, où chaque n_i $(1 \leq i < l)$ est un nœud parent de n_{i+1}. L'annotation $\boldsymbol{ann}(A_1, A_2) = N_h$ indique que tout sous-arbre enraciné à n_2 est inaccessible et aucun élément sous n_2 peut remplacer cette annotation. Ainsi, si certaine annotation $\boldsymbol{ann}(A_i,$

$A_{i+1})=Y|[Q]$ est explicitement définie alors le nœud élément n_{i+1} reste inaccessible même si $n_{i+1} \vDash Q$. Toutefois, l'annotation $\boldsymbol{ann}(A_1, A_2)=[Q_2]_h$ indique que les annotations définies sur les types descendants de A_2 prendront effet seulement si Q_2 est valide. En d'autres termes, compte tenu de l'annotation $\boldsymbol{ann}(A_i, A_{i+1})=Y$ (resp. $[Q_{i+1}]$), l'élément nœud n_{i+1} est accessible si et seulement si : $n_2 \vDash Q_2$ (resp. $n_2 \vDash Q_2 \wedge n_{i+1} \vDash Q_{i+1}$).

Exemple 3.8 Supposons que l'hôpital veut imposer certaines restrictions qui permettent à une certaine infirmière d'accéder seulement à l'information des patients qui sont traités dans le département des *soins intensifs* et résidant dans le service *421*. En outre, toutes les données de la fratrie devraient être inaccessibles. Cette politique peut être spécifiée en utilisant la spécification d'accès $S = (D, \boldsymbol{ann})$ où D est la DTD de l'hôpital et la fonction $\boldsymbol{ann}$ définit les trois annotations suivantes :

R_1 : $\boldsymbol{ann}(hospital, department)=[\underbrace{\downarrow::name = \text{“critical care”}}_{Q_1}]_h$

R_2 : $\boldsymbol{ann}(department, patient)=\boldsymbol{ann}(parent, patient)=[\underbrace{\downarrow::wardNo = \text{“421”}}_{Q_2}]$

R_3 : $\boldsymbol{ann}(patient, sibling)=N_h$

Selon cette spécification, la vue des données de la figure 3.4 est extraite et représentée sur la figure 3.5. Cette vue affiche seulement les données autorisées à l'infirmière. Toutes les données du département *ENT* sont cachées, à savoir le sous-arbre enraciné à l'élément $departement_2$. Comme R_1 est downward close et $departement_2 \nvDash Q_1$, alors R_2 ne peut pas être appliquée au niveau de l'élément $patient_6$ qui reste inaccessible même avec $patient_6 \vDash Q_2$. Notons que $departement_1 \vDash Q_1$, ce qui signifie que l'élément $departement_1$ est inaccessible et l'outrepassement des annotations est accordé pour ses descendants. Ainsi, les éléments $patient_1$ et $patient_3$ sont accessibles avec leurs enfants immédiats comme Q_2 est valide au niveau de ces éléments, alors que l'élément $patient_2$ (avec $patient_2 \nvDash Q_2$) remplace l'annotation Y héritée à partir de $patient_1$ et devient inaccessible avec tous ses enfants immédiats. De cette façon, l'élément $patient_3$ apparaît à la vue de la figure 3.5 comme enfant immédiat de $parent_1$. Enfin, comme l'élément $sibling_2$ est concerné par l'annotation downward close R_3 avec la valeur N_h, alors tout sous-arbre enraciné à $sibling_2$ est inaccessible et l'annotation R_2 ne peut prendre effet sur les éléments $patient_4$ et $patient_5$. ■

Notre langage de spécification d'accès est plus expressif que ceux qui existent déjà dans la littérature. Les spécificités de ces approches peuvent être décrites dans notre langage en utilisant seulement quelques valeurs d'annotation comme indiqué dans la table 3.1. Par exemple, la politique de l'exemple 3.8 ne peut pas être spécifiée dans le fragment $\mathcal{X}$ en utilisant les langages de spécification présentés dans [36, 68]. Cela peut être fait en utilisant un fragment plus expressif, comme $\mathcal{X}^{\Uparrow}$, mais les annotations peuvent être plus verbeuses et difficiles à gérer.

La *complétude* et la *consistance* des politiques de contrôle d'accès ont été définies dans [106] comme suit. Soient P une politique et T un arbre XML. Si un nœud n dans T n'est pas concerné par toute règle d'accès de P alors P est *incomplète*. En outre, s'il y a à la fois une règle négative et une règle positif pour le même nœud n (à savoir, n est à la fois accessible et inaccessible) alors P est *inconsistante*. Considérons nos spécifications d'accès de la définition 3.9, nous définissons les notions de *complétude* et *consistance*, dans le même sens que [36, 68], comme suit :

Définition 3.10 Etant donnés une spécification d'accès $S=(D, \boldsymbol{ann})$ et un arbre XML $T \in \mathcal{T}(D)$. Nous disons que S est *complète* et *consistante* ssi *l'accessibilité* de chaque nœud dans T est uniquement défini, c-à-d, il est soit *accessible* soit *inaccessible*. ■

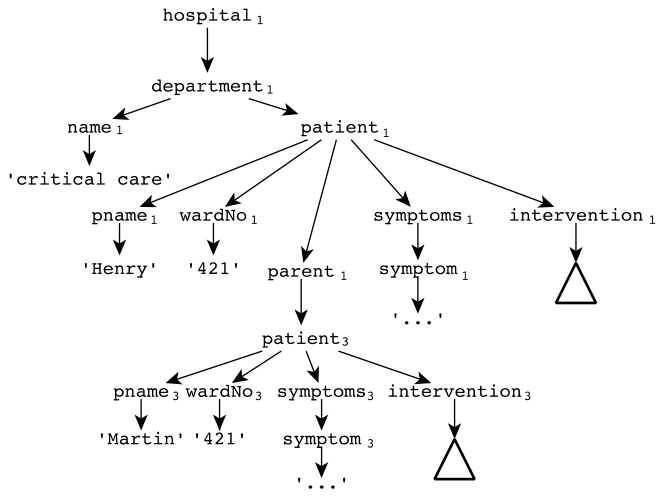


FIGURE 3.5 – Vue de l'arbre de la figure 3.4 par rapport la politique de l'exemple 3.8.

Politique d'accès	**Valeurs requises**					**Remarque**
	Y	N	N_h	$[Q]$	$[Q]_h$	
[36, 37, 38]	✓	✓			✓	
[68]	✓	✓		✓		cas des politiques top-down
[49]	✓	✓		✓		
[28]	✓		✓		✓	
[76]	✓	✓				
[44]	✓		✓		✓	*Priorité à la révocation* pour la résolution des conflits dans la politique
[90]	✓		✓		✓	*Révocation vers le bas* pour la consistance

TABLE 3.1 – Approches existantes spécifiées avec notre langage.

PROPOSITION 3.2 Les politiques de contrôle d'accès basées sur la définition 3.9 sont *complètes* et *consistantes*. ■

Accessibilité. La mise en œuvre de nos politiques de contrôle d'accès repose principalement sur la définition de l'*accessibilité du nœud*. Inspiré de [44, 49], nous définissons un filtre XPath unique, qui peut être construit pour toute spécification d'accès, et vérifie si un nœud XML donné est *accessible* ou non par rapport à cette spécification.

DÉFINITION 3.11 Soit n un élément de type B qui est le fils d'un élément de type A. Une annotation $\boldsymbol{ann}(A, B)$ est *valide* ssi $\boldsymbol{ann}(A, B)=Y|[Q]|[Q]_h$ avec $n \vDash Q$. Sinon, elle est *invalide*, c-à-d, $\boldsymbol{ann}(A, B)=N|N_h|[Q]|[Q]_h$ avec $n \nvDash Q$. ■

Si $\boldsymbol{ann}(A, B)=[Q]_h$ avec $n \vDash Q$ (resp. $\boldsymbol{ann}(A, B)=N_h|[Q]_h$ avec $n \nvDash Q$) alors nous parlons

d'annotation downward close qui est *valide* (resp. *invalide*). Compte tenu de ce qui précède, nous définissons l'accessibilité du nœud comme suit :

DÉFINITION 3.12 Soient une spécification d'accès S=(D, $\boldsymbol{ann}$), T est une instance de D et n un élément de type B dans T ayant un parent de type A. L'élément n est *accessible* par rapport à S ssi les conditions suivantes sont satisfaites :

i) Soit il existe une annotation explicite $\boldsymbol{ann}(A, B)$ qui est valide à n, ou soit la première annotation définie explicitement sur les ancêtres de n est valide.

ii) Il n'y a aucune annotation downward close et invalide définie sur les ancêtres de n. ■

Plus précisément, considérons les nœuds $n_1, \ldots, n_k$ ($k \geq 2$) de types respectivement $A_1, \ldots, A_k$ où n_1 est le nœud racine. Prenons le cas de l'élément nœud n_k, la condition (i) de la définition 3.12 se réfère à l'un des trois cas suivants :

a) Seulement l'annotation par défaut $\boldsymbol{ann}(A_1)$=Y est définie sur les types $A_1, \ldots, A_k$. Ainsi, n_k hérite son accessibilité de la racine n_1.

b) L'annotation $\boldsymbol{ann}(A_{k-1}, A_k)$ est explicite et valide au niveau de n_k.

c) L'annotation $\boldsymbol{ann}(A_{i-1}, A_i)$ est explicite et valide au niveau de l'élément n_i ($1 < i < k$), et aucune annotation n'est définie sur les types $A_{i+1}, \ldots, A_k$. Ainsi, n_k hérite son accessibilité à partir de son ancêtre n_i.

La condition (ii) de la définition 3.12 implique que pour toute annotation downward close $\boldsymbol{ann}(A_{i-1}, A_i)$ définie sur l'ancêtre n_i de n_k (avec $1 < i < k$), soit $\boldsymbol{ann}(A_{i-1}, A_i) \neq N_h$ ou soit $\boldsymbol{ann}(A_{i-1}, A_i) = [Q]_h$ avec $n_i \vDash Q$. Enfin, notons qu'un nœud texte est accessible ssi son élément parent est accessible.

DÉFINITION 3.13 Etant donnée une spécification d'accès S=(D, $\boldsymbol{ann}$), nous définissons deux prédicats de $\mathcal{X}^{\Uparrow}$, $\mathcal{A}_1^{acc}$ et $\mathcal{A}_2^{acc}$, comme suit :

$\mathcal{A}_1^{acc} := \uparrow^* :: *[allAnn][1][validAnn]$, où :
$allAnn := \varepsilon :: root \bigvee_{ann(A',A) \in \boldsymbol{ann}} \varepsilon :: A/\uparrow :: A'$
$validAnn := \varepsilon :: root \bigvee_{(\boldsymbol{ann}(A',A)=Y) \in \boldsymbol{ann}} \varepsilon :: A/\uparrow :: A' \bigvee_{(\boldsymbol{ann}(A',A)=[Q]|[Q]_h) \in \boldsymbol{ann}} \varepsilon :: A[Q]/\uparrow :: A'$

$\mathcal{A}_2^{acc} := \bigwedge_{(\boldsymbol{ann}(A',A)=[Q]_h) \in \boldsymbol{ann}} \neg (\uparrow^+ :: A[\neg (Q)]/\uparrow :: A') \bigwedge_{(\boldsymbol{ann}(A',A)=N_h) \in \boldsymbol{ann}} \neg (\uparrow^+ :: A/\uparrow :: A')$

Les prédicats $\mathcal{A}_1^{acc}$ et $\mathcal{A}_2^{acc}$ satisfont respectivement les conditions (i) et (ii) de la définition 3.12. ■

Le premier prédicat contrôle si le nœud n est explicitement concerné par une annotation valide (cas **b**) ou hérite son accessibilité à partir d'une annotation valide définie sur ses ancêtres (cas **a** et **c**). Le second prédicat contrôle si le nœud n n'est pas dans la portée d'une annotation downward close et invalide. Le prédicat [$allAnn$] se compose d'une disjonction de toutes les annotations, tandis que [$validAnn$] est la disjonction des annotations valides. Plus précisément, l'évaluation du prédicat $\uparrow^* :: *[allAnn]$ au niveau du nœud n renvoie un ensemble ordonné de nœuds N quit contient le nœud n et/ou certains de ses ancêtres tel que chacun est "explicitement" concerné par une annotation de S, c-à-d, $N \subseteq \{n\} \cup \boldsymbol{ancestors}(n)$ [16], et $\forall m \in N$, m est de type B et possède un nœud parent de type A où $\boldsymbol{ann}(A, B)$ est explicitement définie dans S. Le prédicat $\uparrow^* :: *[allAnn][1]$ (c-à-d, $N[1]$) renvoie le premier nœud N, c-à-d,, soit le nœud n (s'il

16. Nous utilisons $\boldsymbol{ancestors}(n)$ pour faire référence à tous les ancêtres du nœud n.

est explicitement concerné par une annotation), le premier ancêtre de n qui est explicitement concernée par une annotation, ou le nœud racine (si seulement l'annotation par défaut est définie).

Le dernier prédicat $[validAnn]$ vérifie si l'annotation définie sur le nœud $N[1]$ est valide : cela signifie que le nœud n est explicitement concerné par une annotation valide ou il hérite son accessibilité à partir de l'un de ses ancêtres qui est concerné par une annotation valide (condition (i) de la définition 3.12). L'utilisation du second prédicat $\mathcal{A}_2^{acc}$ est évidente : si $n \vDash \mathcal{A}_2^{acc}$, alors toutes les annotations downward closes définies sur $\boldsymbol{ancestors}(n)$ sont valides (condition (ii) de la définition 3.12).

LEMME 3.1 Etant donnée une spécification d'accès S=(D, $\boldsymbol{ann}$), nous définissons le *prédicat d'accessibité* $\mathcal{A}^{acc}$:=$\mathcal{A}_1^{acc} \wedge \mathcal{A}_2^{acc}$ tel que : pour tout arbre XML $T \in \mathcal{T}(D)$, un nœud n de T est accessible ssi $n \vDash \mathcal{A}^{acc}$. ■

Selon ce lemme, pour toute spécification d'accès S=(D, $\boldsymbol{ann}$) et tout arbre XML $T \in \mathcal{T}(D)$, la requête $\downarrow^* :: *[\mathcal{A}^{acc}]$ sur T retourne l'ensemble de tous les nœuds accessibles de T où $\mathcal{A}^{acc}$ est calculé par rapport à S.

EXEMPLE 3.9 Considérons la politique des infirmières définies dans l'exemple 3.8 avec les annotations suivantes d'accès :

$\boldsymbol{ann}(hospital, department)=[\underbrace{\downarrow::name = \text{“}critical\ care\text{”}}_{Q_1}]_h$

$\boldsymbol{ann}(department, patient)=\boldsymbol{ann}(parent, patient)=[\underbrace{\downarrow::wardNo = \text{“}421\text{”}}_{Q_2}]$

$\boldsymbol{ann}(patient, sibling)=N_h$

Selon ces annotations, les prédicats $\mathcal{A}_1^{acc}$ et $\mathcal{A}_2^{acc}$, qui composent $\mathcal{A}^{acc}$, sont définis comme suit :

$\mathcal{A}_1^{acc}$:= $\uparrow^* :: *[allAnn][1][validAnn]$, où :
$allAnn := \varepsilon :: root \vee \varepsilon :: department/\uparrow :: hospital \vee \varepsilon :: patient/\uparrow :: department \vee \varepsilon :: patient/\uparrow :: parent \vee \varepsilon :: sibling/\uparrow :: patient$

$validAnn := \varepsilon :: department[Q_1]/\uparrow :: hospital \vee \varepsilon :: patient[Q_2]/\uparrow :: department \vee \varepsilon :: patient[Q_2]/\uparrow :: parent \vee \varepsilon :: root$

$\mathcal{A}_2^{acc}$:= $\neg\ (\uparrow^+ :: departement[\neg\ (Q_1)]/\uparrow :: hospital) \wedge$
$\neg\ (\uparrow^+ :: sibling/\uparrow :: patient)$

Considérons le cas de l'élément $patient_1$ de la figure 3.4. Le prédicat $\uparrow^* :: *[allAnn]$ au niveau $patient_1$ retourne l'ensemble N=$\{patient_1, departement_1, hospital_1\}$ (chaque élément est concerné par une annotation explicite). Nous avons $N[1]$= $\{patient_1\}$ et le prédicat $[validAnn]$ est valide à $patient_1$ (puisque $patient_1 \vDash Q_2$). Ainsi, le prédicat $\mathcal{A}_1^{acc}$ est valide à $patient_1$. Il est clair de voir que $\mathcal{A}_2^{acc}$ est aussi valide à $patient_1$. Nous concluons que $patient_1 \vDash (\mathcal{A}_1^{acc} \wedge \mathcal{A}_2^{acc})$; ce qui signifie que l'élément $patient_1$ est accessible. Considérons maintenant l'élément $patient_2$, $\uparrow^* :: *[allAnn]$ à $patient_2$ retourne l'ensemble $N^{'}$=$\{patient_2, patient_1, departement_1, hospital_1\}$, $N^{'}[1]$=$\{patient_2\}$. Cependant, le prédicat $[validAnn]$ n'est pas valide à $patient_2$ (comme $patient_2 \nvDash Q_2$). Par conséquent, $patient_2 \nvDash \mathcal{A}_1^{acc}$ et ensuite l'élément $patient_2$ n'est pas accessible. Par ailleurs, bien que $patient_4 \vDash \mathcal{A}_1^{acc}$, $patient_4$ est inaccessible comme $patient_4 \nvDash \mathcal{A}_2^{acc}$ (c-à-d, $patient_4$ est le descendant de l'élément $sibling_2$ qui est concerné par une annotation downward close et invalide. Enfin, la requête $\downarrow^* :: *[\mathcal{A}^{acc}]$ sur la figure 3.4 donne tous les éléments accessibles qui composent la vue de la figure 3.5. ■

3.3.2 Réécriture des requêtes

Dans cette section, nous discutons le principe de base de notre approche de contrôle d'accès XML. Nous rappelons que le fragment $\mathcal{X}$ (voir la définition 3.4) est utilisé dans notre approche pour la spécification des politiques de contrôle d'accès ainsi que pour la formulation des requêtes des utilisateurs. Cependant, nous utilisons des fragments étendus de XPath pour surmonter le problème de réécriture des requêtes XPath sur des vues de sécurité présenté dans la section 3.2.2. Plus précisément, les politiques de contrôle d'accès basées sur la définition 3.9 sont appliquées au moyen d'une technique de réécriture. Soit S=(D, $\mathtt{ann}$) une spécification d'accès, T une instance de D, T_v la vue virtuelle de T calculée par rapport à S, et Q une requête définie dans $\mathcal{X}$. Notre objectif est de définir une fonction de réécriture $\mathtt{Rewrite}$ tel que :

$$\mathcal{X} \longrightarrow \mathcal{X}^{\Uparrow}_{[n,=]}$$
$$Q \longmapsto \mathtt{Rewrite}(Q) \text{ avec } \mathcal{S}[\![\mathtt{Rewrite}(Q)]\!](T)=\mathcal{S}[\![Q]\!](T_v)$$

Requêtes sans prédicats. Considérons des requêtes sans prédicats définies dans $\mathcal{X}$ de la forme $\alpha_1 :: \eta_1/\cdots/\alpha_k :: \eta_k$ ($k \geq 1$) où $\alpha_i \in \{\varepsilon, \downarrow, \downarrow^*, \downarrow^+\}$ et η_i peut être tout type d'élément, étiquette $*$, ou la fonction *text()*. L'union des requêtes est discutée plus tard. Nous montrons d'abord que la limitation de réécriture pour ce genre de requêtes est rencontrée lorsque nous manipulons l'axe $\downarrow$. Cependant, les axes restants peuvent être réécrits de manière simple en utilisant uniquement le prédicat d'accessibilité.

