UFR mathématiques et informatique | École doctorale IAEM Lorraine

Département de formation doctorale en informatique

Modèles de Contrôle d'Accès pour Les Applications Collaboratives

THÈSE

présentée et soutenue publiquement le 26 Novembre 2012

pour l'obtention du

Doctorat de l'université Nancy 2

(spécialité informatique)

par

Asma CHERIF

Composition du jury

Président :	Kamel SMAILI	Pr. Université de Lorraine
Rapporteurs :	Frédéric CUPPENS	Pr. ENST Bretagne
	Achour MOSTEFAOUI	Pr. Université de Nantes
Examinateurs :	Ladjel BELLATRECHE	Pr. LIAS/ISAE ENSMA Poitiers
	Sophie CHABRIDON	Mcf. Télécom Sud Paris
Directeurs de thèse :	Abdessamad IMINE	Mcf. Université de Lorraine
	Michaël RUSINOWITCH	Dr. Inria Nancy Grand Est

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

Mis en page avec la classe thloria.

Acknowledgments

I would like to express my gratitude to my supervisor Abdessamad Imine, whose expertise and patience, added considerably to my graduate experience. I really appreciate his knowledge and skills especially his assistance in writing my thesis report. This thesis could not have been written without Dr. Abdessamad Imine who not only served as my supervisor but also encouraged and challenged me throughout my thesis.

A very special thanks goes out to my thesis director, Michaël Rusinowitch, without whose motivation and encouragement I would not have finished my graduate career in computer science research. It was though his understanding and kindness that I succeed to completed this work. He and Dr. Abdessamad Imine, guided me through the dissertation process, never accepting less than my best efforts. I thank them all.

I would like to thank Prof. Achour Moustefaoui and Prof. Frédéric Cuppens for taking effort and time out from their busy schedule to serve as my external readers. Finally, I also would like to thank the other members of my committee for accepting reading and monitoring my work: Prof. Kamel Smaili, Prof. Ladjel Bellatreche and Dr. Sophie Chabridon.

I would also like to thank the members of Cassis research team, past and present, for their help and friendship. I would also like to thank the members of Score research team. Especial thanks go to Dr. Claudia Ignat for her advices and relevant remarks. I would like to thank Dr. Stéphane Weiss who responded to my questions when needed.

I would like to acknowledge and extend my heartfelt gratitude to all computer science department of Nancy 2, faculty members and staff especially Prof. Kamel Smaili and Prof. Odile Thiery for their advices and support.

Many thanks to my friends in Loria, particularly, Hanen Maghrebi and Ben Slimane Jamila for our exchanges of knowledge and skills, which really helped enrich the experience. I would also like to thank Charlène Khun, Asma Kerkeni and Asma Ben Abacha for their support and help.

Finally, I would like to acknowledge and extend my heartfelt gratitude to all my family, most especially my mother for the support she provided me through my entire life, words alone cannot express what I owe her. I must also express my deepest thanks to my husband, Aymen, without whose love, encouragement and support, I would not have finished this thesis.

To my mother, my husband and my children.

Contents

Chapter 1

General Introduction

Contents

1.1 Motivations

The project summarized in this thesis aims at developing techniques to support access control in distributed collaborative editors. The ever-increasing role of Computer Supported Cooperative Work (CSCW) in academic, industry and society comforts the expansion of data sharing and raises growing concerns about the security of these shared data. The last technological advances have allowed for a distributed storage of the shared data which open up opportunities to communicate and cooperate on a given task while being geographically dispersed. With more and more data shared between many users, controlling access to this data has emerged as one of the main challenges of computer security. Consequently, access control in collaborative applications is increasingly attracting the attention of researcher intrigued by the multidisciplinary aspect and reach of such applications. Indeed, CSCW can be deployed on a wide range of applications such as collaborative editing, data sharing, video conferencing, workflow management, and so on. One major issue when considering security in collaborative applications is the management of a dynamic environment where the security policy may evolve over time, and of course, the shared document may be frequently updated. Moreover, an access control model has to satisfy the collaborative applications requirements namely, the distribution and the high local responsiveness. However, maintaining security policy while meeting these requirements is really a hard task since, policy view may be different from one site to another which may lead to security holes, not to mention convergence issues. Indeed, if security policy can be temporarily inconsistent, any given action may be authorized at one site and yet denied at another. This is troublesome since it may lead to permanent divergent state of the shared document.

Collaborative Editors. Collaborative editing systems are famous applications allowing people distributed in time and space to work together on shared documents in order to perform a given task. These systems are more and more used since they have many benefits such as shorting the production time of the final document, improving the final result by reducing errors, getting different viewpoints and skills [67, 103, 113]. Collaborative editors are especially used by committees producing reports and scientists collaborating on a research project [67]. The most famous collaborative editor is Google Docs which enables many users in different locations to collaborate simultaneously on the same document.

Access Control. Access control specifies mechanisms to enforce a security policy that regulates the actions a user can perform. For instance, the *read*, *write* and *execute* permissions associated with Unix files represent a typical example of access rights allowing users and programs to securely share resources of the same system. Most modern operating systems define sets of permissions that are variations or extensions of these three basic types of access. It is worth to mention that privacy and security issues are the foremost arguments for controlling access to data. Access control especially aims at preventing the dissemination of sensitive information which may threat the security of both individuals and companies.

Many researches were conducted towards formalizing access control since year 1971, shortly after the commercialization of time sharing systems. The access matrix model [38, 42, 57], represents the policy for user authentication, and has several implementations such as access control lists (ACL) and capability lists (CL). It is used to describe which users have access to what objects. This model provides the basic framework to describe protection systems [89]. However, it does not specify the exact nature of the subjects, objects or access rights [89]. Another important model for access control is the Role Based Access Control (RBAC) [84, 112] model where a role-based policy is a policy that regulates access control of users to resources or objects in the system according to the organizational activities and responsibilities of each user in the system. RBAC is appropriate to business activities since it naturally maps to an organization's structure [80].

Few researches have addressed access control issues in collaborative applications. A collaborative environment has to manage the frequent changing of access rights by users. Unfortunately, Access Control Lists (ACL) and Capability Lists (CL) do not suit very well dynamic change of permissions [107]. As a matter of fact, the administrator of collaborative environments often sets stricter permissions, as multiple users with different levels of privileges will try to access shared resources [89].

Approaches based on RBAC [84] overcome some problems with dynamic change of rights thanks to the session notion [52]. However, the "session" concept also prevents a dynamic reassignment of roles since roles cannot be changed within a single session. To solve this problem, Spatial Access Control (SAC) has been proposed [17] where instead of splitting users into groups as in RBAC, SAC divides the collaborative environment into abstract spaces. However, this needs prior knowledge of the practice used in the collaborative system. Moreover, this solution requires locking data-structures since every access needs to check the underlying access data-structures which reduces collaborative work performance.

Other works addressed security in peer-to-peer (P2P) environments ($e.g.$ [11]). Some of them rely on eXtensible Access Control Markup Language (XACML) policies [24, 25] to specify the access restrictions on the data offered by the peers for hierarchical P2P networks. However, none of these solutions considers existing access control components and collaborative applications residing on the peers [93].

In database area, some works propose a replicated access control [13, 79, 116]. For maintaining authorization consistency, these works generally rely on concurrency control techniques that are suitable for database systems such as explicit locking or transaction processing which are inappropriate for Real Time Collaborative Editors (RCE) [32]. For instance, [79] proposes an access control model for replicated databases. However, authorizations are timestamped and hence the solution does not scale.

Bouganim et al. [16] proposed a client-based evaluator of access control rules for regulating access to XML documents based on the access control model of Samarati [26] and [15]. However, this evaluator only concerns passive users that only have the right to read shared documents and can not update them. The approach of Wobber et al. [115] dealt with security issues in weakly consistent state-based replicated systems. The proposed authorization policy allows only for positive rights, *i.e* no explicit access denial is supported. Although delegation is allowed, the revocation is limited in order to avoid ambiguity. Furthermore, policy enforcement relies on a trust anchor for all collaborating items. The crash of this single root would create security problems and blocks the collaboration unless it delegates all his capabilities to another user.

1.2 General Context

A collaborative system provides a set of shared textual or graphical objects that may be accessed by users at any time. According to the communication type offered by a CSCW system, two kinds of systems are known:

- Synchronous systems which allow users to cooperate in a real time fashion. In fact, updates of the shared objects are carried out and broadcast immediately to other users. Collaborative editors represent a typical example of these systems since they provide interactivity between users [32,78, 94,102]. Indeed, each user edits his local document then sends his modifications to other users in order to see immediately the effect of his updates on their copies of the shared document.
- Asynchronous systems which enable users to collaborate at different times [9]. Thus, users contributions may be observed with a delay. Versions management tools such as CVS [18] and file synchronizers such as Unison [72] support asynchronous communications. For instance, a file synchronizer allows users to edit a shared file at different times then merge their changes later in order to get the same final view of the shared file.

In our work, we focus on real-time collaborative editors which are distributed systems based on the interaction of several users trying to edit simultaneously shared documents, such as articles, wiki pages and program source code. To improve availability of data, each user has a local copy of the shared documents. In general, the collaboration is performed as follows: each user's updates are locally executed in a non blocking manner and then are propagated to the other sites in order to be executed on remote copies. Although being distributed applications, RCE are specific in the sense that they must consider human factors. So, they are characterized by the following requirements [48]:

- *High local responsiveness*: the system has to be as responsive as a single-user editor [32,99,100];
- *High concurrency*: the users must be able to concurrently and freely modify any part of the shared document at any time [32,99];
- *Consistency*: the users must eventually be able to see a converged view of all copies [32,99];
- *Decentralized coordination*: all concurrent updates must be synchronized in a decentralized fashion in order to avoid a single point of failure;
- *Scalability*: a group must be dynamic in the sense that users may join or leave the group at any time [48].

It is very difficult to meet these requirements when deploying RCE in networks with high communication latencies (*e.g.* Internet). Due to replication and arbitrary exchange of updates, consistency maintenance in a scalable and decentralized manner is a challenging problem. Traditional concurrency control techniques, such as (pessimistic/optimistic) locking and serialization, turned out to be ineffective because they may ensure consistency at the expense of responsiveness and loss of updates [32,58,99].

To maintain consistency, while maintaining concurrency, an *Operational Transformation* (OT) approach has been proposed in [32,98]. Another approach was proposed in [3], it is based on Commutative

Replicated Data Type (CRDT). For such data type, convergence is obtained for free [3]. The main issue here is how to uniquely identify the shared document's elements in order to enforce a natural commutativity between concurrent updates.

In our thesis, we focus on OT-based RCE since OT is application-independent and used in many collaborative editors including Joint Emacs [78] (a groupware based on Emacs text editor), CoWord [100] and CoPowerPoint [100] and a file synchronizer [66] distributed with the LibreSource Community[1] and Google Wave [2]. OT also has been proposed as a consistency model for replicated mobile computing [40].

To deal with concurrent operations, OT uses an algorithm, called *inclusive transformation* [98] and denoted by function IT, to merge these operations regardless reception order. Let op_1 and op_2 be two concurrent operations. Intuitively, $IT(op_1, op_2)$ transforms op_1 against op_2 in order to *include the effect* of op_2 in op_1. The transformed form of op_1 is then executed after op_2. For instance, consider the scenario presented in Figure 1.1 where two users work on a shared document. Just to simplify, we consider that the document is represented by a sequence of characters. These characters are addressed from 1 to the end of the document. Initially, both copies hold the string "efecte". User 1 executes operation $op_1 = Ins(2, f, 1)$ to insert the character 'f' at position 2. Concurrently, user 2 performs $op_2 = Del(6, 2)$ to delete the character 'e' at position 6. At site 1, op_2 needs to be transformed in order to include the effect of op_1: $op_2' = IT((Del(6, 2), Ins(2, f, 1)) = Del(7, 2)$. The deletion position of op_2 is incremented because op_1 has inserted a character at position 1, which is before the the position of the character deleted by op_2.

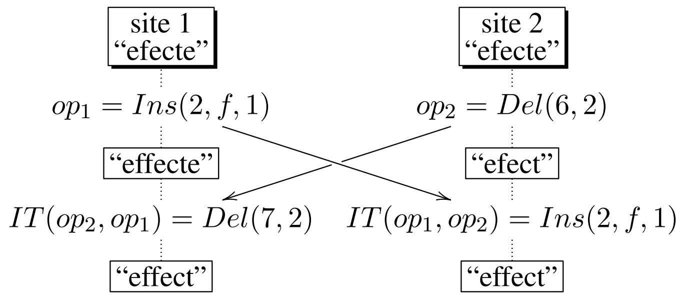


Figure 1.1: Serialization of concurrent update with OT approach

1.3 Access Control Issues and Requirements in Distributed Collaborative Editors

Access Control represents an essential part of any information sharing system. In particular, coordinating access to shared information is currently a hot topic in CSCW. As more and more people are working together in a collaborative manner in order to edit the same shared document, across wide areas, it becomes increasingly important to provide collaborative editing applications with flexible yet simple access mechanisms to keep shared information secure. Indeed, collaborative applications provide information and resources characterized by different degrees of sensitivity such as customer data in a financial application or patient data in a healthcare application.

However, it is really hard to balance the competing goals of collaboration and security. Indeed interaction in collaborative applications is aimed at making shared documents available to all who need

[1]http://dev.libresource.org
[2]http://www.waveprotocol.org/whitepapers/operational-transform

it (by replicating the shared document), whereas access control seeks to ensure this availability only to users with proper authorization.

The major latency problem in access control for collaborative editors is due to the use of a shared data-structure containing access rights that is stored on a central server as shown in Figure 1.2 where we consider N users collaborating to edit the same document using a collaborative editing system that satisfies the requirements mentioned in Section 1.2. We assume that each user holds a replica of the shared document and has the ability to alter the state of his replica with respect to an access control policy. In Figure 1.2, the security policy is enforced at site i, so controlling access consists in locking the security data-structure stored at site i and verifying whether this access is valid. Subsequently, high responsiveness is lost because every update must be granted by some authorization coming from the central server, namely user i.

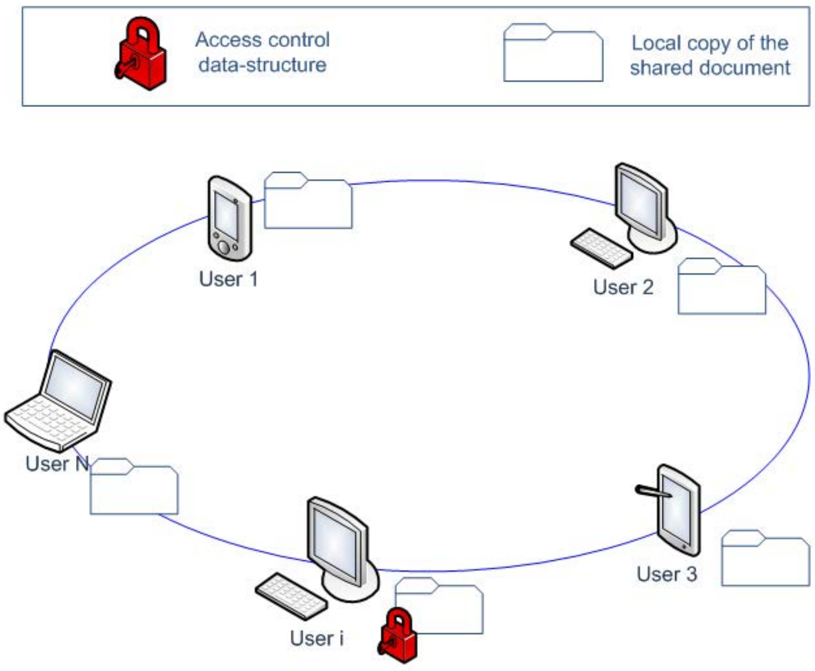

Access control data-structure
Local copy of the shared document
User 1
User 2
User N
User 3
User i

Figure 1.2: Access Control Problematic.

To overcome the latency problem, we propose an access control model based on replicating the access data-structure on every site. Thus, a user will own two copies: the shared document and the access data-structure (see Figure 1.3). It is clear that this replication enables users to gain performance since when they want to manipulate (read or update) the shared document, this manipulation will be granted or denied by controlling only the local copy of the access data-structure.

However, moving access control from the central server to every collaborating site leads to several issues. Indeed, unlike traditional single-user models, collaborative applications have to allow for dynamic change of access rights, as some users can join and leave the group in an ad-hoc manner. Likewise, there are many situations where access control rules are user specific and difficult to predict [16].

Combining dynamic access control with consistent replication is a challenging task if the resulting system is to support consistency of the shared document. Indeed, the shared document's updates and the access data-structure's updates are applied in different orders at different user sites. The absence of safe coordination between these different updates may cause security holes (*i.e.* permitting illegal updates or rejecting legal updates on the shared document). Without a careful design, permanently divergent document's state may be produced.

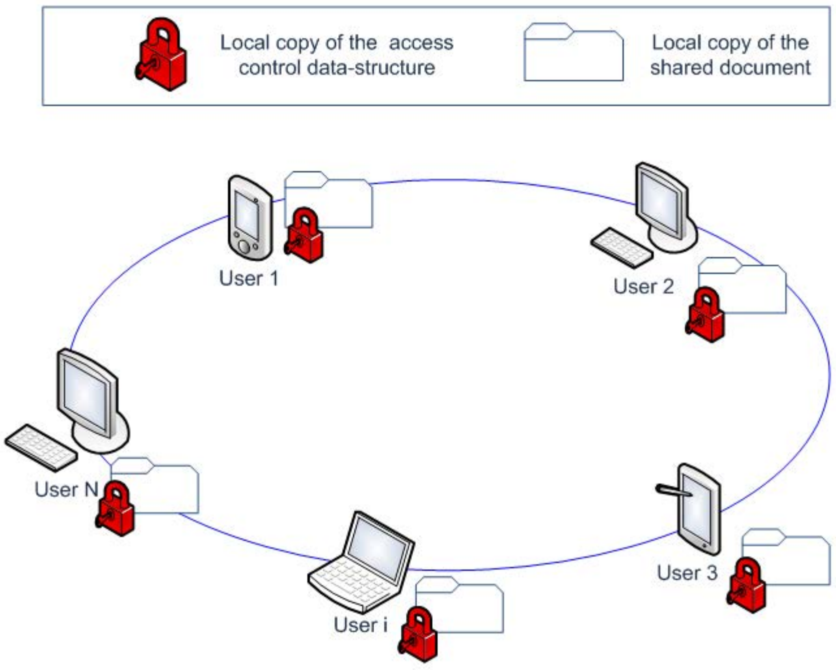

Local copy of the access control data-structure
Local copy of the shared document
User 1
User 2
User N
User 3
User i

Figure 1.3: Replicating Access Control Policy.

Consequently, providing an access control model for collaborative editing solutions is a hard task since it must meet the following requirements:

- Unlike traditional systems where access control has been explored, access decisions in a collaborative application have to consider dynamic policy changes that are concurrent to document updates.
- All requirements of RCE (as previously mentioned) should be preserved. For instance, a security layer atop RCE must incur significantly less overhead to maintain the high local responsiveness.
- The security layer has to be generic and appropriate for existing collaborative editors regardless the underlying coordination algorithm.
- A significant requirement of the security architecture for emerging applications is that security policy is enforced in all sites.

First Goal: The main objective of the thesis is to propose a generic model for access control in real-time distributed collaborative editors. To deal with dynamic policies and preserve all RCE's requirements, we propose an optimistic approach that tolerates momentary violation of access rights but then ensures the copies to be restored in valid states with respect to the stabilized access control policy.

This thesis is part of a larger effort from the community to develop an optimistic access control for real time collaborative editors relying on replicating the access control policy. To the best of our knowledge, our work is the first that addresses the frequent and dynamic updates of both replicated access control policy and shared documents.

1.4 Undoability in Distributed Collaborative Editors

Undoing operations is an indispensable feature in interactive applications [68,90]. It is a standard feature in most single-user interactive applications and is becoming more and more available in collaborative applications [5,12,22,28,74,77,78,96,97,101]. In particular, it represents a very useful mechanism for collaborative editors since it allows for error recovery by providing the ability to restore a correct state of the shared data after erroneous operations.

Providing an undo feature for collaborative editing applications decreases the gap between single-user and multi-user collaborative editors since the former allow users to undo operations.

A history has to be maintained at every collaborating site for supporting undo. Then a user can select the operation to be undone from the history buffer. Selective undo allows users to undo the effect of an isolated action selected from all past actions [12] by reordering operations in the history. Hence, undo is generally combined with OT approach as it allows to rearrange operations [97,99]. However supporting undo in collaborative applications is a challenging problem because of the interleaving between updates performed by multiple users in a collaborative computing environment [22,74,97].

To better illustrate the complexity of undo, consider the scenario of Figure 1.4 where we suppose that two users collaborate to edit the same sequence of characters initially equal to "undo last operation". User 1 alters the document's state by executing the operation $op_1 = Del(6, "last")$ which deletes the word last at the position 6. Concurrently, user 2, inserts the word "any" to get the state "undo anylast operation". The OT principle consists in integrating the effect of concurrent operations at remote sites. Thus, operation op_1 is transformed to $op'_1 = Del(9, "last")$ at site 2 as for op_2, it is transformed to $op'_2 = op_2$ at site 1. Clearly, both sites 1 and 2 converge to the state "undo any operation".

To simplify, suppose that undo request is generated to undo the last operation. Unfortunately, last operation is not the same at all sites, since operations are not always stored in execution order, then considering the undo request as an error recovery for the last operation would produce divergence. Obviously, undoing the last operation at site 1 undoes op'_2 while it undoes op'_1 at site 2 (see Figure 1.4). Additional issues are encountered when the undo request interleaves with other document updates.

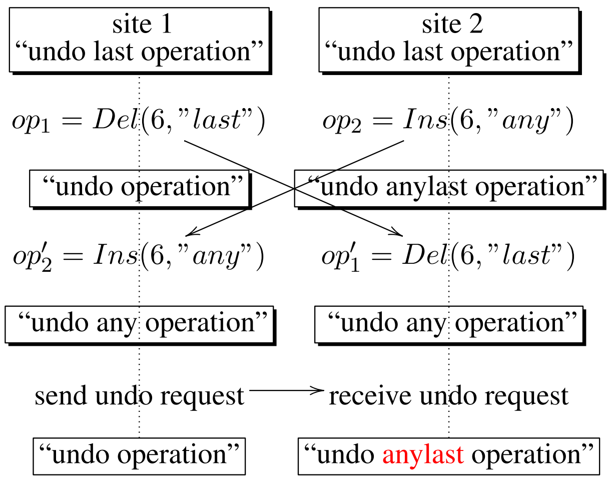


Figure 1.4: Undo last operation produces divergence.

Accordingly, the correct way to manage undo in collaborative applications is to select the operation to be undone. Hence, instead of undoing the last operation as illustrated in Figure 1.4, an $undo(op)$ operation is executed and broadcast to other sites (see Figure 1.5) where op_2 is undone at site 2 but its new form op'_2 is undone at site 1. The challenging task in selective undo, is to maintain convergence by producing the same final state after undoing two different forms of the same operation.

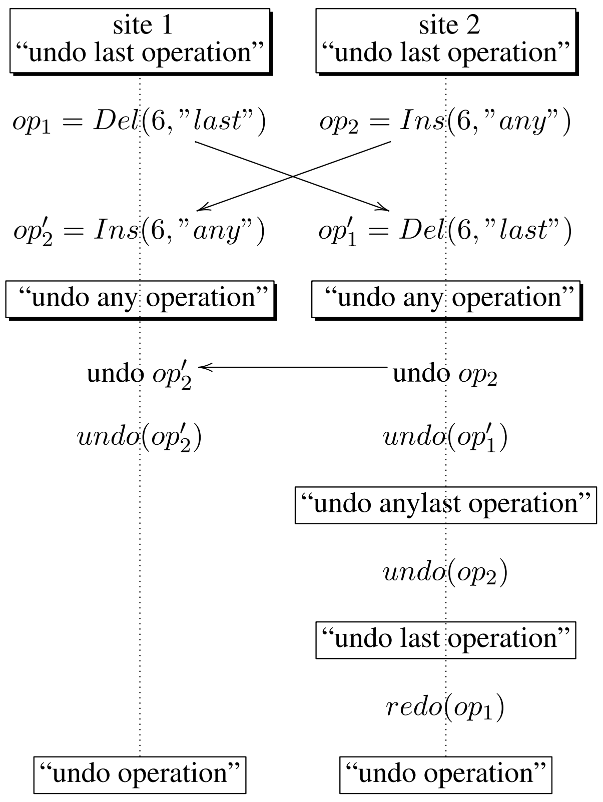


Figure 1.5: Undo/Redo approach.

An intuitive solution would be to undo all the operations in the inverse chronological order, $i.e$ from the last to the wanted operation as it is proposed in [77]. So, considering the same example presented in Figure 1.4, undoing op_2 at site 2 would require to undo op'_1, op_2 then, redo op_1. As shown in Figure 1.5, both sites converge to the same final state "undo last operation".

However, the undo/redo scheme is expensive since it requires to perform many steps to achieve undo. Moreover, it does not allow undo in all cases. In fact, an operation may not be undoable if another later operation performed by the same user has not been undone.

Another solution consists in generating the inverse of the operation to be undone then transform it against the following operations in the history so as to take into account the effect of all operations that follows in the log using OT approach. However, combining OT and undo approaches while ensuring data convergence is difficult as there are many properties to be preserved for both OT and undo. To be correct, OT functions must fulfil two transformation properties namely, $\mathbf{TP1}$ and $\mathbf{TP2}$ [74,78]. Also, an undo command has to fulfil some inverse properties $\mathbf{IP1}$, $\mathbf{IP2}$ and $\mathbf{IP3}$ [36, 74, 97, 101]. Consequently, undoing operations may itself lead to divergence cases called undo *puzzles* [97] due to the violation of one of these properties.

Significant researches have been made to address OT-based selective undo [74, 77, 97, 101]. Even though various undo solutions have been proposed over the recent years, either they do not allow to undo any operation at any time *e.g* [77] or have exponential complexity such as AnyUndo and COT algorithms [97, 101] which is not of relevance in distributed collaborative applications since they require high responsiveness.

Furthermore, verifying correctness of an existing undo solution is error-prone due to the absence of formal guidelines for undo. Providing such guidelines is a hard task, since there are many constraints to be considered. That is why some approaches resort to avoid some constraints at the expense of performance degradation [101].

Accordingly, undo in collaborative applications remains an open and challenging issue. Providing an undo solution for collaborative applications has to take into account three main issues:

- Formalizing the correctness criteria of an undo solution.
- Designing an undo algorithm and prove its correctness.
- Ensuring efficiency of the undo algorithm.

Second Goal: The second objective of the thesis is to provide a theoretical study of the undoability problem in the context of collaborative applications. This study enables us to design a generic and safe undo solution appropriate for collaborative editing applications.

1.5 Contributions

The main contributions of this thesis are as follows:

Access Control. We survey existing access control solutions and present the access control requirements in the context of real-time collaborative editors. This study would be useful for future investigation in this area. Based on the previous study, we discuss the main security and convergence issues arising when adding an access control layer to a collaborative writing framework. The main contribution of this dissertation is an optimistic and decentralized access control model that overcomes these issues [21,49]. Furthermore, our access control model is generic and enables programmers to separate the access control layer from the coordination layer, thereby greatly simplifying algorithmic development of the access control on the top of a given coordination framework and enabling to reuse the access control for different collaborative applications.

We also propose a garbage collection mechanism in order to improve the performance of our solution [64] since our access control model is based on logging access permissions. Consequently, our model can be easily deployed on mobile devices.

The design goals of any access control model in RCE are:

- Simplifying the editing and management of access control permissions;
- Separating access control data structures and coordination algorithms;
- Being easily extensible and reusable.

Our access control model achieves these goals through its careful separation between access control and coordination layers. All algorithms were implemented on the top of the collaboration framework OPTIC [47,48] to demonstrate the feasibility of our model. Additionally, we provide performance measurements of the above algorithms to highlight the efficiency of our solution. To the best of our knowledge, these measurements are the first demonstration of access control implementation for decentralized collaborative editors.

Undoability for Collaborative Applications. A significant part of this work was dedicated to the undoability problem in distributed collaborative applications since undo represents a main feature of our optimistic access control model.

On the one hand, we addressed the principles of undoability in collaborative applications from a theoretical point of view. We show that designing an undoable object is a combinatorial problem and propose a novel approach to analyse this problem based on constraint programming. As a main result, we define a necessary and sufficient condition for undoing replicated objects based on operational transformation approach with respect to inverse and transformation properties..We showed that it is impossible to achieve correct undoability unless the updates performed on the shared object commute. Although it is an impossibility result, it is interesting for OT-based object developers as well as researchers working on undoability for collaborative applications.

On the other hand, we propose an enhanced set of operations in order to achieve undoability in a reasonable time. Our solution has a linear complexity and thus is appropriate for collaborative applications. Moreover, it could be integrated in any existing collaborative editing algorithm.

Publications. The results of this thesis have been partially published in fourth conference papers:

- Our access control model was first introduced in [49][3]. This paper also discusses the access control requirements for RCE and proposes an optimistic approach to regulate access in a collaborative application. However, the paper only considers a mandatory access control policy.
- An extension of the previous paper was presented in [21][4]. This paper proposes a multi-administrator access control and investigates the generic aspect of the access control layer to meet a large variety of collaborative editing solutions.
- Since our model was implemented on the top of the OPTIC framework based on log usage [47,48], we investigate a garbage collection scheme in [64][5]. This paper presents the issues raised by the garbage collection in a collaborative application and our solutions to address these issues as well as an implementation of the solution on mobile devices.
- Our first contribution to the undoability problem for collaborative editors appeared in [20][6]. This papers proposes a new semantic for idle operations in order to achieve undoability.

1.6 Thesis organization

The remainder of this thesis is structured as follows.

In **Chapter 2**, we present the state of the art of collaborative editing as well as their requirements . We also give an overview on existing access control models and compare them.

In **Chapter 3**, we propose our generic and optimistic access control model for distributed collaborative editors. In this chapter, we stress the security and convergence issues that motivated our conceptual choices.

In **Chapter 4**, we detail our algorithms and prove the correctness of our access control model. Experimental study is given in **Chapter 7** to demonstrate the feasibility of our access control model and show its good performance and scalability.

The second part of this dissertation is dedicated to the undoability problem in collaborative applications as a main feature of our access control model and is organized as follows:

[3]Abdessamad Imine, Asma Cherif and Michaël Rusinowitch : A Flexible Access Control Model for Distributed Collaborative Editors. SDM 2009: 89-106.