Exemple 3.10 Considérons l'arbre XML de la figure 3.4 et sa vue représentée sur la figure 3.5 qui est calculée par rapport à la politique de l'exemple 3.8. Nous supposons que l'infirmière formule la requête $\downarrow^+ :: \mathit{departement}/\downarrow^+ :: \mathit{patient}$ sur la vue qui retourne les nœuds $\mathit{patient}_1$ et $\mathit{patient}_3$. Il est facile de voir que cette requête peut être réécrite sur les données d'origine en $\downarrow^+ :: \mathit{departement}[\mathcal{A}^{acc}]/\downarrow^+ :: \mathit{patient}[\mathcal{A}^{acc}]$ où le prédicat $\mathcal{A}^{acc}$ est donné dans l'exemple 3.9. De toute évidence, cette requête réécrite sélectionne les premiers éléments accessibles *department* de la figure 3.4, c-à-d, l'élément $\mathit{department}_1$, et ensuite retourne tous ses descendants accessibles de type *patient*, c-à-d, $\mathit{patient}_1$ et $\mathit{patient}_3$. L'accessibilité de ces nœuds est vérifiée à l'aide $\mathcal{A}^{acc}$. Considérons maintenant une autre requête sur la vue de données des infirmières définie par $\downarrow^* :: \mathit{parent}/\downarrow :: *$ et qui doit retourner seulement le nœud $\mathit{patient}_3$. Comme il y a un cycle entre les éléments *patient* et *parent* de la DTD hôpital, cette dernière requête ne peut pas être réécrite en utilisant seulement le prédicat de l'accessibilité. Plus précisément, la requête $\downarrow^* :: \mathit{parent}[\mathcal{A}^{acc}]/\downarrow :: *[\mathcal{A}^{acc}]$ sur le document originel ne retourne pas d'éléments puisqu'elle sélectionne d'abord l'élément accessible parent_1, alors que son fils immédiat $\mathit{patient}_2$ n'est pas accessible. En outre, un cycle ne peut pas être capturé par le remplacement des axes $\downarrow$ par les axes $\downarrow^*$. La requête $\downarrow^* :: \mathit{parent}[\mathcal{A}^{acc}]/\downarrow :: *[\mathcal{A}^{acc}]$ sur le document originel retourne à la fois le nœud $\mathit{patient}_3$ ainsi que d'autres éléments : pname_3, $\mathit{symptoms}_3$, $\mathit{symptom}_3$, etc. ■

Nous montrons, dans ce qui suit, comment les axes vers le haut (en anglais "upward") et le prédicat de position du prédicat du fragment XPath $\mathcal{X}^{\Uparrow}_{[n]}$ peuvent être utilisés pour surmonter la limitation de réécriture rencontrée lorsque nous considérons des requêtes sans prédicats de $\mathcal{X}$.

Définition 3.14 Etant donnés une spécification d'accès S=(D, $\mathtt{ann}$) et un type d'élément B, alors nous définissons deux prédicats définis dans $\mathcal{X}^{\Uparrow}_{[n]}$, $\mathcal{A}^+$ et $\mathcal{A}^B$, avec : $\mathcal{A}^+ := \uparrow^+ :: *[\mathcal{A}^{acc}]$, et $\mathcal{A}^B := \uparrow^+ :: *[\mathcal{A}^{acc}][1]/\varepsilon :: B$. Pour tout élément n, l'évaluation $\mathcal{S}[\![\mathcal{A}^+]\!](\{n\})$ retourne tous les ancêtres accessibles de n, alors que $\mathcal{S}[\![\mathcal{A}^B]\!](\{n\})$ retourne le premier ancêtre accessible de n dont le type est B. ■

Enfin, nous donnons les détails de notre fonction de réécriture. Etant donnée une spécification d'accès $S{=}(D,\ \texttt{ann})$, nous définissons $\texttt{Rewrite} : \mathcal{X} \longrightarrow \mathcal{X}^{\Uparrow}_{[n]}$ qui réécrit toute requête Q de $\mathcal{X}$, de la forme $\alpha_1 :: \eta_1/\cdots/\alpha_k :: \eta_k$ $(k \geq 1)$, en une autre définie dans le fragment $\mathcal{X}^{\Uparrow}_{[n]}$ comme suit :

$$\texttt{Rewrite}(Q) := \downarrow^* :: \eta_n[\mathcal{A}^{acc}][prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_k :: \eta_k)]$$

Le qualificatif $prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_k :: \eta_k)$ exprime une réécriture récursive d'une manière descendante où chaque sous-requête $\alpha_i :: \eta_i$ est réécrite sur toutes les sous-requêtes qui la précèdent dans Q. En d'autres termes, pour chaque sous-requête $\alpha_i :: \eta_i$ $(1 \leq i \leq k)$, $prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_{i-1} :: \eta_{i-1})$ est déjà calculé et utilisé par calculer $prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_i :: \eta_i)$ comme suit [17] :

— $\alpha_i = \downarrow$:
$prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_i :: \eta_i) := \mathcal{A}^{\eta_{i-1}}[prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_{i-1} :: \eta_{i-1})]$

— $\alpha_i \in \{\downarrow^+, \downarrow^*\}$:
$prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_i :: \eta_i) := \alpha_i^{-1} :: \eta_{i-1}[\mathcal{A}^{acc}][prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_{i-1} :: \eta_{i-1})]$

— $\alpha_i = \varepsilon$:
$prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_i :: \eta_i) := \varepsilon :: \eta_{i-1}[prefix^{-1}(\alpha_1 :: \eta_1/\cdots/\alpha_{i-1} :: \eta_{i-1})]$

Comme cas particulier, la première sous-requête est réécrite sur le type de la racine. Ainsi, nous avons $prefix^{-1}(\downarrow :: \eta_1){=}\mathcal{A}^{root}$, $prefix^{-1}(\downarrow^+ :: \eta_1){=}\uparrow^+ :: root$. Pour les axes restants, $\alpha_1 \in \{\varepsilon, \downarrow^*\}$, $prefix^{-1}(\alpha_1 :: \eta_1)$ est vide.

Exemple 3.11 Considérons la requête $Q{=}\downarrow^* :: parent/\downarrow :: *$ de l'exemple 3.10 posée sur la vue de la figure 3.5. Par rapport à la spécification d'accès de l'exemple 3.8, cette requête peut être réécrite comme suit : $\texttt{Rewrite}(Q){=}\downarrow^* :: *[\mathcal{A}^{acc}][\mathcal{A}^{parent}]$. En remplaçant $\mathcal{A}^{parent}$ par sa valeur, nous obtenons : $\downarrow^* :: *[\mathcal{A}^{acc}][\uparrow^+ :: *[\mathcal{A}^{acc}][1]/\varepsilon :: parent]$. Il faut rappeler que la définition du prédicat $\mathcal{A}^{acc}$ par rapport à la spécification d'accès de l'exemple 3.8 est donnée dans l'exemple 3.9. L'évaluation de la requête $\downarrow^* :: *[\mathcal{A}^{acc}]$ sur le document originel de la figure 3.4 retourne un ensemble de nœuds N composé par tous les nœuds accessibles, illustré dans la figure 3.5. L'évaluation de $[\mathcal{A}^{parent}]$ sur l'ensemble N retourne uniquement les éléments ayant comme premier ancêtre accessible, un élément de type *parent*. Ainsi, la requête $\downarrow^* :: *[\mathcal{A}^{acc}][\mathcal{A}^{parent}]$ sur le document originel renvoie l'élément $patient_3$ qui est l'unique élément satisfaisant le prédicat $[\mathcal{A}^{parent}]$: $\mathcal{S}[\![\mathcal{A}^{parent}]\!](\{patient_3\})$ retourne l'élément $parent_1$, c-à-d, $patient_3 \vDash \mathcal{A}^{parent}$. Par conséquent, la requête $\texttt{Rewrite}(Q)$ sur le document originel de la figure 3.4 retourne seulement l'élément $patient_3$ comme le fait la requête Q sur la vue de la figure 3.5. ■

Réécriture des prédicats. Nous discutons dans cette section la réécriture de prédicats du fragment $\mathcal{X}$ pour compléter la description de notre approche de réécriture. Etant donnée une spécification d'accès $S{=}(D,\ \texttt{ann})$, nous définissons la fonction $\texttt{RW\_Pred} : \mathcal{X} \rightarrow \mathcal{X}^{\Uparrow}_{[n,=]}$ qui réécrit tout prédicat P de $\mathcal{X}$ de la forme $\alpha_1 :: \eta_1/\cdots/\alpha_k :: \eta_k$ $(k \geq 1)$, en un autre prédicat défini dans le fragment $\mathcal{X}^{\Uparrow}_{[n,=]}$. De manière descendante, $\texttt{RW\_Pred}(P)$ est récursivement définie sur les sous-prédicats de P comme suit :

— $\alpha_i = \downarrow$:
$\texttt{RW\_Pred}(\alpha_i :: \eta_i/\cdots/\alpha_k :: \eta_k) :=$
$\downarrow^+ :: \eta_i[\mathcal{A}^{acc}][\texttt{RW\_Pred}(\alpha_{i+1} :: \eta_{i+1}/\cdots/\alpha_k :: \eta_k)]/\mathcal{A}^+[1]{=}\varepsilon :: *$

17. Pour $\alpha_i \in \{\downarrow^+, \downarrow^*\}$, si $\alpha_i{=}\downarrow^+$ alors $\alpha_i^{-1}{=}\uparrow^+$ sinon $\alpha_i^{-1}{=}\uparrow^*$.

— $\alpha_i \in \{\downarrow^+, \downarrow^*\}$:
$\texttt{RW\_Pred}(\alpha_i :: \eta_i / \cdots / \alpha_k :: \eta_k) :=$
$\alpha_i :: \eta_i[\mathcal{A}^{acc}][\texttt{RW\_Pred}(\alpha_{i+1} :: \eta_{i+1} / \cdots / \alpha_k :: \eta_k)]$

— $\alpha_i = \varepsilon$:
$\texttt{RW\_Pred}(\alpha_i :: \eta_i / \cdots / \alpha_k :: \eta_k) := \varepsilon :: \eta_i[\texttt{RW\_Pred}(\alpha_{i+1} :: \eta_{i+1} / \cdots / \alpha_k :: \eta_k)]$

Comme cas particulier, le prédicat $\alpha :: \eta / text()='c'$ (comparaison du contenu texte) est réécrit, selon l'axe α, comme suit :

— $\texttt{RW\_Pred}(\downarrow :: \eta / text()='c') := \downarrow^+ :: \eta[\mathcal{A}^{acc}][self :: * / text()='c'] / \mathcal{A}^+[1] = \varepsilon :: *$

— Pour $\alpha \in \{\downarrow^+, \downarrow^*\}$, $\texttt{RW\_Pred}(\alpha :: \eta / text()='c') := \alpha :: \eta[\mathcal{A}^{acc}] / text()='c'$

— $\texttt{RW\_Pred}(\varepsilon :: \eta / text()='c') := \varepsilon :: \eta / text()='c'$

Exemple 3.12 Considérons la spécification d'accès de l'exemple 3.8 et la vue de la figure 3.5. Il est clair que le prédicat $\underbrace{[\downarrow :: patient / \downarrow :: wardNo = \text{“}421''}_{P}]$ est satisfait que sur le nœud de l'élément $parent_1$. Ce prédicat est réécrit en $[\texttt{RW\_Pred}(P)]$ comme suit :

— $[\texttt{RW\_Pred}(P)] = [\downarrow^+ :: patient[\mathcal{A}^{acc}][\texttt{RW\_Pred}(\downarrow :: wardNo=\text{“}421\text{”})] / \mathcal{A}^+[1]=\varepsilon :: *]$

— $[\texttt{RW\_Pred}(\downarrow :: wardNo=\text{“}421\text{”})] =$
$[\downarrow^+ :: wardNo[\mathcal{A}^{acc}][\varepsilon :: * / text()=\text{“}421\text{”}] / \mathcal{A}^+[1]=\varepsilon :: *]$

Considérons le document XML de la figure 3.4, il est facile de vérifier que le prédicat $[\texttt{RW\_Pred}(P)]$ est satisfait que sur l'élément nœud $parent_1$. ■

Enfin, nous généralisons la définition de la fonction $\texttt{Rewrite}$ pour prendre en compte toutes les requêtes du fragment $\mathcal{X}$. Etant donnée la spécification d'accès S=(D, $\texttt{ann}$), la fonction $\texttt{Rewrite} : \mathcal{X} \longrightarrow \mathcal{X}^{\Uparrow}_{[n,=]}$ est redéfinie pour réécrire toute requête Q de $\mathcal{X}$, de la forme $\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_k :: \eta_k[p_k]$ ($k \geq 1$), dans une autre requête définie dans le fragment $\mathcal{X}^{\Uparrow}_{[n,=]}$ comme suit (où p_i^t=$\texttt{RW\_Pred}(p_i)$ pour $1 \leq i \leq k$) :

$$\texttt{Rewrite}(Q) := \downarrow^* :: \eta_k[\mathcal{A}^{acc}][p_k^t][prefix^{-1}(Q)]$$

Le qualificateur $prefix^{-1}(Q)$ est récursivement défini comme suit :

— $\alpha_i = \downarrow$:
$prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_i :: \eta_i[p_i]) :=$
$\mathcal{A}^{\eta_{i-1}}[p_{i-1}^t][prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_{i-1} :: \eta_{i-1}[p_{i-1}])]$

— $\alpha_i \in \{\downarrow^+, \downarrow^*\}$:
$prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_i :: \eta_i[p_i]) :=$
$\alpha_i^{-1} :: \eta_{i-1}[p_{i-1}^t][\mathcal{A}^{acc}][prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_{i-1} :: \eta_{i-1}[p_{i-1}])]$

— $\alpha_i = \varepsilon$:
$prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_i :: \eta_i[p_i]) :=$
$\varepsilon :: \eta_{i-1}[p_{i-1}^t][prefix^{-1}(\alpha_1 :: \eta_1[p_1] / \cdots / \alpha_{i-1} :: \eta_{i-1}[p_{i-1}])]$

Comme cas particulier, la requête dans $\mathcal{X}$ de la forme $Q_1 \cup \cdots \cup Q_k$ ($k \geq 1$) est réécrite en $\texttt{Rewrite}(Q_1) \cup \cdots \cup \texttt{Rewrite}(Q_k)$.

EXEMPLE 3.13 Considérons la spécification d'accès définie dans l'exemple 3.8. La requête dans $\mathcal{X}$, $Q{=}{\downarrow^+} :: patient/{\downarrow} :: patient[\underbrace{{\downarrow}:: pname = \text{“}Martin''}_{P}]$, sur la vue de la figure 3.5 est réécrite sur le document originel de la figure 3.4 comme suit :

$$\texttt{Rewrite}(Q){=}{\downarrow^*} :: patient[\mathcal{A}^{acc}][\texttt{RW\_Pred}(P)][{\uparrow^+} :: *[\mathcal{A}^{acc}][1]/\varepsilon :: parent]$$

$$\texttt{RW\_Pred}(P) = [{\downarrow^*} :: pname[\mathcal{A}^{acc}][\varepsilon :: */text(){=}\text{“}Martin\text{”}]/\mathcal{A}^{+}[1]{=}\varepsilon :: *]$$

L'évaluation de la requête $\texttt{Rewrite}(Q)$ sur le document originel renvoie l'élément nœud $patient_3$ comme le fait la requête Q sur la vue. □

Nous insistons sur le fait que la généralisation de la fonction `RW_Pred` pour manipuler les prédicats complexes est assez simple. Par exemple, $\texttt{RW\_Pred}(P_1 \vee P_2)$ est donné par $\texttt{RW\_Pred}(P_1) \vee \texttt{RW\_Pred}(P_2)$. En outre, $\texttt{RW\_Pred}(P_1[P_2])$ est donné par $\texttt{RW\_Pred}(P_1[\texttt{RW\_Pred}(P_2)])$.

Résultats théoriques. Il faut noter que nous avons aussi étendu notre fonction de réécriture `Rewrite` pour manipuler des requêtes de $\mathcal{X}^{\Uparrow}$ (de plus amples détails sont donnés dans [78, 82]).

Nous présentons ici brièvement quelques résultats qui concernent l'évaluation du temps de réponse global de notre approche de réécriture, ainsi que sa correction.

LEMME 3.2 Toute requête Q dans $\mathcal{X}^{\Uparrow}_{[n,=]}$ peut être évaluée sur un document XML T en un temps $O(|Q|.|T|)$. ■

La preuve de ce lemme est basée sur les résultats de l'analyse de la complexité des requêtes XPath détaillée dans [48].

THÉORÈME 3.2 Soient une spécification d'accès $S{=}(D, \texttt{ann})$, un arbre XML $T \in \mathcal{T}(D)$ et sa vue virtuelle T_v calculée par rapport à S. Il existe un algorithme `Rewrite` qui réécrit toute requête Q dans $\mathcal{X}$ sur T_v en une requête Q^t dans $\mathcal{X}^{\Uparrow}_{[n,=]}$ au plus dans un temps $O(|Q|)$. En plus, Q^t peut être évaluée sur T au plus dans un temps $O(|Q|.|\texttt{ann}|.|T|)$. ■

THÉORÈME 3.3 L'approche de la réécriture des requêtes est correcte pour toute requête du fragment $\mathcal{X}$. ■

Plus précisément, le théorème 3.3 stipule que pour toute spécification d'accès $S{=}(D,\texttt{ann})$ et tout arbre XML $T \in \mathcal{T}(D)$ muni de sa vue virtuelle T_v, notre algorithme de réécriture `Rewrite` transforme toute requête Q dans $\mathcal{X}$ sur T_v en une requête correcte Q^t sur T telle que : $\mathcal{S}[\![Q]\!](T_v){=}\mathcal{S}[\![Q^t]\!](T)$.