[4]Asma Cherif, Abdessamad Imine and Michaël Rusinowitch: Optimistic Access Control for Collaborative Editing Systems. ACM Symposium on Applied Computing SAC 2011.

[5]Moulay Driss Mechaoui, Asma Cherif, Abdessamad Imine and Fatima Bendella: Log Garbage Collector-based Real Time Collaborative Editor for Mobile Devices. CollaborateCom 2010.

[6]Asma Cherif, Abdessamad Imine : Undo-Based Access Control for Distributed Collaborative Editors. CDVE 2009: 101-108.

In **Chapter 5**, we present the undoability challenges as well as existing solutions and their limits. Then, a theoretical study of the undoability problem is provided in **Chapter 6** in order to well understand the undoability problem and provide guidelines of the undo command. In this chapter, we formalised the undoability problem as a Constraint Satisfaction Problem [108] and a provide a necessary and sufficient condition to achieve undoability. Moreover, we presented our solution to overcome this impossibility result. It consists in a generic framework for undoing operations in OT-based collaborative frameworks.

Finally, **Chapter 8** discuss our achievements and provide directions that could be conducted in the future.

To more illustrate the organization of our thesis we present different chapters in Figure 1.6.

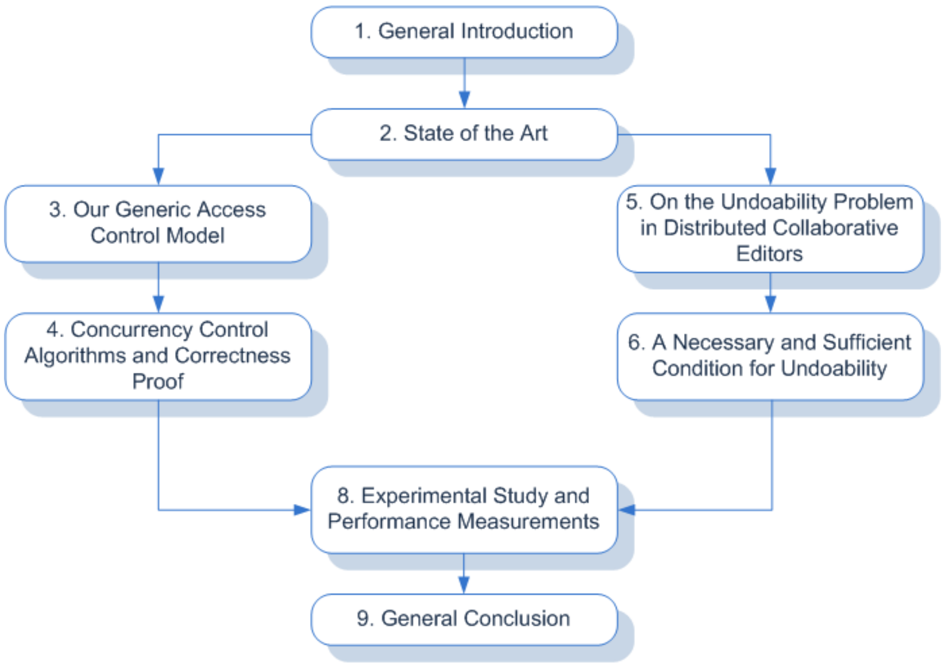

1. General Introduction
2. State of the Art
3. Our Generic Access Control Model
4. Concurrency Control Algorithms and Correctness Proof
5. On the Undoability Problem in Distributed Collaborative Editors
6. A Necessary and Sufficient Condition for Undoability
8. Experimental Study and Performance Measurements
9. General Conclusion

Figure 1.6: Thesis Organizatin.

Chapter 2

State of the Art

Contents

Collaborative editors are very important applications since they allow many dispersed users to collaboratively edit the same shared document. However editing the same document collaboratively is a complex [46] and dynamic group process in which many considerations and issues must be addressed [61]. Such applications necessitate a careful design and a good attention since they allow for a human-computer-human interaction. Adding an access control layer on the top of any collaborative editing solution is a hard task. In fact, the access control must respect the requirements of such applications without additional overhead in order to allow for the high responsiveness required by collaborative editors. Moreover, consistency of the shared document must be preserved. Accordingly, security measurements must not lead to divergence cases and provide the same view of the shared document for all users. Furthermore, the security model must be designed in a decentralized fashion.

In this chapter we begin by giving an overview on collaborative editors in Section 2.1 in order to illustrate the requirements of such systems. Then we discuss the access control solutions in Section 2.2. We present the most popular solutions and discuss their limitations in order to well define an access control layer for collaborative editors that fulfil the requirements discussed below.

2.1 Collaborative Editors

Most usually, collaborative editing is applied to textual documents such as wikis or programmatic source code such as version control. Generally, these applications allow for recording changes in order to see who contributed what changes. The most famous collaborative editing framework is Google Docs.

It provides collaborative both synchronous and asynchronous editing functionalities based on revision control. Also, Wikipedia represents a large scale collaborative editing project.

The advantages of collaborative writing are:

- provide projects that are richer and more complex than those produced by individuals which is very useful for academic as well as commercial communities.
- improve learning experiences and the process of reviewing documents.

There are two kinds of collaborative editors (see Figure 2.1): (i) real-time collaborative editors (RCE) are a class of distributed systems based on the interaction of several users trying to edit simultaneously shared documents (ii) and non-real-time collaborative editors which do not allow editing of the same document at the same time, thus being similar to revision control systems.

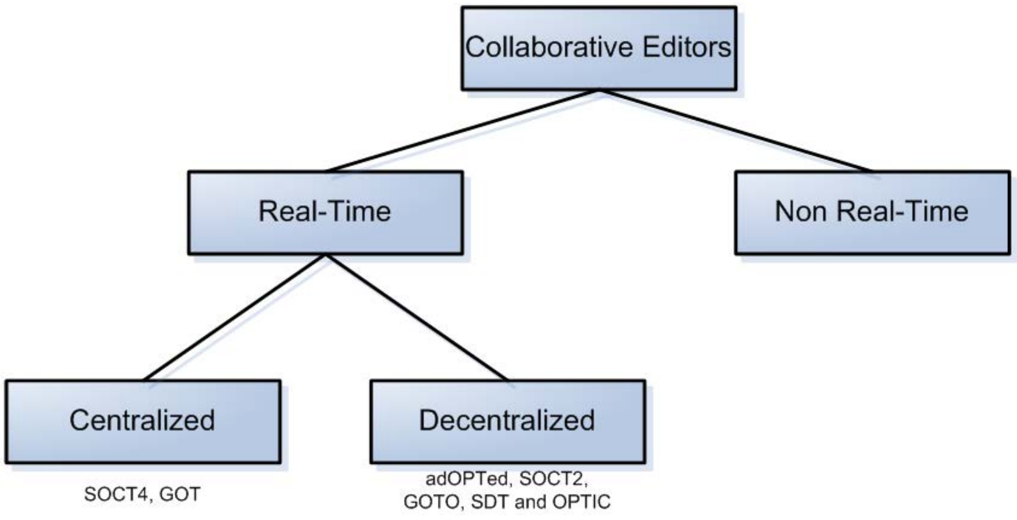


Figure 2.1: Different kinds of collaborative editors

Real-time collaborative editors are based on two kinds of collaborative editing algorithms: centralized and decentralized algorithms.

Centralized algorithms. Editors based on these algorithms require the presence of a central server to coordinate between updates performed on the shared document. For instance, Google Docs, SOCT4 algorithm [109], GOT [99] as well as COT [101] rely on client-server architecture in order to get a global and unique order of execution, hence do not scale well.

Decentralized algorithms. The second approach allows concurrent requests to be executed in any order, but does not allow necessarily for an arbitrary number of users. For example in adOPTed [78], SOCT2 [95], GOTO [98], and SDT [58], the use of state vectors is necessary to detect causality relationship between different updates. Consequently, these solutions do not scale well and it would be difficult to adapt them for a P2P context. To the best of our knowledge, the first coordination algorithm allowing for an arbitrary number of users is OPTIC [47, 48] thanks to the use of semantic dependency in order to coordinate concurrent updates.

In our thesis, we focus on real time collaborative editors regardless the algorithm used to coordinate different updates. However, we aim at providing a decentralised access control model in order to reach the high responsiveness as well as scalability requirements. In the following, we are interested in these requirements.

2.1.1 Real Time Collaborative Editors

Real-time Collaborative Editors (RCE) are special class of distributed applications that allow users to collaboratively edit a shared document. These applications allow for a human-computer-human interaction which necessitates a careful design.

Although being distributed applications, RCE are specific in the sense that they must consider human factors [48]. So, they are characterized by the following requirements:

1. *High local responsiveness*: the system has to be as responsive as its single-user editors [32,99,100]. The response to the local user's actions must be quick, even though collaborating users may reside on different machines connected via the Internet with a long and non-deterministic communication latency. High responsiveness is very important because it presents a main indicator of the good quality of a collaborative system. Moreover, poor responsiveness decreases the system's effectiveness in supporting collaborative work.

2. *High concurrency*: the users must be able to concurrently and freely modify any part of the shared document at any time [32, 99] in order to facilitate the flow of information among collaborating users.

3. *Consistency*: the users must eventually be able to see a converged view of all copies [32, 99]. Maintaining consistency is the major challenge since it is hard to manage the multiple streams of concurrent activities performed by multiple users which may lead to conflicting scenarios [48].

4. *Decentralized coordination*: all concurrent updates must be synchronized in a decentralized fashion in order to avoid a single point of failure.

5. *Scalability*: a group must be dynamic in the sense that users may join or leave the group at any time [48].

It is very difficult to meet these requirements when deploying RCE in networks with high communication latencies (*e.g.* Internet). Due to replication and arbitrary exchange of updates, consistency maintenance in a scalable and decentralized manner is a challenging problem. Traditional concurrency control techniques, such as (pessimistic/optimistic) locking and serialization, turned out to be ineffective because they may ensure consistency at the expense of responsiveness and loss of updates [32,48,58,99].

Since collaborative editors allow multiple users to concurrently edit the same shared document, divergence cases may be encountered. Consistency maintenance in the face of concurrent accesses and updates to shared document is one of the main issues in the design of collaborative editing systems. Operational transformation (OT) is a technique originally invented to ensure consistency and avoid divergence occurring when many users edit concurrently the same document.

2.1.2 Overview on the Operational Transformation (OT) Approach

OT is an optimistic replication technique originally invented to allow many users (or sites) to concurrently modify the shared data and next to synchronize their divergent replicas in order to obtain the same data.

This approach is considered as the efficient and safe method for consistency maintenance in the literature of collaborative editors. Indeed, it is aimed at ensuring copies convergence even though the users's updates are executed in any order on different copies.

After two decades of research, this technique has extended its capabilities and expanded its applications to include group undo, locking, conflict resolution, operation notification and compression, group-awareness, HTML/XML and tree-structured document editing, collaborative office productivity tools, application-sharing, and collaborative computer-aided media design tools. Recently, OT has been adopted as a core technique behind the collaboration features in Google Wave and Google Docs, which are taking OT to a new range of web-based applications[7].

OT approach requires that every site stores all executed operations in a buffer also called a *log*. It is known that collaborative editors manipulate shared objects that own a linear data-structure [32, 98, 100] (*e.g.* a list). This list is a sequence of elements from some data type, such as a character, a paragraph, a page, an XML node, etc. In [100], it has been shown that this linear structure can be easily extended to a range of multimedia documents, such as MicroSoft Word and PowerPoint documents [48].

Two primitive operations are generally used to modify the shared document:

- $Ins(p, e, s)$ to insert the element e at position p;
- $Del(p, s)$ to delete the element at position p.

The parameter s is the identity of the site issuing the operation. Every site has a unique identity. The set of site identities is assumed totally ordered.

To deal with concurrent operations, OT uses an algorithm, called *inclusive transformation* [98] and denoted by function IT, to merge these operations regardless reception order. Let op_1 and op_2 be two concurrent operations. Intuitively, $IT(op_1, op_2)$ transforms op_1 against op_2 in order to *include the effect* of op_2 in op_1. The transformed form of op_1 is then executed after op_2. For instance, here are two transformation cases given in the IT algorithm proposed by Ressel et *al.* [78]:

```
1: $IT(Ins(p_1, e_1, s_1), Ins(p_2, e_2, s_2))$:
2: if ($p_1 < p_2$ or ($p_1 = p_2$ and $s_1 < s_2$)) then
3:     return $Ins(p_1, e_1, s_1)$
4: else
5:     return $Ins(p_1 + 1, e_1, s_1)$
6: end if
7: $IT(Ins(p_1, e_1, s_1), Del(p_2, s_2))$:
8: if ($p_1 > p_2$) then
9:
10:    return $Ins(p_1 - 1, e_1, s_1)$
11: else
12:    return $Ins(p_1, e_1, s_1)$
13: end if
```

Algorithm 1: Inclusive transformation of two insertions [78].

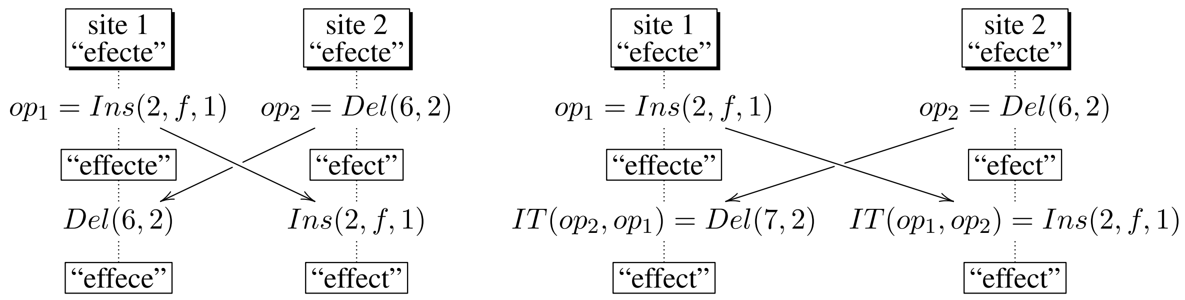
The site identities are used to tie-break conflict situations (*e.g.* two concurrent operations inserting elements at the same position).

To illustrate the importance of the OT approach, consider the scenario in Figure 2.2.(a) where two users work on a shared document represented by a sequence of characters. These characters are addressed from 1 to the end of the document. Initially, both copies hold the string “efecte”. User 1 executes operation $op_1 = Ins(2, f, 1)$ to insert the character ‘f’ at position 2. Concurrently, user 2 performs $op_2 = Del(6, 2)$ to delete the character ‘e’ at position 6. When op_1 is received and executed on site 2, it produces the expected string “effect”. But, at site 1, op_2 does not take into account that op_1 has been executed before it and it produces the string “effece”. The result at site 1 is different from the result of site 2 and it apparently violates the intention of op_2 since the last character ‘e’, which was intended to be deleted, is still present in the final string. It should be pointed out that even if a serialization protocol [32]

[7]http://en.wikipedia.org/wiki/Operational_transformation

was used to require that all sites execute op_1 and op_2 in the same order (*i.e.* a global order on concurrent operations) to obtain an identical result *effece*, this identical result is still inconsistent with the original intention of op_2.

In Figure 2.2.(b), we illustrate the effect of IT on the previous example. At site 1, op_2 needs to be transformed in order to include the effects of op_1: $op_2' = IT((Del(6, 2), Ins(2, f, 1)) = Del(7, 2)$. The deletion position of op_2 is incremented because op_1 has inserted a character at position 1, which is before the character deleted by op_2.



(a) Incorrect integration.

(b) Correct integration.

Figure 2.2: Serialization of concurrent updates

Many collaborative applications are based on the OT approach such as Joint Emacs [78] (a groupware based on text editor Emacs), CoWord [100] (a collaborative Microsoft word processor) and CoPowerPoint [100] (a real-time collaborative multimedia slides creation and presentation system) and a file synchronizer [66] distributed with the industrial collaborative development environment LibreSource Community[8]. OT also has been proposed as a consistency model for replicated mobile computing [40].

2.1.2.1 OT Properties

Using an IT algorithm requires to satisfy two properties $TP1$ and $TP2$ in order to ensure convergence [78].

Definition 2.1.1 (TP1).

For all op, op_1 and op_2 pairwise concurrent operations with $op_1' = IT(op_1, op_2)$ and $op_2' = IT(op_2, op_1)$, it must be that:

$$[op_1 \,;op_2'] \,\equiv\, [op_2 \,;op_1'],$$

i.e. sequences $[op_1 \,;op_2']$ and $[op_2 \,;op_1']$ are equivalent and have the same effect on the document.

Property $TP1$ defines a *state identity* and ensures that if op_1 and op_2 are concurrent, the effect of executing op_1 before op_2 is the same as executing op_2 before op_1. This property is necessary but not sufficient when the number of sites is greater than two.

[8]`http://dev.libresource.org`

Definition 2.1.2 (TP2).

For all op, op_1 and op_2 pairwise concurrent operations with $op'_1 = IT(op_1, op_2)$ and $op'_2 = IT(op_2, op_1)$, it must be that:

$$IT(IT(op, op_1), op'_2) = IT(IT(op, op_2), op'_1)$$

Property $TP2$ defines an *operation identity* and ensures that transforming op along equivalent and different operation sequences will give the same operation.

Properties $TP1$ and $TP2$ are sufficient to ensure the convergence for *any number* of concurrent operations which can be executed in *arbitrary order* [62, 78]. Accordingly, by these properties, it is not necessary to enforce a global total order between concurrent operations because data divergence can always be repaired by operational transformation.

To better understand our work, all examples given in the following use characters as elements to be inserted/deleted.

2.1.2.2 Integration Algorithms

In OT-based coordination frameworks, every site is equipped with two main components: the integration algorithm and the transformation algorithm. The first algorithm is responsible for receiving, broadcasting and executing operations. It is independent of the collaborative objects semantics. Several integration algorithms were proposed in the literature of collaborative editors, such as dOPT [32], adOPTed [78], SOCT2,4 [95, 109], GOTO [98], OPTIC [48], and COT [101]. Each of these integration algorithms uses a set of transformation algorithms for serializing concurrent operations defined on the same state. In contrast to integration algorithms, transformation algorithms depends on the semantics of the shared objects.

Every site generates document updates, executes them on its local copy of the shared document, then stores them in a local history also called log. When a site receives a remote operation let op, the integration algorithm proceeds as follows:

- Find all operations in the local log that are concurrent to op.
- Integrate op against all concurrent operations using the transformation functions.
- Execute the integrated form of op at the current document state.
- Store the integrated form of op in the local log.

Thus, integration algorithms allow for building the local logs while preserving the causal relation between different document updates. Obviously, logs are different from one site to another but they are equivalent thanks to properties $\mathbf{TP1}$ and $\mathbf{TP2}$.

Given a coordination framework equipped with integration and transformation algorithms, our objective is to build on the top of this coordination layer, an access control model to ensure the security of the shared documents. In the following, we investigate the requirements of access control models in collaborative editors. Then, we survey existing models proposed for collaborative applications.

2.2 Access Control for Collaborative Editors

Protecting data and resources is one of the most important requirements of any information management system. Indeed, it ensures (i) *confidentiality*: protecting data from unauthorized disclosure and (ii) *integrity*: protecting data from unauthorized or improper modifications.

To enforce protection, every access to the data must be controlled. Moreover, only authorized accesses can be allowed. To develop an access control system, we need to define the regulation of the access to the data as well as the functions to be executed in order to implement these regulations in a computer machine. In general, the development process of access control systems relies on the concepts cited below [81]:

1. *Security policy* defines the rules of the access control, for instance saying that a site s has the right to modify the shared document is a rule.
2. *Security mechanism* defines the functions that will be executed on the computer machine. These functions illustrate the access control policy defined by the security policy and formalized by the security model.

It should be pointed out that defining a security policy model is not a trivial task [81]. It is a subtle and challenging task. Indeed, designing a security model relies on interpreting real world security policies which are often ambiguous and complex in a unambiguous and simple way in order to be executed on a computer machine [81]. Moreover, a security policy depends on the nature of the application in which it will be enforced and must take into consideration malicious behaviour.

Controlling access is more and more complex in collaborative applications since they have special requirements. In the following, we detail these requirements and show how they complicate the access control task.

2.2.1 Access Control Issues and Requirements in Collaborative Applications

There are many issues faced when trying to apply an access control model for a collaborative application. First of all, it is difficult to balance the competing goals of collaboration and security [107]. In fact, collaboration aims at allowing many users to access to shared resources by building useful connection among users, tools and information. However, security aims at ensuring confidentiality and integrity of shared information. Several requirements must be addressed in order to ensure security for such systems where the behavior of users is unpredictable and interactions between users and resources are unexpected. Consequently, protecting resources in a collaborative application needs to address many requirements not raised by normal single-user applications.

Moreover, collaborative systems contain resources with different degree of sensitivity. For instance, some resources could be top secret such as military documents or information of a patient in a healthcare application while other resources are just confidential such as academic emails. Thus, it is difficult to manage access control for different resources at the same time.

Furthermore, some policies require real time updates which means that the policy is changed while it is in effect. However updating the policy while it is deployed may inevitably lead to security holes. To further illustrate this issue, suppose that in a financial company, the user Bob has the right to update a given resource but concurrently the policy is updated to revoke this right. Since the policy update is not enforced immediately, Bob may perform updates while policy is changing which obviously lead to policy violation and may introduce financial loss to the company.

Another important issue that must be kept in mind is that policy enforcement should be managed in a distributed fashion. Meanwhile, inconsistencies as well as security breaches or unavailability of resources due to latencies introduced by the remote check of access rights must be avoided in order to respect the real time aspect of the collaboration while enforcing the security of the shared resources.

Finally, access control introduces additional processing time which must be very low in a collaborative application. High responsiveness must be taken into account to meet human factor requirements that become more important during collaborations since the user is responsible of the group progress in contrary of a single-user application.

Several research studies addressed the requirements for access control in collaborative systems [4, 30, 35, 53, 107]. In the following, we state the main requirements that must be taken into account before the design of an access control model for any collaborative application:

1. Distribution: access control must be enforced in a distributed system. Thus it must meet the requirements of such a system. Otherwise, the full potential of the distributed application will be limited.

2. Generic: the model for access control should meet a large variety of cooperative applications. It should be generic with the minimal changes from one application to another in order to be deployed on different applications easily.

3. Transparency and flexibility: the access control must be transparent for users having correct access rights. The management of access rules has to be as flexible as possible without constraining the application on which it is deployed.

4. High level specification: helps to concentrate on access layer independently from the collaboration complexity of the underlying coordination layer.

5. Dynamic: since collaborative applications are designed for dynamic groups, access control must take into account the dynamic aspect of the collaboration and allows dynamic change of the policy in order to meet the frequent changes of access rights.

6. Good performance: we stress the fact that access control must be kept with low overhead to meet the real-time aspect of communications.

7. Scalability: it should be possible for the access control model to be deployed in a large scale in order to support a large number of users and operations in a collaborative application as well as the churn (*i.e* users can join and leave the application in an ad-hoc manner).

In the following, we illustrate different access control classes that have been proposed in the literature also investigating their limitations with regards to the requirements mentioned before.

2.2.2 Classes of Access Control Policies

In this section, we present the main access control classes [81, 107]:

1. Discretionary (DAC): in this class the policy is based on authorizations to define access control rules stating what user is allowed to do what action.

2. Mandatory (MAC): Mandatory policies control access based on regulations defined by a central authority.

3. Role-based (RBAC): users are divided up into different roles and accesses are granted according to the rules defining what actions are allowed to a given role.

4. Task-based (TBAC): this model incorporates contextual information into access rules in order to better recognize the context in which security access arises.

5. Team-based (TMAC): this approach considers that the appropriate way to group users is based on the notion of team instead of roles since, work inside organizations is managed in teams.

6. Spatial: this model suites environments that can be represented in terms of regions. The access into regions is based on credentials.

7. Context-aware: this model is also an extension of the RBAC model in order to export its usage for context-aware applications.

2.2.2.1 Discretionary Access Control (DAC)

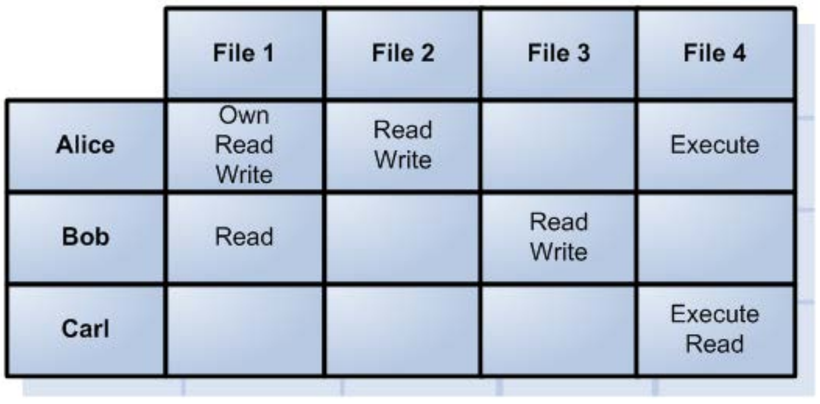
Discretionary policies define access control on the basis of access right requestor identity. Access rules are specified to assign access rights to requestors. Then, accesses are granted or denied based on these access rules. In this security model, users can have the right to delegate their access control capabilities where from the name discretionary.

DAC policy is characterized by its flexibility and is widely used in many sectors. The most famous discretionary model that was proposed in the literature is the access matrix model. This model is based on the subject-object distinction which represents the key concept in this model [82]. It was first proposed in [57] and allows for describing discretionary access control in operating system context. Then, the model was refined by Graham [38]. The formalization of the access matrix model called HRU model was proposed by Harisson, Ruzo and Ullman [42].

Three kinds of entities are defined: (i) Objects: which represent the resources to be protected such as files in file systems or shared documents in collaborative editors; (ii) Subjects: are the users of the objects to be protected; (iii) Access rights: are the operations to be executed by subjects on the protected objects.

An access matrix is a matrix containing subjects in rows and objects in column. Each entry of the matrix denotes the rights a subject has on a given object. To check an access rule, an action is granted to a user if and only if his access matrix entry contains the appropriate rights allowing action performance.

Figure 2.3, represents an example of the access control model with a small set of subjects ({Alice, Bob, Carl}) having the possibility to own, read, write and execute rights on four files. This matrix specifies for example, that Alice has the right to read and write the file number 2.



	File 1	File 2	File 3	File 4
Alice	Own Read Write	Read Write		Execute
Bob	Read		Read Write	
Carl				Execute Read

Figure 2.3: Example of Access Matrix Model

ACLs are by far the most common approach for implementing DAC policies; [45]. An ACL associates the permitted operation to an object and specifies all the subjects that can access the object. That is, each entry in the list specifies access rights given to a subject. From another point of view, this implementation stores the matrix by columns [80]. An example of the ACL implementation is given in Figure 2.4 showing the ACLs corresponding to the access matrix of Figure 2.3.

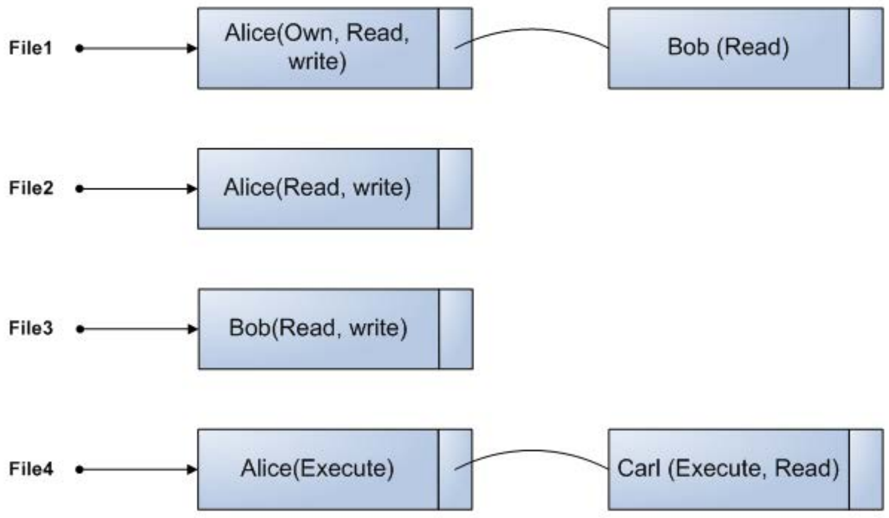


Figure 2.4: Access Control Lists (ACL)

ACLs are used in many commercial operating systems such as Microsoft Windows 2000, 2003, and XP. UNIX (and UNIX variants such as LINUX and FreeBSD) also support the implementation of ACLs [45].

Among the collaborative frameworks that use ACLs, we cite:

1. RTCAL: RTCAL (Real-Time CALendaring) [39] is a collaborative application that allows users to collaboratively schedule meeting times. RTCAL is based on two kinds of data: private and public. Each user has access to a shared and a private column. The first one displays whether the time slot is free for all users or not. The second one shows details of any appointment of that user at that time. In this approach there are two kinds of commands: application commands and conference-control commands. The former manipulate the calendar where only the controller has the ability to enter these commands at any time. The latter are performed by participants in order to pass control, enter and leave the conference. Note that there is a special role (chairperson) that has the authority to end the conference and determine who is the current controller.

2. Grove Outline Editor: Like RTCAL, GROVE (GRoup Outline Viewing Editor) [32], is an almost WYSWIS group outline editor. However, it allows for fine-grained access control. Grove represents a simple outline-only editor for small groups and works as a tightly-coupled synchronous editor. GROVE outline is based on a recursive structure consisting of structured items. It is based on three important concepts: (i) View: each user has access to a view which is a subset of Grove that may be private (only visible locally), public (visible by all), or shared (by invitation only). This is specified by the user when starting a Grove session. Private and shared views may be dynamically forked from existing shared and public views.(ii) Viewer: a viewer is a window that displays an outline view. (iii) Viewport: Each user displays a view of the outline in a viewport.

All changes made by a user are immediately transmitted to the collaborators. It is also possible for many users to change an item simultaneously. Grove ensures that they see these changes in the same logical order in an optimistic way (transformation functions) without centralization or aborts usage. Moreover, it does not support locking, and relies on social protocol to prevent conflicts. However, accidental deletion may occur enough which makes Undo necessary.

3. Diva [91]: is a prototype virtual office environment replacement for the ubiquitous graphical user interface (GUI) desktop that provides integrated support for awareness of coworkers, tools for performing shared tasks, and facilities to support communication, in both the synchronous and asynchronous modes of work. To manage access control, DIVA provides availability states for rooms. Consequently, users have the control over both their availability and the awareness information about others. The model includes access lists to give users control over the use of rooms and documents.

Another approach to implement access control matrix are Capability lists (CL). This approach is dual to ACL approach and stores the matrix by rows. It associate a list with appropriate access on object for each subject in the matrix. Figure 2.5 shows the CLs corresponding to the access matrix of Figure 2.3.

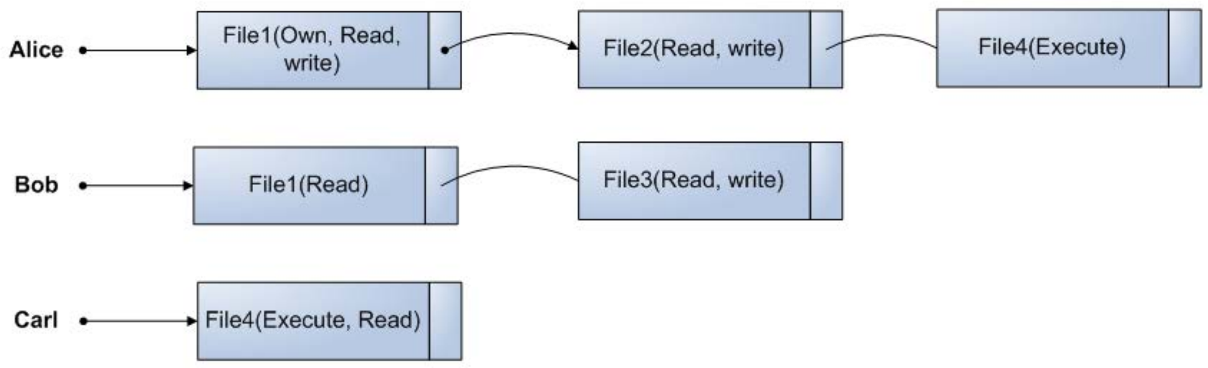


Figure 2.5: Capability Lists (CL)

In the following, we state the shortcomings of this model and its implementation.
Although it represents a good abstraction of access control, the matrix model has some weaknesses:

- Inappropriate to complex access policies: indeed sophisticated policies such that policies that regulate access based on competency or conflict-of-interest rules are difficult to describe with access matrix because they generally associate credentials to subject when granting or revoking actions [107].

- Static access rules: ACLs and CLs are not adequate to implement dynamic changes of access rights which represents a mean feature of a collaborative application where rules may change at any time of the collaboration according to the behavior of users [107]. Moreover, with ACLs, it is easy to revoke all access rights of an object by deleting the corresponding list. However, determining all granted access to a given user is very difficult to achieve since it needs to examine all ACLs in the system. Consequently, revoking all rights of a subject is very expensive [80, 107]. The dual problem is faced in the case of CL, it is very difficult to revoke all accesses on a given object.

- ACLs are platform-dependent since different systems have their own format of ACL, which is inconvenient, subject to error, slow, and makes it difficult to identify or model the overall “policy” that is enforced by the system [44, 45].

- In some collaborative applications, ownership must not be at the discretion of the user in which case the system must own resources. However, ACLs as well as CLs are mostly appropriate to file systems and do not allow to define access rights on some attributes of a resource content, attribute or contextual information [107].

2.2.2.2 Mandatory Access Control (MAC)

The mandatory access control model manages access to resources based on a central authority. The need for a MAC model arises when protection decisions must not be decided by the object owner as in DAC model but rather enforced by the system.

The most common form of mandatory policy is the multilevel security policy [80] based on the classification of *subjects* and *objects* in the controlled system where subjects request access to the objects. The notion of subjects is different from that of DAC model. In fact, users or groups may be considered as subjects in DAC model. However, in MAC model, there is a distinction between users (human beings accessing to objects) and subjects that represent processes performed by users. This allows for controlling indirect access resulting from processes execution since the same user is able to execute many processes [80].

In the MAC model, an access class is assigned to each subject. Thus, the policy is specified as a set of partially ordered set of classes where each class dominates the following one according to the partial order. An example of classes would be Top Secret (TS), Secret (S), Confidential (C) and Unclassified (U) where TS dominates S which dominates itself C which dominates U [80].

The MAC model offers a high level of security and is generally is used in multilevel systems to protect highly sensitive data such as military information.

2.2.2.3 Role-based Access Control (RBAC)

The RBAC model is based on groups or roles rather than individuals to regulate access to resources. The main idea of RBAC is grouping privileges. This idea was firstly introduced by [112] for an SQL-based framework. In general, roles are statically created, then users are assigned to roles according to their responsibilities and/or qualifications. Different approaches [6, 51, 69, 85, 86] of RBAC have been made with the concept "role" that may be related to the user's job or to designate a task ($i.e$ order processing, administration, *etc*).

In this model, there is a distinction between two logical parts where the first assigns users to roles while the second assigns access rights to roles (see Figure 2.6). In RBAC, access rights are grouped by role name and access to resource is regulated according roles membership where users are granted membership into roles based on their role in the organization. For instance, in an academic system, the role of teacher can include operations to prepare courses and/or exams, correct exams while the role of student can be limited to consulting information for studies, answer online exams/exercises.

Consequently, access is regulated according to the role instead of specifying rules for each action performed by each user. The way access is enforced by assigning first roles to users then assigning access rules to roles simplifies the security policy management. Indeed, user membership into roles can be revoked easily. Furthermore, updating roles does not require to update the privileges for every user. Moreover, in many applications, there is a natural hierarchy between roles which can be exploited to imply authorizations. This greatly simplifies the access control specification and management.

The RBAC model is very practical for traditional as well as collaborative environments, it provides a promising paradigm for many commercial and government organizations. However, it is limited by the following shortcomings:

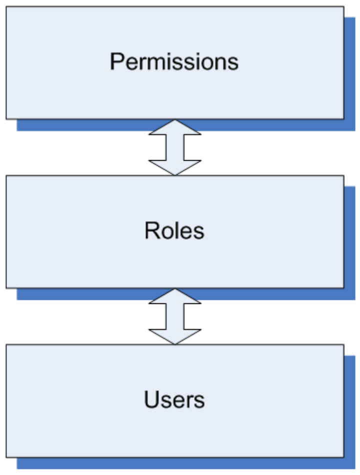


Figure 2.6: RBAC model.

- Static role assignment: the earlier implementations of RBAC model defines roles and assign users to them at the beginning. Hence, dynamic changes of role assignments were badly managed. One can designate this kind of roles as static roles lacking flexibility and not responding to dynamic applications requirements [107].
- Static session roles: the family of models RBAC96 presented in [84] overcomes some problems with dynamic change of rights since it introduces the notion of session associated to each user [53]. A session starts when the user authenticates with the system and ends when he exits the system and allows him/her to activate the permissions of a subset of roles that he/she belongs to. For every session, users can be assigned new permissions. However, the concept of session also prevents a dynamic reassignment of roles since the user roles cannot be changed within a single session. Indeed, every user has to authenticate again in order to obtain new permissions.
- Lack of fine-grained control: in collaborative systems, users may need specific access rights independently of the role to which they belong or additional access rights inside their membership. This is not supported by the RBAC model since it is not possible to define fine-grained access control on individuals or objects. For instance, suppose that the user having the role *secretary* need to replace her manager during his absence. Suppose that she needs only some administrative rights granted to the role *manager* and not all of them. In this situation, it is not tolerated from a security point of view, to assign all the rights of *manager* role to the secretary.
- In some systems mainly those that do not follow organizational scheme, least privilege is difficult or costly to achieve due to ambiguity and difficulty to tailor access based on various attributes or constraints. In fact, the concept of least privilege requires many steps: first, the user's job functions must be identified, then the minimum set of privileges has to be defined , finally each user must be restricted to a domain with those privileges.

2.2.2.4 The Intermezzo and Suite Models

In the following, we briefly discuss two famous approaches based on both access lists and roles in conjunction.

The Intermezzo model. Edwards et al. [30] propose a system for implementing a variety of policies mainly in the areas of awareness and collaboration. The model supports dynamic policies and is implemented atop the Intermezzo collaboration environment [31]. Intermezzo framework supports replication and coordination in the form of session management and provides awareness information for users. It provides access control for reading, writing, removing and testing the existence for objects. Denial of access is achieved thanks to a special right NONE.

Policies are described in terms of access control rights on data objects, and are assigned to roles (groups of users) where roles represent not only statically-defined collections of users, but also dynamic descriptions of users evaluated at the run-time of the application. This dynamic aspect of roles allows them to meet flexibly the collaboration dynamism. This is achieved by determining users membership at the moment of evaluation of a given request thanks to predicate functions instead of using static membership lists.

The implementation of the access policies are based on poor scaling ACL. Moreover, access rights are validated at a remote server which degrades the performance of the application. Furthermore, author did not discuss how to manage security when the server crashes in which case, users may be prevented to perform actions.

The SUITE model. The SUITE access control model [89] proposed by Shen and Dewan is an extension for the matrix model that was implemented in the collaborative Suite multi-user framework allowing for collaborative editing [27].

The model is characterized by:

- A large set of access rights that meets the SUITE framework [27]. For instance, in addition to classical access rights (read, write, delete) access rights like viewing, eliding and hiding was introduced [89].
- Negative rights: the notion of negative rights was previously introduced in database area [75]. This notion allows for explicit denial of access and facilitates access right specifying in presence of a large amount of controlled objects and/or subjects. For instance, consider the scenario where a user u wants to grant the read right to all group members except the user v. Without negative right specification, u would require to grant read right to each one of collaborators except v. Thanks to negative rights, the problem is easily solved by granting a positive read right to all users and an explicit negative read right to v.
- Inheritance-based specification: the model allows for access right specification for groups of users, objects and rights. Moreover, entries of the access matrix are not necessarily specified and can be inferred from other values in the matrix which meets very well the dynamic aspect of collaboration [89].

This approach is a good mixture between matrix and RBAC model since it allows for grouping users. However, since it specifies a large set of rights, inheritance and implication relations were defined in order to infer access rules and simplify access specification to the user. Unfortunately, these two relations introduce conflicts when two contradictory access control actions (grant and revoke) may be inferred at the same time. This requires conflict resolution rules which complicates the model as well as its mechanisms and makes it difficult to be generalized for any collaborative application.

2.2.2.5 Task-based Authorization Control (TBAC)

TBAC is an access control model proposed for active and enterprize-oriented authorization management. It models access control from a task-oriented perspective rather than the traditional subject-object perspective where access mediation involves authorizations at various points during the completion of tasks.

It is supposed to meet the requirements of agent-based distributed computing and workflow management [106].

Preliminary ideas for TBAC were presented in [87]. Then a workflow authorization model named WAM was proposed in [8]. These works recognized the need for active security and just-in-time permissions. However, TBAC is more comprehensive and conceived better than WAM since the latter proposes a primitive authorization concept [106]. Furthermore, TBAC provides many features such as holding permissions temporarily, tracking the usage of permissions, *etc*.

Access control in this model is managed in many steps related to the progress and life-cycle of tasks. A protection state is associated to each step (see Figure 2.7). This protection state contains a set of permissions that change according to the task evolution. Consequently, the model is considered as an active model since it deals with constant changes of a task allowing by this for dynamic management of permissions. Moreover, authorizations in the TBAC model have a strict runtime usage, validity, and expiration characteristics defined according to the task and its protection state.

As shown in Figure 2.7, the main components of the TBAC model are entities of the workfolw and protection states associated to each of them. TBAC is based on the use of type-based access control and dynamic check-in and out of permissions from protection states.

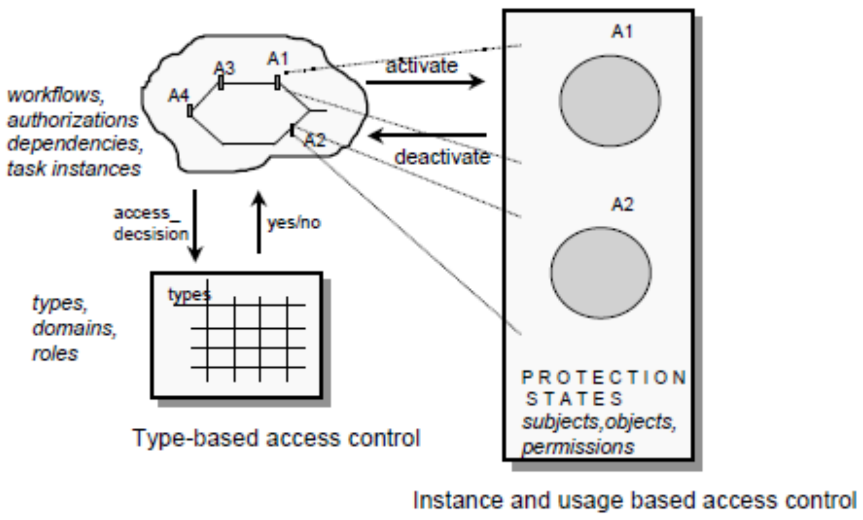


Figure 2.7: TBAC as an Active Security Model [106].

Shortcomings. The TBAC model has some weaknesses [107]:

- TBAC systems mainly suit task-based or workflow applications. Nevertheless, collaborative work is not always based on tasks. To illustrate this point, consider a simple collaborative application allowing for editing shared text document such as a wiki. In this application, each user has the ability to update the document without the need of assigning users to tasks or following a workflow scheme.

- Permissions are activated and deactivated in a just-in-time fashion which may cause problems in the case of a centralized access control management. In fact, if the central authority crashes, security holes may occur. This necessitates additional constraints to well manage just-in-time permissions.

- Specification of complex policies as well as delegation and revocation of authorizations are very primitive in TBAC.
- Even though TBAC meets enterprise applications more than the subject-object approach, it needs to be used within other access control models. For instance, in [54, 71], authors associate TBAC and RBAC to define access control models for inter-organizational workflow.

2.2.2.6 Team-based Access Control (TMAC)

In contrary with RBAC model, the TMAC model is based on a more natural way to group users among organizations and enterprises based on teams. The approach proposes a fine-grained access control allowing access assignments to individuals on object instances which is not supported by RBAC. This model was proposed by [105]. As shown in Figure 2.8, this model introduces two important notions:

1. User context: aims at identifying users playing a role into a given team at any given moment;
2. Object context: identifies which objects are needed by a collaboration session.

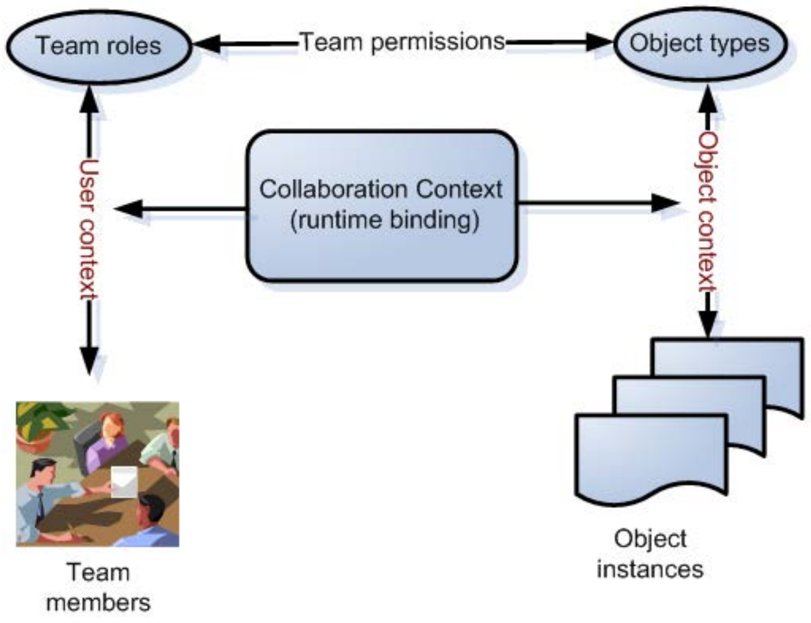


Figure 2.8: Main Concepts in TMAC Model [105].

The TMAC model allows for grouping users as well as fine-grained access control in contrary to RBAC model. It was extended by the C-TMAC approach [37] in order to take into account contextual information (time, place, etc).

Even though the TMAC and C-TMAC support contextual specifications, they have the following shortcomings [107]

- The models have not been fully developed and tested to demonstrate they are applicable.
- There is a lack of a fine grained access right specification allowing to specify access assignments concerning entities and relations.

- The model was used in conjunction with RBAC for Hypermedia environments [111] where roles are defined across teams. This proves that TMAC model needs some extension to be applicable in real word applications.

2.2.2.7 Spatial Access Control (SPACE)

Spatial Access Control (SPACE) has been proposed to solve the problem of role migration within a session [17]. Instead of splitting users into groups as in RBAC, SPACE divides the collaborative environment into abstract spaces in which subjects and objects reside. Users migrate from space to space in a session based on a set of predefined rules. SPACE is modelled by an access graph, where nodes are the spaces and the arcs are the rules. The access graph aims at defining constraints to move from one region to another. The model also uses credential to allow access within regions, these credentials label the edges of the graph (see Figure 2.9).

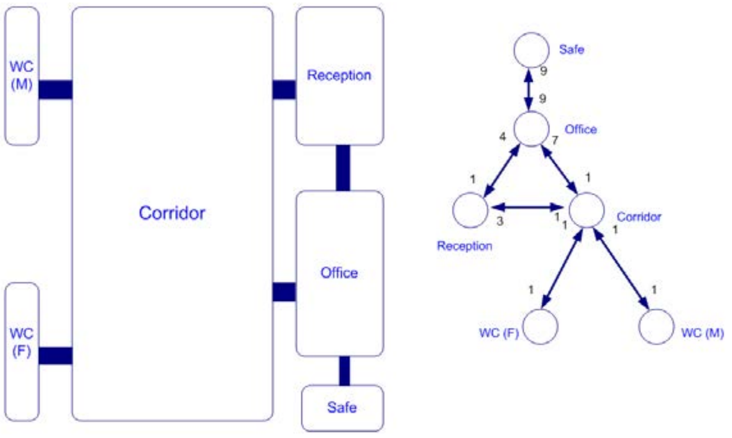


Figure 2.9: Space Model Abstraction of an Office Environment [17].

The SPACE model was implemented in the graphical collaborative virtual environment Spline [4]. It has the following limits:

- Not generic: SAC implementation needs prior knowledge of the practice used in some collaborative system, in order to produce a set of rules that are generic enough to match most of the daily access patterns. This model is appropriate only for the collaborative environments with regions and boundaries.
- Locking: Every access needs to check the underlying access data-structures which requires locking data-structures and reduces collaborative work performance.
- Lack of fine-grained access specification: This model does not provide a fine-grained control since it concerns only navigational access requirements *i.e* “Can i get into this office?”, hence does not suit applications with specific controls or very large number of objects such as collaborative editors [17].

2.2.2.8 Context-aware Access Control

The RBAC model was extended in order to provide security for context-aware applications. In [23], RBAC was extended with the environment notion, *environment role* [23] were used instead of traditional roles in order to capture environment state of a role (see Figure 2.10). Roles are activated according to environmental conditions (*e.g* time, location) when a request is issued. Permissions are assigned to a set of roles where a set may include both subject and environment roles. Thus, the approach is considered as a generalization of the RBAC traditional model.

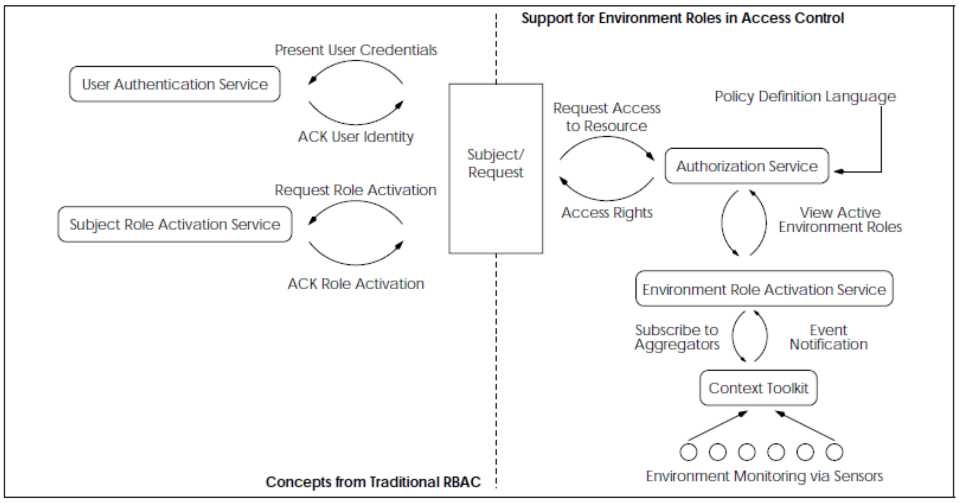

Support for Environment Roles in Access Control
Present User Credentials
User Authentication Service
ACK User Identity
Request Role Activation
Subject Role Activation Service
ACK Role Activation
Subject/ Request
Request Access to Resource
Access Rights
Policy Definition Language
Authorization Service
View Active Environment Roles
Environment Role Activation Service
Subscribe to Aggregators
Event Notification
Context Toolkit
Environment Monitoring via Sensors
Concepts from Traditional RBAC

Figure 2.10: Environment Roles in Context-aware Access Control [23].

This approach is useful for ubiquitous computing. However, it requires to be tested whether it meets collaborative applications or not [107]. Moreover, as in RBAC, the system administrator must define environment roles at the beginning. Furthermore, each subject or client in the model must contact a server to obtain authorization to access the desired service. The administration of access is centralized due to the nature of the targeted systems (ubiquitous computing systems such as Aware Home).

Other works have focused on extending RBAC with context-awareness and dynamic activation of permissions. The DRBAC [118] model provides context aware access control for pervasive applications. DRBAC also extends the RBAC model by dynamically adjusting role and permission assignments based on context information. However, DRBAC must be combined with feasible authentication mechanisms to secure pervasive applications in the real world [118].

The GTRBAC model [14] provides mechanisms for enabling and disabling roles based on temporal constraints. The CA-RBAC [55] uses context-based constraints that include temporal and spatial constraints. They are specified as part of role admission/validation and role operation preconditions. This approach supports fine-grained access control requirements. Indeed, it is possible to selectively revoke a user's membership from a role, or activate/deactivate specific role permissions, instead of enabling/disabling a role.

The Or-BAC [33] also generalizes the RBAC model by introducing two abstractions of action and object called activity and view. The central concept in Or-BAC is the Organization that can be seen as an organized group of subjects. Furthermore, the Or-BAC model allows for specifying contextual information to better manage situations specific to a given context (*e.g.* bad connexion, temporal information, *etc.*). These extensions have introduced the notion of contextual or environmental roles to handle some events in the system. Consequently, it is possible to assign rôles dynamically.

2.2.3 Access Control in Database Area

An important requirement for collaborative applications is the high response time. To avoid additional time processing needed by the security policy in order to validate accesses, we believe that security policy must be replicated on all collaborating sites. Hence, we study the solutions that replicate the policy and allow for concurrent updates targeting the policy.

Samarati et al. [79] addressed authorization replication in database area. They dealt with the propagation of authorizations in distributed relational database systems. Authors proposed an optimistic replica control algorithm ensuring consistency even though transient inconsistency are allowed. Indeed, the replica control algorithm allows the out of order execution between different updates performed on the authorizations policy. Consistency is maintained thanks to reprocessing updates that have been recognized as received out of order without resorting to undo-redo mechanism but with time stamps usage. Time stamps may be defined in a centralized fashion or in a distributed one. The latter may be achieved through the Lamport clock [56]. However even though it allows for distributed timing, it assumes a fixed number of users which limits the scalability of the system.

Moreover, the presented model allows for discretionary administration. It should be pointed out that managing discretionary administration is a hard task in a distributed and replicated access control model. Indeed, the out of order execution between policy updates leads inevitably to violations and divergence. However, since the model is based on using time stamps it is always possible to know the correct order of different updates and grants performed on the policy.

However, the proposed model does not allow for grouping users. Instead, it deals with individual subjects (users) and objects (tables/tuples). Moreover the model presented above is not generic since it only deals with relational databases and resorts to Lamport time stamps.

Another work of relevance is that of Ray and Xin [76, 117] where authors focus on real time update of access control policies for database systems. In [76], two algorithms were proposed: syntax-based and semantic-based in order to allow for concurrent and real time update of the security policy.

The security model is based on a simple authorization-based policy where an access policy is associated with each object. Access rights are represented in a binary n-element vector, n being the number of access rights. For instance, suppose that two access rights (Read and write) are allowed. The binary vector [01] of a given object allows write but not read on that object.

In [117], a third algorithm were proposed to deal with conflicting situations when multiple access right policies are specified over the same object. Priorities are assigned to policies to overcome this problem. However, even priorities are subject to concurrent modifications in which case a lattice-based approach is used to distinguish between policy relaxation and restriction.

Similarly to [79], access rights are specified only for one user on one object. Hence, it is not possible to define access rights for group of users or objects. Moreover, while the approach addresses concurrency in updating security policy, it does not allow users to update at any time since it is based on locking mechanisms for both shared data and policy object in order to prevent inconsistencies of both objects which leads to poor performance and many aborts of operations. To increase concurrency, authors propose to distinguish between policy restriction and relaxation in order to limit aborts. Nevertheless, cooperative systems need to exploit the full potential of concurrency and avoid locks in order to offer high response time as well as availability of both data and security policy. Moreover, this approach does not allow for explicit revocations and replication.

2.2.4 Access Control for P2P Systems

Several works addressed security in peer-to-peer (P2P) environments. In [65], authors propose an encryption-based access control mechanism for a p2p file sharing system. A framework for enforcing access control policies on published XML documents was proposed. In this framework the owner publishes a partially encrypted single data instance. The solution is based on a declarative language for access policies. The data owner enforces an access control policy by granting keys to users. This approach concerns the sharing of data and not concurrent updates of the shared data by several users.

Another approach was presented in [16] for managing access control in a distributed fashion in a p2p network. This approach is based on the access control model of Bertino [15] and Samarati [26]. It proposes a client-based access control evaluator motivated by the emergence of hardware and software security elements on client devices as well as the erosion of trusting servers due to internal and external attacks. Even though, this model manages the replication of access rules in client devices, it only concerns sharing of data. Moreover, the abstraction of target applications makes no assumption on how the policy is updated and by whom.

Another approach based on trusted computing is proposed in [83] in order to enforce access control policies in application layer of a P2P application. The proposed framework is an abstract platform beyond the layer mechanisms of trust computing (including kernel architecture, hardware and attestation mechanism). This framework is based on using cryptography in conjunction with RBAC model with the aim to control data dissemination in order to restrict the misuse of data. Note that obtaining certificate for a given role requires a central server called *role server* as well as a trusted third party (certificate authority) or direct anonymous attestation to certify users. Examples of targeted applications would be a document dissemination control application or P2P VoIP applications. The work does not discuss the concurrent modification of shared objects.

Another security approach for information sharing systems was proposed in [11]. This solution distributes the authorization and authentication enforcement in order to avoid single point of failure. However it relies on centralized knowledge to enforce policy to overcome inconsistency problems caused by autonomous authentication.

Some other approaches rely on eXtensible Access Control Markup Language (XACML) policies [24, 25] to specify the access restrictions on the data. These restrictions are stored in super nodes which limits the solution to hierarchical P2P networks.

It should be pointed out that none of the solutions presented above considers existing access control components and collaborative applications residing on the peers [93]. Moreover, administration distribution is not supported by these approaches which represents an important feature of access control in a p2p application.

Sturm et al. [93] provide a fine grained access control mechanism similar to those available in relational database systems. It supports delegation of access rights and is based on XACML. Moreover, the solution ensures secure authentication thanks to a public key infrastructure (PKI) with certificates to avoid "sybil" Attacks [29]. However, it does not allow for replicating the security policy which may introduce additional processing time to check access against the central policy. Users do not have the ability to check locally their access rights and have to wait for the reception of the access decision.

The solution of [115] is proposed to handle security in weakly consistent state-based replicated systems wherein many items collaborate concurrently (*e.g* laptop, phone, cloud, etc) and then synchronize periodically. They propose a logic access control policy formalized with SecPal [10]. The policy is replicated and allows only for positive rights, *i.e* no explicit access denial is supported. Although delegation is allowed, the revocation is limited in order to avoid ambiguity. To more illustrate this, neither revocations nor delegation of revocations can be revoked which is important in such situation. For instance, if we delegate the revocation right at a given user then we discover it is a malicious one, we can not revoke this action and hence this user may prevent some trusted collaborators to perform actions. Furthermore, policy enforcement relies on a single root of authority trusted by all collaborating items. The crash of this single root would create security problems and block the collaboration if it does not delegate all its capabilities to another user.

2.2.5 Comparing Existing Solutions

After presenting previously some background on main classes of policies in different areas, we are now in a position to discuss and compare them to better understand the differences between these models. This step is very important and aims at well defining the appropriate access control model for our work.

The main assessment criteria for access control in collaborative editors are the following

- Ease of use: this criteria indicates the simplicity of the model from user point of view [107]. Users should feel comfortable with the access control component and be able to specify access definitions easily [89]. the security model should require less effort from users [30].
- Expressiveness: the expressive power of a model allows for specifying complex security scenarios and deals with the ability of the model to capture the range of needs of system users [107].
- Applicability: this criteria shows the possibility for the model to be deployed in real world collaborative applications [107].
- Groups of users support: it indicates whether the model supports the notion of groups and allows to specify access rights for groups or not [107].
- Policy specification: policy specification is very important and allows for scalability and easy extension of the model [107].
- Policy enforcement: it is very important to provide means that ensures a correct enforcement of the policy specification [107].
- Fine-grained control: the system should support fine-grained subjects, objects and access rights [89]. This makes the model able to specify access rules not only for roles but also for individuals on one or many controlled objects [107].

Table 2.1 extends the comparative study given in [107] and resumes the different access control models as well as the differences between them.

Since our objective is to develop an access control model that meets RCEs requirements, we aim at defining a security layer characterized by its dynamic aspect and high responsiveness. Moreover, our access control model has to provide most importantly high distribution, high concurrency, availability and high responsiveness. Consequently, the proposed model has take into account these additional criteria:

- Simplicity and flexibility: this is a very important aspect since, we need to study many collaboration scenarios, the chosen model should be simple to allow for an exhaustive study of different issues that may cause divergence.
- Generic: it is desirable for an access control model to be generic so that any collaborative system can deploy it to protect easily shared data without redesigning an access control model each time.
- Policy replication: Replicating the access control policy is very important in collaborative applications. Indeed, this allows to respect the high responsiveness requirement of RCE. Moreover, it allows to benefit from the full potential of replicating the shared data.
- Open group: it is very important in a dynamic collaborative application to allow for open groups. In fact, many collaboration systems offer to users the ability to join and leave the application at any time (churn). Consequently, the collaborating group must be open and must have a variable size.