Notre algorithme `Rewrite` et le détail des preuves sont donnés dans [78, 82]. En outre, une comparaison de notre solution avec celle de [38] utilisant XPath régulier (ou Regular XPath) est donnée dans [78, 82].

3.4 Implémentation et Evaluation

Nos résultats sur le contrôle accès en lecture ont été étendus avec succès pour sécuriser les opérations de mise à jour de XQuery Update Facility [104] (voir [79, 80]). Nous avons développé SVMAX (Secure and Valid MAnipulation of XML), un système qui facilite la spécification et l'exécution des droits d'accès en lecture et mise-à-jour pour les données XML. Il fournit des modèles généraux et expressifs pour le contrôle d'accès qui dépassent les limites des approches

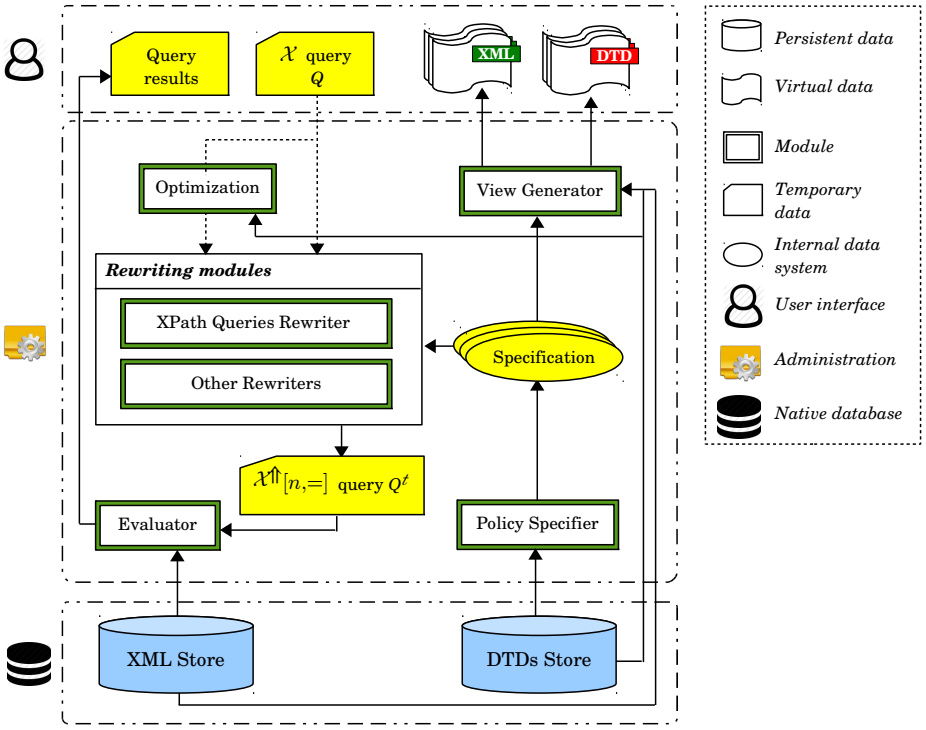


FIGURE 3.6 – Notre système SVMAX.

existantes. Les droits de lecture et mise-à-jour de SVMAX sont définis par annotation des grammaires DTD et appliqués en utilisant le principe de la réécriture. SVMAX est bien adapté pour réécrire efficacement de telles requêtes et mises à jour, et d'être intégré dans les systèmes de base de données qui supportent les normes W3C : XPath et XQuery Update Facility.

3.4.1 Système SVMAX

La figure 3.6 présente notre système SVMAX. Il est composé par les principaux modules suivants : 1) *Specifier Policy* pour la spécification de privilèges de lecture et de mise-à-jour ; 2) *View Generator* pour la génération de DTD et des vues de données ; 3) *XPath Rewriter* [81] et 4) *XQuery Update Rewriter* [80], pour la réécriture, respectivement, des requêtes de lecture et mise à jour ; 5) *Validator* qui applique une validation supplémentaire du schéma de la DTD après chaque opération de mise à jour. Ces modules sont mis en œuvre comme une API permettant à SVMAX d'être intégré dans les systèmes existants de base de données NXD (utilisant du XML natif) qui manipulent la structure de données XML et les standards du W3C.

D'autre part, SVMAX peut fonctionner en mode autonome grâce à son outil visuel. Celui-ci est un outil graphique qui surveille les modules précédents. Plus précisément, il est utilisé par l'administrateur pour spécifier les politiques de lecture et mise-à-jour, générer des vues virtuelles de la DTD et les données XML, et de fournir ces vues à l'utilisateur. Les requêtes des utilisateurs (requêtes XPath ou des opérations de mise à jour XQuery) sont réécrites, en utilisant le module "Rewriter" adéquat, pour être évaluées en toute sécurité sur les données XML d'origine, puis les résultats d'évaluation sont retournés à l'utilisateur (voir [83] pour plus de descriptions).

3.4.2 Mesures de performance

Dans cette section, nous décrivons brièvement l'évaluation de SVMAX. Notre système est fourni à la fois comme une API Java et outil visuel. En utilisant ce dernier, on peut choisir un document DTD, spécifier les politiques d'accès et de mise à jour, et faire respecter ces politiques sur les données XML sous-jacentes. Nous nous sommes concentrés dans nos expériences sur le temps global nécessaire pour la *réécriture* et l'*évaluation* des requêtes XPath. L'étude est menée sur les aspects suivants : 1) mesure du passage à l'échelle et de la dégradation de nos approches de réécriture, et 2) la comparaison de SVMAX avec l'approche naïve en terme de temps global de réponse. Étant donné que notre système peut être intégré au sein de NXDs (base de données XML Native) existants, l'autre préoccupation de l'expérimentation est 3) une étude de l'efficacité de l'intégration. Dans ce qui suit, nous allons présenter seulement le premier aspect, à savoir le passage à l'échelle. De plus amples détails sur les autres aspects sont donnés dans [78, 82].

Nous mesurons le temps nécessaire à SVMAX pour réécrire des requêtes générales de XPath. Nous utilisons une DTD récursive utilisée pour structurer des données réelles **GedML** [18] et nous générons de manière aléatoire 10 spécifications d'accès en variant le nombre d'annotations (de 20 jusqu'à 200). Ensuite, nous définissons différentes requêtes XPath de taille [19] 400 qui comprennent la plupart des caractéristiques du fragment XPath $\mathcal{X}^{\Uparrow}$: avec $\downarrow^*$-axes (Q_1) ; avec $\downarrow^*$-axes et prédicats (Q_2) ; avec $\downarrow$-axes (Q_3) ; avec $\downarrow$-axes et prédicats (Q_4) ; avec $\downarrow^*$-axes, prédicats, et étiquettes $*$ à l'intérieur des prédicats (Q_5) ; avec $\downarrow$-axes, prédicats, et étiquettes $*$ à l'intérieur des prédicats (Q_6). Notons que les prédicats utilisés contiennent différents opérateurs (par exemple, $\wedge$, $\vee$, et comparaison de texte).

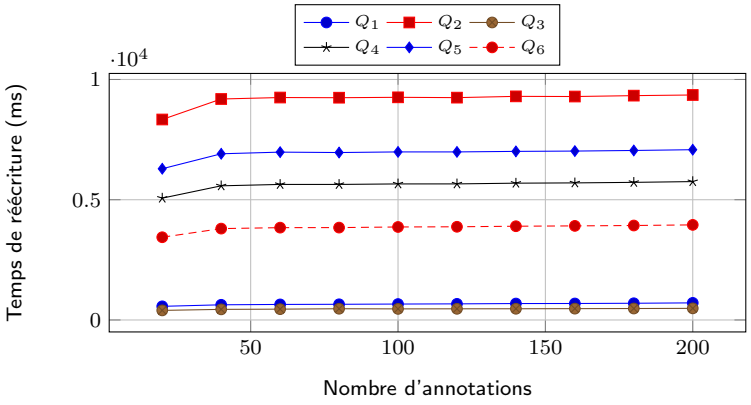
En utilisant SVMAX, nous réécrivons ces requêtes en fonction de chacune des spécifications d'accès générées précédemment. La figure 3.7 montre les temps globaux de réécriture. Il faut noter que le temps de réécriture obtient une nature constante, c-à-d, le temps n'augmente pas avec l'accroissement du nombre d'accès des annotations. Ceci peut être expliqué par le fait que, pour une requête XPath en entrée, notre module "Rewriter" analyse toutes ses sous-requêtes (avec la forme *axe* :: *label*) et les réécrit en utilisant le *prédicat d'accessibilité*. Le temps de calcul de ce dernier est négligeable (moins de 10 *ms* pour les grandes spécifications d'accès). Notre temps de réécriture dépend donc essentiellement de l'analyse de la requête, puis sur la taille de la requête. Comme nos requêtes ont la même taille, le temps global de réécriture ne dépend pas du nombre d'annotations d'accès et reste constant à un moment donné. De plus, notons qu'en général une requête avec des axes $\downarrow^+$ nécessite plus de temps que la réécriture d'une requête avec des axes $\downarrow$ (Q_1 par rapport à Q_3). Une requête avec des prédicats consomme également un peu de temps supplémentaire (Q_2 par rapport à Q_1 ; et Q_4 par rapport à Q_3). Les étiquettes $*$ nécessitent moins de temps de réécriture (Q_2 par rapport à Q_5 ; et Q_4 par rapport à Q_6).

3.5 Etat de l'Art

Au début, les approches utilisées dans [89, 90] consistait à annoter naïvement les données XML avec des étiquettes de sécurité pour spécifier quelles actions peuvent être effectuées sur quels nœuds XML, et donc de limiter l'accès aux données sensibles par le biais de ces étiquettes. Bien que, quelques améliorations dans [30, 31] ont été faites afin d'éviter la ré-annotation coûteuse des données, ces approches naïves sont lourdes et généralement difficiles à appliquer, par exemple,

18. Genealogy Markup Language : `http://xml.coverpages.org/gedml-dtd9808.txt`.

19. La taille d'une expression XPath est le nombre d'occurrences de tous les types d'éléments, étiquette $*$, et les fonctions text().



(a)

FIGURE 3.7 – Mesure de passage à l'échelle pour SVMAX.

dans le cas où il y a plusieurs utilisateurs et de multiples actions à effectuer ainsi que des politiques dynamiques. D'autres modèles ont été proposés dans [44, 43] qui définissent les politiques d'accès sans étiquetage des données, et appliquent ces politiques lors de l'évaluation des requêtes des utilisateurs (requêtes d'accès en lecture ou des opérations de mise à jour). Une politique d'accès est définie comme un ensemble d'expressions XPath, chacune se réfère à un ensemble de nœuds XML sur laquelle l'utilisateur peut exécuter certains actions (lecture ou mise à jour). Les requêtes des utilisateurs sont *réécrites* par rapport aux stratégies d'accès sous-jacentes en ajoutant quelques prédicats XPath afin d'exécuter l'action demandée uniquement sur les données autorisées (à savoir des données qui peuvent être interrogées et/ou mises à jour). Ces approches fondées sur XPath surpassent celles basées sur les instances dans la plupart des cas. Cependant, la principale limite de ces modèles est le manque de support pour les utilisateurs autorisés à accéder aux données : le schéma de données accessibles est nécessaire pour les utilisateurs afin de formuler et d'optimiser leurs requêtes, ainsi que pour l'administrateur de sécurité pour comprendre comment la vue autorisée des données XML, pour un groupe d'utilisateurs, ressemblera réellement.

Pour surmonter les limites de la protection nœud-étiquetage et la protection à base de XPath, Stoica et Farkas [117] introduisent la notion de *vue de sécurité XML* qui consiste à définir, pour chaque groupe d'utilisateurs, une vue sur le document XML qui affiche uniquement l'information accessible. Cette notion a été affinée plus tard et utilisée de différentes façons en fournissant à chaque groupe d'utilisateurs (1) une *vue matérialisée* des données accessibles, (2) une *vue virtuelle*, ou (3) une vue qui consiste en une combinaison de sous-vues matérialisées et virtuelles [125]. Fan et al. [36] a proposé un langage expressif qui vise à définir ces points de vue de la sécurité. Ce langage est basé sur la notion d'annotations de schéma. En gros, le schéma des données XML est jumelé avec une collection d'expressions XPath qui, lorsqu'il est évalué sur les données, il extrait uniquement l'information accessible. Le serveur définit, pour chaque groupe d'utilisateurs, ces collections d'expressions XPath représentant les politiques d'accès des utilisateurs. Selon chaque politique d'accès, le schéma (par exemple, une DTD) est ensuite filtrée en éliminant les informations de données inaccessibles, ce qui résulte en une *vue du schéma* qui est fournie aux utilisateurs pour la formulation et l'optimisation de leurs requêtes. Alors que les utilisateurs peuvent interroger les vues, ils ne sont pas autorisés à interroger directement les données XML sous-jacentes. Une question importante est de répondre aux requêtes posées sur les vues et d'assurer l'exposition sélective de données à différentes catégories d'utilisateurs.

Une façon de le faire est de fournir à chaque groupe d'utilisateurs une *vue matérialisée* de toutes (et seules) des données accessibles (comme étudié dans [68]), qui est utilisée pour évaluer les requêtes des utilisateurs et offre un accès plus rapide aux données. Cependant, lorsque les données XML et/ou les politiques d'accès sont modifiées, toutes les vues des utilisateurs doivent être (incrémentalement) maintenues [52, 53, 10, 91]. On notera que, dans certains cas, la maintenance incrémentale des vues matérialisées conduit aux mêmes performances du re-calcul des vues à partir de zéro. En plus du coût de la maintenance, la matérialisation de toutes les vues des utilisateurs au sein du serveur est consommatrice de temps et de mémoire.

La *virtualisation de vue* est la solution adéquate et plus évolutive en cas de données énorme, de nombre important d'utilisateurs, et des politiques dynamiques. Fan et al. [36] a défini la notion de *réécriture de requête* qui consiste à traduire des requêtes posées sur des vues virtuelles en requêtes équivalentes à évaluer sur les données originelles. Comme les DTD trouvées dans la pratique sont souvent récursives [20], de nombreux auteurs ont affiné ce travail pour utiliser des langages de requêtes plus expressif [37, 38, 49], à savoir XPath régulier (ou Regular XPath). XPath régulier est plus expressif que XPath standard et permet la définition de chemins récursifs. Cependant, les solutions basées sur XPath régulier restent comme un acquis théorique et elles sont moins pratiques. En effet, la réécriture des expressions XPath régulières peut atteindre un coût exponentiel comme nous l'avons montré dans [78, 82]. En outre, XPath régulier n'est pas couramment utilisé en pratique [20] et la plupart des systèmes de bases de données commerciales (par exemple eXistdb) sont basés sur les normes W3C : XPath et XQuery. En effer, la manipulation des requêtes basées sur la standard XPath est la pratique la plus courante.

3.6 Conclusion

Nous avons fourni une solution pour le problème de la réécriture des requêtes XPath en présence de DTDs récursives. Nous avons étendu la classe "downward" de XPath avec des axes et des opérateurs, et nous avons montré que le fragment résultant $\mathcal{X}^{\Uparrow}_{[n,=]}$ peut être utilisé pour réécrire de manière efficace toute requête de $\mathcal{X}$, posée sur une vue, en une autre qui peut être évaluée sur le document originel. Notre proposition donne la première solution pratique au problème de réécriture. L'expérimentation menée a montré l'efficacité de notre approche. Plus précisément, la traduction des requêtes de $\mathcal{X}$ en requêtes $\mathcal{X}^{\Uparrow}_{[n,=]}$ n'a pas d'impact sur les performances du système pour répondre aux requêtes.

Par ailleurs, il convient de souligner qu'en cas de DTD récursives, la génération de vue DTD n'est pas toujours garantie [49] ou peut être de taille exponentielle [67]. Ainsi, le fait de cacher certaines informations de la DTD peut entraîner une grammaire non-contextuelle qui ne peut être capturée avec une grammaire régulière [21]. Dans de telles situations, notre module "*View generator*" génère une vue DTD *approximative*. Nos approximations sont basées sur les conditions suffisantes bien connues pour la régularisation des grammaires non-contextuelle [4].

20. Notons qu'aucun outil n'existe dans la pratique pour évaluer les requêtes XPath régulières.
21. Il est indécidable, en général, de trouver une solution régulière pour une grammaire non-contextuelle.

Chapitre 4

Sondage Collaboratif dans les Réseaux Sociaux

Sommaire

4.1 Introduction

Les réseaux sociaux en ligne offrent aux participants une variété d'activités concernant le monde des affaires, le divertissement et les événements à caractères social et culturel comme, par exemple, l'organisation des fêtes, le tissage des liens d'amitiés, l'édition et le partage de l'information. Ces plateformes constituent aussi une tribune pour exprimer des opinions sur des sujets variés. A ce titre, le sondage est une pratique utile et très prisée. En général, il consiste à soumettre une ou plusieurs questions et fournir à tous les participants les résultats d'un scrutin réalisé entre eux, donnant ainsi le choix le plus favori. Chaque participant peut exprimer sa préférence en soumettant un vote, tous les votes sont agrégés et l'option majoritaire sera choisie comme résultat final. Comme situation typique, c'est l'exemple d'une université qui vient juste de lancer une nouvelle procédure administrative et elle veut sonder ses étudiants sur l'utilité de cette procédure en utilisant un réseau social comme Facebook. Aussi, chaque étudiant choisira une option entre "Oui" (ou "+1") et "Non" (ou "+1").

Le but d'étudier le problème de sondage est de mettre au point un protocole effectuant un scrutin sécurisé et précis pour dépouiller les votes initiaux avec la présence d'utilisateurs malhonnêtes, qui essaient de biaiser le résultat final et divulguer les choix des votants honnêtes.

Malgré les caractéristiques simples de ce problème, il occupe une place importante et critique dans l'intégration en ligne de l'avis de l'utilisateur. Actuellement, plusieurs études et solutions pour traiter de ce problème, utilisant des collaborations centralisées et distribuées, sont proposées. Dans un réseau social centralisé, par exemple Facebook Poll [22] et Doodle [23], un tel processus de calcul peut être facilement atteint par un serveur central qui sert à recueillir les votes des utilisateurs avant de les résumer pour obtenir le résultat final. Néanmoins, cette solution souffre des défaillances de serveur et, en particulier, des problèmes de confidentialité. En effet, l'utilisateur désire généralement que son vote ne soit pas connu par une entité centrale, et il n'est pas garanti que cette entité ne corrompt pas le sondage et ne divulgue pas les votes des utilisateurs.

Nous nous intéressons ici à la conception d'un protocole de sondage basé sur les réseaux sociaux décentralisés où les données ne sont pas concentrées dans un point central, et par conséquent, la vie privée de l'utilisateur est moins exposée. Pour ce faire, nous ne voulons pas nous appuyer sur l'utilisation des techniques cryptographiques pour assurer la vie privée ainsi que l'exactitude du résultat final du vote. En effet, ces techniques sont très gourmandes en calcul et peuvent avoir un impact négatif sur l'usage pratique et le passage à l'échelle des protocoles de sondage. En outre, certains problèmes traditionnels du calcul distribué peuvent être résolus sans cryptographie, comme cela a été motivée dans [84, 102].

Dans ce chapitre, nous proposons la conception d'un protocole simple de sondage décentralisé basé sur le partage de secret et ne nécessitant pas d'autorité centrale ni de la cryptographie. Notre protocole est asynchrone et s'appuie sur le graphe social pour établir une collaboration entre les utilisateurs. Pour permettre une diffusion efficace de messages à travers les liens sociaux, nous avons recouru à une propriété basée sur le tri topologique des graphes sociaux. Ensuite, nous définissons les conditions nécessaires et suffisantes pour que le vote soit sécurisé et précis en agrégeant les votes initiaux avec la présence d'utilisateurs malhonnêtes, qui tentent de biaiser le résultat final et divulguer les votes des honnêtes. Ce chapitre est composé comme suit : la section 4.2 décrit notre modèle de sondage ainsi qu'une famille de graphes sociaux pour y déployer notre protocole sondage qui est présenté dans la section 4.3. Dans cette même section, nous analysons la correction de ce protocole sans et avec la présence de comportements malhonnêtes. La section 4.4 discute l'impact des problèmes de défaillance des nœuds et la perte des message sur le comportement de notre protocole. Nous présentons également quelques exemples de graphes sociaux sur lesquels nous pourrons déployer notre protocole. Enfin, nous passons en revue l'état de l'art dans la section 4.5 et nous terminons le chapitre par une conclusion.

4.2 Modèle de Sondage

Cette section décrit les comportements des utilisateurs et les modèles de graphes sociaux. Il convient de noter que nous considérons les mêmes hypothèses que celles indiquées dans [50, 51].

4.2.1 Interactions sociales

Le problème du sondage se compose d'un système modélisé comme un graphe non-orienté $G = (V, E)$, que nous appelons *graphe social*, avec $N = |V|$ est l'ensemble des nœuds (ayant des identités uniques) représentant les utilisateurs et E est l'ensemble de liens sociaux.

Chaque participant (ou utilisateur) n exprime son opinion par le biais d'un vote $v_n \in \{-1, 1\}$ [24]. Après la collecte des votes de tous les nœuds, le résultat attendu est $\sum_n v_n$.

22. http ://apps.facebook.com/opinionpolls/
23. http ://www.doodle.com/
24. Le choix "1" (resp. "−1") désigne la réponse "oui" (resp. "non").

Dans ce travail, nous considérons les hypothèses suivantes :

Nous considérons un modèle *asynchrone* où chaque nœud peut communiquer (envoyer/recevoir des messages) avec ses voisins (par exemple, des amis directs). Certains messages peuvent arriver à destination avec un certain retard. Tous les nœuds doivent envoyer/recevoir/ transmettre des messages s'ils sont sollicités.

Chaque nœud est soit *honnête* ou *malhonnête*. Le nœud honnête respecte complètement le protocole et prend soin de sa vie privée en ce sens que la valeur de son vote n'est pas divulguée. Tous les nœuds se soucient de leur *réputation* : les informations relatives à un utilisateur sont considérées comme reflétant intiment la vraie personne associée à ce nœud. En particulier, les nœuds malhonnêtes ne montrent jamais une mauvaise conduite (ou un mauvais comportement) qui mettra en péril leur réputation. Tous les nœuds malhonnêtes peuvent former une coalition pour obtenir une connaissance complète de la topologie du réseau et essayer de tout faire pour atteindre les objectifs suivants sans être détectés : (i) biaiser le résultat du sondage pour la promotion de leurs votes ou de changer (ou corrompre) les valeurs qu'ils ont reçues de la part d'honnêtes nœuds ; (ii) déduire (ou inférer) les opinions des autres nœuds.

Afin d'unifier les opinions, tous les nœuds malhonnêtes pourraient former une coalition $\mathcal{D}$ de taille D. Cependant, ils veulent aussi protéger leur réputation sans être affectés. Ils sont égoïstes dans le sens où chaque nœud malhonnête préfère prendre soin de sa propre réputation pour se couvrir des autres [50, 51]. Contrairement à des nœuds byzantins, les nœuds malhonnêtes sont plutôt limités et représentatifs du comportement humain [69].

Pour tolérer l'existence de nœuds malhonnêtes avec une corruption de vote limitée, nous supposons que chaque nœud a au moins un ami honnête, mais il ne sait pas lequel est honnête. Pour dissuader les mauvais comportements venant des nœuds malhonnêtes, une action affectant le profil d'un nœud se comportant mal peut être déclenchée. Plus précisément, si le nœud u est détecté comme malhonnête par un nœud v, alors le profil de u est tagué avec la déclaration "v accuse u d'être malhonnête" et le profil de v contient la déclaration "u est un méchant". Notons que dans les réseaux sociaux, personne ne voudrait être étiqueté comme malhonnête. En outre, nous ne prenons pas en compte les "Sybil" attaques et la gestion des spams comme ces types de mauvais comportements peuvent être détectés par plusieurs systèmes tels que SybilGuard [136], SybilLimit[135], et [87, 116] (pour atténuer les spams).

Par ailleurs, comme dans [50, 51], nous supposons que les nœuds malhonnêtes ne peuvent pas blâmer injustement les autres nœuds. En effet, lorsque cette accusation à tort est faite abusivement (sans preuve concrète), elle apporte aux utilisateurs malhonnêtes plus de problèmes que d'avantages et affecte leur réputation. Ces utilisateurs mal intentionnés veulent participer au processus de sondage afin de divulguer le vote des utilisateurs honnêtes et biaiser le résultat final *mais sans compromettre leur réputation.*

4.2.2 Graphe social

Dans ce qui suit, nous définissons d'abord les termes et notations du graphe social utilisés tout au long de ce chapitre. Ensuite, nous décrivons les graphes à faible coût de communication. Enfin, nous donnons la description de notre famille de graphes.

Notations. Chaque nœud n possède un ensemble de voisins directs $\Gamma(n)$ de taille d_n, et deux sous-ensembles de $\Gamma(n)$: un ensemble $\mathcal{S}_n$ de *consommateurs* contenant les nœuds qui reçoivent des messages de n, et un ensemble $\mathcal{R}_n$ de *producteurs* concernant les nœuds pour lesquels n agit comme un consommateur (ou reçoit des messages). Ces ensembles pourraient ne pas être disjoints, c-à-d, $\mathcal{S}_n \cap \mathcal{R}_n \neq \varnothing$, comme illustré sur la figure 4.1 (où z est un nœud à la fois producteur et consommateur).

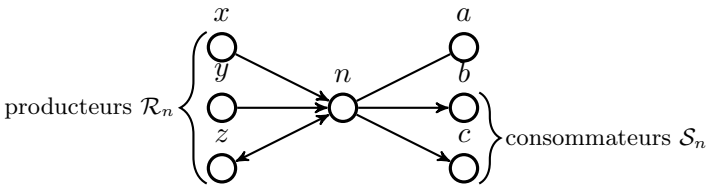


FIGURE 4.1 – Producteurs et consommateurs de n.

Le diamètre d'un graphe social G est noté Δ_G. Notons que les algorithmes distribués pour calculer le diamètre exact d'un graphe sont de complexité $\mathcal{O}(N)$ [57, 93].

Graphes à faible coût de communication. Considérons dans un réseau l'opération de diffusion de messages initiée par un nœud unique, appelé *source*. Ce dernier veut transmettre une donnée à tous les nœuds du réseau. Si l'on utilise une approche naïve : à la réception d'un message venant d'un voisin, un nœud stocke la donnée puis la propage à travers ses liens. Cependant, un nœud peut recevoir/envoyer autant de messages dupliqués (qui peuvent être transmis par de nombreux chemins) de/vers d'autres nœuds. Cela conduit à inonder le réseau. Pour surmonter ce problème, nous proposons une méthode de diffusion efficace utilisant le concept que nous appelons : la *propriété m-diffusion*. Un graphe satisfait la propriété *m-diffusion*, pour un entier positif m tel que $1 \leq m \leq d_{min}$ avec d_{min} est le degré minimal, si pour chaque nœud source, il existe un ordre topologique sur les nœuds du graphe de telle sorte que chaque nœud est connecté soit directement à la source, soit à m nœuds qui le précèdent selon l'ordre défini par rapport à la source. En conséquence, au lieu d'accepter tous les messages provenant d'une source, un nœud reçoit et stocke seulement m messages passés par des chemins ordonnés.

Soit $\beta_n(s)$ l'ensemble des voisins pour un nœud n qui le précèdent selon un ordre établi par rapport à un nœud source s [25] (nous omettons parfois d'indiquer la source lorsqu'aucune confusion ne se pose).

Graphes basés sur le partage de secret. Nous présentons la famille des graphes avec et sans la présence de nœuds malhonnêtes. Nous utilisons un paramètre prédéfini $k \in \mathbb{N}$ (comme dans [50, 51]), appelé *paramètre de confidentialité*, et un autre paramètre $m \in \mathbb{N}$ pour présenter les caractéristiques de nos réseaux sociaux. Soit $G = (V, E)$ un graphe social ayant les propriétés suivantes :

PROPRIÉTÉ 4.1 (P_{g_1}) $d_n \geq 2k + 1$, $|\mathcal{S}_n| = 2i + 1$ et $|\mathcal{R}_n| \leq 2k + 1$ où $0 \leq i \leq k$, pour tout $n \in V$. ■

PROPRIÉTÉ 4.2 (P_{g_2}) G satisfait la propriété m-diffusion. ■

PROPRIÉTÉ 4.3 (P_{g_3}) Pour un nœud source, chaque autre nœud possède moins de $m/2$ nœuds malhonnêtes que le précèdent selon un ordre défini par rapport à une source donnée. ■

Selon la propriété P_{g_1}, l'ensemble des consommateurs et l'ensemble des producteurs d'un nœud ont la taille d'*au plus* $2k + 1$ et pourraient *ne pas être disjoints*. Cette condition distingue notre famille de graphes des autres structures dans [35, 50, 51] et elle est plus flexible par rapport aux graphes utilisées dans [56] car ils ont tous considéré la restrictive condition, à savoir : chaque nœud a *exactement* $2k + 1$ consommateurs. De plus, elle diffère aussi des structures utilisées dans [7, 8] qui n'imposent aucune borne supérieure sur le nombre de producteurs (que doit

25. La liste des voisins est obtenue par une étape de pré-traitement non détaillée ici.

avoir un nœud). Ainsi, un nœud malhonnête peut envoyer des valeurs arbitraires (par exemple, la somme des votes) aux autres et la précision du résultat global du sondage sera facilement affectée.

La propriété P_{g_2} permet de réduire le coût de la communication dans le système. Il est également noté que cette condition implique implicitement la condition que G est un graphe honnête comme mentionné dans [56], à savoir, pour tous les nœuds honnêtes u et v, il existe un chemin entre u et v contenant des nœuds intermédiaires qui sont honnêtes. La propriété P_{g_3} assure que chaque nœud honnête obtient toujours une version correcte des données provenant d'autres nœuds honnêtes. Cette propriété nous permet également de limiter le nombre des utilisateurs malhonnêtes, c-à-d $D \leq \frac{m-1}{2}\Delta_G$ (présentée dans le théorème 4.7).

A partir de ces propriétés, nous caractérisons deux familles de graphes :

(i) $\mathcal{G}_1 = \{G \mid \mathcal{D}(G) = \varnothing$ et G satisfait P_{g_1} et $P_{g_2}\}$.

(ii) $\mathcal{G}_2 = \{G \mid \mathcal{D}(G) \neq \varnothing$ et G satisfait P_{g_1}, P_{g_2} et $P_{g_3}\}$.

Les graphes dans $\mathcal{G}_1$ contiennent seulement des nœuds honnêtes et satisfont les propriétés P_{g_1} et P_{g_2}. Quant aux graphes dans $\mathcal{G}_2$, ils contiennent les deux types de nœuds (honnête et malhonnête) et vérifient les propriétés P_{g_1}, P_{g_2} et P_{g_3}.

4.3 Protocole de Sondage

4.3.1 Description

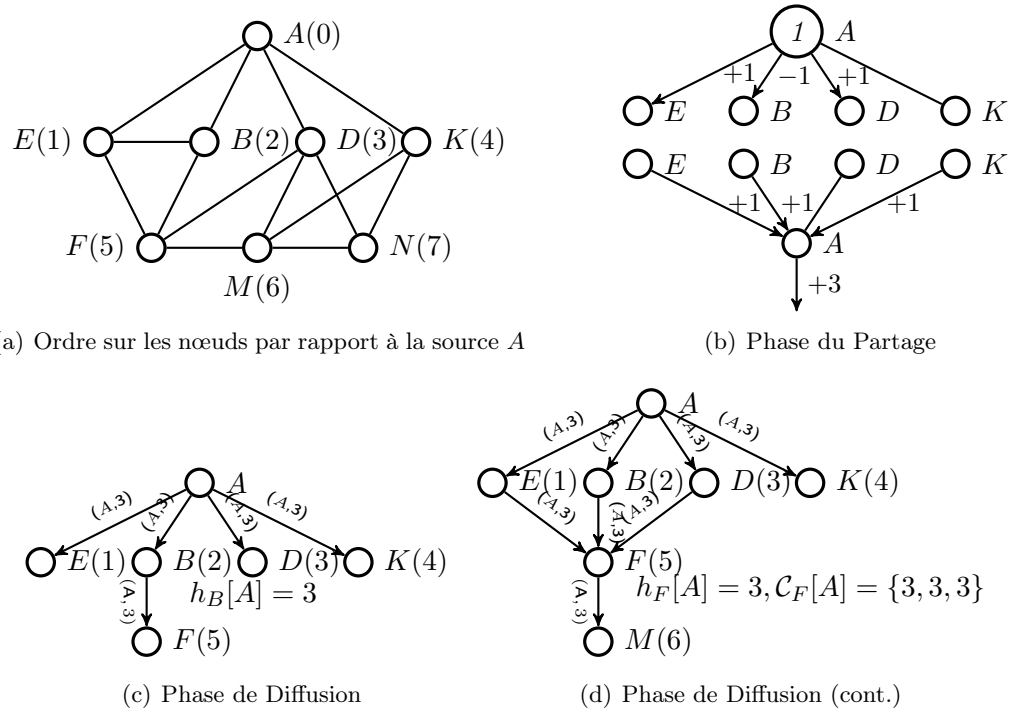
Notre protocole de sondage *EPol* comprend trois phases (pour plus de détails voir [5, 55]) : (i) La phase *Partage* décrit la génération et la distribution du vote d'un nœud vers ses voisins ainsi que la collecte des votes. (ii) La phase *Diffusion* permet à chaque nœud de transmettre les messages contenant la somme des votes à ses voisins directs et indirects. (iii) La phase *Aggrégation* finalise le sondage en calculant le résultat final du vote.

Présentation du protocole

Dans ce qui suit, nous présentons les trois phases.

Phase 1 : Partage. Dans cette phase, chaque nœud n donne son opinion en envoyant un ensemble de *parts*, exprimant son vote $v_n \in \{-1, 1\}$, à ses consommateurs. Cette génération des parts est inspirée du système de partage proposé dans [29]. Plus précisément, n choisit d'abord de manière *aléatoire* i tel que $i \in \{0, 1, ..., k\}$. Cette valeur i n'est pas connue par d'autres nœuds. Ensuite, il génère $2i+1$ parts $\mathcal{P}_n = \{p_1, p_2, ..., p_{2i+1}\}$, où $p_j \in \{-1, 1\}$ et $1 \leq j \leq 2i+1$, contenant : $i+1$ parts de la valeur v_n, et i parts de l'opposé de v_n. L'intuition derrière cette création est de régénérer (reconstruire) le vote v_n lorsque toutes les parts sont additionnées. Plus tard, il génère aléatoirement une permutation μ_n de $\mathcal{P}_n$, et envoie les parts à $2i+1$ consommateurs.

Par ailleurs, le nœud reçoit aussi $|\mathcal{R}_n|$ parts de ses producteurs. Contrairement aux travaux [7, 8, 35, 50, 51], les consommateurs $\mathcal{S}_n$ et les producteurs $\mathcal{R}_n$ pourraient ne pas être disjoints. Après que chaque nœud collecte les parts de ses producteurs, et calcule leur somme c_n, cette phase est terminée. Il faut également noter que, parce que les votes et leurs parts génératrices appartiennent à l'ensemble $\{-1, +1\}$, les nœuds ne peuvent pas distinguer entre un vote et une part. Par conséquent, si un nœud choisit une valeur $i = 0$ et génère seulement $2i + 1 = 1$ part, le consommateur malhonnête recevant un message de ce nœud ne pourra pas distinguer si une telle part a été générée toute seule ou elle fait partie de plusieurs parts générées par ce nœud.



(a) Ordre sur les nœuds par rapport à la source A

(b) Phase du Partage

(c) Phase de Diffusion

(d) Phase de Diffusion (cont.)

FIGURE 4.2 – Protocole de sondage pour $i = k = 1$ et $m = 3$.

La figure 4.2 illustre un exemple de notre protocole pour $i = k = 1$. La figure 4.2(a) présente le réseau et un ordre entre parenthèses sur les nœuds par rapport à la source A. La figure 4.2(b) montre la phase de partage pour le nœud A. Ce dernier souhaiterait voter $+1$. Ainsi, il génère un ensemble de $2i + 1 = 3$ parts $\{+1, -1, +1\}$. Le nœud A reçoit les parts de ses producteurs et calcule la somme $c_A = 3$.