To achieve our goals, we deeply concentrate in [79] since it discusses security policy replication and [89] since it resorts to a policy specification that allows for a dynamic access control management. These two works are related to our work and served as a basis of the security model that we propose. The solution of [115] was published recently assuming it is the first work that address policy and data replication while our first work was published one year before. However, it relies on partial and state-based replication in the context of weakly consistent replication while we propose an access control model for collaborative editors where data is totally replicated.

2.3 Conclusion

Collaborative applications are becoming more and more pervasive. As a matter of fact, many applications are designed in a distributed fashion in order to meet collaborative work requirements. In our work, we focus on collaborative editors with the aim to extend them with a security layer since there is a lack of an adequate access control concept that ensures security of shared documents.

In this chapter, we have discussed the main requirements of a collaborative editing work in real-time context. Then, we have surveyed the most important access control models in order to well define an access control model for a distributed collaborative editor. Furthermore, we have highlighted their weaknesses to meet RCEs requirements. Indeed, controlling access in a decentralized fashion for such systems is still a challenging problem, as they need dynamic access changes and low latency access to shared documents.

In the following Chapter, we propose our model that fulfils the main requirements discussed before. To the best of our knowledge, our model is the first generic access control model based on replicating the shared document and its authorization policy that addresses document updates.

Criteria	Matrix	RBAC	**SUITE [89]**	TBAC	TMAC	**Space**	Context Aware	**Samarati et al. [79]**	**Ray and Xin [116]**	**Policy-based AC [115]**
Ease of use	Medium	High	Medium	Medium	High	Low	High	High	High	Medium
Expressiveness	Medium	High	High	High	High	Medium	High	High	High	High
Applicability	Medium	High	Medium	Medium	Medium	Low	High	High	High	High
Groups of users support	Low	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Policy specification	Low	Yes	Yes	Low	Yes	Yes	Yes	Yes	Yes	Yes
Policy enforcement	Low	Yes		Low	Yes	Low	Yes	Yes	Yes	Yes
Fine grained control	No	Low	Yes	Low	Yes	No	Yes	No	Yes	Yes
Generic	No	No	No	No	No	No	No	No	No	No
Open Group	No	No	No	No	No	No	No	No	Yes	Yes
Policy replication	No	No	No	No	No	No	No	Yes	No	Yes

Table 2.1: Comparison between Access Control Models

Chapter 3

Our Generic Access Control Model

Contents

Controlling access in RCE while meeting their requirements discussed in Chapter 2 is a challenging task. The major problem of latency in access control-based collaborative editors is due to using one shared data-structure containing access rights that is stored on a central server. So controlling access consists in locking this data-structure and verifying whether this access is valid.

Thus, a centralized solution is to be discarded since it limits the potential of the collaborative application. Indeed, a central server for verifying access rights leads to high latencies.

To overcome the latency problem, we propose an access control model based on replicating the access data-structure on every site. Thus, a user will own two copies: the shared document and the access control policy. It is clear that this replication enables users to gain performance since when they want to manipulate (read or update) the shared document, this manipulation will be granted or denied by controlling only the local copy of the access data-structure.

However, combining access control with totally consistent replication presents a real challenge if the resulting system is to be consistent. Indeed, if authorization policy can be temporarily inconsistent then any given operation may be authorized at one node and yet denied at another. This is troublesome because it leads to security holes as well as the divergence of the shared document.

Furthermore, unlike traditional single-user models, collaborative applications have to allow for dynamic change of access rights, as users can join and leave the group in an ad-hoc manner.

In this chapter, we describe the design of an access control system for replicated collaborative editing systems hat satisfies RCE requirements (see Chapter 2). Our access control model allows for the specification of a fine-grained access control over a collection of replicated copies of the shared document. The consistency of the replicated data is preserved despite the temporary inconsistency of the access control policy.

Our contributions are as follows:

1. Access to shared documents is controlled in a decentralized and optimistic fashion in the sense that every user sees the effect of his modifications immediately even though they were remotely revoked. Then a correct state is restored after revocations are received locally.

2. Security policy is replicated at every collaborating site in order to instantly control the access to shared documents.

3. Our model is generic since it meets existing coordination protocols. Moreover, it is characterized by its good performance since our algorithms have a low complexity.

This chapter is organized as follows: Section 3.1 is dedicated to the access control model and its ingredients. Section 3.2 discusses security and convergence issues. In Section 3.4, we propose a garbage collection protocol to clean logs. In Section 3.3, we demonstrate the generic aspect of our solution. Finally, Section 3.5 concludes.

3.1 Coordination Model

Every existing collaborative editing solution defines a data object that is shared by all collaborators having the ability to update it concurrently. In the following, we propose a shared security object to be deployed on the top of a given shared data object in order to ensure access control to shared objects.

3.1.1 Shared Data Object

We consider collaborative objects shared by many users in order to perform a common task simultaneously. Operations altering the shared object are called *cooperative operations* and referred to with the letter c. Let $\mathcal{C}$ be the set of all cooperative operations. These operations are generated locally by each site and then broadcast to other collaborators.

Example 3.1.1 (Collaborative Text Editors.). *For instance a shared object may be a text document. The document may be considered as a sequence of paragraphs where each paragraph has a unique identifier and represents an element of the whole shared data object.*

Practically, text editor algorithms consider two cooperative operations:

- $Ins(p, e)$ *where* p *is the insertion position and* e *the element to be added at position* p.

- $Del(p, e)$ *which deletes the element* e *at position* p.

Some approaches allow for a third operation $Upd(p, e, e')$ *which replaces the element* e *at position* p *by the new element* e' *[19, 48, 102].*

In our work, we consider coordination models allowing for concurrent execution of cooperative operations. Such systems, generally, use the OT approach to synchronize cooperative operations. Subsequently, we assume that every site has a local buffer (see Chapter 2) of cooperative operations also called *cooperative log* and referred to with letter H (see Definition 3.1.1).

Definition 3.1.1 (Cooperative log)**.**

Each site s maintains a buffer H that stores all cooperative operations. A cooperative log is represented by the sequence of cooperative operations $c_1 \cdot c_2 \cdot \ldots \cdot c_n$ executed at site s.

Given any shared data object based on log usage, we aim at extending it with a security layer to allow for access control to the different replica hold by collaborators. In the following, we describe the design of our access control model.

3.1.2 Shared Policy Object

We consider an access control model based on *authorization policies*. Three sets are used for specifying authorization policies, namely:

1. S is the set of *subjects*. A subject can be a user or a group of users.

2. O is the set of *objects*. An object can be the whole shared document, an element or a group of elements of this shared document.

3. R is the set of *access rights*. Each right is associated with an operation that a user can perform on shared document. For instance, we can consider the right of inserting an element (i), deleting an element (d) and updating an element (u). The read right is out of the scope of this thesis but we plan to give an outlook on future work.

An authorization policy specifies the operations that a user can execute on a shared document.

Definition 3.1.2 (Policy)**.**

A *policy* is a function that maps a set of subjects and a set of objects to a set of signed rights. We denote this function by $P : \mathcal{P}(S) \times \mathcal{P}(O) \to \mathcal{P}(R) \times \{+, -\}$, where $\mathcal{P}(S)$, $\mathcal{P}(O)$ and $\mathcal{P}(R)$ are the power sets of subjects, objects and rights respectively. The sign "$+$" represents a right *attribution* and the sign "$-$" represents a right *revocation*.

We represent a policy P as an indexed list of authorizations. Each authorization $l_i \in P$ is a quadruple $\langle S_i, O_i, R_i, \omega_i \rangle$ where $S_i \subseteq S$, $O_i \subseteq O$, $R_i \subseteq R$ and $\omega_i \in \{-, +\}$. An authorization is said *positive* (resp. *negative*) when $\omega = +$ (resp. $\omega = -$). Negative authorizations are just used to accelerate the checking process. We use a first-match semantics: when a cooperative operation is generated, the system checks it against its authorizations one by one, starting from the first authorization and stopping when it reaches the first authorization l that matches the cooperative operation. If no matching authorizations are found, the operation is rejected.

Strings "All" and "Doc" refer to the set of all subjects and all objects respectively. For instance the authorization $l = \langle All, Doc, i, + \rangle$ assign a positive right to insert new objects for all users.

We suppose that the user who assigns authorizations is able to perform *administrative operations*. An administrative operation is simply an operation that updates the policy by adding or deleting an authorization.

Definition 3.1.3 (Administrative Operations)**.**

The state of a policy is represented by a triple $\langle P, S, O\rangle$ where P is the list of authorizations. The administrator can alter the state policy by the following set of *administrative operations*:

- $AddAuth(p, l)$ to add authorization l at position p;
- $DelAuth(p, l)$ to remove authorization l at position p;

An administrative operation a is called *restrictive* iff $a = AddAuth(p, l)$ and l is negative or $a = DelAuth(p, l)$ and l is positive.

Notations $a.p$ and $a.l$ refer to the position and authorization of the administrative operation a.

The policy object is *replicated* in order to avoid high latencies introduced by checking whether a cooperative operation is valid or not with respect to the policy. The replication aims at satisfying the high local responsiveness requirement of RCE.

Collaboration Protocol. We consider the collaboration protocol presented in the flow chart of Figure 3.1. This protocol consists of the following steps:

1. When a user manipulates the local copy of the shared document by generating a cooperative operation, this operation will be granted or denied by only checking it against the local copy of the policy object (step (1) in Figure 3.1).

2. Once granted and executed, the local cooperative operations are then broadcast to other users. A user has to check whether or not the remote cooperative operations are authorized by its local policy object before executing them (step (2) in Figure 3.1).

3. When an administrator modifies its local policy object by adding or removing authorizations, he sends these modifications to the other users in order to update their local policy copies (steps (3) in Figure 3.1).

We assume that messages are sent via secure and reliable communication network, and users are identified and authenticated in order to associate correctly access to these users.

Although this protocol seems very simple, we show in Section 3.2 that many security and convergence problems are encountered during collaboration.

Another important aspect of the access control policy is whether the model is mandatory or not. Indeed, two approaches are possible when building the security layer based on the protocol presented above: the *single-administrator* approach and the *multi-administrator* one. Each approach has its advantages and shortcomings as illustrated subsequently.

3.1.3 Single and Multi-Administrator Approaches

When designing a security model, an important question arises: *"is there only one administrator or more?"* even though the answer depends on the targeted application, it is better for our access control model to be flexible and *easily* extensible from single to multi-administrator model in order to satisfy a wide variety of applications.

In the following, we discuss both single and multi-administrator approaches based on motivating examples for both approaches.

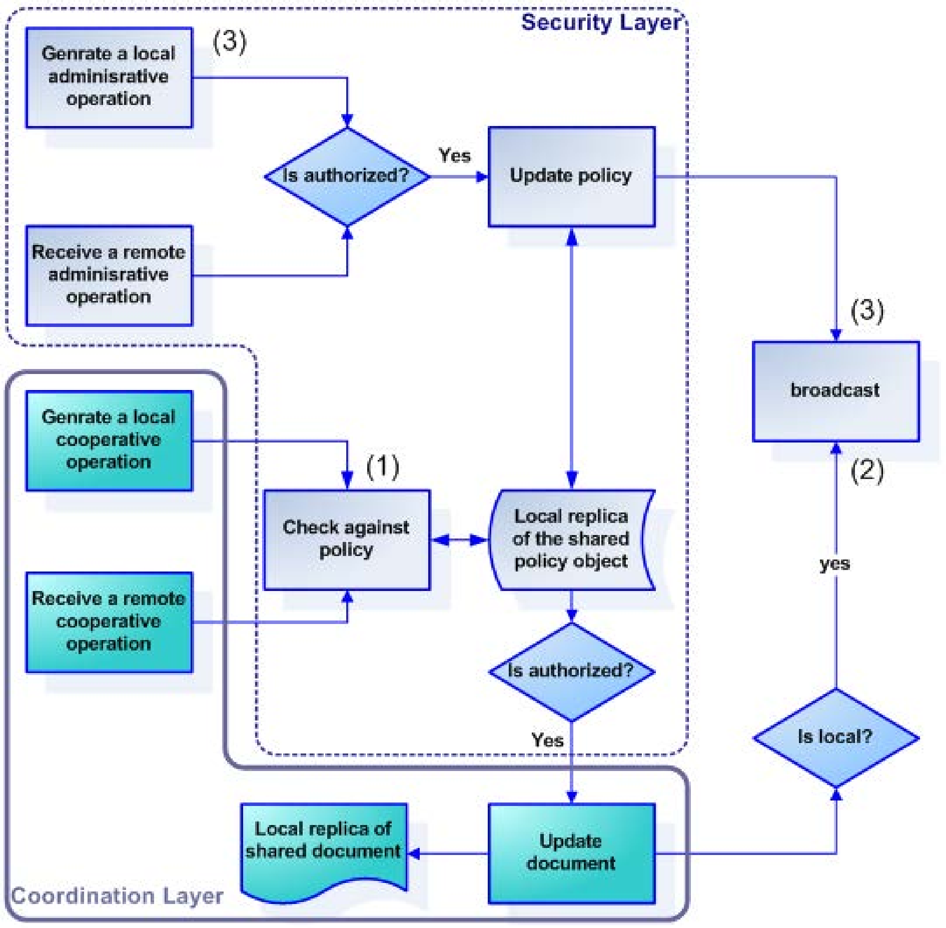

Security Layer
Genrate a local adminisrative operation
(3)
Is authorized?
Yes
Update policy
Receive a remote adminisrative operation
(3)
broadcast
Genrate a local cooperative operation
(1)
(2)
Check against policy
Local replica of the shared policy object
yes
Receive a remote cooperative operation
Is authorized?
Yes
Is local?
Local replica of shared document
Update document
Coordination Layer

Figure 3.1: Collaboration Protocol.

Single-Administrator Approach. Many distributed applications require only one administrator in the group. This form of control is useful to maintain group focus, especially when working toward a strict deadline [61]. In a single-administrator application, the administrator is generally the user creating the group or supervising it. In practice, some editing areas requires a single administrator, generally, the group creator. In such applications, the group creator prefers to have a full control over the application wherein mandatory model is of relevance. This is especially required when the administrator is the owner of the shared document. Among these applications, we cite that of preparing online exams where only the course's supervisor needs to assign rights to other users. To illustrate more, consider the example of the master degree specialized in "Natural Language Processing" taught in common by the Computer Science Departments of the Henri Poincaré University and the University of Nancy 2. Suppose that the course of Web Technologies is taught by two professors, the professor Bob gives theoretical courses and the professor Carl ensures practical class. When the supervisor prepares the exam for students, he wants that each professor prepares only his dedicated part (Bob for the theoretical part and Carl for the practical one). In such a situation, since the supervisor administrates the online exam, he has the ability to give each professor the appropriate access of his part of the document.

Another example would be a community of programmers where the supervisor of a development team has the whole authority on the collaborating group. He must be able to assign rights to programmers according to his needs and to the competence of each programmer while programmers should not have the right to assign right to other programmers. Let Bob be a supervisor administrating a group of two programmers Alice and Carl where Carl is a new programmer in the company. Consequently, Carl is less experienced and trusted than Alice and has just the right to develop some parts of the code that are not critical.

The main advantage of such a model is its simplicity. Indeed, it allows for an easy update of the policy since only one administrator changes access rights. However, from a security point of view, it is harder to control all users mainly in the case of large collaboration groups. Moreover, the absence of the administrator site due to crashes or physical absence inevitably leads to a static policy. For these reasons, the mandatory model does not meet all applications' requirements and depends on the application preferences and policy.

Multi-Administrator Approach. There are several distributed collaborative applications that require a multi administrator feature. In such applications, each user must be able to control the access to his objects since users may leave and join the group at any time without being members of the same community. Even though collaborative applications aim at making data available for many users, users may need to protect their data according to the nature of the collaborative application.

Shared calendars fall under the collaborative applications umbrella, where a multi-administrator policy would be of relevance. For instance, in professional context, a secretary may need to instantly see the activity of his/her manager in order to organize his meetings. On the other side, the manager can see the activities of his employees and so on. To underline the necessity of a multi-administrator policy in such an application, let us consider the case when the leader of a company is unable to edit his calendar and that in the meantime one of his/her customers wants to change the date of an important meeting with him/her. It would be more practical if the secretary were able to update the appropriate meeting of the manager's calendar. Hence, it is interesting to give the leader the ability to grant update right on all or parts of his shared calendar to his secretary. The secretary (his/her)self may need to grant some rights on his/her calendar to other secretaries or employees in order to get help.

Another motivating example is that of a collaborative application allowing for a community of researchers, doctors, patients and students to share a piece of information concerning a special disease. In such an application a researcher may allow doctors to read his scientific results for diagnosing reasons and students for analysing reasons. Moreover, a doctor may grant writing on his diagnoses to researchers.

According to these two examples, it is crucial to allow for a multi-administrator access control in existing collaborative applications.

Considering these motivations, we extend the shared policy object in order to take into account multi-administrator access control. In the multi-administrator access control, we refer to the *user ownership* approach in the sense that only the owner of an object has the right to assign rights to other users on that object. Otherwise the administrative operation is rejected. Hence, for a multi-administrator access control model, we consider the following meta-access-control policy:

1. Each user who joins the collaborating group is able to create new objects.
2. Each user who creates an object has the right to assign rights to other users on that object.

Accordingly, each user has an *owner policy* and every policy evolutes independently from other policies. Each administrator is able to specify access rights to other users on his objects. Owner policies are defined below.

Definition 3.1.4 (Owner Policy)**.**

An *owner policy* of a user i referred to as P_i is a policy administrated by user i on its own objects.

Moreover each user is able to perform the same administrative operations as in the single-administrator model to alter the state of its own policy P_i.

Definition 3.1.5 (Global Policy)**.**

A *global policy* is a set of a finite number n of couples (P_i, L_i) where P_i is the owner policy of the site i and L_i is its owner administrative log. We denote this set by $P_g = \{(P_1, L_1), (P_2, L_2), \dots, (P_n, L_n)\}$ where n represents the size of the collaborating group.

Let o and c be a shared object and a cooperative operation respectively such that c alters o. We consider the primitive $Administrator(c)$ which returns the owner of the object o altered by the cooperative operation c. This primitive allows for identifying the owner policy of o's administrator in order to correctly check a remote cooperative operation.

Note that the multi-administrator approach is more general than the single-administrator one and is required by many applications since it offers a higher flexibility and allows many users to edit access rights.

In the following, we investigate the issues raised by our protocol presented in Figure 3.1 for both approaches and present our solutions to overcome these issues.

3.2 Concurrency and Security Issues

The replication of the shared data object and the policy object is twofold beneficial: (i) firstly it ensures the availability of the shared document, (ii) and secondly it allows for flexibility in access rights checking.

However, this replication may create violation of access rights which inevitably leads to violate the consistency of shared data required by RCE as mentioned in chapter 2. Indeed, the cooperative and administrative operations are performed in different orders on different copies of the shared data and security objects.

In the following, we illustrate the issues raised by the performance of the administrative operations concurrently to cooperative ones and we present our solutions to address these issues. To better illustrate the problem, we use the shared text given in Example 3.1.1.

3.2.1 Out-of-order Execution of Cooperative and Administrative Operations

Performing cooperative and administrative operations in different orders at every user site may inevitably lead to security holes. and, most importantly, data divergence. To underline these issues we will present in the following four scenarios.

3.2.1.1 First scenario: Divergence caused by administrative operations

Consider a group composed of three users s_1, s_2 and s_3. Initially, the three sites have the same shared document "abc" (where the characters "a", "b" and "c" are inserted by s_1, s_2 and s_3 respectively) and the owner policies P_{s_1}, P_{s_2} and P_{s_3} respectively where P_{s_1} authorizes s_2 to delete the character "a" (see Figure 3.2). Suppose that s_1 revokes the deletion right to s_2 and sends this administrative operation to s_2 and s_3 so that it is applied on their local policy copies. Concurrently s_2 executes a cooperative operation $Del(1, a)$ to derive the state "bc" as it is granted by its local policy. When s_1 receives the s_2's operation, it will be ignored (as it is not granted by the s_1's owner policy) and then the final state still remain "abc". As s_3 receives the s_2's delete operation before its revocation, he reaches the state "bc" that will be unchanged even after having executed the revocation operation. We are in presence of data divergence even though the policy object is the same in all sites due to out-of-order execution of administrative and cooperative operations.

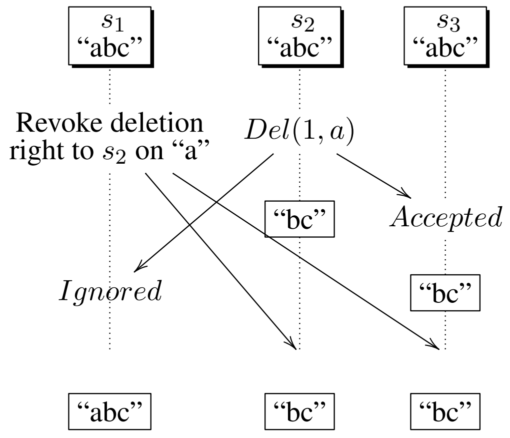


Figure 3.2: Divergence caused by introducing administrative operations

The new policy object is not uniformly enforced among all sites because of the out-of-order execution of administrative and cooperative operations. Thus, security holes may be created. For instance some sites can accept cooperative operations that are illegal with respect to the new policy which is the case of sites s_2 and s_3.

As our objective is to deploy such model in a decentralized environment, the solution based on enforcing a total order between both operations is to be discarded as it would require a central server or a total-order broadcast protocol [43] which is not appropriate for dynamic groups. Achieving this objective raises a critical question: *how the enforcement of the new policy is performed with respect to concurrent cooperative operations?* It should be pointed out that this enforcement may be delayed by either the latency of the network or malicious users.

To solve this problem, we apply the principles of optimistic security [73] in such a way that the enforcement of the new policy may be retroactive with respect to concurrent cooperative operations. This means that users are able to violate the policy temporarily until receiving concurrent administrative operations. Then, operations that violate the new policy enforcement are undone. Hence, only illegal operations are undone. For instance, in Figure 3.2, $Del(1, a)$ should be undone in s_2 and s_3 after the execution of the revocation.

To detect concurrency, each operation generated to alter an element e locally by a non owner of e is considered as a *tentative*. Hence, a remote administrative operation coming from a user s will be concurrent to all tentative operations performed on the objects administrated by s.

3.2.1.2 Second scenario: Causality between cooperative and administrative operations

In the scenario of the Figure 3.3, we address the causality between a cooperative and administrative operation. We firstly consider that the user s_2 has not the right to delete characters. Then the s_1 grants him the deletion right for character "a". Thus, he becomes able to perform a delete operation (in our case the operation $Del(1, a)$ is permitted). Now, if we suppose that the deletion arrives before the grant at the site s_1, we will obviously diverge since the deletion is rejected.

Intuitively, our solution consists in capturing the causal relations between cooperative operations and the policy copies on which they are generated. The causality reflects a dependency between cooperative

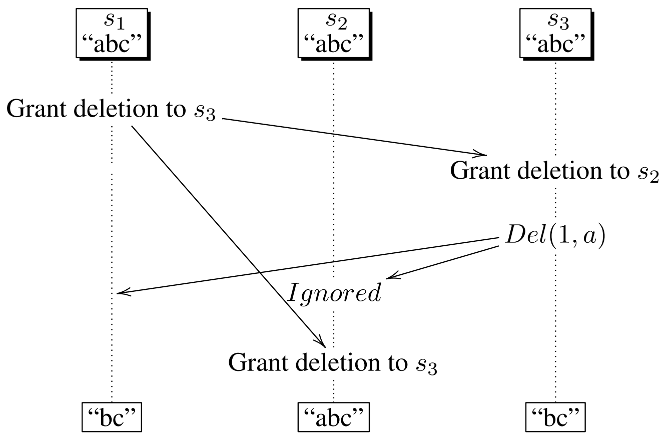


Figure 3.3: Causality between administrative and cooperative operations.

and administrative operations. Since every cooperative operation is generated at a given context (the version of the local copy of the policy object), every local policy copy has to maintain a monotonically increasing counter that is incremented by generating local or receiving remote administrative operations. If each granted cooperative operation is associated with the local counter of the policy object at the time of its creation, then we can correctly integrate it in every remote site.

3.2.1.3 Third scenario: Necessity of administrative logs

In this scenario, we depict the necessity of using administrative logs in order to keep a local trace of policy updates. In the scenario of Figure 3.4, three users see initially the same document "abc" and they use the same policy object containing only one authorization $l = \langle \{s_3\}, \{a\}, d, + \rangle$ allowing user s_3 to delete characters from the shared document. Firstly, s_1 revokes the deletion right to s_3 by removing the authorization (the policy becomes empty). Concurrently, s_3 performs $Del(1, a)$ to obtain the state "bc". Once the revocation arrives at s_3, it updates the local policy copy and it enforces the new policy by undoing $Del(1, a)$ and restoring the state to "abc".

The question that arises here is *how to integrate the remote operation $Del(1, a)$ at s_1 and s_2?* Before to execute this operation, if we check it directly against the local policy at s_1, it will be rejected since the policy is empty. After a while of receiving and ignoring operation $Del(1, a)$, s_1 decides to grant once again the deletion right to s_3. At s_2, the execution of both administrative operations leads to a policy copy allowing s_3 to delete characters. Before to execute $Del(1, a)$, if we check it directly with respect to the local policy of s_2 then it will be granted and its execution will lead to data divergence. Indeed, the generation context of $Del(1, a)$ *i.e* the local policy on which it was checked at s_3 is different from the current execution context at s_1 and s_2 due to precedent executions of concurrent administrative operations.

To overcome this issue, when the cooperative operation's counter is less than the policy copy's counter of another site then this operation need to be checked with respect to precedent concurrent administrative operations before its execution and not with respect to the policy. Therefore, we propose in our model to store administrative operations in a log at every site in order to validate the remote cooperative operations at appropriate context. For instance, in Figure 3.2, we can deduce that $Del(1, a)$ will be ignored at s_2 by simply checking it against the first revocation. Otherwise if the cooperative operation counter is greater than the policy version, the operation should wait until it is at the correct context. Finally, if both counters are equal, we deduce that the policy is the same at the sender and receiver sites. Consequently, it is considered as granted and is executed.

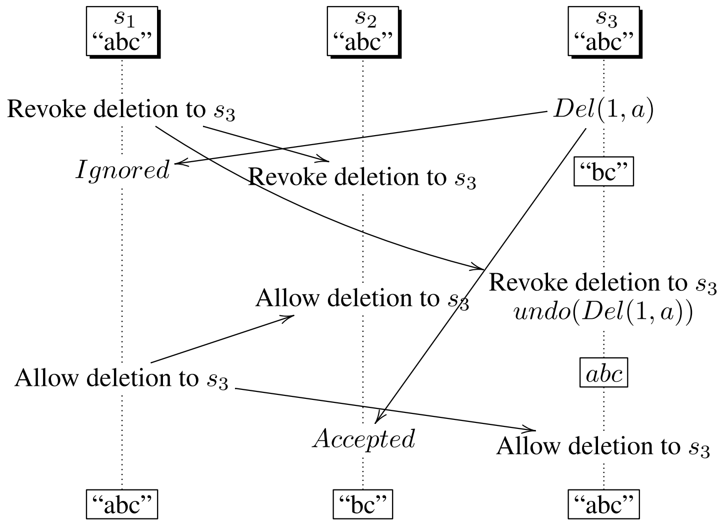


Figure 3.4: Necessity of administrative log.

3.2.1.4 Fourth scenario: Necessity of the "Validate" administrative operation

Using the above solution, the administrative operations will be totally ordered as only the administrator modifies the policy object and we associate to every version of this object a monotonically increasing counter.

Consider the scenario illustrated in Figure 3.5 where s_1 is initially authorized to insert any character. Suppose that s_1 performs the operation $Ins(1, x)$. When adm revokes the insertion right to s_1, he has already seen the effect of the s_1's insertion. If s_2 receives the revocation before the insertion, he will ignore this insertion as it is checked against the revocation. It is clear that the insertion may be delayed at s_2 either by the latency of the network or by a malicious user. We observe that there is a causal relation at adm between the insertion and the revocation. This causal relation is not respected at s_2 and the out-of-order execution of operations creates a security hole as s_2 rejects a legal insertion.

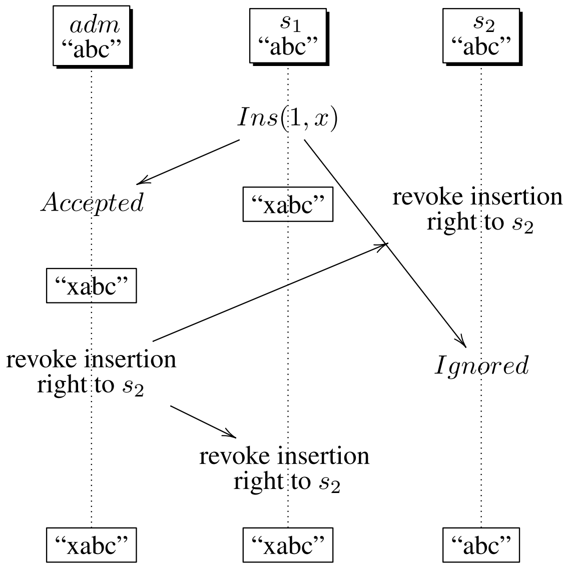

adm "abc"
s_1 "abc"
s_2 "abc"
$Ins(1, x)$
$Accepted$
"xabc"
revoke insertion right to s_2
"xabc"
revoke insertion right to s_2
$Ignored$
revoke insertion right to s_2
"xabc"
"xabc"
"abc"

Figure 3.5: Validation of cooperative operations

Before it is received at the administrator site, we consider a cooperative operation as tentative. So, our solution consists of an additional administrative operation that does not modify the policy object but increments the local counter. This operation validates each received and accepted cooperative operation at the administrator site. Consequently, every administrative operation is concurrent to all tentative operations. The policy modifications done after the validation of a cooperative operation are executed after this operation in all sites, as administrative operations are totally ordered.

In case of our scenario in Figure 3.5, the revocation received at s_2 will not be executed until the validation of the insertion is received. This avoids blocking legal operations and data divergence.

Accordingly, we define three kinds of cooperative operations in order to detect the concurrency between administrative and cooperative operations (see Section 3.2):

- *Valid*: a cooperative operation c is valid if it is generated by $Administrator(c)$ or by another user and validated by $Administrator(c)$; For instance, the operation $Ins(1, x)$ in Figure 3.5 is valid since it is accepted at the administrator site.