Phase 2 : Diffusion. Dans cette phase, chaque nœud doit diffuser les données collectées à tous les autres nœuds de telle manière que tout autre nœud obtient finalement que des données correctes. Chaque nœud n utilise un tableau h_n (indexé par les nœuds du graphe) pour enregistrer toutes les valeurs collectées durant la phase précédente. Dans l'approche naïve, à la réception d'un message du voisin, un nœud stocke les données puis les transmet à travers ses liens. Malgré l'utilisation de structures plus riches de graphe social, et avec la présence de nœuds malhonnêtes qui peuvent corrompre les données, un nœud peut recevoir/envoyer autant de messages dupliqués (qui peuvent être transmis par de nombreux chemins) de/vers d'autres nœuds. Cela conduit à inonder le stockage local. Comme motivé précédemment, nous proposons une méthode pour diffuser efficacement des messages en utilisant la propriété m-diffusion. Pour un graphe satisfaisant la propriété m-diffusion, chaque nœud n envoie ses données recueillies auprès de tous les voisins. Puis, lors de la réception du message, contenant les données recueillies de la source s, par l'intermédiaire du voisin r qui le précède dans l'ordre (par rapport à source s), le nœud n fait l'une des activités suivantes :

- $r = s$: Il analyse la donnée provenant de la source s pour stocker sa valeur collectée c_s dans $h_n[s]$. Une fois enregistrée localement, cette valeur est ensuite transmise à tous les $d_n - \beta_n(s)$ successeurs de n selon l'ordre. Dans la figure 4.2(c), le nœud A envoie sa donnée qui est reçue par B et stockée dans $h_B[A]$. Ensuite, elle est transmise vers F.
- $r \neq s$: Pour éviter le cas où la valeur $h_n[s]$ pourrait être calculée (et diffusée) deux fois

pour le voisin direct s, le nœud n considère seulement le cas $r \neq s$ et $s \notin \Gamma(n)$. Si cette condition est satisfaite, il ajoute la valeur c_s au multi-ensemble $\mathcal{C}_n[s]$ contenant les possibles valeurs collectées par s.

Lorsque le nœud n a reçu le nombre prévu de m possibles données collectées par une source donnée s, il décide sur les données recueillies en choisissant la valeur la plus représentée dans $\mathcal{C}_n[s]$ et la met dans $h_n[s]$. Comme la décision est basée sur la valeur la plus représentée, au lieu d'attendre pour recevoir toutes les m données transmises, le nœud n peut décider des données provenant de la source lors de la réception de plus de $m/2$ données identiques. Le nœud n transmet ensuite les données à tous les $d_n - \beta_n(s)$ nœuds qui le succèdent dans l'ordre.

La figure 4.2(d) montre le nœud F qui reçoit des messages de ses voisins. Il a quatre amis, mais reçoit seulement $m = 3$ messages à partir des prédécesseurs et voisins E, B, D. Comme toutes les valeurs dans $\mathcal{C}_F[A]$ sont 3, le nœud F décide que cette valeur est la donnée collectée de A et la stocke dans $h_F[A]$. Ensuite, il la transmet à son successeur voisin M.

Lorsqu'un nœud décide sur les données recueillies de s et n'a pas d'ami successeur, la valeur $h_n[s]$ n'est plus transmise. Cette phase est terminée si un nœud détermine les données recueillies de tous les autres nœuds dans le réseau.

Phase 3 : Aggrégation. Le résultat final du sondage est obtenu à chaque nœud en effectuant tout simplement le calcul : $\mathsf{result} = c_n + \sum_{j \neq n} h_n[j]$. Ce qui termine le sondage.

Propriétés du protocole

Pour un système de N nœuds, nous présentons dans cette partie les propriétés de notre protocole ainsi qu'une analyse de complexité dans le cas où tous les participants sont honnêtes.

Terminaison. Dans la phase de Partage, chaque nœud reçoit un nombre fini ($|\mathcal{R}|$) de messages. Durant la phase de Diffusion, pour une source s, chaque voisin direct reçoit seulement un message et chaque voisin indirect réceptionne m messages. Comme tout nœud envoie le nombre requis de messages qui arrivent finalement à destination, chaque phase est assurée de terminer. Etant donné que le protocole a un nombre fini de phases, la terminaison est garantie.

Précision du résultat final. La terminaison du protocole doit assurer que chaque nœuds obtient le résultat final $\sum_{n=0}^{N-1} v_n$. Dans la phase Partage, chaque nœud n envoie un ensemble de parts $\mathcal{P}_n = \{p_{n_1}, p_{n_2}, ..., p_{n_{2i+1}}\}$ à ses consommateurs, où $\sum p_{n_j} = (i+1).v_n + i.(-v_n) = v_n$. De ses producteurs ($l = |\mathcal{R}_n|$), n reçoit $\{p'_{n_1}, p'_{n_2}, ..., p'_{n_l}\}$ pour une donnée collectée ayant la valeur $c_n = \sum_{j=1}^{l} p'_{n_j}$. Avec l'hypothèse qu'il n'y a pas de défaillance ou perte de message, chaque message de la source atteint avec succès la destination, et donc l'ensemble de toutes les parts envoyées de tous les nœuds coïncidera exactement avec l'ensemble de toutes les parts reçues de tous les nœuds, à savoir :

$$\bigcup_{V} \{p_{n_1}, p_{n_2}, ..., p_{n_{2i+1}}\} = \bigcup_{V} \{p'_{n_1}, p'_{n_2}, ..., p'_{n_l}\}$$

Durant la phase de Diffusion, chaque nœud n transmet sa donnée collectée à ses voisins. Ensuite, ils propagent (honnêtement) cette valeur vers les voisins des voisins de n (ceux qui les succèdent dans l'ordre par rapport à la source n) et ainsi de suite. Les messages sont finalement réceptionnés par tous les voisins directs et indirects. Durant cette phase, le tableau h_n est maintenu pour contenir toutes les données collectées de la part de tous les nœuds du système. Par

conséquent, le calcul final nous donne :

$$\mathsf{result} = c_n + \sum_{\substack{0 \leq j < N \\ j \neq n}} h_n[j] = \sum_{j=0}^{N-1} c_j = \sum_{j=0}^{N-1} \sum_{t=1}^{l} p'_{j_t} = \sum_{j=0}^{N-1} \sum_{t=1}^{2i+1} p_{j_t} = \sum_{j=0}^{N-1} v_j.$$

Analyse de complexité. Pour la complexité spatiale, l'espace total que doit gérer localement chaque nœud est $\mathcal{O}(mN)$. Quant à la complexité des messages, le nombre de message envoyé par un nœud est $\mathcal{O}(N(d_n - m))$.

4.3.2 Correction

Dans cette section, nous étudions l'impact des nœuds malhonnêtes sur la vie privée et la précision du résultat du sondage quand EPol est déployé sur les graphes de $\mathcal{G}_2$ dans les pires et moyens cas. Toutes les preuves sont données dans [5, 55].

Analyse de confidentialité

La propriété de la vie privée exprime la capacité du système pour empêcher la divulgation des informations privées (dans notre cas le vote des utilisateurs) aux nœuds malhonnêtes. En d'autres termes, la coalition des malhonnêtes ne pourrait révéler toute information d'un nœud honnête au-delà de ce qu'elle peut déduire de son propre vote, le résultat des calculs et les parts des votes.

Soit γ_i la proportion des nœuds votant avec une génération de $2i+1$ parts durant la phase de Partage, où $0 \leq i \leq k$ et $\sum_{i=0}^{k} \gamma_i = 1$. Le vote d'un nœud peut être révélé avec (i) *certitude* si les nœud malhonnêtes sont sûrs de leur divulgation ; (ii) sinon avec *incertitude*. Supposons que le nombre de nœuds malhonnêtes est $D \leq (m-1)\Delta_G/2$ (présenté ultérieurement dans le théorème 4.7). Nous considérons les cas suivants de divulgation de votes :

Divulgation avec certitude. Nous discutons le cas où le vote d'un nœud donné peut être divulguée avec certitude dans deux cas : (i) tous les nœuds envoient $2k+1$ parts, et (ii) les nœuds peuvent envoyer un nombre impair $2i+1$ parts où $0 \leq i \leq k$ (en générant $i+1$ des parts du vote désiré et i parts opposées).

Théorème 4.1 (Divulgation sûre) Supposons qu'une coalition D de nœuds malhonnêtes connaît le nombre de parts envoyé par un nœud ayant voté avec $2i+1$ parts ($0 \leq i \leq k$). La probabilité P_{ce} pour que D révèle correctement et avec certitude le vote d'un nœud honnête est au plus de $\gamma_i \left(\frac{D}{N}\right)^{i+1}$. ■

Corollaire 4.1 Si tous les nœuds envoient $2k+1$ parts durant la phase de Partage, alors la probabilité pour que D divulgue avec certitude le vote d'un honnête nœud est au plus de $\left(\frac{D}{N}\right)^{k+1}$. ■

Nous traçons la limite P_{ce} comme une fonction $f(\gamma_i, i) = \gamma_i \left(\frac{D}{N}\right)^{i+1}$ dans la figure 4.3 pour $k = 3$, $N = 100$ et $D = 20$. Nous remarquons que P_{ce} croit avec l'augmentation de γ_i et la diminution de i. Ainsi, nous obtenons une confidentialité maximale lorsque tous les nœuds génèrent $2k+1$ parts, et une confidentialité minimale lorsque tous les nœuds produisent une seule part.

Si le résultat du sondage est N (resp. $-N$), il est trivial de déduire que tous les nœuds ont voté "+1" (resp. "−1") et dans ce cas particulier tous les votes sont divulgués. De plus, supposons

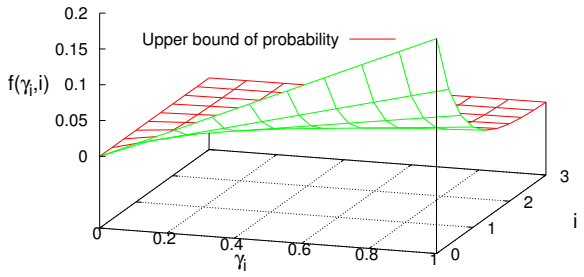


FIGURE 4.3 – Probabilité pour révéler avec certitude le vote d'un nœud.

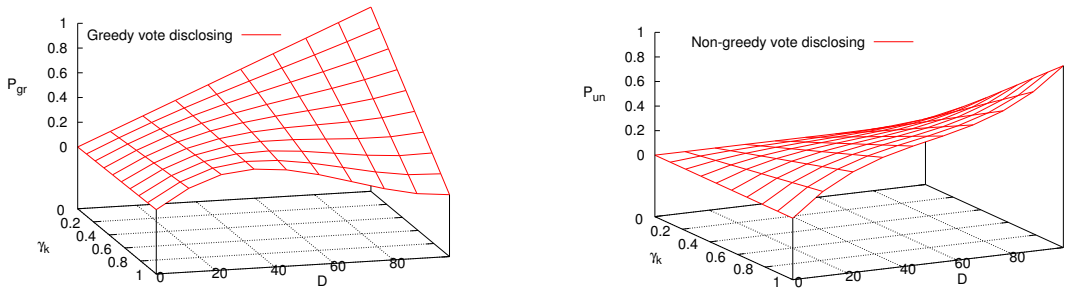
que tous les nœuds malhonnêtes votent "-1". Si le résultat final est $N-2D$, alors on peut deviner que le vote des nœuds honnêtes est "$+1$". Sans tenir compte de ces cas (à savoir, tous les nœuds honnêtes ne votent pas pour la même option), le théorème 4.2 nous montre le nombre maximum de votes qu'une coalition de malhonnêtes pourrait découvrir.

THÉORÈME 4.2 Si tous les nœuds honnêtes ne votent pas la même option, une coalition de D malhonnêtes nœuds peut révéler au plus le vote de $2D$ honnêtes nœuds.

Divulgation avec incertitude. Cette partie examine le cas où des nœuds malhonnêtes complotent pour révéler le vote d'un nœud honnête sans certitude. La coalition décide sur le vote d'un nœud sur la base des parts reçues (ils ont reçus soit toutes les parts soit quelques parts du vote). Ainsi, ils choisissent une des *stratégies* suivantes : (a) des $\rho+1$ parts identiques (pour certains $0 \leq \rho \leq k$) émises par un nœud donné, elles seront considérées comme son vote ; (b) Après avoir reçu la totalité des parts à partir d'un nœud donné, la valeur la plus représentée des parts reçues sera considérée comme son vote. La première stratégie est utilisée par de "gourmands" utilisateurs malhonnêtes qui veulent révéler rapidement le vote d'un utilisateur honnête (même s'ils viennent juste de recevoir une part). Quant à la deuxième stratégie, elles est utilisée par des utilisateurs malhonnêtes "non gourmands" qui sont patients et attendent pour recevoir toutes les parts de l'utilisateur honnête avant d'essayer de divulguer son vote. Nous présentons les probabilités qu'une coalition de nœuds malhonnêtes divulgue le vote d'un nœud honnête pour ces situations dans les théorèmes 4.3 et 4.4. Rappelons que chaque nœud ne connaît pas le nombre de parts générées par les autres et Δ_G est le diamètre de réseau.

THÉORÈME 4.3 (STRATÉGIE GOURMANDE) Supposons qu'une coalition de D nœuds malhonnêtes se base sur la règle suivante " à la réception de $\rho+1$ parts identiques ($0 \leq \rho \leq k$) à partir d'un nœud donné, elles seront considérées comme le vote de ce nœud". La probabilité pour que cette coalition révèle correctement le vote d'un nœud est $P_{gr}(\rho) = \sum_{i=\rho}^{k} \gamma_i \cdot \binom{D}{\rho+1} \sum_{j=0}^{\rho} \binom{D-\rho-1}{j} / \binom{N}{j+\rho+1}$ et qui est bornée par $\sum_{i=\rho}^{k} \gamma_i \frac{N+1}{N-D+\rho+2} (\frac{D}{N-D+\rho+1})^{\rho+1}$. ■

Dans le théorème 4.3, le vote v d'un nœud honnête est découvert si ce nœud a envoyé $2i+1 \geq 2\rho+1$ parts dans lesquelles $\rho+1$ parts identiques représentant v et jusqu'à ρ parts de valeur $-v$ ont été reçues par les consommateurs malhonnêtes. En outre, par le théorème 4.3,



(a) Gourmand (b) Non-gourmand

FIGURE 4.4 – Probabilité pour révéler le vote d'un nœud avec incertitude.

la valeur P_{gr} augmente quand γ_i diminue (et i augmente) et D augmente. Par exemple, avec $N = 100$, $k = 1$ (c-à-d, chaque nœud vote, avec une part ou $2k + 1 = 3$ parts) et $\rho = 0$, nous traçons la probabilité P_{gr} en fonction de D et γ_k sur la figure 4.4(a). Comme prévu, la confidentialité du vote diminue (à savoir, P_{gr} augmente) lorsque γ_k diminue et D augmente.

THÉORÈME 4.4 (STRATÉGIE NON GOURMANDE) Supposons qu'une coalition de D nœuds malhonnêtes se base sur la règle suivante "la valeur la plus représentée des parts reçues à partir d'un nœud donné sera considérée comme le vote de ce nœud". La probabilité pour que la coalition révèle correctement ce vote est $P_{un} = \sum_{i=0}^{k} \gamma_i \cdot \sum_{j=1}^{i+1} \sum_{t=0}^{j-1} \binom{D}{j}\binom{D-j}{t}/\binom{N}{j+t}$, qui est bornée par $\left(\frac{D}{N}/(1-2\frac{D}{N})\right)\left[1-\sum_{i=0}^{k}\gamma_i(2\frac{D}{N})^{2i+1}\right]$. ■

Par le théorème 4.4, la quantité P_{un} croit lorsque les deux valeurs γ_i et D augmentent (et i aussi augmente). Pour $N = 100$ et $k = 1$, la figure 4.4(b) montre l'impact de D sur P_{un}. Selon ce résultat, comme prévu, la confidentialité du vote diminue (à savoir P_{un} augmente) lorsque les deux valeurs D et γ_k croissent.

Combinaison de la divulgation avec et sans certitude. L'objectif de cette partie est de présenter la situation où les nœuds malhonnêtes peuvent essayer de révéler le vote d'un nœud honnête soit avec certitude ou incertitude. Supposons qu'ils respectent les règles de la divulgation du vote avec et sans certitude. Du point de vue des nœuds malhonnêtes, ils veulent toujours que leur détection de vote soit aussi sûre que possible. Plus précisément, ils préfèrent que le vote soit révélé de manière plus sûre que d'autres cas. Par conséquent, leur stratégie est la suivante : ils essaient d'abord de divulguer un vote de nœud avec certitude. S'ils ne parviennent pas (par exemple, en raison d'un manque de messages), ils examineront donc la façon de détecter ce vote sans certitude. Cette stratégie hybride implique dans ce cas que la probabilité pour qu'un vote soit divulgué est $P_{com} = \max\{P_{ce}, P_{gr}, P_{un}\}$.

Précision du résultat final

Dans cette partie, nous examinons les impacts maximal et moyen de la coalition des malhonnêtes sur la précision du résultat final et ce lorsque nous déployons le protocole *EPol* sur la famille des graphes $\mathcal{G}_2$. Un nœud malhonnête peut influer sur le résultat du scrutin avec les *mauvais comportements* suivants :

(i) Comme un nœud ne peut générer et envoyer des parts qu'à ses propres consommateurs (sinon les parts sont abandonnées) et il y a au plus $2k+1$ consommateurs pour chaque nœud, ce dernier doit envoyer au plus $2k+1$ parts dans lesquelles au plus $k+1$ sont identiques. D'où, un nœud malhonnête peut ne pas se conformer au protocole en envoyant plus de $k+1$ (mais pas supérieure à $2k+1$) parts identiques.

(ii) Il corrompt les parts reçues en remplaçant chaque part "+1" par "−1" pour diminuer la valeur des données collectées.

(iii) Il modifie les données collectées provenant d'un nœud honnête ou envoie un message forgé (ou corrompu) dans la phase de Diffusion.

Pour pouvoir limiter l'impact des mauvais comportements des nœuds malhonnêtes sur le bon déroulement du sondage, les utilisateurs peuvent déclencher des *procédures de vérification*. A ce titre, les attaques (i) et (ii) ne peuvent être détectées sans l'inspection du contenu des parts. En effet, dans la première attaque, le pire des cas est lorsqu'un nœud malhonnête envoie toutes les $2k+1$ parts avec la valeur "−1". Dans l'attaque (ii), un nœud reçois au plus $2k+1$ parts (comme il possède au plus $2k+1$ producteurs). Aussi, la valeur calculée pour les données collectées doit être dans l'intervalle $[-(2k+1), 2k+1]$. Par contre, pour l'attaque (iii), un nœud malhonnête ne peut pas créer un message forgé contenant l'identité d'un autre nœud. De plus, cette activité ne porte pas atteinte au résultat final étant donné qu'un nœud reçoit directement un message de la source s (s'il est un voisin de s) ou obtient une majorité de messages reçus ($\lceil (m+1)/2 \rceil$) contenant les données correctes de s. Nous montrons l'impact de ces mauvais comportements sur la précision du sondage dans les théorèmes 4.5 et 4.6.

THÉORÈME 4.5 (IMAPCT MAXIMAL) Tout nœud malhonnête pourrait affecter le résultat final jusqu'à $6k+4$. ■

THÉORÈME 4.6 (IMPACT MOYEN) Soit α la proportion des nœuds ayant voté "+1". L'impact moyen du à un mauvais comportement d'un nœud malhonnête est $I_{avg} = \left[\sum_{i=0}^{k} \gamma_i (2i+1)\right]\left[1 + 2\sum_{i=0}^{k} \gamma_i \frac{i+\alpha}{2i+1}\right] + 1$. ■

La quantité I_{avg} est minimisée lorsque tous les nœuds génèrent le même nombre de parts (par exemple $2i+1$), et ainsi, $I_{avg} = 2(2i+\alpha+1)$. Dans le pire des cas, un nœud malhonnête envoie $2k+1$ parts, et l'impact moyen minimisé est $I_{avg} = 2(2k+\alpha+1)$.