- *Invalid*: a cooperative operation c is invalid if it violates the policy or is undone by a concurrent administrative operation. For example, the operation $Del(1, a)$ generated at site s_3 (see Figure 3.4) is invalid since it was concurrent to the revocation originated by the owner site s_1. Thus, the operation is ignored at site s_1 and undone at site s_3.

- *Tentative*: a cooperative operation c is tentative if it is not generated by $Administrator(c)$. For instance, the operation $Ins(1, x)$ generated at site s_2 in Figure 3.5 is tentative since s_2 is not the owner of the character ”x“.

Let $\mathcal{V}$, $\mathcal{I}$ and $\mathcal{T}$ be the sets of valid, invalid and tentative operations respectively. Note that, the set $\mathcal{V}$ is sufficient to describe the set of all operations since the two others may be calculated from $\mathcal{V}$ and the cooperative Log[9].

3.2.2 Joint Issue

Consider the scenario illustrated in Figure 3.6 where we have initially a group of two collaborating sites s_1 and s_2. When s_1 grants the delete right on “a” to s_2, s_3 is not yet a member of the group. Suppose that s_3 joins the group just before s_2 receives the grant from s_1 and downloads the global policy from s_2. This means that the global policy replica owned by s_3 is different from that owned by s_1. Then s_2 receives the administrative operation from s_1. Consequently, the security shared object diverges. This leads to data divergence. In fact, when s_2 deletes the character “a”, this cooperative operation will be executed at s_1 and s_2 but rejected by s_3.

It should be pointed out that this issue does not concern only administrative operations but also cooperative ones. In fact, a user may loose a cooperative operation when he downloads the log and the state concurrently to the generation of a cooperative request by another user in the group.

To overcome this issue, there are two possible solutions: (1) The new user must request the policy object (P_i, L_i) from its possessor i and not from the nearest user in the P2P network as well as the shared data object. This solution is costly and to be discarded; (2) The new user must wait the time θ before requesting shared objects from the nearest user. The time θ corresponds to the maximum bound needed so that a message traverses the network from any sender to its receiver (see Figure 3.7).

[9] $H \setminus \mathcal{V} = \mathcal{T} \bigcup \mathcal{I}$, H being the cooperative log storing the cooperative operations.

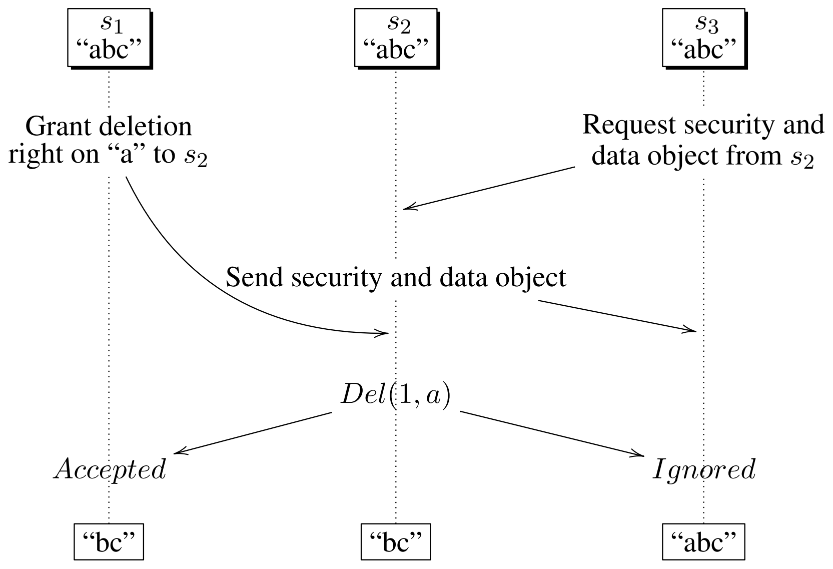

s_1 "abc"
s_2 "abc"
s_3 "abc"
Grant deletion right on "a" to s_2
Request security and data object from s_2
Send security and data object
$Del(1, a)$
$Accepted$
$Ignored$
"bc"
"bc"
"abc"

Figure 3.6: Divergence caused by a new user

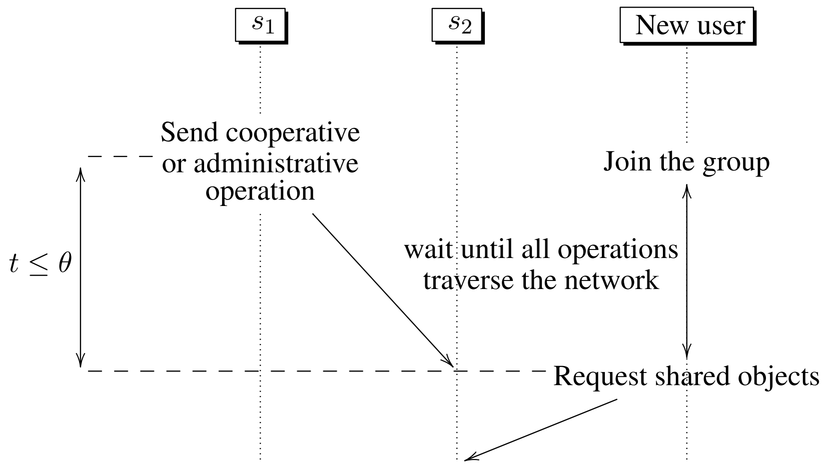

s_1
s_2
New user
Send cooperative or administrative operation
Join the group
$t \leq \theta$
wait until all operations traverse the network
Request shared objects

Figure 3.7: Time required by a new user to request shared objects.

In our model, we consider that θ is a parameter depending on message size and network configuration on witch the application is running. For instance, $\theta \leq 175\,ms$ for 1KB message propagated through HTTP connection on the P2P network JXTA 2.0 [41][10].

The same problem occurs when a user disconnects then reconnects to the group. In this situation two cases are possible. A disconnected user may be regarded as a new user to whom we assign a new identifier. This approach is avoided since policies must be updated each time a user disconnects and reconnects by replacing old identity by the new one. The second possibility, is to assign the same identity to the user. However, owner policies of other collaborators may change during his absence. Consequently, his local replica of the global policy must be updated in the same way as a new user who joins the group by downloading the new objects from the nearest peer after time θ.

[10]JXTA is the P2P platform that we used to implement our solution.

3.2.3 Remote Check and Unnecessary Undo

In this section, we focus on the unnecessary undo issue raised by the remote check. Indeed, tentative cooperative operations are undone when a concurrent restrictive administrative operation is found at the receiver site during the check process. However, a cooperative operation c may be revoked due to the concurrent deletion of an administrative operation granting c while c is legal with respect to the policy if there are other concurrent administrative operations granting c. This issue occurs due to the fact that we invalidate or undo tentative operations as soon as a concurrent restrictive administrative operation is found regardless the semantic of this restrictive operation.

In the following, we detail the scenarios of unnecessary undo operations and show how to exploit the administrative operations semantics to overcome this issue.

3.2.3.1 First case of useless undo

Consider the scenario presented in Figure 3.8 where two sites s_1 and s_2 edit the same shared data object. Suppose that s_1 and s_2 begin the collaboration with the initial owner policy of s_2, P_{s_2} containing $\{l_1 = \langle\{s_1\}, \{o\}, u, +\rangle; l_2 = \langle All, \{o\}, u, +\rangle\}$ where o is an object administrated by s_2. Authorisation l_1 only grants update right to user s_2 while l_2 grants update right to all users. We also assume that the authorization list is parsed from the top to the bottom.

Suppose now that user s_1 performs a tentative cooperative operation $c = Upd(p, o)$ to update the object o at position p. According to the ownership principle, s_2 performs the restrictive administrative operation $a_3 = DelAuth(0, l_1)$ in order to delete the permission l_1. Then, broadcasts the restrictive administrative operation a_3 that revokes the cooperative operation c.

When a_3 is received at site s_1, it leads to undoing c since it is tentative and revoked by a_3. At site s_2, the remote check of c returns a negative result, since a_3 is a revocation of c generated at a version greater than that of c.

Note that c is undone while authorized by the permission l_2 inserted on the top of the authorization list. The same issue would be faced if site s_2 inserts a permission $l_0 = \langle s_1, \{o\}, u, -\rangle$ at a position less or equal to l_2's position wherein c is invalidated even though it is legal with respect to the authorization list.

To overcome this problem, we propose to track administrative dependency for every cooperative operation. Tracking this dependency inside administrative logs enables us to correctly integrate remote cooperative operations since a cooperative operation depends on the permission allowing its execution. For instance in the scenario of Figure 3.8, c is validated thanks to l_2 and not l_1. Subsequently, the deletion of l_2 instead of l_1 should invalidate c. This administrative dependency should be updated when parsing the administrative log during a remote check process in order to take into account concurrent administrative operations executed on the local policy of the receiver site.

3.2.3.2 Second case of useless undo

Consider the scenario of Figure 3.9 where two sites s_1 and s_2 collaborate with the initial policy object such as $L_{s_2} = a_1$ with $a_1 = AddAuth(0, l_1)$ and $P_{s_2} = \{l_1 = \langle\{s_1\}, \{o\}, d, +\rangle\}$. We suppose that s_1 performs the cooperative operation $c = Del(p, o)$ verifying $Administrator(c) = s_2$ then broadcasts c to be executed at site s_2. After a while, s_2 generates two administrative operations a_2 and a_3 concurrent to c. The administrative operation a_2 inserts the permission $l_2 = \langle All, \{o\}, d, +\rangle$ on the top of l_1 which also grants c since l_2 is more general than l_1. However, the administrative operation $a_3 = DelAuth(0, l_1)$ deletes the permission l_1 on which depends c.

According to the first useless undo case, c should be undone at site s_1 and invalidated at site s_2 since its administrative dependency l_1 was deleted concurrently. However, the administrative operation

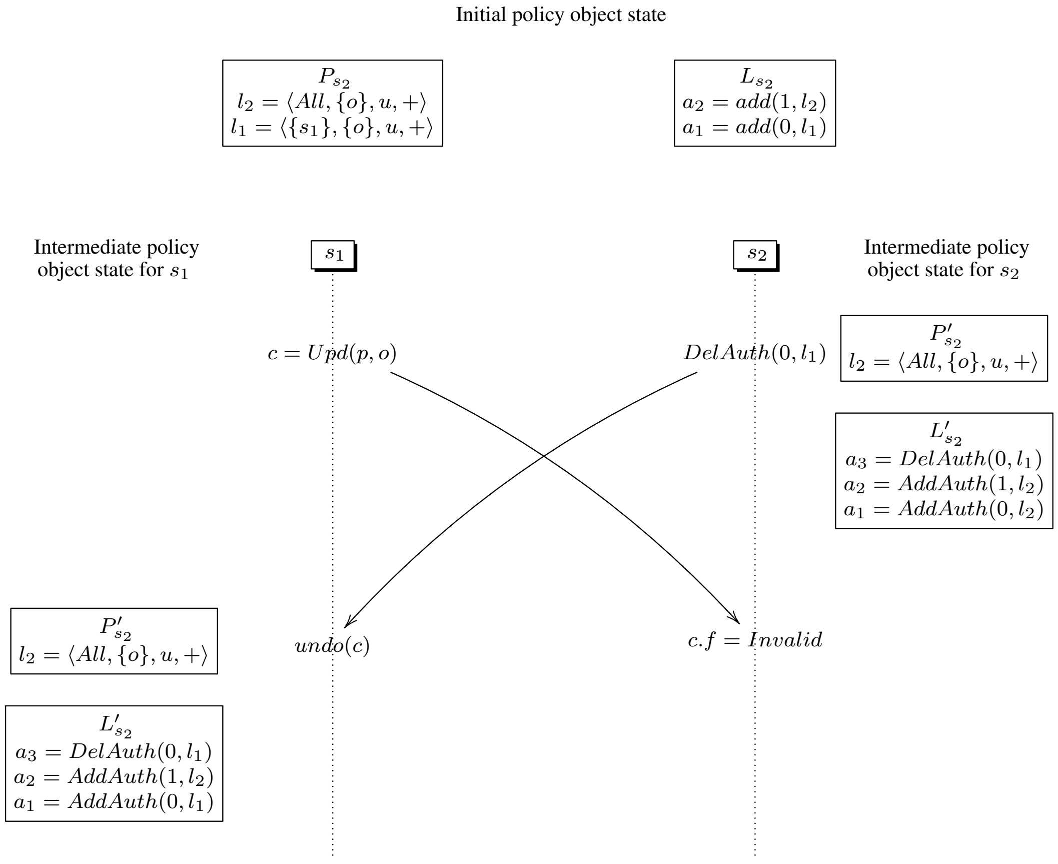


Figure 3.8: Useless Undo: First Case.

a_2 generated after a_1 also grants c. Consequently, the cooperative operation c must not be undone even though its dependency was deleted concurrently if there are at least one administrative operation authorizing its execution. Subsequently, when checking c remotely, we have to take into account the administrative operation a_2.

To overcome this issue, we propose to track l_2 in the administrative dependency of c during the remote check process. Consequently, the administrative dependency is seen as a list of integers referring to the positions of all authorizations granting the execution of a given cooperative and inserted on the top of its initial dependency. This dependency is referred to with $c.ad$.

Initially, the dependency set of an operation c contains the authorization on which it depends locally. Next, during the remote check procedure, the set of administrative dependencies is updated by adjusting, adding or removing positions according to the kind of the encountered administrative operation ($AddAuth()$ or $DelAuth()$). Accordingly, given a cooperative operation c, the administrative dependency of c represents an ordered set of integers where the minimum and the maximum refer to the first and last administrative operations granting c respectively.

To better illustrate how we update the dependency of a given cooperative operation, we consider the following example:

Example 3.2.1. *Consider a cooperative operation c and the following administrative log where c is initially granted by a_0 (e.g c is a delete operation). Then c is checked against the administrative log as*

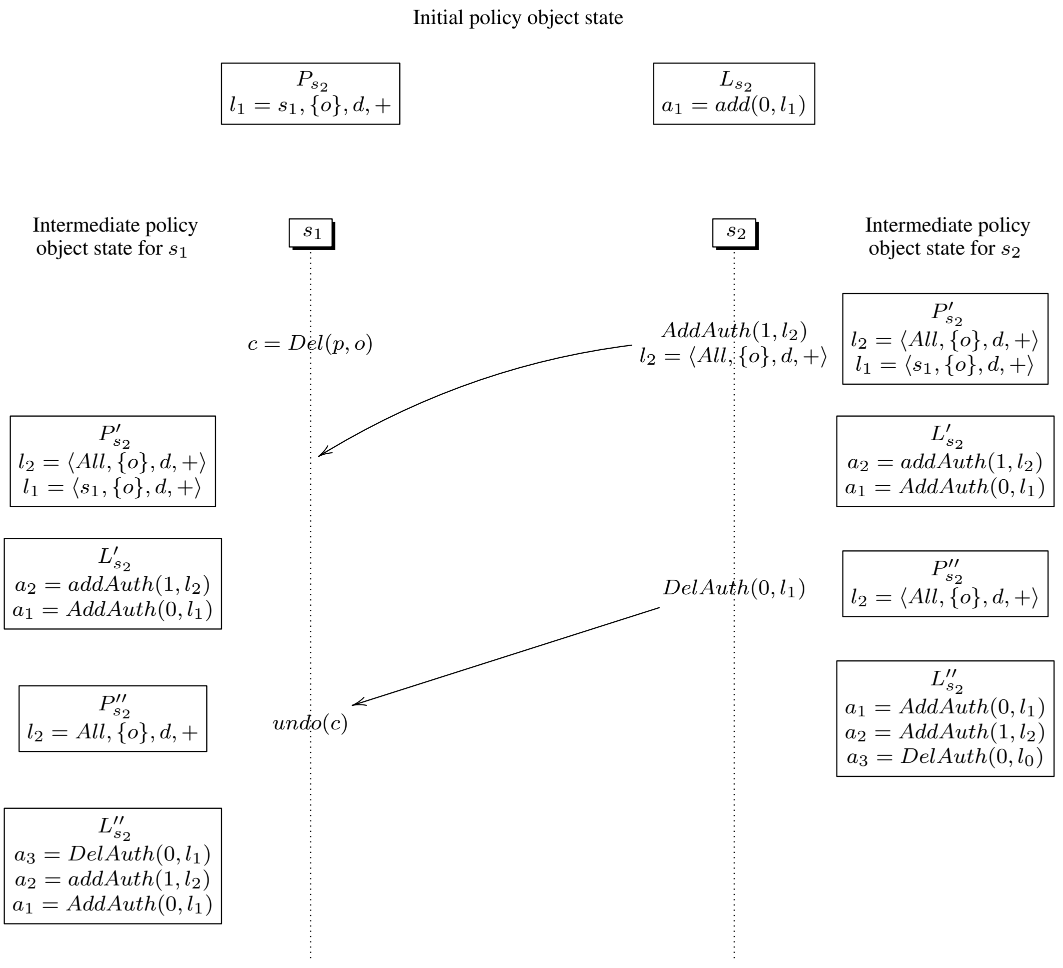


Figure 3.9: Useless Undo: case 2

follows: Initially $ad = [0]$*, it is updated when c is transformed against* a_1 *and becomes* $[0, 1]$ *(since also* a_1 *grants c). Finally, the deletion of the authorization at position 1 leads to* $ad = [0]$.

$$\begin{array}{lcl}
a_2 = DelAuth(1, \langle All, Doc, d, + \rangle) & \rightarrow & c.ad = [0] \\
a_1 = AddAuth(1, \langle All, Doc, \{d, u\}, + \rangle) & \rightarrow & c.ad = [0, 1] \\
a_0 = AddAuth(0, \langle All, Doc, d, + \rangle) & \rightarrow & c.ad = [0]
\end{array}$$

This transformation is applied both at the remote check and at the reception of a remote administrative operations. In the first case, a cooperative operation is transformed against concurrent administrative operations stored in the administrative log in order to update the administrative dependency. Thus, the remote check will proceed correctly by avoiding useless invalidation scenarios. Similarly, the reception of remote administrative operations updates the administrative dependency of tentative cooperative operations.

For instance, the cooperative operation c in Figure 3.9 must be updated at site s_1 at the reception of the administrative operation a_2 (*i.e* $c.ad = [0, 1]$). Similarly, the reception of a_3 has to update the administrative dependency of c by deleting 1 (*i.e* $ad = [0]$) since the new position of l_2 is 0 after the deletion of l_1.

Accordingly, a cooperative operation c is undone only when its dependency becomes empty (*i.e* $ad = \emptyset$) after being transformed against a concurrent administrative operation.

3.2.3.3 Third case of useless undo

The last scenario of useless undo is illustrated in Figure 3.10. In this scenario, we suppose that two sites s_1 and s_2 have the initial policy P_{s_2} allowing s_1 to delete all objects administrated by s_2. Suppose s_1 performs a cooperative operation $c = Del(p, o)$ where o is administrated by s_2.

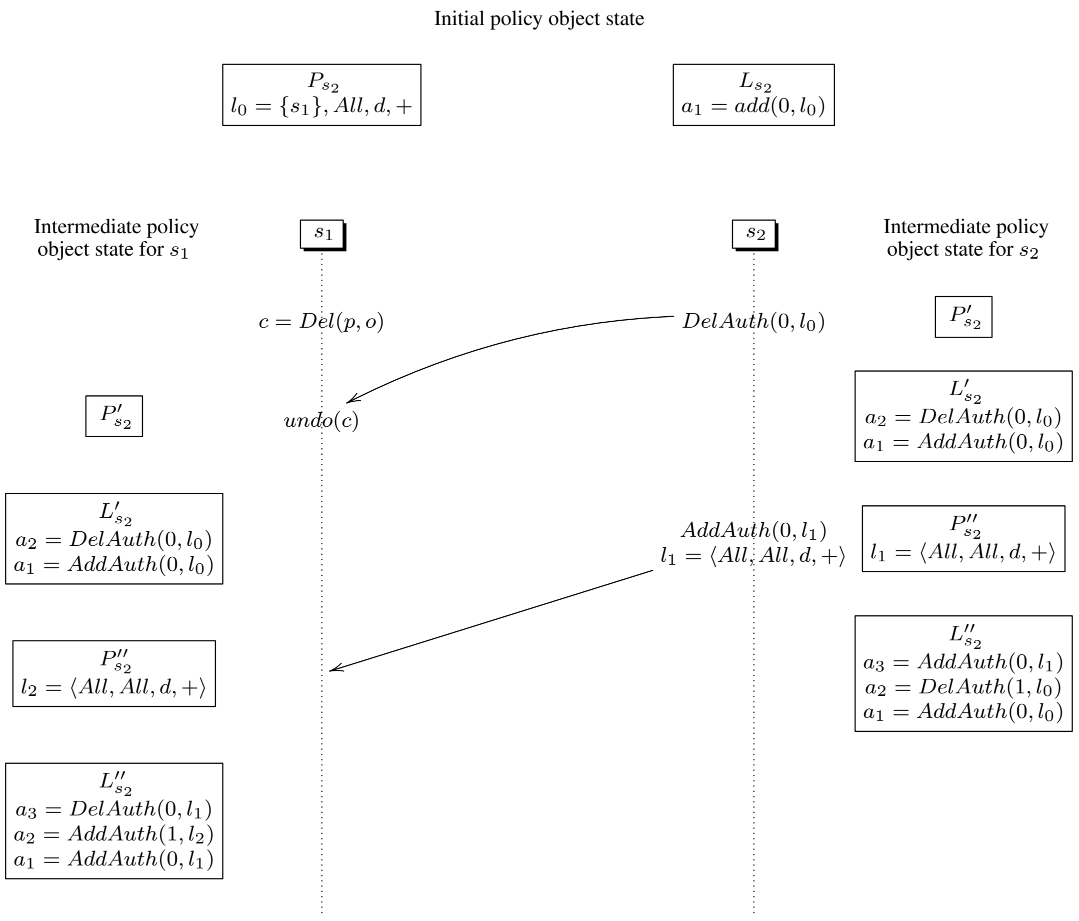


Figure 3.10: Useless Undo: case 3

Concurrently, s_2 deletes the permission l_0 on which depends c (by generating the administrative operation $a_2 = DelAuth(0, l_0)$ then adds the permission $l_1 = \langle All, All, d, + \rangle$ which is more general than l_0 and grants c. Hence, c will be undone due to the administrative operation $a_2 = DelAuth(1, l_0)$ generated by s_2 although the policy allows its execution (thanks to a_3).

For simplicity reasons, we chose to invalidate a cooperative operation if an invalidation is found before any new administrative operation granting this operation. Details and algorithms of the remote check are given in Chapter 4.

3.3 A Generic Security Model

In this Section, we present the generic architecture of our access control model that can be deployed atop any log-based collaborative editor. To illustrate the generic aspect of our access control model, we define an editing application as a structure of three different layers (see Figure 3.12):

1. The Coordination Layer denoted as CL;

2. The Access Control Layer denoted by ACL and;

3. The Generic Interface (GI) that connects CL and ACL;

The *Coordination Layer e.g.* OPTIC, SDT, ... (see Chapter 2) implements the coordination algorithm responsible of coordinating concurrent updates. This layer is seen as a black box and is independent of other layers. It ensures the concurrency control as illustrated in Figure 3.11, where every site first executes locally generated document update, then broadcasts them to other users. The receiver site, extracts remote operations from network and integrates them locally by the CL.

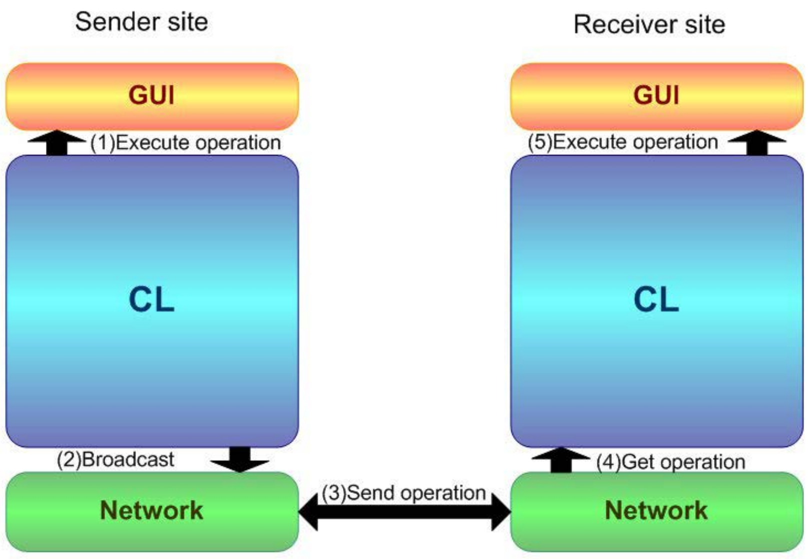


Figure 3.11: Underlaying Coordination Architecture.

Note that we only address CLs using logging mechanisms to store different updates performed on the shared document and that identify each created object with a unique identifier.

To build a generic access control layer (ACL) on the top of a given CL while preserving convergence, we need to well define the Generic Interface (GI) that connects both layers: ACL and CL. For the sake of our results, we define the following set of operations that handle the connections between the different layers mentioned before (see Figure 3.12):

- $Check(c)$ checks whether the cooperative operation c communicated to the generic interface (GI) is authorized by ACL or not;

- $Apply(c)$ launches the execution of the cooperative operation c on the actual data state;

- $Undo(c)$ to undo every cooperative operation c in the cooperative log *conflicting*[11] with the concurrent restrictive administrative operation a received by GI.

We describe below the flow of interactions between the three layers:

1. When a cooperative operation c is received from the network, it is firstly redirected to the GI instead of the CL in order to be checked against the policy object through the $Check(c)$ module.

2. If c is authorized by the ACL, then it is communicated to the module $Apply(c)$ of the generic interface GI. This module will ensure its execution on the current document state state. Otherwise, the operation is rejected from the ACL and then ignored by CL.

3. When a remote administrative operation a is received, it is communicated to the GI in order to be applied in the ACL so to modify the local copy of the shared policy object.

4. If there are cooperative operations concurrent with r_a and conflicting with it, then the generic layer will process the $Undo(c)$ module in order to undo all cooperative operations c that are no more legal after the reception of r_a. This step is crucial since it allows for the data convergence by restoring the correct state seen by the administrator at the moment of the generation of the administrative operation a.

5. When a cooperative operation c is locally generated, it is first executed then redirected to the GI to check if it is authorized by the ACL or not. If not, it is undone thanks to the $Undo(c)$ module[12]. Otherwise its effect remains on the document state. Finally, the operation c is broadcast through the network.

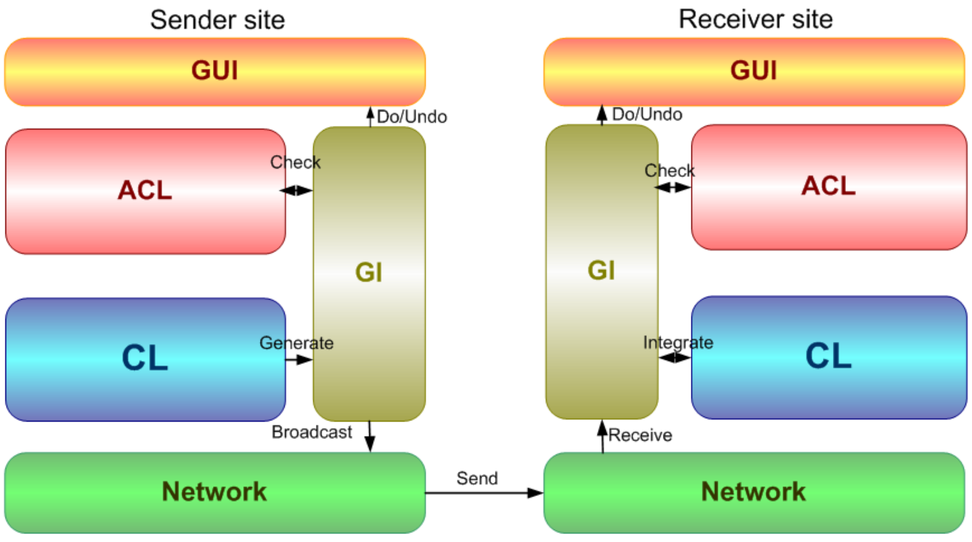


Figure 3.12: Different layers of secure RCE model.

[11]We say that a cooperative operation c is conflicting with an administrative operation when c is concurrent with a, $Administrator(c)$ is the issuer of a and c violates the rights restricted by a (a is restrictive).

[12]We suppose that a user is aware of his/her capabilities to limit the repetition of a do/undo mechanism

3.4 Distributed Garbage Collection for Administrative Logs

Our model requires every site s to maintain an administrative log containing administrative operations of which site s is informed, *i.e.* administrative operations generated by s or originated from other sites and then communicated to s. As discussed in Section 3.2, the site may need to reconsider the administrative operations it has received in order to properly check a remote cooperative operation upon reception of out of order administrative ones. Furthermore, shared objects may not have been received by other users such as new users or disconnected ones; hence every site must remember the operations it has received so to communicate them to other sites.

Accordingly, enforcing security requires to store administrative operations in a buffer history, as it is necessary to track policy updates in order to achieve convergence of the shared data. Unfortunately, with the continuous increasing of log size, the system performance may be degraded which is inappropriate for low storage capacity devices and may not be useful after producing many versions of the access control policy. Moreover, storage capacity is not infinite as memory always has limited size. Secondly, the motivation for maintaining a log at each site is that a remote administrative operation must be executed at the same security context on all receiver sites as we have already shown in Section 3.2. However, not all operations received by a site must be kept in its log. In particular, an administrative operation can be safely deleted from a site's log if it is already received in all other sites.

In this section, we focus on garbage collection as a feature for our access control model. We discuss issues raised by garbage collection mechanism in collaborative editing context. Finally, we devise a distributed garbage collection scheme to clean administrative logs.

3.4.1 Garbage Collection Issues

Administrative logs are identical at all sites, since all administrative operations are executed in the same order dictated by the administrator[13].

Let GARBAGE_ADM_LOG(s,v) be the request allowing for initiating a garbage process for administrative logs where s is the owner site's identity and v is the policy version. Such a message is causally ready at a site s_1 if $v = v' + 1$, v' being the version of the owner policy copy P_s of site s_1.

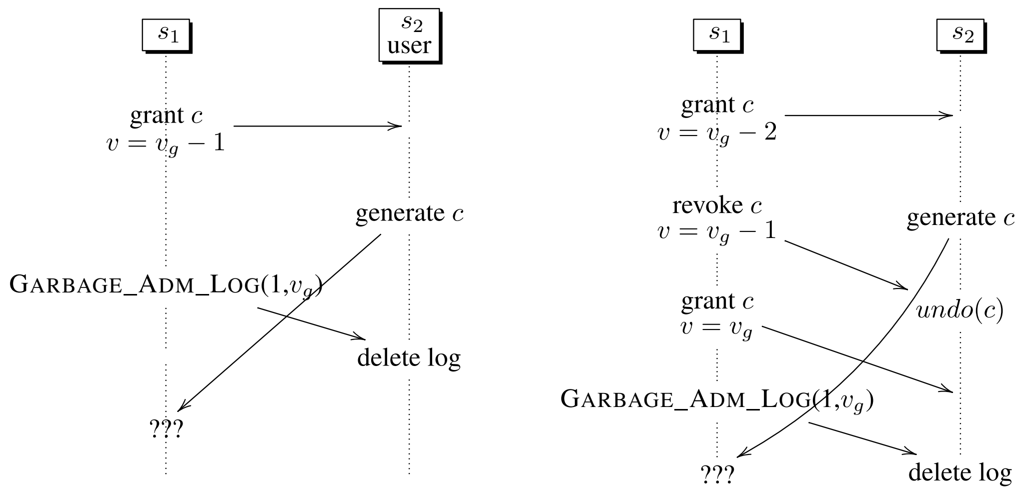
The out of order execution of a garbage request and a cooperative operations inevitably leads to divergence of the shared data object since cooperative operations are checked against the administrative log. To illustrate this issue, consider the scenario of Figure 3.13 in which we give the impact of deleting the administrative log of a site s_1 on the remote check of cooperative operations received after the log removal.

In Figure 3.13.(a), two sites s_1 and s_2 begin the collaboration with the same document and policy objects. Let c be a cooperative operation originated by s_2. Suppose that $Administrator(c) = s_1$ and that c is granted at version $v_g - 1$. After he reaches the version v_g, site s_1 generates the request GARBAGE_ADM_REQUEST(s_1, v_g) in order to garbage the administrative log L_1. We also assume that the garbage request is immediately executed at the originating site s_1. Consequently, the local copy of the access control policy becomes empty. Obviously, checking the remote cooperative operation c already executed at site s_2 presents a big issue.

The same issue is encountered for illegal cooperative operations as illustrated in Figure 3.13.(b). In this case, we suppose that s_1 grants r_c at version $v_g - 2$ but revokes later c from s_2. Then s_1 generates again an administrative operation that grants c. Finally, s_1 performs GARBAGE_ADM_LOG(s_1, v_g) in order to garbage the administrative log and broadcast the garbage request so to be executed at site s_2.

The cooperative operation c is invalid because of the revocation performed at version $v_g - 1$. In fact, once the revocation is received at site s_2, c is undone since it is a tentative operation concurrent to the

[13]This order reflects the policy version.



(a) Case of a legal coop. operation. (b) Case of an illegal coop. operation.

Figure 3.13: Interleaving Between a Garbage Request and Cooperative Operation

revocation performed by the administrator s_1. However, when c arrives at site s_1, the log is empty so we can not correctly check this remote cooperative operation. Unfortunately, such a situation inevitably leads to divergence of the shared document as well as policy violation (if c is executed at site s_1).

To overcome this problem, we have to correctly address the following two issues:

1. What should be version of the policy after processing a GARBAGE_ADM_LOG?
2. How to check remote cooperative operations that are concurrent to a GARBAGE_ADM_LOG request?

In an attempt to answer these questions, we focus on policy version and show how to avoid the divergence caused by the interleaving between garbage request and cooperative operations.

Policy Version. Restarting the version 0 after cleaning logs is to be discarded since it does not allow to remember previous garbage requests. Consequently, a delayed remote cooperative operation concurrent to a GARBAGE_ADM_LOG request may be executed at given site s that has processed a garbage collection as if it was not the case. For instance, consider a cooperative operation c with version v. Suppose that c is delayed by a malicious user while $Administrator(c)$ performs a GARBAGE_ADM_LOG request and restarts the version of its owner policy to 0. Then $Administrator(c)$ performs many administrative requests until he reaches the version v. Clearly, when $Administrator(c)$ receives c, he will integrate it locally as if it was generated after the garbage. Consequently, the remote check may produce an erroneous result that violates the policy and leads to divergence.

To overcome this problem, we define the policy version v as a couple (v_g, v_a), where v_g represents the counter of garbage requests *i.e* v_g is incremented each time a GARBAGE_ADM_LOG request is generated while v_a represents the counter of administrative requests *i.e* it is incremented when an administrative operation is generated and reset to 0 each time v_g is incremented. For example, the version $v = (5, 2)$ refers to an administrative log containing two administrative operations after the fifth garbage of the administrative log. We use the dot notation to denote each parameter of the version ($v = (v_g, v_a)$).

Accordingly, a cooperative operation administrated by the site s having the version $v_s = (v_g, v_a)$ is ready with respect to the policy if its version number v verifies (i) $v.v_g = v_s.v_g$ *i.e* it is generated after the same number of garbage requests. (ii) $v.v_a \geq v_s.v_a$ *i.e* s has reached the policy context on which c was generated. Thus, we are able to correctly check a remote cooperative operation against the administrative log even if it was removed by a GARBAGE_ADM_LOG request.

Checking Remote Cooperative Requests Concurrent to a Garbage Request. We have already shown in Figure 3.13, that it is impossible to correctly integrate remote cooperative requests concurrent to a garbage request. To overcome this issue, our solution consists in undoing cooperative operations that are concurrent to a GARBAGE_ADM_LOG request even if they are legal operations. This highlights the optimistic aspect of our model in the sense that a user may introduce temporary divergence situations while the system ensures permanently the convergence of the shared document.

The causal relation between garbage and cooperative requests is defined as follows:

- Every cooperative operation c with version v received at a site s with version v_s such that $v.v_g < v_s.v_g$ is considered as invalid since it was generated concurrently to a garbage request. Note that c is a tentative cooperative operation otherwise it would be validated before the execution of the garbage request thanks to the administrative operation $Validate(c)$.
- When a GARBAGE_ADM_LOG request is received at a remote site, it is executed after undoing all tentative operations from the cooperative log.

For instance, in Figure 3.13, c must be undone before the execution of GARBAGE_ADM_LOG(s_1,v_g) which leads to convergence. Thus, a garbage request behaves in the same manner as a restrictive administrative operation.

Garbage Collection Protocol. To remove administrative logs, we follow these steps:

1. The site s generates the administrative operation GARBAGE_ADM_LOG(s,v) and broadcasts it to all users. This request contains the identity of the issuer as well as its version.
2. When a remote site s' receives GARBAGE_ADM_LOG(s,v), it stores it until it is causally ready. Once it is causally ready, s' undoes all tentative operations then removes the administrative log and updates the policy version. Thus the counter of administrative operations is reset to zero and that of garbage administrative logs is incremented.
3. Each cooperative operation received at site s or at any other collaborating site after the garbage request is ignored.

3.5 Conclusion

In this chapter, we have proposed a generic and decentralized framework for controlling access in distributed collaborative editors. We designed our solution in a replicated and optimistic fashion so that users are able to collaborate in editing the shared access control policy. Our model relies on the optimistic aspect of the collaboration in the sense that users may violate temporarily the policy. Illegal operations are undone in order to restore the correct state.

Chapter 4

Concurrency Control Algorithms and Correctness Proof

Contents

This chapter presents a protocol for optimistic access control model proposed for RCE that enables us to address issues such as security holes and data divergence presented in Chapter 3.

The outline of this chapter is as follows: In Section 4.1, we present our algorithms as well as an illustrative example. Section 4.2 gives the correctness proof of our protocol.

4.1 Concurrency Control Algorithms

In our collaboration protocol, we consider that a user maintains two copies: the shared document and its access policy object. Even though our access control model seems simple, we have shown in Section 3.2 that the policy enforcement is very tricky.

In the following, we formally present the different components of our algorithm as well as its asymptotic time complexity.

4.1.1 Cooperative Requests

A cooperative operation c is associated to a *cooperative request* $r_c = (s, c, v, f, ad)$ where:

1. s is the identity of the generator site.
2. c is the cooperative operation of r_c referred to with $r_c.c$.
3. v is the number version of the policy copy referred to with $r_c.v$.
4. f is the kind of cooperative operation denoted by $r_c.f$ and takes values “Tentative”, “Valid” and “Invalid”.
5. ad is the administrative dependency of c and is referred to with $r_c.ad$.

The object altered by a given request r_c is referred to as $r_c.o$. Furthermore, $r_c.t$ reflects the request type *e.g* considering the set of operations insert, delete and update $r_c.t = i$ if c is an insert, $r_c.t = d$ if it is a delete and $r_c.t = u$ if it is an update. Let $\mathcal{C}_r$ be the set of all cooperative requests.

We consider coordination models allowing for concurrent execution of cooperative operations. Such systems systematically rely on logging cooperative operations. Subsequently, every site keeps a local log for cooperative operations also called *cooperative log* and referred to with letter H.

Definition 4.1.1 (Cooperative log)**.**

Each site s maintains a buffer H that stores all cooperative requests. A cooperative log is represented by the sequence of cooperative requests $r_{c1} \cdot r_{c2} \cdot \ldots \cdot r_{cn}$ executed at site s.

4.1.2 Administrative Requests

Every administrative operation a is associated with an *administrative request* r_a where r_a is the triple $r_a = (s, a, v)$ such that:

1. s is the identity of the site referred to with $r_a.s$;
2. a is the administrative operation denoted by $r_a.a$;
3. v is the version number of the owner policy P_s denoted by $r_a.v$ and is the couple (v_g, v_a) where v_g is the number of garbage collections while v_a is the number of administrative requests.

Moreover, the notation $r_a.t$ refers to the type of a administrative operation $r_a.a$ associated with r_a *e.g.* $r_a.t = AddAuth$ if $r_a.a = AddAuth(p, l)$ and so on. Finally, $r_a.l$ refers to the authorization inserted or deleted by the administrative request r_a if $r_a.t = AddAuth \vee r_a.t = DelAuth$ and $r_a.c$ refers to the identifier of the cooperative request validated by r_a if $r_a.t = Validate$.

Let $\mathcal{A}_r$ be the set of all administrative requests. Note that administrative requests are stored in a log called *administrative log* (see Section 3.2, Chapter 3).

Definition 4.1.2 (Administrative Log)**.**

An *administrative log* L is a buffer that stores all administrative requests.

In order to enforce access control, we need to define both $Grant$ and $Revoke$ predicates that allow to deduce whether an administrative request grants or revokes a given cooperative request.

Both predicates requires a matching rule, to see if the administrative request really concerns the cooperative request or not. This matching rule is given in Definition 4.1.3.

Definition 4.1.3 (Matching rule)**.**

Given an authorization $l_i = \langle S_i, O_i, R_i, \omega_i \rangle$ and a cooperative request r_c. The predicate $Matches(l, r_c)$ returns true iff:

$$(r_c.s \in S_i \vee S_i = \mathit{All}) \wedge (r_c.o \in O_i \vee O_i = \mathit{Doc}) \wedge (r_c.t \in R_i \vee R_i = \mathit{All})$$

Next, we define the boolean predicate $Grant(r_a, r_c)$ and its dual boolean predicate $Revoke(r_a, r_c)$ as follows:

Definition 4.1.4 (Grant predicate)**.**

Given an administrative request r_a and a cooperative request r_c, the boolean function $Grant(r_a, r_c) : \mathcal{A}_r \times \mathcal{C}_r \mapsto \{true, false\}$ returns true iff

$$r_a.a \text{ is not restrictive and } Matches(r_a.l, r_c) = true.$$

For instance, consider the positive authorization $l = \langle \mathit{All}, \mathit{Doc}, \{i, d\}, + \rangle$ and its associated administrative operation $a = \mathit{AddAuth}(p, l)$ where p is the top of the authorization list P ($p = |P|$). Let r_a be the administrative request corresponding to a. In this case, for every cooperative insert or delete request r_c (*i.e* $r_c.t = i \vee r_c.c = d$), we have $Grant(r_a, r_c) = true$.

In the following, we present the revoke predicate, that takes true value if an administrative request revokes a cooperative request.

Definition 4.1.5 (Revoke predicate)**.**

Given an administrative request r_a and a cooperative request r_c, the boolean function $Revoke(r_a, r_c) : \mathcal{A}_r \times \mathcal{C}_r \mapsto \{true, false\}$ returns true iff

$$r_a.a \text{ is restrictive and } Matches(r_a.l, r_c) = true.$$

For better understanding, let $l_1 = \langle \{s\}, \mathit{Doc}, \{i, d\}, + \rangle$ deleted by the administrative operation $a = \mathit{DelAuth}(p, l_1)$ where p is the position of l_1 in the corresponding authorization list P. Consider a cooperative request r_c verifying $r_c.t = d \vee r_c.t = i$ and administrative request r_a such that $r_a.a = a$, thus $Revoke(r_a, r_c) = true$. Similarly, consider $l_2 = \langle \{s\}, \mathit{Doc}, \{i, d\}, - \rangle$ and the administrative request r_{a1} that adds l_2 on the top of P. Consequently, we have $Revoke(r_a, r_c) = true$.

4.1.3 Causality Between Cooperative and Administrative Requests

A cooperative request is causally ready if it has the same garbage as the receiver site and a policy version greater than the policy version of the receiver site. We define the causality of a cooperative operation in the following:

Definition 4.1.6 (Causality between cooperative and administrative operations)**.**

Let r_c and r_a be two cooperative and administrative requests respectively such that $r_a.s = Administrator(r_c)$. The cooperative request r_c depends causally on r_a iff $r_c.v_g = r_a.v_g \wedge r_c.a > r_a.v_a$, *i.e.* r_c has already seen the effect of r_a. If r_c is tentative then it is concurrent to r_a, *i.e.* the administrator has not yet seen the effect of r_c when it generates r_a.

A remote cooperative request is executed at the receiver site only when it is causally ready. We define the boolean function $Ready(r_c)$ as follows:

Definition 4.1.7.

Given a cooperative request r_c received by a site s, the boolean function $Ready(r_c) : \mathcal{C}_r \mapsto \{true, false\}$ returns true iff r_c is causally ready according to the Definition 4.1.6 (*all administrative requests on which* r_c *depends causally are executed on* s).

Similarly, remote administrative requests are buffered and executed only when they are causally ready. By abuse of notation, we use the same function used for cooperative requests.

Definition 4.1.8.

Given an administrative request r_a with version $v = (v_g, v_a)$ received by a site s such that the owner policy of $r_a.s$ has the local version $v_r = (v_{gr}, v_{ar})$. Then r_a is causally ready iff (i) $r_a.t \neq Validate \wedge r_a.v_g = v_{gr} \wedge r_a.v_a = v_{ar} + 1$ or (ii) $r_a.t = Validate \wedge r_a.v_g = v_{gr} \wedge r_a.v_a = v_{ar} + 1 \wedge r_a.c \in H$. Consequently, boolean function $Ready(r_a) : \mathcal{A}_r \mapsto \{true, false\}$ returns true.

4.1.4 Control Procedures

In a secure collaborative editor, a group consists of N user sites (where N is variable in time) starting a collaboration session from the same initial shared data state. Each site stores all cooperative requests in a cooperative log and all administrative requests in an administrative log.

Our concurrency control procedure is given in Algorithm 2. In the following, we detail the main steps of this algorithm.

Joining the group. Each user must join an existing group. Even the group creator, creates the group then joins it. In Algorithm 2, the first step is the JOIN procedure. We define two different states for each user: $passive$ and $active$ state. A user is active when he is able to begin the collaboration with other users. Otherwise he is passive. First, the user waits until receiving both data and security objects. He has the initial state passive until he receives both data and security objects from the nearest peer on the collaborating group (of course if there is already at least a member in the group).

In the following, we illustrate the generation and reception of administrative requests.

Generation of local cooperative request. In Algorithm 3, when a cooperative operation c is locally generated at a site s, we begin by forming $r_c = (s, c, f, v_s, [])$ associated with it where v_s is the version of the policy administrating r_c. Once the request r_c is formed, it is considered either as valid when the

```
 1: Main:
 2: Join
 3: Initialization of the coordination layer
 4: Initialization of the security layer {see Algorithm 8}
 5: while not aborted do
 6:    if there is a generated cooperative operation $c$ then
 7:       Generate_Coop_Request($c$)
 8:    else
 9:       if there is a generated administrative operation $a$ then
10:          Generate_Admin_Request($a$)
11:       else
12:          Receive_Remote_Request
13:          Integrate_Remote_Request
14:       end if
15:    end if
16: end while
17: Receive_Remote_Request:
18: if there is a request $r$ from a network then
19:    $F \leftarrow F + r$
20: end if
21: Integrate_Remote_Request:
22: if $\exists\, r \in F \mid Ready(r) = true$ then
23:    {see Definition 4.1.6 and 4.1.8}
24:    $F \leftarrow F - r$
25:    if $r \in \mathcal{C}_r$ then
26:       Receive_Coop_Request($r$) {see Algorithm 4}
27:    else
28:       if $r \in \mathcal{A}_r$ then
29:          Receive_Admin_Request($r$) {see Algorithm 9 and 10}
30:       else
31:          $r$ is a garbage message
32:          Receive_GC_Message($r$) {see Algorithm 11}
33:       end if
34:    end if
35: end if
36: Join:
37: if $N > 1$ then
38:    $state \leftarrow passive$
39:    wait until all requests traverse the network
40:    request shared data and security object
41:    wait until receiving shared data object
42:    for all cooperative requests $r_c$ in $H$ do
43:       Receive_Coop_Request($r_c$)
44:    end for
45: end if
46: $state \leftarrow active$
```

Algorithm 2: Concurrency Control Algorithm

issuer s is $Administrator(r_c)$ (note that $Administrator(r_c)$ is simply the group's administrator in the case of a single-administrator approach) or otherwise as tentative. The request r_c is immediately executed on its generation state (*i.e.* $Apply(r_c)$ computes the resulting state when executing operation c on the document state) then checked to verify whether it is authorized or not in which case it is undone. The operation is checked against the local copy of the policy (*i.e.* using boolean function CHECK_LOCAL detailed in Algorithm 5). We choose to execute the operation immediately before checking it against the local copy of the policy for generality reasons. Indeed, all existing coordination algorithms execute a locally generated operation immediately. This allows our model to be generic and to be deployed on the top of any coordination algorithm without any modification of the coordination layer. However, we suppose that users are aware of their rights and consequently generating illegal operations is not frequent.

Reception of remote cooperative request. Each site has the use of queue F to store the remote requests coming from other sites (line 19 of Algorithm 2). Request r_c generated on site s_i is added to F when it arrives at site s_j (with $s_i \neq s_j$). In Algorithm 4, to preserve the causality dependency with respect to precedent administrative requests and precedent cooperative requests, r_c is extracted from the queue when it is *causally-ready* (see Definition 4.1.6). If there is no concurrent garbage collection for the administrative log we check the cooperative request. Using function CHECK_REMOTE(r_c,$L_{Administrator(r_c)}$), r_c is checked against the administrative log $L_{Administrator(r_c)}$ (note that this log is the log of the administrator of the group in the case of a single-administrator approach) to verify whether or not r_c is granted. If r_c is executed by $Administrator(r_c)$ then it is validated and a validation $Validate(r_c)$ request is generated in order to broadcast it to other sites. Note that if a cooperative request has the same version as the receiver site, it is valid since it is has the same context of the receiver site. In this case there is no need to check the operation.

```
1: GENERATE_COOP_REQUEST($c$):
2: $r_c \leftarrow (s_i, c, f, v_{s_i}, [])$
3: $s_j \leftarrow Administrator(r_c)$
4: if $s_j = s_i$ then
5:     $r_c.f \leftarrow Valid$
6:     $\mathcal{V} \leftarrow \mathcal{V} + r_c$
7: else
8:     $r_c.f \leftarrow Tentative$
9: end if
10: Apply($r_c$)
11: if CHECK_LOCAL($r_c$, $P_j$) then
12:     {see Algorithm 5}
13:     broadcast $r_c$
14: else
15:     Undo($r_c$)
16: end if
```

Algorithm 3: Generation of Cooperative Request at the Site s_i

4.1.5 Check Procedures

In this Section, we illustrate how to check local and remote cooperative requests.

In order to avoid useless undo cases (see Chapter 3, section 3.2.3), we define the transformation function $IT^a(r_c,r_a)$ presented in Algorithm 7 that updates the administrative dependency of a cooperative request when checked against the local copy of the policy (see algorithm 7) or at the reception of concurrent administrative operations.

```
1: RECEIVE_COOP_REQUEST($r_c$):
2: $s_j \leftarrow Administrator(r_c)$
3: if ($v_j.v_g = r_c.v_g$ and CHECK_REMOTE($r_c$,$L_{s_j}$))(see Algorithm 6) then
4:     Apply($r_c$)
5:     if $s_i = s_j$ then
6:         $\mathcal{V} \leftarrow \mathcal{V} + r_c$
7:         $a \leftarrow Validate(r_c)$
8:         GENERATE_ADMIN_REQUEST($a$)
9:     else
10:        $\mathcal{T} \leftarrow \mathcal{T} + r_c$
11:    end if
12: else
13:    $\mathcal{I} \leftarrow \mathcal{I} + r_c$
14: end if
```

Algorithm 4: Reception of a Remote Cooperative request at the Site s_i

The Algorithms 5 and 6, show how to check the legacy of local and remote cooperative requests with respect to the access control layer.

Check of locally generated cooperative request. Algorithms 5 shows that a cooperative request r_c generated locally is checked against the authorization list (the owner authorization list in the case of multi-administrator approach). The authorization list is parsed from the top to the end in order to check r_c against each authorization l verifying $Matches(l, r_c) = true$. If l grants r_c then the check returns true, the cooperative request is authorized and $rc.ad$ takes the position of l. If a revocation is found the cooperative request r_c is rejected since the result of the check returns false. Similarly, the local check function returns false if the end of the policy is reached.

Check of remote cooperative request. Algorithm 6 shows how we can deduce if a received operation is still authorized or not according to the administrative log owned by $Administrator(r_c)$. When receiving a cooperative request r_c with the version $r_c.v$ and administrative dependency $c.ad$ at a site s_i, we check r_c against the appropriate administrative log L from the index $r_c.v_a$ to guess whether it was invalidated concurrently or not.

According to Section 3.2, there are some cases where the cooperative request r_c is legal even though a revocation is performed concurrently. The remote check proceeds as follows:

1. The administrative log is parsed from $r_c.v_a$ in order to find concurrent revocations.

2. If an administrative request verifying $Revoke(r_a, r_c) = true$ is encountered such that $r_a.p > Max(r_c.ad)$, then r_c is considered as an invalid operation and is not executed. This is achieved thanks to the $Invalidate()$ predicate which will be used to determine if a cooperative request conflicts with an administrative request in order to correctly check remote cooperative requests against the administrative log. Formally, the boolean function $Invalidate(r_a, r_c) : \mathcal{A}_r \times \mathcal{C}_r \mapsto \{true, false\}$ takes value true iff $Administrator(r_c) = r_a.s \wedge r_c.f = Tentative \wedge Max(r_c.ad) < p \wedge Revoke(r_a, r_c) \wedge r_a.t = AddAuth$. The $Invalidate()$ predicate allows to decide whether a cooperative operation is invalid during a remote check algorithm or not. It is also used to decide whether a tentative cooperative request must be undone or not when receiving a concurrent remote administrative operation.

3. If an administrative request r_a verifying $Grant(r_a, r_c) = true$ is encountered such that $r_a.p > Max(r_c.ad)$, we store the new administrative dependency and continue the check (line 5 of algo-

rithm 7). Otherwise, if a $DelAuth$ request is found such that the request belongs to the administrative dependencies set, $r_c.ad$ is updated by deleting this dependency (line 8 Algorithm 7). In both cases, if $r_a.p < Maxr_c.ad$, we update the administrative dependencies set in order to get correct positions (line 15 Algorithm 7).

4. Finally, if $c.ad \neq \emptyset$, we deduce that the cooperative operation r_c is still legal with respect to the policy and is executed at the receiver site.

```
 1: CHECK_LOCAL($r_c, P_{s_j}$): Boolean
 2: $r_c.v_a \leftarrow v_{s_j}$
 3: for ($k = |P_{s_j}|$); $k >= 0$ ; $k - -$) do
 4:     $l \leftarrow P_{s_j}[k]$
 5:     if $Grant(l, r_c)$ then
 6:         $r_c.ad \leftarrow [k]$
 7:         return true
 8:     else
 9:         if $Revoke(l, r_c)$ then
10:             return false
11:         end if
12:     end if
13: end for
14: return false
```

Algorithm 5: Local Check Against the Authorization List

```
 1: CHECK_REMOTE($r_c$,$L$): Boolean
 2: $decision \leftarrow true$
 3: for (int $k = r_c.v_a + 1$ ; $k < |L|$; $k + +$) do
 4:     $r_c \leftarrow IT^a(r_c, L[k])$ {see Algorithm 7}
 5:     if $r_c.ad = \emptyset$ or $r_c.ad \neq \emptyset$ and $Invalidate(L[k], r_c)$ then
 6:         $decision \leftarrow false$
 7:         return decision
 8:     end if
 9: end for
10: return decision
```

Algorithm 6: Remote Check Algorithm Against Administrative Log L

4.1.6 Administrative Procedures

In this section , we give the algorithms for generating and receiving administrative requests as well as the garbage procedure allowing for cleaning the administrative log.

4.1.6.1 Single-Administrator

In case of single-administrator approach, each group consists of one administrator and several users. Only the administrator can specify authorizations in the policy object. It can also modify directly the shared documents. As for users, they only modify the shared document with respect to the local policy object.

```
$IT^a\,(r_c, r_a) : r'_c$
$r'_c \leftarrow r_c$
$a \leftarrow r_a.a$
if $a.p > Max(r'_c.ad) \wedge Grant(r_a, r_c)$ then
    $r_c.ad \leftarrow r_c.ad \cup \{a.p\}$
else
    if $a.p \in r_c.ad \wedge r_a.t = DelAuth$ then
        $r_c.ad \leftarrow r_c.ad \setminus \{a.p\}$
    end if
    offset $\leftarrow (r_a.t = AddAuth)?1 : -1$
    for $(d \in r_c.ad \,|\, d \geq r_a.p)$ do
        if $(d = a.p)$ then
            $d \leftarrow d + 1$
        else
            $d \leftarrow d +$ offset
        end if
    end for
end if
return $r'_c$
```

Algorithm 7: Transformation algorithm of a remote cooperative operation against administrative operation at the i-th site

In the following we detail the initialization, generation and reception of administrative requests procedures.

Initialization. At the initialization step (see Algorithm 8), i is set to the identifier of the site. The policy as well as administrative log are empty. The policy counter is set to 0. Finally, the coordination component is initialized.

```
INITIALIZATION OF THE SECURITY LAYER:
$s \leftarrow$ Identification of local site
if single-administrator approach then
    $v \leftarrow 0$ {Initial Version of the policy copy of the site}
    $(P \leftarrow [], L \leftarrow [])$ { initial policy and administrative log }
else
    {multi-administrator approach}
    $v_s \leftarrow 0$ {Initial Version of the owner policy of the site}
    $(P_s \leftarrow [], L_s \leftarrow [])$ {Owner policy}
    $P_g \leftarrow (P_s, L_s)$ {Global policy }
end if
$F \leftarrow []$ {Queue buffer}
```

Algorithm 8: Initialization

Generation and Reception of administrative request. In Algorithm 9, the policy copy maintains a version counter v that is incremented by the administrative request generated by the administrator and performed on the policy copy. This request is next broadcast to other users in order to enforce the new policy.

When a remote administrative request r_a is causally ready at site s (*i.e.* $r_a.v = v_s + 1$ and if r_a is a validation of a cooperative request r_c then r_c must have been already executed on this site), it is extracted from F. Otherwise, $r_a.t$ is either $AddAuth$ or $DelAuth$ in which case: (i) it is performed on the policy copy; and, (ii) it undoes the tentative cooperative requests that are no longer granted by the new policy. However, if r_a is a validation of cooperative request r_c then it sets $r_c.f$ to valid.

```
GENERATE_ADMIN_REQUEST($a$):
if $s$ is the administrator then
    if $a$ is a garbage administrative log request then
        GARBAGE_ADM_LOG()
    else
        Apply $a$ to $P$
        $r_a \leftarrow (s, a, v)$
        $L \leftarrow L + r_a$
        broadcast $r_a$
    end if
    $v.v_a \leftarrow v.v_a + 1$
end if

RECEIVE_ADMIN_REQUEST($r_a$):
if $(r_a.t = AddAuth \vee r_a.t = DelAtuh)$ then
    Apply $a$ to $P$
    for All $r_c \in \mathcal{T}$ do
        $r_c \leftarrow IT^a(r_c, r_a)$
        if $Invalidate(r_a, r_c)$ or $r_c.ad = \emptyset$ then
            Undo($r_c$)
            $\mathcal{T} \leftarrow \mathcal{T} - r_c$
            $\mathcal{I} \leftarrow \mathcal{I} + r_c$
        end if
    end for
else
    if $r_a.a = Validate(r_c)$ then
        $r_c.f \leftarrow$ "$Valid$"
        $\mathcal{V} \leftarrow \mathcal{V} + r_c$
    end if
end if
$v.v_a \leftarrow v.v_a + 1$
```

Algorithm 9: Generation and Reception of Administrative Requests: single-administrator approach

4.1.6.2 Multi-Administrator

In this approach, we consider that each user administrates his own objects. Hence, he can specify authorizations in the policy object by modifying his owner policy. All users modify the shared document with respect to the local policy object associated with the modified element. Our collaboration protocol proceeds as follows:

1. When a user manipulates the local copy of the shared document by generating a cooperative request r_c, this operation will be granted or denied by only checking the local copy of the owner policy owned by the $Administrator(r_c)$.

2. Once granted and executed, the local operations are then broadcast to other users.

3. A user has to check whether or not the remote operations are authorized by the copy of his local policy object before executing them by checking each operation against the appropriate owner log.

4. When a user modifies his owner policy by adding or removing authorizations, he sends these modifications to other users in order to update their local copies.

Initialization. At the initialization step (see Algorithm 8), s is set to the identifier of the site. The owner policy as well as owner administrative log are empty. The owner policy counter is set to 0. Then the global policy is initialized with the local owner policy. Finally, the coordination component is initialized.