COROLLAIRE 4.2 Le résultat attendu biaisé est dans l'intervalle $[(2\alpha-1)N - (6k+4)D; (2\alpha-1)N]$. Plus particulièrement, si tous les nœuds envoient $2k+1$ parts, alors ce résultat biaisé est $(2\alpha-1)N - (4k+2\alpha+2)D$. ■

Sur la base des théorèmes 4.1-4.6 et les corollaires 4.1 et 4.2, pour un paramètre fixe k, le nombre d'utilisateurs votant avec un nombre élevé de parts (par exemple, $2k+1$ parts) influe sur la confidentialité des votes et de la précision du résultat du sondage. Plus concrètement, si nous nous soucions de la confidentialité du vote, nous devons augmenter le nombre de nœuds générant $2k+1$ parts puisque la probabilité de divulguer le vote d'un nœud avec certitude (P_{ce}) et une incertitude gourmande (P_{gr}) va diminuer. Mais cela augmente la probabilité P_{un} de la divulgation non gourmande de vote ainsi que l'impact sur le résultat final. En revanche, si nous prenons soin de l'exactitude du résultat final, nous devrions réduire le nombre de nœuds votant avec 2k +1 parts puisque cela réduit l'impact sur le résultat final. Il diminue aussi P_{un}. Cependant, ceci augmente les valeurs P_{ce} et P_{gr}.

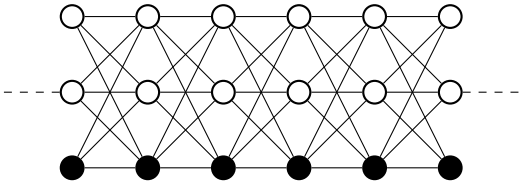


FIGURE 4.5 – Un graphe où notre protocole tolère $\frac{m-1}{2}\Delta_G$ malhonnête nœuds.

Tolérance aux nœuds malhonnêtes

Dans cette partie, nous calculons le nombre maximum de nœuds malhonnêtes que notre protocole *EPol* peut admettre. Rappelons que Δ_G est le diamètre du réseau.

THÉORÈME 4.7 (BORNE DES NŒUDS MALHONNÊTES) Le nombre maximum de nœuds malhonnêtes que *EPol* peut tolérer est $\frac{m-1}{2}\Delta_G$. ■

Un exemple de graphe qui tolère $D = \frac{m-1}{2}\Delta_G$ nœuds malhonnêtes est illustré dans la figure 4.5 pour $m = 3$.

Il est également à noter que notre protocole peut tolérer plus de $\sqrt{N}$ nœuds malhonnêtes pour une structure à base d'anneau qui a été proposée dans [50, 51] (ces travaux tolèrent moins de $\sqrt{N}$ nœuds malhonnêtes). En effet, comme cette structure a le diamètre $\Delta_G = \sqrt{N}$, et avec le paramètre $m \geq 3$, la borne supérieure de D n'est pas moins de $\sqrt{N}$.

COROLLAIRE 4.3 Si $D \leq \frac{m-1}{2}\Delta_G$ alors un nœud décide à tort sur les données collectées d'un autre nœud avec la probabilité qui converge exponentiellement vite vers 0 en N (et ΔG). ■

4.4 Sondage en pratique

4.4.1 Défaillance et perte de messages

Nous supposons que les nœuds communiquent par UDP (User Datagram Protocol) et ils peuvent subir une perte de message sur les canaux de communication. En outre, les nœuds peuvent être peu fiables, ce qui pourrait mener à la perte des messages en raison de situation de défaillances de l'expéditeur. En considérant la présence de pannes des nœuds et des pertes de messages dans le système, cette partie analyse l'effet de ces facteurs sur la précision final du sondage ainsi que la terminaison du protocole. Ici, nous supposons que le système ne contient pas de nœuds malhonnêtes.

Impact sur la terminaison. Nous considérons l'effet sur la terminaison du protocole en calculant la probabilité pour qu'un nœud défaillant se prononce sur le résultat final, à savoir, il n'a pas décidé sur au moins une des données collectées par une certaine source s.

Nous supposons qu'un nœud est défaillant avec une probabilité r (et il n'a jamais repris après cet incident) et un message est perdu (après la transmission d'un expéditeur) avec une probabilité l. Nous définissons une probabilité q pour qu'un nœud ne parvienne pas à envoyer des parts dans la phase de Partage, et nous avons $q = r + (1 - r)l$.

Un nœud n *échoue à transmettre* des données collectées provenant de la source s à un nœud si : (i) soit il est défaillant, (ii) il n'a pas lui-même décidé sur cette valeur, ou (iii) il a transmis un message, mais il est perdu lors de la transmission. En outre, un nœud n aussi *échoue à décider* sur des données recueillies de la source s si l'un des événements suivants apparaît :

1. $n = s$: s n'arrive pas à faire des calculs sur ses données collectées c_s.
2. $n \in \Gamma(s)$: n ne reçoit pas un message de transmission de la part de s.
3. $n \notin \Gamma(s)$: n reçoit au plus $(m-1)$ messages à partir des voisins (qui le précèdent selon l'ordre défini par rapport à la source s), c-à-d plus de $\beta_n(s) - m$ (précédents) voisins échouent pour transmettre les données collectées.

Nous définissons par e_{n_i} et z_{n_i}, respectivement, la probabilité pour un nœud n à distance i de la source s échouant à transmettre et décider sur les données collectées de s. Nous avons donc : $e_{n_i} = r + (1-r)[z_{n_i} + (1-z_{n_i})l]$, où $z_{n_0} = z_s = \sum_{j=0}^{|\mathcal{R}_s|-1} \binom{|\mathcal{R}_s|}{j}(1-q)^j q^{(2k+1)-j}$, $z_{n_1} = e_{n_0}$, et $z_{n_i} = \sum_{j=0}^{m-1}\Big[\binom{\beta_n(s)}{j}\prod_{t=1}^{j}(1-e_{n_{t_{l_t}}})\prod_{p=j+1}^{\beta_n(s)} e_{n_{p_{l_p}}}\Big]$

où $i \geq 2$, $\{n_1, n_2, \ldots, n_{\beta_n(s)}\}$ et $\{l_1, l_2, \ldots, l_{\beta_n(s)}\}$ sont respectivement les ensembles des précédents amis de n (selon l'ordre par rapport à s) et leurs distances par rapport s. Il est à noter que l_j pourrait être plus grand que i pour $j = 1, 2, ...\beta_n(s)$.

Par exemple, pour le graphe de la figure 4.6(a), si $i \geq 2$: $z_{n_i} = \sum_{j=0}^{m-1}\binom{\beta_n(s)}{j}(1-e_{i-1})^j e_{i-1}^{\beta_n(s)-j}$.

Dans le pire des cas, la source s possède $|\mathcal{R}_s| = 2k+1$ et un nœud ne décide pas sur les données collectées de s avec une probabilité $z_{n_{Rad(s,G)}}$ où $Rad(s,G) = \max_{u\in V}\{dist(s,u)\}$ et $dist(u,v)$ est la distance entre les nœuds u et v. Ainsi, un nœud ne parvient pas à décider sur le résultat final s'il a échoué à décider sur au moins une donnée collectée d'une certaine source, c-à-d $z_{n_{\Delta_G}}$ avec $\Delta_G = \max_{s\in V}\{Rad(s,G)\} = \max_{u,v\in V}\{dist(u,v)\}$.

Pour notre protocole $EPol$, la terminaison est détectée en comptant simplement le nombre de messages reçus. Cependant, à cause des défaillances des nœuds et la perte de message d'un expéditeur, d'autres nœuds pourraient ne pas savoir quand ils devraient cesser d'attendre des messages et démarrer la prochaine phase. Ils sont incapables de calculer la somme de leurs données recueillies, ou de prendre des décisions sur des données recueillies provenant d'une certaine source et donc sur le résultat final en raison du manque d'information. Par conséquent, nous supposons que : lorsque le nombre possible des données recueillies reçues de la source s croît au-delà d'un seuil donné $\lambda.m$ où $0 < \lambda \leq 1$, le nœud prend Δt secondes pour décider. La raison de l'utilisation de cette hypothèse vient du fait qu'un nœud choisit des données collectées en tant que valeur ayant une majorité dans l'ensemble $\mathcal{C}_n[s]$. Cela signifie que le nœud peut décider sur les données à chaque fois qu'il obtient plus que $m/2$ messages contenant les mêmes valeurs. Et ainsi, nous choisissons souvent $\lambda \geq \frac{1}{2} + \frac{1}{m}$.

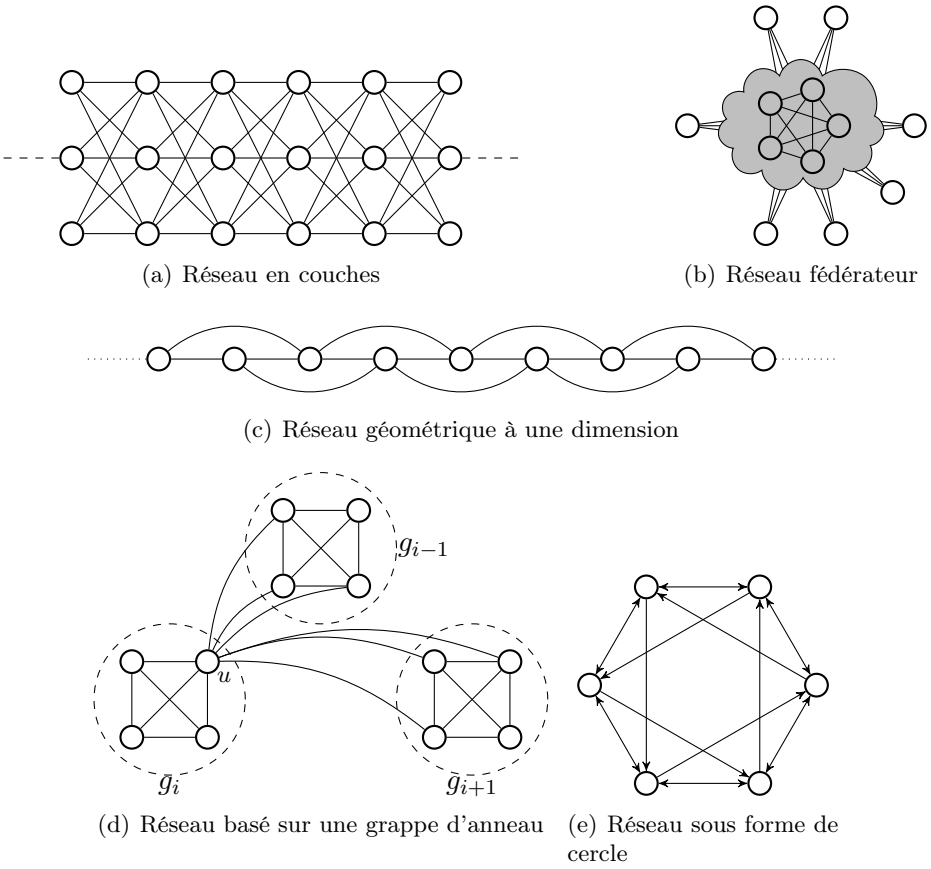
Impact sur la précision du résultat final. Le théorème 4.8 donne l'impact sur le résultat du sondage.

Théorème 4.8 L'impact maximum d'une panne de nœud sur la précision du scrutin est de $3k+2$. ■

Une panne de nœud affecte le résultat final lorsque ce nœud a des informations uniques qui n'ont pas encore été répliquées, typiquement les parts de votes. Il se bloque (i) pendant l'envoi de ses parts aux consommateurs, ou (ii) après avoir calculé la somme des parts des producteurs. Le premier cas affecte le résultat final jusqu'à $k+1$ (s'il se bloque après l'envoi de k parts de $-v$). Le dernier cas affecte jusqu'à $2k+1$. Par conséquent, l'impact d'un tel incident sur le résultat final est au plus $3k+2$.

4.4.2 Exemples de réseaux

Notre protocole $EPol$ nécessite des graphes dans une famille satisfaisant la propriété de m-diffusion. Ici, nous illustrons quelques graphes particuliers de cette famille. Les exemples de ces



(a) Réseau en couches

(b) Réseau fédérateur

(c) Réseau géométrique à une dimension

(d) Réseau basé sur une grappe d'anneau

(e) Réseau sous forme de cercle

FIGURE 4.6 – Exemples de graphes satisfaisant la propriété 3-diffusion.

graphes pour $m = 3$ sont également représentés sur la figure 4.6 et ils peuvent être généralisés pour toute valeur de m.

(a) **Réseau en couches.** Chaque couche contient au moins m nœuds et chaque nœud est relié vers tous les autres nœuds dans la couche voisine. Le nombre de nœuds dans chaque couche peut être différent. Pour chaque nœud source, un ordre (croissant) répondant à la propriété m-diffusion (par rapport à cette source) peut être défini sur la base de la distance séparant tout nœud de la source.

(b) **Réseau fédérateur.** Le graphe comprend une *colonne vertébrale* qui est densément graphe connexe, et d'autres nœuds en dehors de la colonne qui se connectent directement à au moins m nœuds dans la colonne. Un ordre répondant à la propriété m-diffusion (par rapport à une source) est l'ordre déterminé dans le cadre du sous-graphe de la colonne, suivi par tous les nœuds restants qui ne sont pas dans la colonne vertébrale.

(c) **Réseau géométrique à une dimension.** Tous les nœuds sont agencés selon une ligne, et il y a une connexion entre deux nœuds si leur distance est inférieure à un seuil fixé. Chaque nœud a au moins m connexions. Un ordre sur les nœuds (par rapport à une source) est l'ordre de leur distance euclidienne qui les séparent de la source.

(d) **Réseau basé sur une grappe en anneau.** Les N nœuds sont regroupées en $r = \sqrt{N}$ groupes *ordonnés*, à partir de g_0 à g_{r-1}. Chaque groupe est une clique. Un nœud n dans le groupe g_i est également lié à un nombre fixe de $\mathcal{S}_n$ nœuds dans le groupe suivant ($\mathcal{S}_n \subset$

$g_{i+1 \bmod r}$), et un nombre fixe de $\mathcal{R}_n$ nœuds dans le groupe précédent où $|\mathcal{S}_n| = |\mathcal{R}_n| = 2k+1$. Ainsi, tous les groupes forment pratiquement un anneau avec g_0 étant le successeur de g_{r-1}. En fait, cette structure est un cas particulier des réseaux en couches présenté ci-dessus, dans laquelle chaque couche est une clique de taille $\sqrt{N}$, et toutes les couches forment pratiquement un anneau. Ainsi, pour chaque nœud source, un ordre (croissant) répondant à la propriété m-diffusion (par rapport à une source) est l'ordre des nœuds par rapport à la distance de la source. Cet ordre est également établi en fonction de la distance entre la source et la direction d'envoi de messages. Par exemple, d'un groupe g_i à $g_{i+1 \bmod r}$. Notons que ces réseaux sont utilisés dans [8, 35, 50, 51].

(e) **Réseau sous forme de cercle.** Considérons un graphe $G_0 \in \mathcal{G}_2$ de taille $N > 2k+1$ où chaque nœud n a un ensemble de consommateurs $\mathcal{S}_n$ qui est la sortie de la fonction :

$$f\colon A \to 2^A$$
$$n \mapsto \{(n-1) \bmod N, (n+1) \bmod N, \ldots, (n+2k) \bmod N\}$$

avec $A = \{0, 1, 2 \ldots, (N-1)\}$. L'ensemble des producteurs est $\mathcal{R}_n = \{(n+1) \bmod N, (n-1) \bmod N, (n-2) \bmod N, \ldots, (n-2k) \bmod N\}$ de taille $2k+1$. La figure 4.6(e) montre un graphe avec $N = 6$ et $k = 1$ dans lequel un arc de u à v indique que le nœud v est un consommateur de u. Pour chaque nœud source, un ordre (croissant) répondant à la propriété m-diffusion (par rapport à cette source) est défini selon la distance séparant les nœuds de la source.

Tous les protocoles dans [7, 8, 35, 50, 51] ne peuvent pas être déployés dans ce graphe puisque la condition $\mathcal{S}_n \cap \mathcal{R}_n = \varnothing$ (proposée dans ces travaux) n'est pas satisfaite pour tout nœud n.

4.5 Etat de l'Art

Plusieurs travaux récents relatifs au partage du secret et aux sondages distribués ont été proposés. Nous présentons quelques travaux qui ne sont pas basés sur une structure de recouvrement et des calculs expensifs. Les systèmes de partage de secret [6, 113] peuvent être utilisés pour l'interrogation par rapport à l'addition. Cependant, comme ils n'offrent pas de protection pour les parts initiales, le résultat est probablement affecté par la présence de nœuds malhonnêtes. Le procédé du partage de secret vérifiable (VSS : Verifiable Secret Sharing Scheme) et le protocole multi-parties (MPC : Multi-Party Computation) dans [21, 96] calculent de manière privée les parts d'un nœud et donnent une sortie avec une petite erreur si la majorité des nœuds est honnête. Néanmoins, sans condition sur l'entrée initiale, un nœud malhonnête peut partager des données arbitraires, et biaiser la sortie. Ces protocoles utilisent également la cryptographie. Cet inconvénient est également valable pour d'autres études basées sur MPC tels que [24, 25, 26, 27] même si les complexités en temps et communication sont améliorées. Les auteurs de [85] ont proposé AMPC qui offre aux utilisateurs l'anonymat sans utiliser la cryptographie, mais ce travail utilise la notion de groupe. Basé sur AMPC et des vecteurs de contrôle améliorés, le protocole de vote électronique [84] est le protocole sécurisé en terme de la théorie de l'information. Mais il définit des rôles différents pour les utilisateurs, et donc il est différent de notre problématique. Les schémas de classement distribués sont également liés à notre préoccupation. Cependant, ils essaient de concevoir un mécanisme de classement précis plutôt que de fournir des systèmes de vote efficaces [54, 105] et traiter du problème de la vie privée [32]. Dans [112], une notion similaire à la propriété m-diffusion permet une propagation minimale des parts d'un secret. Mais, dans notre travail, ce concept est utilisé pour créer une majorité pour décider de la valeur correcte lors de la phase de diffusion de notre protocole.

Algorithme	Graphe	Impact Max.	Privacy	Nb. nœuds Malhonnêtes	Complexité		Crash
					Spatiale	Message	
DPol [50]	Recouvrement	$(6k+2)D$	$(D/N)^{k+1}$	$D < \sqrt{N}$	$\mathcal{O}(rk + \|g_i\|)$	$\mathcal{O}(rk + \|g_i\|)$	Oui
DPol* [51]	Recouvrement	$(6k+4)D$	$(D/N)^{k+1}$	$D < \sqrt{N}$	$\mathcal{O}(rk + \|g_i\|)$	$\mathcal{O}(rk + \|g_i\|)$	Oui
Pol* [56]	Originel	$(6k+4)D$	$(D/N)^{k+1}$	$D < N/5$	$\mathcal{O}(N^2)$	$\mathcal{O}(k + N^2)$	Non
MPOL* [35]	Recouvrement	$(6k+2)D$	$(D/N)^{k+1}$	$D < \sqrt{N}$	$\mathcal{O}(rk^2\|g_i\|)$	$\mathcal{O}(rk + \|g_i\|)$	Non
PDP [8]	Recouvrement	$2(k+N)D$	$(D/N)^{k+1}$	$D \geq k+1$	$\mathcal{O}(rk + \|g_i\|)$	$\mathcal{O}(rk + \|g_i\|)$	Non
DiPA [7]	Recouvrement	$2(k+N)D$	$(D/N)^{k+1}$	$D \geq k+1$	$\mathcal{O}(rk + \|g_i\|)$	$\mathcal{O}(rk + \|g_i\|)$	Non
EPol*	Originel	$(6k+4)D$	$(D/N)^{k+1}$	$D \leq (m-1)\Delta_G/2$	$\mathcal{O}(mN)$	$\mathcal{O}(N(d_0 - m))$	Oui

TABLE 4.1 – Comparaison des protocoles distribués de sondage où "Impact Max." : la différence maximale entre la sortie et le résultat attendu, "Privacy" : la probabilité de divulgation du vote, "Nb. Nœuds Malhonnêtes" : le nombre de nœuds malhonnêtes que le système peut tolérer, "la complexité spatiale" : l'espace total qu'un nœud doit détenir, "la complexité du message" : le nombre de messages envoyés par un nœud, r : nombre de groupes, $|g_i|$: taille du groupe, d_0 : degré maximum de nœud, Δ_G : diamètre du réseau. Les entrées marquées du symbole (*) montrent les résultats pour les sondages binaires.

DPol [50, 51] est un protocole simple de vote distribué sans utiliser de cryptographie, où les nœuds sont préoccupés par leur réputation. Il garantit la confidentialité et la précision malgré la présence de nœuds malhonnêtes en combinant le partage de secret et des procédures de vérification. Par ailleurs, le protocole proposé est basé sur une construction de recouvrement, qui est au dessus du graphe social réel. Il ne prend pas en compte tous les liens sociaux entre les nœuds dans le sens où il utilise une affectation uniforme des nœuds à un groupe. Ceci n'est pas pratique comme nous devons cibler un réseau particulier utilisant la notion de groupe au lieu de préserver la structure originelle du graphe social. De plus, le nombre de nœuds doit être un carré parfait pour qu'un graphe avec N nœuds soit divisé en $\sqrt{N}$ groupes de taille $\sqrt{N}$. Au contraire, nous proposons un protocole qui peut être déployé sur une structure plus générale. Nous pouvons observer que les graphes de la famille $\mathcal{G}_2$ comprend la structure de recouvrement utilisée dans DPol. Récemment, plusieurs extensions et des protocoles inspirés de l'idée de DPol ont été proposées. PDP [8] et son extension Dipa [7] prennent en compte le nombre de nœuds divisant leurs entrées pour améliorer la complexité et la vie privée. Toutefois, l'impact des participants malhonnêtes sur le résultat final est jusqu'à $2(k+\sqrt{N})D$ (où $D < \sqrt{N}$). Cette borne est si élevée en comparaison à l'impact maximum théorique, qui est $2N$. Par conséquent, un nœud peut toujours obtenir un résultat incorrect même si le système dispose d'un petit nombre de nœuds malhonnêtes. En outre, ils utilisent la même structure de recouvrement que DPol. MPOL [35] permet de calculer exactement les comptes de vote pour chaque candidat au lieu d'un gagnant parmi plusieurs candidats, mais encore une fois, il est basé sur une structure d'anneau. Le protocole dans [45] et *AG-S3* [47] peuvent être utilisés pour le scrutin d'une manière évolutive et sécurisée, mais ils utilisent soit (i) un recouvrement à base d'anneau, ou (ii) la cryptographie. Les auteurs de [56] ont proposé un protocole de sondage distribué qui ne nécessite pas une structure de recouvrement au dessus du réseau social originel. Il peut être déployé sur une famille de réseaux sociaux ayant une structure plus riche et il garantit la confidentialité de vote et l'exactitude du résultat final. Cependant, la complexité de la communication est quadratique en N dans le pire des cas

Nous illustrons dans la table 4.1 la comparaison des contributions entre notre travail et d'autres protocoles de vote répartis. Cette table montre que notre protocole *tolère plus* de nœuds malhonnêtes et a de *meilleures complexités* que d'autres. Plus particulièrement, si le graphe est un cycle structuré, en comparaison à DPol [50, 51], notre protocole a la même complexité de messages, mais tolère plus des nœuds malhonnêtes et calcule plus précisément le résultat du scrutin. Il est également noté que DPol considère seulement l'effet des crash, et non pas la perte de message.

4.6 Conclusion

Dans ce chapitre, nous avons présenté EPol, un protocole simple de sondage distribué et défini sur une famille de graphes sociaux. Nous avons montré que leurs structures se prêtent bien pour assurer la confidentialité des votes et limiter l'impact des comportements malhonnêtes sur la précision des résultats du scrutin. Contrairement à d'autres travaux, notre protocole est déployé sur une famille plus générale de graphes, et nous avons obtenu des résultats similaires (voire même meilleurs dans certains cas). En outre, nous avons introduit des règles de divulgation de vote avec certitude et incertitude pour les nœuds malhonnêtes, et présenté les probabilités de détection de vote dans ces cas. Nous avons également analysé l'effet des pertes de messages et les défaillances des noeuds sur la précision et la terminaison de notre protocole. En utilisant des structures plus riches de graphes sociaux, les complexités spatiale et de communication de EPol sont approximativement linéaires.

Conclusion

Ce document a présenté quelques travaux de recherche que j'ai menés au cours des dernières années autour de la synchronisation et la protection des données partagées dans les systèmes collaboratifs. La conception de protocoles de synchronisation reste une tâche ardue et nécessite donc un cadre formel couvrant toutes les facettes de cette tâche. Par ailleurs, il est très difficile de trouver un modèle général pour protéger les données dans les systèmes collaboratifs. En effet, ces derniers se distinguent par le type de collaboration (centralisée ou distribuée) et surtout, et pas des moindres, la structure des données partagées (linéaire, arborescente, ou graphique). Ces différences imposent donc de trouver des solutions appropriées qui concilient la collaboration et la protection des données partagées.

Dans ce qui suit, je vais dresser un bilan sur mes contributions de recherche présentées dans ce manuscrit et exposer brièvement des perspectives pour de nouvelles contributions.

Bilan

Dans la première contribution, nous avons traité du problème de maintien de cohérence en utilisant le modèle des transformées opérationnelles. Dans ce modèle, chaque objet collaboratif doit posséder sa propre fonction de transformation qui doit satisfaire deux propriétés **TP1** et **TP2**. La satisfaction de telles propriétés (et plus particulièrement **TP2**) reste problématique pour des objets linéaires (tels que les documents textes). A ce titre, nous avons montré l'impossibilité de trouver une fonction de transformation pour assurer la cohérence des objets linéaires altérés par de simples opérations d'insertion et de suppression. Ce résultat négatif a été obtenu grâce la mise en œuvre d'une méthode de synthèse de contrôleur basée sur les automates de jeu. Néanmoins, nous avons enrichi la signature de l'opération d'insertion par une méta-donnée pour définir une nouvelle fonction de transformation dont les deux propriétés **TP1** et **TP2** ont été vérifiées par une technique de model-checking. Ensuite, nous avons mené une étude formelle concernant la combinaison des approches TO et d'annulation des opérations. La description de cette combinaison comme un problème de satisfaction de contraintes (CSP) nous a permis de vérifier l'impact de la commutativité sur les fonctions de transformation pour assurer à la fois la cohérence et l'annulation.

Dans la deuxième contribution, nous avons proposé un modèle générique de contrôle d'accès qui se prête bien aux spécificités des systèmes collaboratifs. Ce modèle repose sur la réplication optimiste de l'objet partagé ainsi que sa politique d'autorisation. Cette double réplication confère à la partie administrative une souplesse et une grande réactivité. Pour valider et démontrer l'efficacité de notre modèle, nous avons développé des applications collaboratives pour évaluer leurs performances sur la plate-forme GRID'5000 distribuée. Ces mesures se sont avérées prometteuses et indiquent que notre modèle induit peu de coûts en terme de traitements. Pour vérifier qu'il n'y a pas d'interférence sur les propriétés sous-jacentes à un système collaboratif, nous avons utilisé

une technique symbolique de model-checking borné pour spécifier formellement l'empilement de notre contrôle d'accès à un système collaboratif. L'analyse d'une telle spécification nous a montré que le contrôle d'accès est uniformément appliqué sur tous les sites et préserve la cohérence. Elle a également permis de valider certains choix conceptuels de notre modèle.

Dans la troisième contribution, nous avons fourni une solution pour le problème de la réécriture des requêtes XPath en présence de la récursivité dans les DTDs. Nous avons étendu la classe "downward" de XPath avec des axes et des opérateurs, et nous avons montré que le fragment résultant $\mathcal{X}^{\Uparrow}_{[n,=]}$ peut être utilisé pour réécrire de manière efficace toute requête de $\mathcal{X}$, posée sur une vue, en une autre qui peut être évaluée sur le document originel. Notre proposition donne la première solution pratique au problème de réécriture. En effet, nous avons développé SVMAX, un système qui facilite la spécification et l'exécution des droits d'accès en lecture et mise-à-jour pour les données XML. Les droits de lecture et mise-à-jour de SVMAX sont définis par annotation des grammaires DTD et appliqués en utilisant le principe de la réécriture. L'expérimentation menée a montré l'efficacité de notre approche. Plus précisément, la traduction des requêtes de $\mathcal{X}$ en requêtes $\mathcal{X}^{\Uparrow}_{[n,=]}$ n'a pas d'impact sur les performances du système pour répondre aux requêtes.

Dans la quatrième contribution, nous avons présenté EPol, un protocole simple de sondage dans les réseaux sociaux décentralisés où le caractère confidentiel des votes et la réputation du votant sont très critiques. Notre protocole est asynchrone et ne nécessite ni autorité centrale ni cryptographie. Il peut être déployé sur une famille riche de graphes sociaux. Nous avons montré que leurs structures se prêtent bien pour préserver la vie privée des utilisateurs et limiter l'impact des comportements malhonnêtes sur la précision des résultats du scrutin. Pour mieux illustrer les comportements des utilisateurs malhonnêtes, nous avons introduit des règles de divulgation de vote avec certitude et incertitude, et présenté les probabilités de détection de vote dans ces cas. Nous avons également analysé l'effet des pertes de messages et les défaillances des nœuds sur la précision et la terminaison de notre protocole.

Perspectives

A l'issue de ces travaux, plusieurs directions de recherche s'avèrent intéressantes à explorer et à développer. Dans ce qui suit, je présente brièvement quelques pistes :

Conception formelle des algorithmes de réplication optimiste

Il sera intéressant d'étudier l'existence des fonctions de transformation pour des objets partagés non-linéaires (arbre, graphe, etc.) en utilisant les techniques de synthèse de contrôleur et model-checking. Ce qui permet de savoir si les signatures des opérations primitives altérant l'état d'un objet sont suffisantes ou non pour définir des fonctions consistantes. Quant aux modèles de transformation fourni par notre méthode basée sur la résolution des contraintes, la généralisation de ces résultats permettra de mettre au point un cadre générique pour un ensemble fini et arbitraire d'opérations. Ce cadre va constituer une base pour les développeurs d'applications collaboratives pour choisir une instance de transformation correcte. De manière analogue, les modèles de transformation produits par notre solveur CSP pourraient être associés à des objets concrets. Ainsi, la vérification de la propriété de cohérence sera effectuée sur des modèles finis.

Délégation optimiste de l'administration des autorisations

Même s'il est flexible et générique, notre modèle de contrôle d'accès ne permet pas la délégation de l'administration des droits qui pourrait être une extension très importante pour la gestion

optimiste des autorisations dans les systèmes collaboratifs. En effet, il peut y avoir plusieurs raisons pour qu'un membre d'un groupe délègue ses droits à un ou plusieurs collaborateurs. Par exemple, un utilisateur peut quitter provisoirement le groupe tout en restant actif grâce à un processus de délégation, de telle sorte que les délégués peuvent définir au nom de cet utilisateur des droits d'accès sur ses objets. En se reconnectant au groupe, notre utilisateur pourra valider ou invalider les actions effectuées par ses délégués. En optant pour le partage de l'administration d'une politique de contrôle d'accès, il serait très intéressant d'examiner toutes les situations occasionnées par la concurrence entre les délégations et les changements de la politique ainsi que les modifications des objets partagés, et ce dans un environnement complètement dynamique.

Interaction entre les privilèges de lecture et de mise à jour

Dans notre modèle de contrôle d'accès pour les données XML partagées, ces dernières peuvent être consultées et modifiées par plusieurs utilisateurs. En plus des droits d'accès pour la lecture, des restrictions sur la mise à jour peuvent être imposées pour spécifier les parties du document XML que les utilisateurs peuvent modifier. Il est facile de montrer que la réécriture d'une opération de mise à jour par rapport à une spécification d'accès peut ne pas préserver les droits d'accès de la lecture. En d'autre termes, l'exécution d'une opération de mise à jour sur une vue virtuelle XML peut révéler à l'utilisateur l'existence de certaines données sensibles qu'il n'a pas le droit de voir. Nous prévoyons de mener une étude approfondie sur l'interaction entre les privilèges de lecture et mise à jour afin de préserver la confidentialité et l'intégrité des données.

Vers un sondage collaboratif avec plus de fonctionnalités

Dans notre travail, nous avons étudié le problème de sondage binaire, à savoir un vote qui considère une question posée ayant deux options (ou réponses) possibles. Nous envisageons donc de généraliser nos solutions pour gérer plusieurs questions et chaque question pourra avoir plusieurs options. Ainsi, le résultat final pour chaque question sera l'agrégation de tous les votes liés à cette question. L'organisation d'un sondage complètement décentralisée à plusieurs questions et plusieurs options va susciter de nouveaux besoins, comme la possibilité de mener des calculs statistiques (par exemple, déterminer le nombre d'utilisateurs ayant voté «oui» pour la première question et «non» pour la troisième question). Aussi, une question importante à étudier est le type de statistiques que l'on peut effectuer sur les résultats finaux sans révéler les votes des utilisateurs.

Bibliographie

[1] R. Abdunabi, I. Ray, and R. France. Specification and analysis of access control policies for mobile applications. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, SACMAT '13, pages 173–184, New York, NY, USA, 2013. ACM.

[2] J. P. Achara, A. Imine, and M. Rusinowitch. Descal - decentralized shared calendar for P2P and ad-hoc networks. In *10th International Symposium on Parallel and Distributed Computing, ISPDC 2011, Cluj-Napoca, Romania, July 6-8, 2011*, pages 223–231, 2011.

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

[4] S. Andrei, W.-N. Chin, and S. V. Cavadini. Self-embedded context-free grammars with regular counterparts. *Acta Inf.*, 40(5) :349–365, 2004.

[5] H. Bao Thien. *On the Polling Problem for Decentralized Social Networks.* Theses, INRIA Nancy ; LORIA - Université de Lorraine, Feb. 2015.

[6] J. C. Benaloh. Secret sharing homomorphisms : Keeping shares of a secret sharing. In *CRYPTO*, pages 251–260, 1986.

[7] Y. Benkaouz and M. Erradi. A distributed protocol for privacy preserving aggregation with non-permanent participants. *Computing*, 2014.

[8] Y. Benkaouz, R. Guerraoui, M. Erradi, and F. Huc. A distributed polling with probabilistic privacy. In *SRDS*, pages 41–50, 2013.

[9] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. Xml path language (xpath) 2.0 (second edition). *W3C Recommendation. Available at : http://www.w3.org/TR/2010/REC-xpath20-20101214/.*, December 2010.

[10] A. Bonifati, M. H. Goodfellow, I. Manolescu, and D. Sileo. Algebraic incremental maintenance of xml views. In *14th International Conference on Extending Database Technology (EDBT)*, pages 177–188. ACM, 2011.

[11] Boolean Satisfiability Research Group at Princeton. zChaff. [Online]. Available : `http://www.princeton.edu/$\sim$chaff/zchaff.html` (Accessed : 10 April 2014).

[12] H. Boucheneb and A. Imine. On model-checking optimistic replication algorithms. In *FMOODS/FORTE*, pages 73–89, 2009.

[13] H. Boucheneb, A. Imine, and M. Najem. Symbolic model-checking of optimistic replication algorithms. In *IFM*, pages 89–104, 2010.

[14] A. Bullock and S. Benford. An access control framework for multi-user collaborative environments. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '99, pages 140–149. ACM, 1999.

[15] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Limei. Efficient on-the-fly algorithms for the analysis of timed games. *CONCUR-LNCS*, 3653 :60–80, 2005.

[16] X. L. Charles Prud'homme, Jean-Guillaume Fages. *Choco3 Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[17] A. Cherif. *Access Control Models for Collaborative Applications.* Theses, Université Nancy 2, Nov. 2012.

[18] A. Cherif and A. Imine. A constraint-based approach for generating transformation patterns. In *Proceedings 14th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2015, Madrid, Spain, 5th September 2015.*, pages 48–62, 2015.

[19] A. Cherif, A. Imine, and M. Rusinowitch. Practical access control management for distributed collaborative editors. *Pervasive and Mobile Computing*, 15 :62–86, 2014.

[20] B. Choi. What are real dtds like ? In *Fifth International Workshop on the Web and Databases (WebDB)*, pages 43–48, 2002.

[21] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS*, pages 383–395, 1985.

[22] J. Clark and S. DeRose. Xml path language (xpath) 1.0. *W3C Recommendation. Available at : http: // www. w3. org/ TR/ xpath/ .*, November 1999.

[23] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20. ACM, 2001.

[24] R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung. Multi-autority secret-ballot elections with linear work. In *EUROCRYPT*, pages 72–83, 1996.

[25] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Trans. on Telecom.*, 8(5) :481–490, 1997.

[26] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, 2008.