Generation and reception of administrative request. In Algorithm 10, we discuss the update of the policy object. The policy copy is formed by the set of owner policies P_i. Each owner policy P_i maintains a version counter v_i that is incremented by the request generated by the administrator i and performed on his copy. This request is next broadcast to other users to enforce the new policy. However, if r_a is a validation of cooperative request r_c then it sets r_c to valid request.

When the received request r_a is causally ready it is extracted from F. If there is no entry of the user $j = r_a.s$ in the global policy then a new entry (P_j, L_j) is added. If $r_a.a$ is $AddAuth$ or $DelAuth$: (i) it is performed on P_j; and, (ii) it launches the $Undo()$ module to undo all tentative cooperative requests performed to alter objects owned by $r_a.s$ that are no longer granted by the new owner policy. If the received administrative operation is a validation, it simply validates the concerned cooperative operation. Finally, r_a is buffered in the owner administrative log L_j and the version v_i is incremented.

Garbage Administrative Log. The garbage administrative log procedures are described in Algorithm 11. The owner administrative log is removed immediately then the garbage request is broadcast to other sites so they remove the owner administrative log of site s. The final step is the update of the policy version v_s. Once RECEIVE_GARGABE_ADM_LOG(s',$v_{s'}$) (see Algorithm 11 line 7) is received at a remote site, all cooperative requests administrated by s and that are tentative are undone. Then the owner log L_s is removed and the version is updated. All requests that are concurrent or generated before RECEIVE_GARGABE_ADM_LOG(s',$v_{s'}$) are ignored (see Algorithm 4) since they do not have the same garbage version.

4.1.7 Asymptotic Time Complexities

We focus on security mechanism in order to study the asymptotic time complexities introduced by the access control layer. We study the complexities of CHECK_LOCAL(), CHECK_REMOTE(), $IT^a()$ as well as GENERATE_ADMIN_REQUEST() and RECEIVE_ADMIN_REQUEST().

The check of a local cooperative request is done against the local copy of the authorization list (see Algorithm 5). In the worst case, the authorization list or the policy P is parsed till the end (either the cooperative request is matched with the last authorization or there is no rule matching it in the policy). The time taken by the matching rule ($Grant()$ and $Revoke()$) is constant C if we use hash sets for both the subset of users and objects in each authorization. Consequently, the complexity of CHECK_LOCAL is linear with the size of P and is equal to $C * O(|P|)$

On the other hand, the check of a remote cooperative request is done against the administrative log. In the single-administrator approach, the worst case consists of parsing all the administrative log which means that the received cooperative operation has the initial version and was delayed for a long time. Since the complexity of $IT^a()$ (line 4 of Algorithm 6) is constant, the final complexity of CHECK_REMOTE is $O(|L|)$.

Concerning the multi-administrator approach, the policy object is formed by the couples (P_{s_i}, L_{s_i}) where P_{s_i} is the policy (authorization list) and L_{s_i} is the administrative log of the site s_i. We resort to hash maps in order to implement the global policy object. Hence, accessing a given policy or administrative log takes a constant time. The complexity of both functions CHECK_LOCAL(), CHECK_REMOTE() are linear with the size of the biggest owner policy and owner administrative log respectively.

```
1: GENERATE_ADMIN_REQUEST($a$):
2: if $s$ is the administrator then
3:     if $a$ is a garbage administrative log request then
4:         GARBAGE_ADM_LOG()
5:     else
6:         Apply $a$ to $P_i$
7:         $r_a \leftarrow (i, a, v_i)$
8:         $L_i \leftarrow L_i + r_a$
9:         broadcast $r_a$
10:    end if
11:    $v_i.v_a \leftarrow v_i.v_a + 1$
12: end if

13: RECEIVE_ADMIN_REQUEST($r_a$):
14: $j \leftarrow r_a.s$
15: if $(P_j, L_j)$ does not exist then
16:     create owner policy $(P_j, L_j)$
17:     $P_g \leftarrow P_g + (P_j, L_j)$
18: end if
19: if $(r_a.t = AddAuth \vee r_a.t = DelAtuh)$ then
20:     Apply $r_a$ to $P_j$
21:     for All $r_c \in \mathcal{T} \mid Administrator(r_c) = j$ do
22:         $IT^a(r_c, r_a)$
23:         if $Invalidate(r_a, r_c) \vee r_c.ad = \emptyset$ then
24:             Undo($r_c$)
25:             $\mathcal{T} \leftarrow \mathcal{T} - r_c$
26:             $\mathcal{I} \leftarrow \mathcal{I} + r_c$
27:         end if
28:     end for
29: else
30:     if $r_a = Validate(r_c)$ then
31:         $r_c.f \leftarrow Valid$
32:         $\mathcal{V} \leftarrow \mathcal{V} + r_c$
33:     end if
34: end if
35: $L_j \leftarrow L_j + r_a$
36: $v_j.v_a \leftarrow v_j.v_a + 1$
```

Algorithm 10: Generation and Reception of Administrative Request at the i-th Site: multi-administrator approach

The generation of a cooperative request has a constant complexity. However, the reception of a remote administrative request depends on the size of the set of tentative requests and on the complexity of the $Undo()$ function. The worst case is reached when $\mathcal{T} = H$. We suppose that the $Undo()$ function has a linear complexity (see Chapter 6). However if each request of the cooperative log H is undone, the final complexity of the RECEIVE_ADMIN_REQUEST() would be quadratic $O|H^2|$. Indeed, undoing a given request requires to parse all requests that follow in H which lead to a complexity equal to $n(n-1)/2$ with $n = |H|$. We stress the fact that such a case is not faced since the communications are in a real time. We assume that the transmission time of requests is very short. Consequently, the size of the cooperative log can not change rapidly between two different versions of the policy. This means that, $\mathcal{T} = H$ is not realistic in practise and that fewer requests are to be undone between two different versions of the policy object.

```
1: GARGABE_ADM_LOG( )
2: $L_s \leftarrow \emptyset$
3: SEND_GARGABE_ADM_LOG($s$,$v_s$)
4: $v_s.v_a \leftarrow 0$
5: $v_s.v_g \leftarrow v_s.v_g + 1$
6: RECEIVE_GARGABE_ADM_LOG($s'$,$v_{s'}$)
7: for all $r_c \in \mathcal{T}$ do
8:     if $Administrator(c) = s'$ then
9:         $Undo(r_c)$
10:    end if
11: end for
12: $L_{s'} \leftarrow \emptyset$
13: $v_{s'}.v_a \leftarrow 0$
14: $v_{s'}.v_g \leftarrow v_{s'}.v_g + 1$
```

Algorithm 11: Garbage Collection Administrative Logs Procedures at site s

4.1.8 Illustrative Example

To highlight the feature of our concurrency control algorithm, we present a slightly complicated scenario in Figure 4.1, where the solid arrows describe the integration order. We have three users s_1, s_2 and s_3 starting the collaboration with the initial state D_0 =“abc” where characters “a”, “b” and “c” are inserted by s_1, s_2 and s_3 respectively. The initial global policy $P_g = \{(P_1, L_1), (P_2, L_2), (P_3, L_3)\}$ with $P_i = \emptyset$ and $L_i = \emptyset$ for $i = 1, 2, 3$. Initially, the cooperative log of each site is empty ($H_i^0 = \emptyset$ for $i = 1, 2, 3$).

Site s_1 generates a cooperative requests where $r_{c_0}.c = Ins(2, y)$. Concurrently, site s_3 generates an administrative request r_{a_1} with $r_{a_1}.a = AddAuth(0, \langle\{s_2\}, Doc, \{u, d\}, +\rangle)$. After receiving r_{a_1} at site s_2 it generates r_{c_1} with $r_{c_0}.c = Del(3, c)$. Concurrently, site s_3 generates another administrative request r_{a_2} with $r_{a_2}.a = AddAuth(0, \langle\{s_2\}, Doc, \{d\}, -\rangle)$. Also site s_1 generates an administrative request r_{a_3} with $r_{a_3} = AddAuth(0, \langle All, Doc, \{u, d\}, +\rangle)$.

Once r_{a_3} is received by s_3, it generates a cooperative request r_{c_2} with $r_{c_2}.c = Del(1, a)$. As soon as r_{c_2} arrives at site s_1 it generates an administrative request r_{a_4} with $r_{a_4}.a = Validate(r_{c_2})$.

The following relations are verified by our set of requests:

- r_{c_0} is concurrent r_{a_1};
- r_{c_1} is causally dependent on r_{a_1};
- r_{c_1} and r_{a_2} and r_{a_3} are concurrent;
- r_{c_2} causally depends on r_{a_4};
- r_{a_4} that also causally depends on r_{a_3}.

We describe the integration of our requests in the following steps:

Step 1. At s_1, the execution of r_{c_0} produces the state “aybc” and the cooperative log $H_1 = r_{c_0}$. When r_{a_1} is received at both sites s_1 and s_2, the owner policy copy of s_3 is updated to $P_3 = \{\langle\{s_2\}, Doc, \{u, d\}, +\rangle\}$ as for the owner log L_3 that becomes $L_3 = r_{a_1}$. When r_{c_1} arrives at site s_2, it results in the state “ybc” and the cooperative log $H_2 = r_{c_0}$. Site s_3 generates an administrative request r_{a_1} with $r_{a_1} = AddAuth(0, \langle\{s_2\}, Doc, \{u, d\}, +\rangle)$. This leads to the update of the global policy by inserting the authorization $\langle\{s_2\}, Doc, \{u, d\}, +\rangle$ into P_3, also the administrative log is updated to

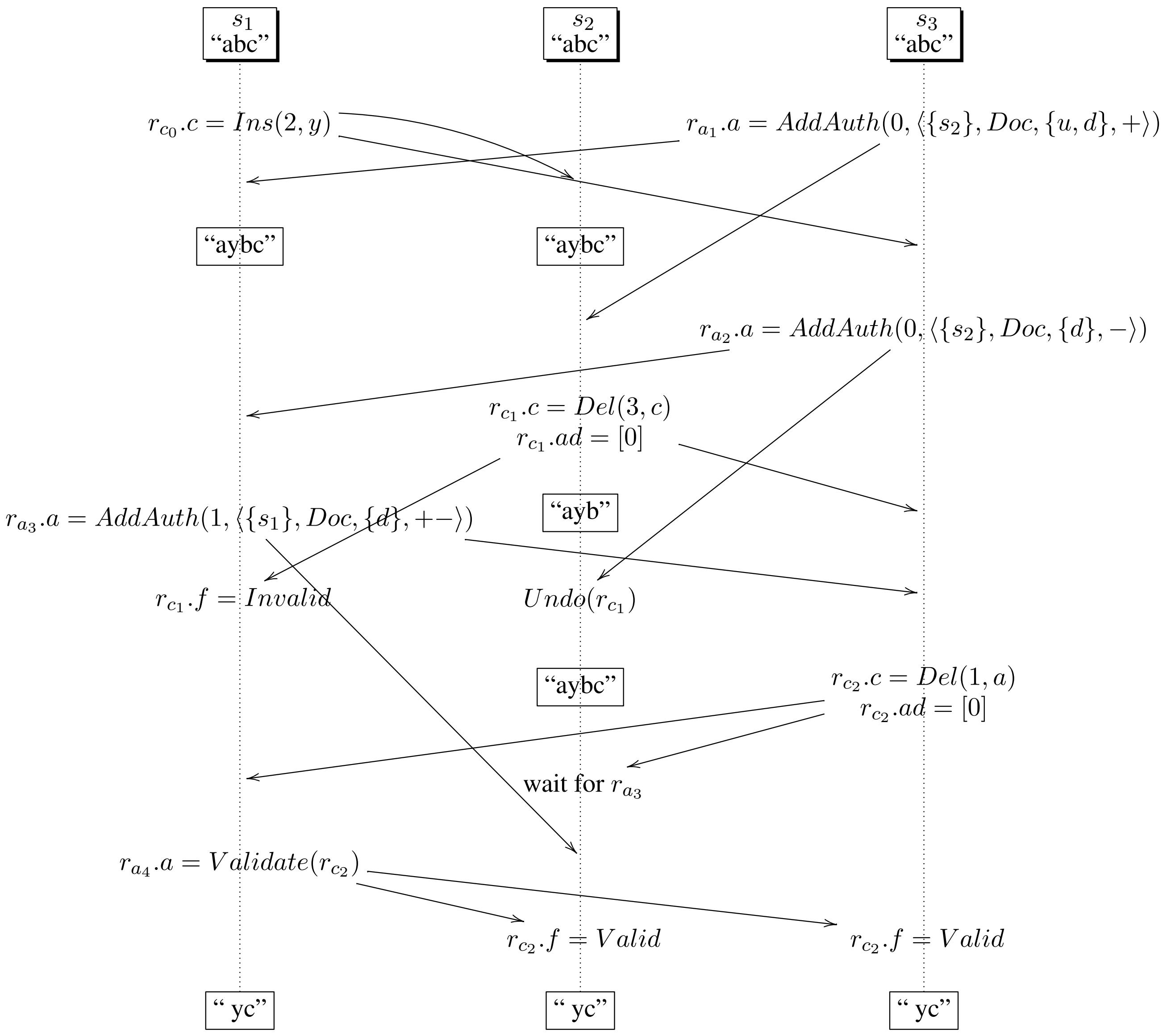


Figure 4.1: Collaboration scenario between an administrator and two sites.

become $L_3 = r_{a_1}$. Similarly, when r_{a_1} is received at site s_1 and s_2, they update their copies of the owner policy and log of site s_3.

Step 2. At s_2, the execution of r_{c_1} gives the state "ayb" and the log $H_2 = r_{c_0} \cdot r_{c_1}$. This operation is authorized since r_{a_1} granted the deletion right to s_2 and $Administrator(r_{c_1}) = s_3$. The administrative dependency of r_{c_1} is set to $r_{c_1}.ad = [0]$ and it is broadcast to other sites.

Concurrently, site s_3 generates a restrictive administrative request r_{a_2} to revoke the deletion right from s_2 with $r_{a_2} = AddAuth(0, \langle\{s_2\}, Doc, \{d\}, -\rangle)$. It is obvious that the policy P_3 is violated by s_2. When r_{c_1} arrives at site s_3 the function CHECK_REMOTE() returns a negative result since $Invalidate(r_{a_2}, r_{c_1}) = true$. However r_{c_1} was executed at site s_2 which lead to divergence. Similarly, at site s_1 when r_{c_1} is is checked against L_3 it is rejected due to the reception of r_{a_2} but is stored in invalid form $r^*_{c_1}$ ($r_{c_1}.f = Invalid$) which has no effect on the local document state. The resulting log is $H_1 = r_{c_0} \cdot r^*_{c_1}$.

Enforcing the new policy requires to undo r_{c_2} at site s_2. Indeed, the reception of r_{a_2} introduces the undo of r_{c_1} since it is a tentative request (not validated yet) and $Invalidate(r_{a_2}, r_{c_1}) = true$. This allows to converge to the same document as site s_3 and s_1. The state is restored to "aybc" and the log is updated by excluding the effect of r_{c_1}.

Step 3. Site s_1 generates an administrative request r_{a_3} to add a positive permission to L_1. Let $r_{a_3}.a = AddAuth(1, \langle\{s_1\}, Doc, \{d\}, +-\rangle)$, this leads to a new version of the policy $v_{s_1} = 1$ with $P_1 = \{\langle\{s_1\}, Doc, \{d\}, +-\rangle)\}$ and $L_1 = r_{a_3}$. Then r_{a_3} is broadcast to s_2 and s_3 so that they update their local copies of the owner policy of s_1. After receiving r_{a_3}, site s_3 generates the cooperative request r_{c_2} with $r_{c_2}.c = Del(1, a)$. Once executed locally, the state becomes "yc". Then this operation is broadcast with $r_{c_2}.v = 1$ and $r_{c_2}.ad = [0]$.

When site s_2 receives r_{c_2}, it stores it in a buffer until it is causally ready. In fact, the version of P_1 at site s_2 is 0. As well as r_{a_3} is received from site s_1, the version of P_1 changes to 1 and r_{c_2} is executed since it is granted.

Once r_{c_2} is received at site s_1 and granted, its execution leads to the state "yc". Then a validation request r_{a_3} is broadcast in order to set r_{c_2} to Valid at all remote sites.

It is clear that all sites converge to the same final state "yc" and policy object.

4.2 Correctness Proof

We will first give some general principles for providing access control in a collaborative editing system. Then we will formalize our correctness criteria.

4.2.1 General Principles

Ensuring security, more precisely access control to shared objects, is more challenging in a collaborative application than in a single-user one due to concurrency. We use the following principle as guidelines in our work:

(a) The security object is replicated: Like the shared document, the policy is also replicated in order to benefit of the full potential of the replication and not introduce additional overhead due to controlling access at a server site.

(b) The security object is updated concurrently with the data object. The modification of the policy object is done independently of the document update. There is no use of locks in order to respect the real time aspect of the collaboration.

(c) Once all requests are received by all sites, users must have the same document state and policy state. Users must converge in terms of data and policy objects.

(d) To enforce access control, we resort to selective $Undo()$. In fact, enforcing the policy requires to undo concurrent cooperative requests that are not legal with respect to the new policy enforced concurrently by the administrator.

(e) $Undo()$ is an operation executed locally and not exchanged between users. Indeed undoing an operation is an administrative action launched and executed at receiver sites.

(f) A cooperative operation can be undone exactly once at each site since this undo is due to policy violation.

4.2.2 The Correctness Criteria

In the following we formalize two correctness criteria, namely causality and access policy preservation. This allows us to provide formal proof of our algorithms correctness.

Following the notations of [32], we define the causality relation between two requests as follows: for any two requests r_1 and r_2, the dependency relation is denoted as $r_1 \to r_2$ if r_2 depends on r_1 (this dependency could be temporal *i.e* happens before relation or based on the semantic of requests *e.g* semantic dependency [48, 50] or on effect relation [88]). If the two requests are concurrent we denote then as $r_1 || r_2$.

Definition 4.2.9 (Causality Preservation)**.**

Given any two requests r_1 and r_2 in $\mathcal{C}_r \cup \mathcal{A}_r$, if $r_1 \to r_2$, then r_1 is invoked before r_2 at any site in the system.

The causality condition is compatible with our principles and it is possible to verify it thanks to the dependency relations we already defined for both cooperative and administrative requests. If r_1 and r_2 are both cooperative requests, the causality condition depends not only on the ACL but also on the CL on the top of which we will implement our model. Some CLs are based on the Lamport clocks [56] which does not scale well and does not allow for open groups. The approach of [48] relies on the semantic dependency notion to define the causality and concurrency relations. In addition to the causality relation imposed by the underlying CL, the ACL adds a new condition of causality which is the policy context (or its version). A cooperative request must wait till the version of the receiver site is greater or equal to its generation's version. Consequently $r_1 \to r_2$ iff $r_2.v_g = r_1.v_g$ and $r_2.v_a \geq r_1.v_a$ where r_1 is an administrative request and r_2 is a cooperative one. As for administrative requests, we have $r_1 \to r_2$ iff $r_2.v_g = r_1.v_g$ and $r_2.v_a \geq r_1.v_a + 1$.

The second correctness criteria is given below:

Definition 4.2.10 (Policy Enforcement)**.**

The execution of any cooperative or administrative request at a site s_i does not violate its local access control policy of the administrator. After the reception of all administrative requests, all sites must have the same document state. More formally, the following statements must be true:

1. $\mathcal{T}_{s_i} = \emptyset$;
2. $\mathcal{V}_{s_i} = \mathcal{V}_{s_j}$;
3. $\mathcal{I}_{s_i} = \mathcal{I}_{s_j}$.

for all sites s_i and s_j.

Using the two correctness criteria cited before, our objective is to satisfy the convergence property defined as follows:

Definition 4.2.11 (Convergence)**.**

When all sites have performed the same set of cooperative and administrative requests, the copies of both shared document and policy object are identical.

In order to ensure convergence of both policy and document state, the condition of policy enforcement must be fulfilled. The statement (1) of Definition 4.2.10 ensures that all tentative requests are seen by the appropriate administrator, this ensures that all requests at the network are received by all sites. Statements (2) and (3) of the same definition ensure that all legal/illegal requests are the same at all sites and that all illegal request are undone. This enforces the policy and allows for document convergence.

In the following, we will present formal proofs with regard to the previous conditions.

Theorem 4.2.1.

Our coordination algorithm satisfies the causality preservation condition.

Proof. In our algorithm each cooperative request r_c is invoked only when it is causally ready (see Definition 4.1.6). That is, all cooperative requests and administrative requests on which depends r_c have been invoked. Similarly, since administrative requests are generated by the same user, thanks to the policy version (administrative requests counter), all these requests are executed in the same order in all sites which respects the causality condition. □

In the following lemma, we prove that the set of tentative requests is empty after the reception of all requests by the collaborating sites.

Lemma 4.2.1.

Given n sites $s_1, s_2, \ldots, s_n$ with the set of tentative operations $\mathcal{T}_1, \mathcal{T}_2 \ldots \mathcal{T}_n$ and p cooperative requests $r_{c_1}, r_{c_2}, \ldots, r_{c_p}$. When the p requests are received by all sites, then

$$\forall i \in [1..n], \mathcal{T}_i = \emptyset$$

Proof. Each set of tentative requests is updated at tree steps of our algorithm:

1. line 9 in Algorithm 3;
2. line 10 in Algorithm 4;
3. line 20 in Algorithm 9 as well as line 25 in Algorithm 10);

We refer to the multi-administrator approach since the single-administrator approach is a special case of the multi-administrator one. The proof is similar in both cases. At the generation step, a cooperative request is either valid, if it is generated by its administrator or tentative. Suppose that none of the requests $r_{c_1}, r_{c_2}, \ldots, r_{c_p}$ is generated by its administrator. In this case. After the reception of all cooperative requests $\mathcal{T}_1 = \mathcal{T}_2 = \ldots = \mathcal{T}_n = \{r_{c_1}, r_{c_2}, \ldots, r_{c_p}\}$. Each time r_{c_i} is received by $s_j = Administrator(r_{c_i})$, two situations are faced:

1. r_{c_i} is granted by L_{s_j}: in this cases r_{c_i} is set to valid, $\mathcal{T}_j$ is updated by removing r_{c_i} and a validation request is sent to every site s_k, $k \in [1..n]$ in order to set r_{c_i} to valid and remove it from $\mathcal{T}_k$.

2. r_{c_i} is invalidated by L_{s_j}: in this case r_{c_i} is set to invalid, $\mathcal{T}_j$ is updated by removing r_{c_i}. This means that a concurrent administrative request r_a conflicting with r_{c_i} is generated at site s_j. When r_a is received by a site s_k, it will undo r_{c_i} (line 24 in Algorithm 10) and update $\mathcal{T}_k$ by removing r_{c_i}.

Accordingly, since the same process is done for all requests, we have $\forall i \in [1..n], \mathcal{T}_i = \emptyset$. □

In the following, we show that after receiving all operations, the set of valid operations is equal at all sites.

Lemma 4.2.2.

Given n sites $s_1, s_2, \ldots, s_n$ with the set of tentative operation $\mathcal{T}_1, \mathcal{T}_2 \ldots \mathcal{T}_n$ and p cooperative requests $r_{c_1}, r_{c_2}, \ldots, r_{c_p}$. When the p requests are received by all sites, then

$$\forall i, j \in [1..n], \mathcal{V}_i = \mathcal{V}_j$$

Proof. Suppose that initially, all sites begin with an empty set of valid operations. Consider a cooperative request r_c and a site s where $s = Administrator(r_c)$. Let v_s be the policy version of s at the moment of reception of the request r_c and L_s be its administrative log (note that $v_s = |L_s|$). Two cases are possible:

1. s does not generate administrative requests concurrently to r_c

2. s has generated k administrative requests $L_s[r_c.v; r_c.v + k]$ concurrently to r_c.

In case (1) the function CHECK_REMOTE(r_c, L_s) returns true since the context is the same. Consequently $\mathcal{V}_s = \{r_c\}$ and a validation request is sent to every site s_j in order to validate r_c and add it to $\mathcal{V}_j = \{r_c\}$. If it is the case for every cooperative request $r_c \in \{r_{c_1}, r_{c_2}, \ldots, r_{c_p}\}$, then after receiving all cooperative and administrative requests we have $\forall i, j \in [1..n], \mathcal{V}_i = \mathcal{V}_j = \{r_{c_1}, r_{c_2}, \ldots, r_{c_p}\}$.

In the second case (2), a request is valid at site s when CHECK_REMOTE(r_c, L_s)=$true$. If $\forall r_a \in L_s[r_c.v; r_c.v + k]$, $Grant(r_a, r_c) = true$, then $r_c.ad \neq \emptyset$ after the remote check and $\forall r_a \in L_s[r_c.v; r_c.v + k]$, $Invalidate(r_c, r_a) = false$. Each time an administrative request $r_a \in L_s[r_c.v; r_c.v + k]$ is received at a site $s_j \neq s$, the procedure RECEIVE_ADMIN_REQUEST(r_a) is called and tentative requests are updated with the transformation function IT^a. Similarly, since $\forall r_a \in L_s[r_c.v; r_c.v + k]$, $Invalidate(r_a, r_c) = false$, $r_c.ad \neq \emptyset$, the request r_c in not undone (line 21-24 in Algorithm 10). As soon as the validation is received from s at site s_j, the request r_c is added to $\mathcal{V}_j$ (line 31 in Algorithm 10). Otherwise, either $r_c.ad = \emptyset$ or it is invalidated ($\exists r_a \in L_s[r_c.v; r_c.v + k]$ verifying $Invalidate(r_a, r_c) = true$). In both cases r_c is set to invalid, thus neither added to $\mathcal{V}_s$ since it is undone (lines 18 in Algorithm 10), nor to $\mathcal{V}_j$ (set of valid requests at the receiver site) due to the remote check. If the same case is faced for every cooperative request $r_c \in \{r_{c_1}, r_{c_2}, \ldots, r_{c_p}\}$, then after receiving all cooperative and administrative requests we have $\forall i, j \in [1..n], \mathcal{V}_i = \mathcal{V}_j = \emptyset$.

If some requests are in case (1) and some others are in case (2), then $\forall i, j \in [1..n], \mathcal{V}_i = \mathcal{V}_j$ and contains the set of requests in case (1). □

Finally, we show that the sets of invalid operations are equal at all sites.

Lemma 4.2.3.

Given n sites $s_1, s_2, \ldots, s_n$ with the set of tentative operation $\mathcal{T}_1, \mathcal{T}_2 \ldots \mathcal{T}_n$ and p cooperative requests $r_{c_1}, r_{c_2}, \ldots, r_{c_p}$. When the p requests are received by all sites, then

$$\forall i, j \in [1..n], \mathcal{I}_i = \mathcal{I}_j$$

Proof. Since at every site s_i, $H_i \setminus \mathcal{V}_\rangle = \mathcal{T}_\rangle \bigcup \mathcal{I}_\rangle$, where H_i is the cooperative log, it comes from Lemmas 4.2.1 and 4.2.2, that after receiving all requests $\mathcal{I}_i = \mathcal{I}_j$ □

Using the precedent lemmas, we deduce that the policy enforcement condition is satisfied.

Theorem 4.2.2.

Our algorithm ensures the policy enforcement condition.

Proof. According to the three precedent Lemmas 4.2.1, 4.2.2 and 4.2.3, the policy enforcement condition is fulfilled. □

Additionally, the convergence condition is ensured according to the following theorem.

Theorem 4.2.3.

Our algorithm ensures the convergence of both data and policy objects.

Proof. We assume that the CL ensures the convergence of the shared document. The ACL may introduce divergence cases as shown in Section 3.2. Our algorithm resolves some of these convergence problems with the use of undo algorithm. Assuming that our undo solution is correct (see Chapter 6), we must prove that the convergence is ensured. For this, all cooperative logs must produce the same document state after the reception of all cooperative and administrative requests by all sites. According to Lemma 4.2.2 and Lemma 4.2.3, we ensure that the same set of requests are executed at all sites, otherwise invalid requests are either not applied to the document copy or undone by the reception of the concurrent administrative request. Consequently, since logs are equivalent according to the CL, convergence of the document state is ensured.

Concerning the policy and the administrative log, since administrative requests are executed at the enumeration order of the administrator site, every administrative request r_a that is causally ready is executed after the execution of all administrative requests that happen before it. □

4.3 Conclusion

In this chapter, we have presented the algorithms of our access control model and calculated their asymptotic time complexities. Finally, we have demonstrated that our approach is correct and ensures convergence.

In the next Chapter, we concentrate on the undo command as a main feature of our access control model. Indeed, since we design an optimistic access control model, we allow users to temporarily violate security policy. Convergence is maintained thanks to the selective undo approach.

Chapter 5

On the Undoability Problem in Distributed Collaborative Editors

Contents

The ability to undo operations performed by a user is a standard and very useful feature allowing to reverse erroneous operations and restoring a correct state without being obliged to redo all the work performed on a document. It represents an indispensable feature for many collaborative applications mainly real time collaborative editors. Furthermore, maintaining convergence in access control-based collaborative editors requires a selective *undo* mechanism [21] as shown in Chapter 3 wherein we have considered an optimistic access-control-based RCE in the sense that we tolerate temporary access right violations. To maintain convergence, we must *undo* illegal updates to alter the document state.

We focus on selective undo which is based on rearranging operations in the history using the OT approach. Three inverse properties that we will detail later, namely $\mathbf{IP1}$, $\mathbf{IP2}$ and $\mathbf{IP3}$ [36, 74, 97, 101] were proposed to achieve undo. Combining OT and undo approaches while ensuring data convergence remains an open and challenging issue. Indeed, undoing operations may itself lead to divergence cases called undo *puzzles* [97]. Moreover, providing an undo solution for collaborative applications has to take into account three main issues: (i) formalizing the correctness criteria of an undo solution, (ii) designing the algorithm and prove its correctness, (iii) ensure performance by providing a low complexity of the algorithm.

Even though many solutions were proposed over the recent years, designing undo schemes in collaborative applications is a hard task since each proposed solution has either a limitation or a counterexample showing it is not correct [88, 101].

This chapter is organised as follows: Section 5.1 presents the notations used to illustrate the undoability problem. Section 5.2 illustrates the principle of undoing operations in collaborative editors. Finally, Section 5.3 gives an overview on related work.

5.1 Notations

An undo framework assumes a collaborative application model in which all updates or operations performed on the shared object are achieved in a history list also called log in order to provide the basis for undoing operations. The operations are reversible and capable of being reordered unless there are dependencies between them.

All applications maintain a current state of the shared object that is being updated concurrently by collaborators. This state depends on the nature of the shared object. We use the notation st to denote the state of the shared object. Let $\mathcal{S}t$ be the set of all possible states for a given shared object.