[27] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[28] E. Damiani, M. Fansi, A. Gabillon, and S. Marrara. A general approach to securely querying xml. *Computer Standards & Interfaces*, 30(6) :379–389, 2008.

[29] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. Secretive birds : Privacy in population protocols. In *OPODIS*, 2007.

[30] M. Duong and Y. Zhang. Dynamic labelling scheme for xml data processing. In *On the Move to Meaningful Internet Systems, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Proceedings, Part II*, volume 5332 of *Lecture Notes in Computer Science*, pages 1183–1199. Springer, 2008.

[31] M. Duong and Y. Zhang. An integrated access control for securely querying and updating xml data. In *Proceedings of the Nineteenth Australasian Database Conference (ADC)*, volume 75 of *CRPIT*, pages 75–83. Australian Computer Society, 2008.

[32] D. Dutta, A. Goel, R. Govindan, and H. Zhang. The design of a distributed rating scheme for peer-to-peer systems. In *P2P Econ*, 2003.

[33] N. Eén and N. Sörensson. MiniSat, The MiniSat Page. [Online]. Available : `http:// minisat.se/`. (Accessed : 10 April 2014).

[34] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

[35] B. Englert and R. Gheissari. Multivalued and deterministic peer-to-peer polling in social networks with reputation conscious participants. In *TrustCom*, 2013.

[36] W. Fan, C. Y. Chan, and M. N. Garofalakis. Secure xml querying with security views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 587–598. ACM, 2004.

[37] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Smoqe : A system for providing secure access to xml. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 1227–1230. ACM, 2006.

[38] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular xpath queries on xml views. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*, pages 666–675. IEEE, 2007.

[39] W. Fan, J. X. Yu, J. Li, B. Ding, and L. Qin. Query translation from xpath to sql in the presence of recursive dtds. *VLDB J.*, 18(4) :857–883, 2009.

[40] L. Fegaras. Incremental maintenance of materialized xml views. In *Database and Expert Systems Applications - 22nd International Conference (DEXA 2011)*, volume 6861 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2011.

[41] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. Sporc : Group collaboration using untrusted cloud resources. In *OSDI*, pages 337–350, 2010.

[42] J. Ferrié, N. Vidot, and M. Cart. Concurrent undo operations in collaborative environments using operational transformation. In *CoopIS/DOA/ODBASE (1)*, pages 155–173, 2004.

[43] I. Fundulaki and S. Maneth. Formalizing xml access control for update operations. In *SACMAT 2007, 12th ACM Symposium on Access Control Models and Technologies*, pages 169–174. ACM, 2007.

[44] I. Fundulaki and M. Marx. Specifying access control policies for xml documents with xpath. In *SACMAT 2004, 9th ACM Symposium on Access Control Models and Technologies*, pages 61–69. ACM, 2004.

[45] S. Gambs, R. Guerraoui, H. Harkous, F. Huc, and A.-M. Kermarrec. Scalable and secure polling in dynamic distributed networks. In *SRDS*, pages 181–190, 2012.

[46] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 21–27. ACM, 2001.

[47] A. Giurgiu, R. Guerraoui, K. Huguenin, and A.-M. Kermarrec. Computing in social networks. *Infor. and Comp.*, 234 :3–16, 2014.

[48] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2) :444–491, 2005.

[49] B. Groz, S. Staworko, A.-C. Caron, Y. Roos, and S. Tison. Xml security views revisited. In *Database Programming Languages - DBPL 2009, 12th International Symposium*, volume 5708 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2009.

[50] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, and M. Monod. Decentralized Polling with Respectable Participants. In *OPODIS*, 2009.

[51] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, M. Monod, and Y. Vigfusson. Decentralized polling with respectable participants. *JPDC*, 72(1) :13–26, 2012.

[52] A. Gupta and I. S. Mumick. Maintenance of materialized views : Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2) :3–18, 1995.

[53] A. Gupta, I. S. Mumick, J. Rao, and K. A. Ross. Adapting materialized views after redefinitions : techniques and a performance study. *Inf. Syst.*, 26(5) :323–362, 2001.

[54] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *NOSSDAV*, New York, NY, USA, 2003. ACM.

[55] B. Hoang and A. Imine. Efficient and decentralized polling protocol for general social networks. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, pages 171–186, 2015.

[56] B.-T. Hoang and A. Imine. On the Polling Problem for Social Networks. In *OPODIS*, pages 46–60, 2012.

[57] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *PODC*, pages 355–364, 2012.

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

[59] H. Hu and G. Ahn. Enabling Verification and Conformance Testing for Access Control Model. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, New York, NY, USA, 2008. ACM.

[60] A. Imine. Coordination model for real-time collaborative editors. In *COORDINATION*, pages 225–246, 2009.

[61] A. Imine, A. Cherif, and M. Rusinowitch. A Flexible Access Control Model for Distributed Collaborative Editors. In W. Jonker and M. Petkovic, editors, *Secure Data Management*, volume 5776 of *Lecture Notes in Computer Science*, pages 89–106, 2009.

[62] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving correctness of transformation functions functions in real-time groupware. In *ECSCW*, pages 277–293, 2003.

[63] A. Imine, M. Rusinowitch, G. Oster, and P. Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, 351(2) :167–183, 2006.

[64] X. Jia. *From Relations to XML : Cleaning, Integrating and Securing Data*. Doctor of philosophy, Laboratory for Foundations of Computer Science. School of Informatics. University of Edinburgh, 2007.

[65] K. Kawagoe and K. Kasai. Situation, team and role based access control. *Journal of Computer Science*, 7(5), 2011.

[66] R. Krishnamurthy, V. T. Chakaravarthy, R. Kaushik, and J. F. Naughton. Recursive xml schemas, recursive xml queries, and relational storage : Xml-to-sql query translation. In *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, pages 42–53. IEEE Computer Society, 2004.

[67] G. M. Kuper, F. Massacci, and N. Rassadko. Generalized xml security views. In *10th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 77–84. ACM, 2005.

[68] G. M. Kuper, F. Massacci, and N. Rassadko. Generalized xml security views. *Int. J. Inf. Sec.*, 8(3) :173–203, 2009.

[69] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3), 1982.

[70] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1-2) :134–152, 1997.

[71] D. Le Berre and A. Parrain. The Sat4j Library, Release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64, 2010. System description.

[72] D. Li and R. Li. Ensuring Content Intention Consistency in Real-Time Group Editors. In *IEEE ICDCS'04*, Tokyo, Japan, March 2004.

[73] D. Li and R. Li. An operational transformation algorithm and performance evaluation. *Computer Supported Cooperative Work*, 17(5-6) :469–508, 2008.

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

[75] P. B. Lowry, A. M. Curtis, and M. R. Lowry. A Taxonomy of Collaborative Writing to Improve Empirical Research, Writing Practice, and Tool Development. *Journal of Business Communication*, 41(1) :66–99, 2004.

[76] B. Luo, D. Lee, W.-C. Lee, and P. Liu. Qfilter : rewriting insecure xml queries to secure ones using non-deterministic finite automata. *VLDB J.*, 20(3) :397–415, 2011.

[77] B. Lushman and G. V. Cormack. Proof of correctness of ressel's adopted algorithm. *Information Processing Letters*, 86(3) :303–310, 2003.

[78] H. Mahfoud. *Efficient Access Control to XML Data : Querying and Updating Problems.* Theses, Université de Lorraine, Feb. 2014.

[79] H. Mahfoud and A. Imine. A general approach for securely updating xml data. In *Proceedings of the 15th International Workshop on the Web and Databases (WebDB 2012)*, pages 55–60, 2012.

[80] H. Mahfoud and A. Imine. On securely manipulating xml data. In *Foundations and Practice of Security - 5th International Symposium (FPS 2012), Revised Selected Papers*, volume 7743 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2012.

[81] H. Mahfoud and A. Imine. Secure querying of recursive xml views : a standard xpath-based technique. In *Proceedings of the 21st World Wide Web Conference, WWW 2012 (Companion Volume)*, pages 575–576. ACM, 2012.

[82] H. Mahfoud and A. Imine. Efficient querying of XML data through arbitrary security views. *Trans. Large-Scale Data- and Knowledge-Centered Systems*, 22 :75–114, 2015.

[83] H. Mahfoud, A. Imine, and M. Rusinowitch. Svmax : a system for secure and valid manipulation of xml data. In *Proceedings of the 17th International Database Engineering & Applications Symposium (IDEAS)*, pages 154–161. ACM, 2013.

[84] D. Malkhi, O. Margo, and E. Pavlov. E-voting without 'cryptography'. In *Financial Cryptography*, pages 1–15, 2002.

[85] D. Malkhi and E. Pavlov. Anonymity without 'cryptography'. In *Financial Cryptography*, pages 108–126, 2001.

[86] M. Marx. Xpath with conditional axis relations. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2004.

[87] A. Mislove, A. Post, P. Druschel, and P. K. Gummadi. Ostra : Leveraging trust to thwart unwanted communication. In *NSDI*, 2008.

[88] MIT Sofware Design Group. Alloy : a language and tool for relational models. [Online]. Available : `http://alloy.mit.edu/alloy/`. (Accessed : 10 April 2013).

[89] M. Murata, A. Tozawa, M. Kudo, and S. Hada. Xml access control using static analysis. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 73–84. ACM, 2003.

[90] M. Murata, A. Tozawa, M. Kudo, and S. Hada. Xml access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3) :292–324, 2006.

[91] A. Nica. Incremental maintenance of materialized views with outerjoins. *Inf. Syst.*, 37(5) :430–442, 2012.

[92] G. Oster, P. Molli, P. Urso, and A. Imine. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *International Conference on Collaborative Computing : Networking, Applications and Worksharing (CollaborateCom)*, pages 1 –10, 2006.

[93] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *ICALP (2)*, pages 660–672, 2012.

[94] D. Povey. Optimistic security : a new access control paradigm. In *NSPW '99 : Proceedings of the 1999 workshop on New security paradigms*, pages 40–45. ACM, 2000.

[95] A. Prakash and M. J. Knister. A framework for undoing actions in collaborative systems. *ACM Trans. Comput.-Hum. Interact.*, 1(4) :295–330, 1994.

[96] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, 1989.

[97] A. Randolph, H. Boucheneb, A. Imine, and A. Quintero. On synthesizing a consistent operational transformation approach. *IEEE Trans. Computers*, 64(4) :1074–1089, 2015.

[98] A. Randolph, A. Imine, H. Boucheneb, and A. Quintero. Spécification et analyse d'un protocole de contrôle d'accès optimiste pour éditeurs collaboratifs répartis. *Ingénierie des Systèmes d'Information*, 19(6) :9–32, 2014.

[99] N. Rassadko. Policy classes and query rewriting algorithm for xml security views. In *Data and Applications Security XX, 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec)*, volume 4127 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2006.

[100] M. Ressel and R. Gunzenhäuser. Reducing the problems of group undo. In *GROUP '99 : Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 131–139, New York, NY, USA, 1999. ACM.

[101] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser. An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. In *ACM CSCW'96*, pages 288–297, Boston, USA, November 1996.

[102] R. L. Rivest. Chaffing and winnowing : confidentiality without encryption. In *RSA Laboratories CryptoBytes 4*, 1998.

[103] J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, and J. Siméon. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation. *Available at : `http://www.w3.org/TR/2008/REC-xml-20081126/`.*, 2008.

[104] J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, and J. Siméon. Xquery update facility 1.0. *W3C Recommendation. Available at : `http://www.w3.org/TR/xquery-update-10/`.*, March 2011.

[105] M. Rodriguez-Perez, O. Esparza, and J. L. Muñoz. Analysis of peer-to-peer distributed reputation schemes. In *CollaborateCom*, 2005.

[106] P. Samarati and S. D. C. di Vimercati. Access control : Policies, models, and mechanisms. In *Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design (FOSAD 2000)]*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2000.

[107] A. Samuel, A. Ghafoor, and E. Bertino. A framework for specification and verification of generalized spatio-temporal role based access control model. Technical report, Purdue University, 2007.

[108] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *Computer*, 29(2) :38–47, 1996.

[109] R. Sandhu and P. Samarati. Access control : principle and practice. *Communications Magazine, IEEE*, 32(9) :40–48, Sept 1994.

[110] R. S. Sandhu and R. K. Thomas. Conceptual foundations for a model of task-based authorizations. In *Seventh IEEE Computer Security Foundations Workshop - CSFW'94, Franconia, New Hampshire, USA, June 14-16, 1994, Proceedings*, pages 66–79, 1994.

[111] Z. Sbaï and K. Barkaoui. Vérification formelle des processus workflow collaboratifs. *CoRR*, abs/1306.4308, 2013.

[112] N. B. Shah, K. V. Rashmi, and K. Ramchandran. Secure network coding for distributed secret sharing with low communication cost. In *ISIT*, 2013.

[113] A. Shamir. How to share a secret. *Commun. ACM*, 22(11) :612–613, 1979.

[114] B. Shao, D. Li, and N. Gu. An algorithm for selective undo of any operation in collaborative applications. In *GROUP*, pages 131–140, 2010.

[115] P. D. N.-M. Shastry. *Integrated Healthcare IHE Pathway for the Patients : Patient Treatment Lifecycle Management (PTLM). Radiology Clinic, United Kingdom (2000)*, October 2012. `http://www.clinrad.nhs.uk/`.

[116] M. Sirivianos, K. Kim, and X. Yang. Socialfilter : Introducing social trust to collaborative spam mitigation. In *INFOCOM*, 2011.

[117] A. Stoica and C. Farkas. Secure xml views. In *Research Directions in Data and Applications Security, IFIP WG 11.3 Sixteenth International Conference on Data and Applications Security*, volume 256 of *IFIP Conference Proceedings*, pages 133–146. Kluwer, 2002.

[118] M. Suleiman, M. Cart, and J. Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *ACM GROUP'97*, pages 435–445, November 1997.

[119] C. Sun. Undo as concurrent inverse in group editors. *ACM Trans. Comput.-Hum. Interact.*, 9(4) :309–361, 2002.

[120] C. Sun and C. Ellis. Operational transformation in real-time group editors : issues, algorithms, and achievements. In *ACM CSCW'98*, pages 59–68, 1998.

[121] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving Convergence, Causality-preservation and Intention-preservation in real-time Cooperative Editing Systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1) :63–108, March 1998.

[122] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13(4) :531–582, 2006.

[123] D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(10) :1454–1470, 2009.

[124] B. ten Cate. The expressivity of xpath with transitive closure. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 328–337. ACM, 2006.

[125] M. Thimma, T. K. Tsui, and B. Luo. Hyxac : a hybrid approach for xml access control. In *18th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2013.

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

[127] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (TBAC) : A family of models for active and enterprise-oriented autorization management. In *Database Securty XI : Status and Prospects, IFIP TC11 WG11.3 Eleventh International Conference on Database Security, 10-13 August 1997, Lake Tahoe, California, USA*, pages 166–181, 1997.

[128] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar. Ensuring Spatio-temporal Access Control for Real-world Applications. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 13–22, New York, NY, USA, 2009. ACM.

[129] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access Control in Collaborative Systems. *ACM Comput. Surv.*, 37(1) :29–41, 2005.

[130] E. Torlak and G. Dennis. Kodkod for Alloy users. In *First ACM Alloy Workshop*, 2006.

[131] E. P. K. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993.

[132] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *ACM CSCW'00*, Philadelphia, USA, December 2000.

[133] S. Weiss, P. Urso, and P. Molli. An Undo Framework for P2P Collaborative Editing. In *4th International Conference on Collaborative Computing*, volume 10, pages 529–544, Orlando États-Unis, 11 2008. Springer Berlin Heidelberg.

[134] J. Woodcock and J. Davies. *Using Z : Specification, Refinement, and Proof.* Prentice Hall, 1996.

[135] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit : A near-optimal social network defense against sybil attacks. *Trans. Netw.*, 18(3) :885–898, 2010.

[136] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman. Sybilguard : defending against sybil attacks via social networks. *Trans. Netw.*, 16(3) :576–589, 2008.

Résumé

Les systèmes collaboratifs permettent la manipulation d'objets partagés (tels que les documents multimédia, les données sociales, etc.) par plusieurs personnes qui sont réparties dans le temps et l'espace. Néanmoins, les besoins identifiés autour de la gestion des données partagées constituent toujours des verrous scientifiques importants. En effet, les utilisateurs requièrent toujours une collaboration sans contraintes qui leur permet des accès rapides, cohérents et sécurisés aux données partagées. Dans cette thèse d'habilitation, je présente quelques travaux de recherche que j'ai menés autour des systèmes collaboratifs. La première partie de ce document aborde les problèmes de la cohérence et l'annulation pour des objets partagés et synchronisés par transformation et elle contient quelques résultats obtenus en utilisant des techniques de synthèse de contrôleur, model-checking symbolique et la résolution des contraintes. La deuxième partie présente un modèle conciliant la collaboration et le contrôle d'accès dans les systèmes collaboratifs et elle décrit un protocole basé sur la réplication de l'objet partagé avec la politique d'accès ainsi que sa vérification formelle. La troisième partie traite du problème d'accès à des informations confidentielles contenues dans des documents XML partagés par plusieurs utilisateurs et elle présente un modèle générique comportant un langage expressif et une technique de réécriture pour, respectivement, éditer et appliquer des politiques d'accès basées sur le standard XPath. Quant à la dernière partie, elle expose le problème de sondage dans les réseaux sociaux décentralisés et décrit la mise au point d'un protocole distribué basé sur le partage de secret, ne nécessitant aucune infrastructure cryptographique et tolérant la présence des utilisateurs malhonnêtes.

Mots-clés: Systèmes collaboratifs, Réplication, Cohérence, Contrôle d'accès, XML, Réseaux sociaux, Vie privée, Vérification formelle.

Abstract

Collaborative systems provide computer support for manipulating shared objects (such as, multimedia documents, social data, etc.) by several users that are temporally and spatially distributed. Nevertheless, the known requirements on managing shared data still raise major challenges. Indeed, users require unconstrained collaboration that allows them quick, consistent and secure access to shared data. In this habilitation thesis, I present some research works I conducted around collaborative systems. The first part of this document addresses the problems of data consistency and undoability for shared objects synchronized by transformation and it contains some results obtained using controller synthesis techniques, symbolic model-checking and constraint solving methods. The second part presents a model reconciling collaboration and access control in collaborative systems and describes a protocol based on the replication of the shared object along with the access policy and its formal verification. The third part deals with the access problem to confidential information stored in XML documents shared by multiple users and presents a generic model with an expressive language and a rewriting technique to edit and enforce respectively access policies based on standard XPath. As for the last part, it exposes the polling problem in decentralized social networks and describes the development of a distributed protocol based on secret sharing, requiring no cryptographic infrastructure and tolerating the presence of dishonest users.

Keywords: Collaborative systems, Replication, Consistency, Access control, XML, Social networks, Privacy, Formal verification.