When we apply an operation to a given state $st \in \mathcal{S}t$, we obtain a new state $st' \neq st$ where also $st \in \mathcal{S}t$. Any given state represents simply the application of a sequence of operations to the initial state referred to as st_0. We use the letter op to denote operations performed to update a given state. Letters and numbers will be used to distinguish between different operations. Let $\mathcal{O}p$ be the set of all possible operations. For a given state $st \in \mathcal{S}t$ and an operation $op \in \mathcal{O}p$, the notation $st \cdot op$ refers to the new state resulting from the application of op on st. The idle operation I is the operation that has no effect on a given state. It verifies the following statement:

$$\forall st \in \mathcal{S}t,\ st \cdot I = st.$$

The sequence of operation $\{op_1, op_2, \ldots, op_n\}$ is denoted by $op_1 \cdot op_2 \cdot \ldots \cdot op_n$. Two sequences of operations are equivalent if they provide the same resulting state when they are performed on the same initial state :

Definition 5.1.1 (Log equivalence)**.**

Two sequences of operations $seq_1 = op_1 \cdot op_2 \cdot \ldots \cdot op_n$ and $seq_2 = op'_1 \cdot op'_2 \cdot \ldots \cdot op'_k$, n and k being two integers, are said to be equivalent denoted as $seq_1 \equiv seq_2$ iff

$$\forall st \in \mathcal{S}t,\ st \cdot seq_1 = st \cdot seq_2$$

To support undo, we suppose that every operation $op \in \mathcal{O}p$ is reversible, *i.e.* it has an inverse noted $\overline{op} \in \mathcal{O}p$[14].

5.2 Undo Approach

This Section presents the principle of undoing an operation in a collaborative context as well as properties to be satisfied in order to achieve a correct undoability.

[14]The inverse of the idle operation I is $\overline{I} = I$.

$$\begin{tikzcd}
 & L & & L' & \text{(after undo)} \\
 & op_1 & & op_1 & \\
 & op_2 & & op_2 & \\
 & \vdots & & \vdots & \\
 & op_{i-1} & & op_{i-1} & \\
 & op_i & \equiv & op_i^* & \\
\overline{op_i} \arrow[ddd, "IT"'] & op_{i+1} & & op'_{i+1} & \\
 & \vdots & & & \\
 & op_n & & op'_n & \\
\overline{op_i}' & & & &
\end{tikzcd}$$

Figure 5.1: Undo Scheme.

5.2.1 Principle

Logging all executed operations is necessary to accomplish an undo scheme. Furthermore, all operations should be undoable (*i.e.*, each operation op has its inverse operation $\overline{op}$). As proposed in [74, 97], to selectively undo operation op_i from the log $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$, we proceed by the following steps:

1. Find op_i in L;

2. Mark op_i as an undone operation: op_i^*;

3. Generate $\overline{op_i}$;

4. Calculate $\overline{op}' = IT^*(\overline{op_i}, op_{i+1} \cdot \ldots \cdot op_n)$ that integrates the effect of operations following op_i in L;

5. Exclude the effect of op_i from the log by including the effect of $\overline{op_i}$ inside the sublog $op_{i+1} \cdot \ldots \cdot op_n$; We can exclude the effect of op_i from the sublog $op_{i+1} \cdot \ldots \cdot op_n$ using this small algorithm:
```
$op \leftarrow \overline{op_i}$
**for** $j$ from $i+1$ to $n$ **do**
$op'_j \leftarrow IT(op_j, op)$
$op \leftarrow IT(op, op_j)$
**end for**
```
   The sequence of operations following op_i^* is then $op'_{i+1} \cdot \ldots \cdot op'_n$;

6. Execute $\overline{op}'$.

5.2.2 Undo Effect

Undoing an operation op at a given state st must restore the previous state of the object as if op had never occur. This can be formalized as follows:

Definition 5.2.2 (Undo Effect)**.**

Given an object state

$$st_n = st_0 \cdot op_1 \cdot \ldots \cdot op_{i-1} \cdot op_i \cdot op_{i+1} \cdot \ldots \cdot op_n,$$

then the effect of undoing op_i must produce the state

$$st'_{n-1} = st_0 \cdot op_1 \cdot \ldots \cdot op_{i-1} \cdot op'_{i+1} \cdot \ldots \cdot op'_n,$$

where op'_j, $j \in i+1, i+2, \ldots, n$ is the operation that would be executed if op_i has never been executed before.

Practically, op'_j is the operation that excludes the effect of op_i. This allows to eliminate the effect of op_i but retains the effect of the operations that follow op_i in the log [97].

Example 5.2.1. *Consider the initial state $st_0 = abcde$ and two operations $op_1 = Del(2, b)$ and $op_2 = Del(2, c)$ (since $st_0 \cdot op_1 = acde$). Then the resulting state is $st' = ade$ and the log is $H = [op_1; op_2]$. Undoing op_1 should produces the state $st'' = abde$. Moreover, the log has to be updated by eliminating the effect of the undone operation op_1 from the log. Thus $H' = [op_1^*; op'_2]$ with $op'_2 = Del(3, c)$ instead of $Del(2, c)$.*

5.2.3 Undo Properties

Three inverse properties **IP1**, **IP2** and **IP3**, have been proposed in the literature (see [36, 74, 97, 101]) to formalize the correctness of a transformation-based undo scheme.

Definition 5.2.3 (Inverse Property 1 (**IP1**))**.**

Given any operation op and its inverse $\overline{op}$, then

$$op \cdot \overline{op} \equiv \emptyset.$$

Property **IP1** means the operation sequence $op \cdot \overline{op}$ should have no effect on the object state.

Definition 5.2.4 (Inverse Property 2 (IP2))**.**

Given a correct transformation function IT and any two operations op_1 and op_2 then:

$$IT(IT(op_1, op_2), \overline{op_2}) = op_1.$$

As the sequence $op_2 \cdot \overline{op_2}$ should have no effect, property **IP2** means transforming op_1 against op_2 and its inverse $\overline{op_2}$ must result in the same operation. This property is generally avoided by placing an inverse just after the undone operation in the log then to consider the sequence $op_2 \cdot \overline{op_2}$ as an empty sequence.

Definition 5.2.5 (Inverse Property 3 (IP3))**.**

Given a transformation function IT and any two operations op_1 and op_2 with $op'_1 = IT(op_1, op_2)$ and $op'_2 = IT(op_2, op_1)$. If sequences $op_1 \cdot op'_2 \equiv op_2 \cdot op'_1$ then

$$IT(\overline{op_1}, op'_2) = \overline{IT(op_1, op_2)}.$$

Property **IP3** means that the operation executed to undo op_1 in $op_1 \cdot op'_2$ is the same as the operation executed to undo the corresponding operation op'_1 in $op_2 \cdot op'_1$.

The violation of one of properties **IP1**, **IP2** and **IP3**, leads to divergence situations referred to as *puzzles* [97] as we illustrate in the following.

5.2.4 Illustrative Examples

To further illustrate the undo properties, we present the following examples:

Example 5.2.2. *Consider a shared integer register altered by two operations $Inc()$ and $Dec()$ which increment and decrement respectively the register state such as the one is the inverse of the other. A correct transformation function is defined as follows:*

$$IT(op_1, op_2) = op_1 \textit{ for all operations } op_1, op_2 \in \{Inc(), Dec()\}.$$

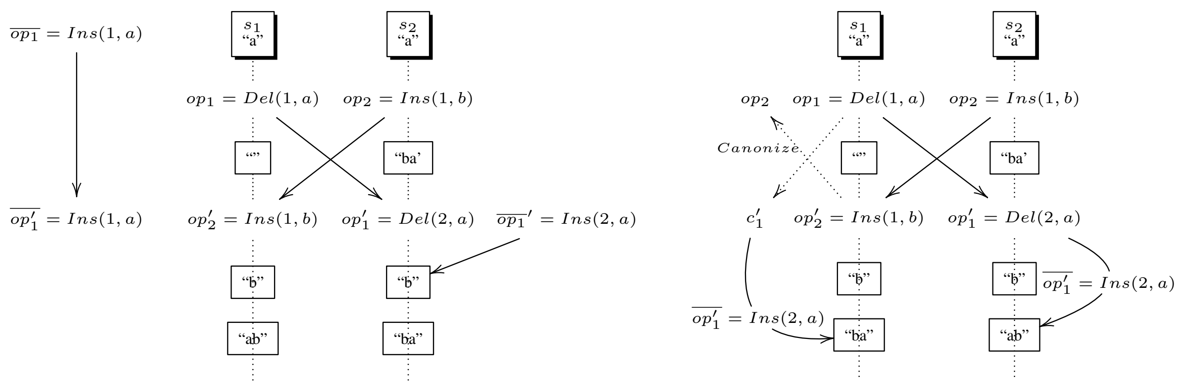
It is trivial that **IP1** *is satisfied as $Inc() \cdot Dec() \equiv Dec() \cdot Inc() \equiv \emptyset$. Furthermore it is easy to verify that* **IP2** *and* **IP3** *are satisfied since $IT(IT(op_1, op_2), \overline{op_2}) = IT(op_1, op_1) = op_1$ and $IT(\overline{op_1}, IT(op_2, op_1)) = \overline{op_1} = \overline{IT(op_1, op_2)}$ for all operations $op_1, op_2 \in \{Inc(), Dec()\}$.*

Example 5.2.3. *Consider an application of collaborative editing of the same shared document altered by the set of the following three operations:*

1. *$Ins(p, e)$ to insert element e at position p*
2. *$Del(p, e)$ to delete element e at position p*
3. *$Upd(e, e', p)$ to replace element e at position p with element e'*

The IT function is given in Appendix B.

Violating **IP2** *leads to an* **IP2** *undo puzzle [97]. This puzzle results when transforming an operation against another operation and its inverse during the couple do-undo pair procedure. The problem occurs when transforming two insertions at the same position since the position is changed arbitrarily according to site identifiers. To illustrate this puzzle, consider a document with the initial state $st_0 = e_0e_1 \ldots e_{p-1}e_pe_{p+1}e_{p+2} \ldots e_n$ and two operations $op_1 = Del(p, e_p)$ and $op_2 = Del(p, e_{p+1})$. Suppose that op_2 depends causally on op_1 then undoing op_2 then op_1 may lead to a divergence case as discussed in [74, 77, 97]. Indeed, after executing op_1 and op_2 at the state s_0, we obtain the state $st_1 = e_0e_1 \ldots e_{p-1}e_{p+2} \ldots e_n$. Now, undoing op_2 produces the state $st_2 = e_0e_1 \ldots e_{p-1}e_{p+1}e_{p+2} \ldots e_n$ and the history log $H = op_1 \ldots op_2 \ldots \overline{op_2}$. Then, if we undo op_1, we have to determine $IT(IT(\overline{op_1}, op_2), \overline{op_2})$. Since $\overline{op_1} = Ins(p, e_p)$ then transforming it against op_2 produces $\overline{op_1}' = Ins(p, e_p)$. Finally, we have to transform $Ins(p, e_p)$ against $Ins(p, e_{p-1})$ which may result in the operation $Ins(p + 1, e_p)$ instead of $Ins(p, e_p)$. Consequently, the effect of undo is not correct since the initial state was equal to $e_0e_1 \ldots e_{p-1}e_pe_{p+1}e_{p+2} \ldots e_n$ and not $e_0e_1 \ldots e_{p-1}e_{p+1}e_pe_{p+2} \ldots e_n$. This divergence case is referred to as the coupled do-undo pair trap [97].*



(a) Undoing op_1 produces a divergence.   (b) Avoiding divergence thanks to canonization.

Figure 5.2: Avoiding divergence thanks to canonization.

As for **IP3***, it is not satisfied by our example since the insert-insert puzzle presented in Figure 5.2 may be encountered. This puzzle occurs when transforming two insertions during the undo procedure where the first is an inverse (see Figure 5.2.(a)). This puzzle may be eliminated in the coordination framework OPTIC [48] since it is based on canonical logs where all insertions precede all deletions as shown in Figure 5.2.(b).*

Example 5.2.4. *Consider a shared binary register where two primitive operations modify the state of a bit from* 0 *to* 1 *and vice versa: (i)* Up *to turn on the register; (ii)* $Down$ *to turn off the register. Intuitively, we can write the transformation* IT *as given in Algorithm 12.*

```
IT($op_1$,$op_2$):$op'_1$
Choice of $op_1$ and $op_2$
  Case: $op_1 = Up$ and $op_2 = Up$
    $op'_1 \leftarrow Up$
  Case: $op_1 = Up$ and $op_2 = Down$
    $op'_1 \leftarrow Up$
  Case: $op_1 = down$ and $op_2 = Up$
    $op'_1 \leftarrow Up$
  Case:$op_1 = Down$ and $op_2 = Down$
    $op'_1 \leftarrow Down$
end choice
```

Algorithm 12: Transformation Cases for the Set of Binary Operations $\{Up, Down\}$

First, we show that the transformation Algorithm 12, is correct i.e respects both transformation properties **TP1** *and* **TP2***.*

Theorem 5.2.1.

Algorithm 12 verifies $TP1$*.*

Proof. Consider a state $st \in St$ and a couple of two concurrent operations $(op_1, op_2) \in \{Up, Down\}^2$. Four cases are possible:

1. $(op_1, op_2) = (Up, Down)$: We have $\forall\, st \in \{0,1\},\ st \cdot Up = 1$. Integrating op_1's effect into op_2 leads to $IT(op_2, op_1) = IT(Down, Up) = Up$. Consequently, the final state is $st \cdot Up \cdot Up = 1$. On the other hand, applying the operation Down to st leads to $st' = st \cdot Down = 0$. The final state is the result of performing $IT(op_1, op_2) = IT(Up, Down) = Up$. Since $\forall\, st \in \{0,1\},\ st.Up = 1$, the final state is $st \cdot Down \cdot Up = 1$. Hence, if $(op_1, op_2) = (Up, Down)$ then $\forall\, st \in \{0,1\},\ st \cdot op_1 \cdot IT(op_1, op_2) = st \cdot op_2 \cdot IT(op_2, op_1)$.

2. $(op_1, op_2) = (Down, Up)$: this case is symmetric to case (1).

3. $(op_1, op_2) = (Up, Up)$: Since $IT(Up, Up) = Up$ and $\forall\, st \in \{0,1\},\ st \cdot Up = 1$, we have: $\forall\, st \in \{0,1\},\ st \cdot op_1 \cdot IT(op_1, op_2) = st \cdot op_2 \cdot IT(op_2, op_1)$ when $(op_1, op_2) = (Up, Up)$.

4. $(op_1, op_2) = (Down, Down)$: this case is symmetric to case (3).

Consequently, for all concurrent couple of operations $(op_1, op_2) \in \{Up, Down\}^2$, we have

$$\forall\, st \in \{0,1\},\ st \cdot op_1 \cdot IT(op_1, op_2) = st \cdot op_2 \cdot IT(op_2, op_1)$$

That is, IT verifies $TP1$. □

Theorem 5.2.2.

Algorithm 12 verifies $TP2$.

Proof. Consider $st \in \mathcal{S}t$ and a pairwise concurrent operations op_1, op_2 and op_3 belonging to $\{Up, Down\}$. Suppose that $op_1 \neq op_2$ (otherwise the proof is trivial) and let $op_3' = IT(op_3, op_1)$ and $op_3'' = IT(op_3, op_2)$. If $(op_1, op_2) = (Up, Down)$, then two cases are possible:

1. $op_3 = Up$: In this case $op_3' = op_3'' = Up$ and $IT(op_2, op_1) = IT(op_1, op_2) = Up$. Consequently, $IT(op_3', IT(op_2, op_1)) = IT(Up, Up) = Up$ and $IT(op_3'', IT(op_1, op_2)) = IT(Up, Up) = Up$. $TP2$ is then verified.

2. $op_3 = Down$: In this case $op_3' = Up$ and $op_3'' = Down$. Moreover, $IT(op_2, op_1) = IT(op_1, op_2) = Up$. Consequently, $IT(op_3', IT(op_2, op_1)) = IT(Up, Up) = Up$ and $IT(op_3'', IT(op_1, op_2)) = IT(Down, Up) = Up$. That is, $TP2$ is also verified.

Consequently, for all concurrent pairwise concurrent operations op_1 ,op_2 and op_3 belonging to $\{Up, Down\}$, we have

$$\forall\, st \in \{0,1\},\ st \cdot op_1 \cdot IT(op_1, op_2) = st \cdot op_2 \cdot IT(op_2, op_1)$$

Consequently, IT verifies $TP2$. □

Property **IP1** *is violated since* $0 \cdot Down \cdot Up = 1 \neq 0$. *As for property* **IP2** *is violated since* $IT(IT(Down, Down), Up) = Up \neq Down$.

To illustrate the violation of **IP3**, *we consider the Figure 5.3 where we illustrate how to undo* op_1 *of the scenario depicted in Figure 2.2.(b). Initially both sites converge to state* 1. *Suppose now that the operation* op_1 *is undone at both sites. At site 1,* $undo(op_1)$ *generates its inverse* $\overline{op_1} = Down$, *then transforms* $\overline{op_1}$ *against* op_2' *which results in* $IT(\overline{op_1}, op_2') = Up$. *Thus, the final state after undoing* op_1 *is* 1 *at site 1. However, at site* 2, *the execution of* $op_1' = Down$ *at state* 1 *gives the state* 0. *Consequently, both copies diverge. This divergence is due to* **IP3** *violation since* $IT(\overline{op_1}, IT(op_2, op_1)) \neq \overline{IT(op_1, op_2)}$.

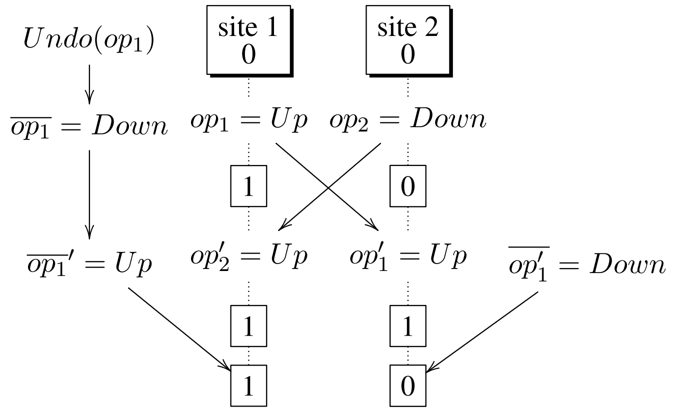


Figure 5.3: **IP3** violation by IT function presented in Algorithm 12.

Accordingly, given a correct transformation function (*i.e.*, they satisfy the transformation properties **TP1** and **TP2**), it is not guaranteed to achieve a correct undo since transformation properties are not sufficient to preserve the data convergence when undoing operations.

Practically, **IP2** violation is discarded by placing inverse operation just after the undone one in the log. The sequence $op \cdot \overline{op}$ is then marked in order to be ignored when transforming another operation against it [97]. Thus the transformation of any operation against an undone operation remains the same. However, the violation of **IP3** cannot be avoided by such a mechanism and must be fulfilled by transformation functions in order to always ensure the data convergence.

5.3 Related Work

Several works proposed undo capability mainly for single user editors. The majority of these solutions are based on log usage. In general, operations are stored in the log according to their execution order. Consequently, many familiar single-user applications such as text editors and design tools allow for undoing operations in chronological order.

In distributed collaborative applications, shared data is usually replicated in order to achieve high local responsiveness as well as high availability. Each user has the ability to perform operations concurrently to other users; which makes undo more and more challenging [22, 74, 97]. In fact, undoing the last operation leads to undoing different operations since the last operation is not the same at all sites (logs are not equal in a collaborative application). Furthermore, undo may be needed to enforce security (see Chapter 3). This obviously motivates the importance of providing a selective undo solution for collaborative applications.

Significant works have been made to address OT-base selective undo [74, 77, 97, 101]. However, the proposed solutions either do not allow to undo any operation *e.g* [77] or is not efficient for real time since it takes time *e.g* [101].

In the following, we summarize the different solutions proposed to undo an operation in single-user editors as well as collaborative editors.

5.3.1 Proposed Undo Solutions for Single-user Editors

Single-step undo. Single-step undo is a feature available in many systems (Macintosh and Windows) as well as editors such *vi* [74]. It allows to undo only the last operation. To further illustrate this approach, consider the following log:

$$op_1 \cdot op_2 \cdot op_3 \cdot op_4$$

In single-step undo approach, it is possible to undo op_4 but a subsequent undo of op_3 is not allowed. To redo the last, the user has simply to undo the last undo it has originated.

Linear undo model and US & R model. The linear undo model [7, 104] allows to undo a sequence of operations by the use of a pointer indicating the following operation to be undone [74]. After undoing a sequence of operations, it is possible to perform new operations then to redo the undone ones.

The US & R (Undo Skip Redo) model [110] is similar to the linear model except the fact that it allows to skip some operations during the redo. This approach is based on the use of a tree structure of the log in order to be able to restore state at any point in the history.

Both approaches are limited by the necessity of undoing the sequence of operations following the operation to be undone then redoing them.

History undo. Like the linear and US & R model, the history model undoes a sequence of operations. However, it appends the inverse operations at the end of the log. For instance, given the following log

$$op_1 \cdot op_2 \cdot op_3 \cdot op_4$$

Undoing op_4 results in

$$op_1 \cdot op_2 \cdot op_3 \cdot op_4 \cdot \overline{op_4}$$

with op_3 as the following operation to be undone. If a normal operation is performed let op_5, the log will be:

$$op_1 \cdot op_2 \cdot op_3 \cdot op_4 \cdot \overline{op_4} \cdot op_5$$

where op_5 is the following operation to be undone. Then if two undo operations are performed, the log is as follows:

$$op_1 \cdot op_2 \cdot op_3 \cdot op_4 \cdot \overline{op_4} \cdot op_5 \cdot \overline{op_5} \cdot \overline{\overline{op_4}}$$

The history undo scheme is used in the *Emacs* editor [92].

5.3.2 Undo Solutions for Collaborative Editors

Various group undo solutions have been proposed. We focus on the following main solutions:

Swap then undo. The first selective undo was proposed in [74]. It consists on placing the selected operation in the end of the history by swapping and then executing its inverse. Consider the state resulting from the following sequence of operations applied on the initial state st_0.

$$st_1 = st_0 \cdot op_1 \cdot op_2 \cdot op_3$$

Undoing op_1 would require to transpose op_1 against op_2 then against op_3 in order to place it at the end of the log then to generate the inverse of the resulting operation. The log will be represented by the following sequence:

$$op'_2 \cdot op'_3 \cdot op'_1 \cdot \overline{op'_1}$$

where op'_2, is the result of transposing op_1 and op_2 and op'_3 is the result of transposing op''_1 and op_3, op''_1 being the result of transposing op_1 with op_2. However in some cases, it may be impossible to swap two operations. For example inserting an element could never be executed after its deletion since the latter operation depends on the former one. To avoid this issue, authors defined a boolean function $conflict(op_x, op_y)$ (where op_x and op_y are successive operations in the log) in order to abort the undo procedure in conflicting situation like that presented above. Hence, the proposed solution does not allow to undo any operation.

Undo/Redo. Another solution is to undo all the operations in the reverse chronological order, $i.e$ from the last to the wanted operation as it is proposed in [77]. So, considering the same example presented above:

$$st_1 = st_0.op_1.op_2.op_3$$

Undoing op_1 would require to undo op_3, op_2 then op_1. The following step imposes to redo op_2 and op_3 in order to restore the last state that excludes op_1's effect. The resulting log will look like:

$$op_1.op_2.op_3.\overline{op_3}.\overline{op_2}.\overline{op_1}.op_2'.op_3'$$

with op_2' and op_3' the new forms of op_2 and op_3 that exclude the effect of op_1.

It is clear that this approach avoids the conflict faced in the first solution. However, it is expensive since it requires to perform many steps to achieve undo. Moreover, it does not allow undo in all cases. In fact, an operation may not be undoable if another later operation performed by the same user has not been undone.

Ferrié's Approach. The undo approach presented in [36] is the same as the work given in [77]. It uses the transformation functions of the algorithm SOCT2 presented in [88, 94]. This algorithm assumes that TP2 is verified, however a counter example showing TP2 violation was presented in [47]. Furthermore, another limitation of the solution is its quadratic complexity.

UNO. The approach of UNO [114] resembles that of Ferrie [36] but is based on TTF [70]. Even though it has a linear complexity with the size of the shared document, the proposed solution only suites a special kind of RCE based on TTF and having the set of operations fixed by the authors (ins, del and undel operations). Moreover, the convergence of UNO assumes the intention preservation which is not proved formally [88].

ANYUNDO-X and COT. The ANYUNDO-X proposed in [97] was the first solution allowing the undo of any operation and solving the known undo problematic. However it has an exponential linear complexity which degrades the performance of a real time application.

Both AnyUndo and COT [101] supports integrated Do and selective Undo. In this approach undo is interpreted as concurrent inverse as in [77] except that an operation op is coupled with its inverse $\overline{op}$ such that they behave as an identity operation. In COT, a contextual relation is introduced to illustrate the relation between an operation, its inverse and the transformed intermediates forms of the inverse. The time complexity is also exponential in the log size.

The difference between ANYUNDO [97] and COT [101] is that the latter discusses the undo in the case of causally dependent operations and not only concurrent ones. However, the solution may violate the effect undo property [97]. This property ensures that the state after undoing a given operation op is the state as if op does not occur. However, if we consider a sequence of characters "abc" and two operations $op_1 = Del(2, b)$ and $op_2 = Ins(2, x)$. The COT-Undo algorithm proceeds as follows: We first generate the inverse of op_1 let $\overline{op_1} = Ins(2, b)$, then we must transform $\overline{op_1}$ against op_2. However, this transformation may lead to inserting character "y" after "b" which is not the same state if op_1 were not performed at the state. This violates the undo effect defined in [97].

ABTU. The ABTU algorithm is presented in [88] where authors propose an undo solution for collaborative editors. The proposed solution is based on the transformation algorithm ABT [60]. Even though the proposed algorithm has a linear complexity, it does not allow to undo any operation since undo is aborted in some cases defined by authors. The transformation algorithm ABT is based on a novel notion named *effect relation* allowing to order document updates in the log. Consequently, all updates are ordered according to their effect relation on the shared document state. Authors assume that this relation ensures convergence. However, in Figure 5.4, we show that there is a divergence case.

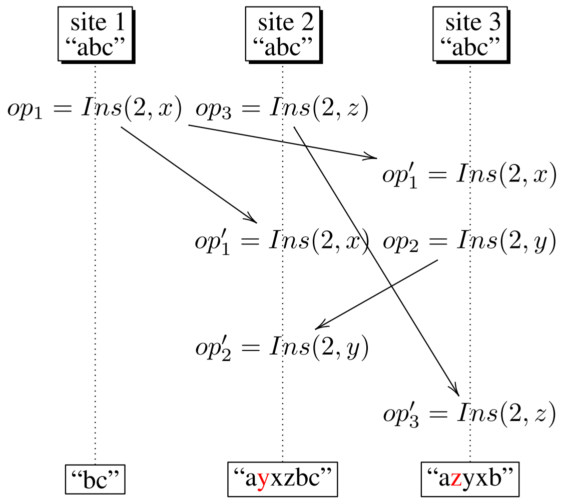


Figure 5.4: Divergence case of the ABTU algorithm.

In this Figure, three sites 1, 2 and 3 begin the collaboration with the same initial sequence of characters "abc" as well as empty logs say H_1, H_2 and H_3 respectively. To simplify, we suppose that the number of the site reflects its identity. Site 1 generates the operation $op_1 = Ins(2, x)$ to derive the state "axbc" and site 2 generates $op_3 = Ins(2, z)$ and gets the state "azbc". Then site 2 receives op_1 as a remote operation from site 1. Note that $op_1 \prec op_3$ according to the effect relation defined by authors (the two operation insert characters at the same position and the identity of site 1 is less than the identity of site 2), the transformed form of op_1 is then $op'_1 = Ins(2, x)$ and the log H_2 is reorganized and updated to be $H'_2 = op'_1(Ins(2, x)) \cdot op'_3(Ins(3, z))$ in order to respect the effect relation. When op_1 arrives at site 3, it is executed as is since the log is empty, $op'_1 = Ins(2, x)$ and gives the state "axbc". Then site 3 generates op_2 which is less than op_1 according to the effect relation, *i.e* $op_1 \succ op_2$. Consequently, the log is updated to $H'_3 = op_2(Ins(2, y)) \cdot op'_1(Ins(3, x))$. Now, when op_3 is received it verifies $op_3 \prec op_2$ (inserting at the same position and site identity of site 2 less than the site identity of site 3), then it is executed as is, the log is updated to $H''_3 = op_3(Ins(2, z)) \cdot op''_2(Ins(3, y)) \cdot op''_1(Ins(4, x))$ and the final state is "azyxbc". On the other side, when op_2 is received at site 2, the log is $H'_2 = op'_1(Ins(2, x)) \cdot op'_3(Ins(3, z))$. The remote integration of op_2 gives $op'_2 = ins(2, y)$. In fact, since $op_2 \prec op_1$, it is executed as is and inserted at position 0 in H'_2. The state is then "ayxzbc" at site 2 which is obviously a divergence since the last state at site 3 is "azyxbc".

Note that the divergence is due to the difference between the two logs at site 2 and 3. Indeed, final logs are $H''_2 = op_2 \cdot op_1 \cdot op_3$ whereas at site 3, we have $H''_3 = op_3 \cdot op_2 \cdot op_1$. However, authors assumed that the effect relation ensure the same order at all sites.

Table 5.1 resumes the main solutions in the context of collaborative editors and compare them according to the following criteria:

1. Simplicity: the simplicity of the algorithm is of relevance and allows its comprehension, verification and integration in any collaborative editing framework.

2. Undo any operation: it is important to allow the undo of any operation, since we need to use the undo procedure in order to enforce the policy (see Chapter 3). If the undo of an illegal operation is abort, we inevitably encounter a security issue and may diverge.

3. Undo causally dependent operations: Undo must allow not only the undo of concurrent operations but also causally dependent ones which is challenging since the undo is based on OT functions that are only defined for concurrent operations.

4. Complexity: the complexity of the undo solution is very important and has to be good since we need to apply it for real time collaborative editors.

5. Correctness: any undo solution must be correct and respect the undo properties defined in Section 5.2.3. A given solution is also considered incorrect if it is based on an incorrect OT-framework even though it respects undo properties.

Criteria	Swap then undo	Undo/Redo	Ferrié	UNo	ANYUNDO/X	COT	ABTU
Simple	Yes	Yes	Yes	Yes	No	No	No
Undo any operation	No	No	No	Yes	Yes	Yes	No
Undo causally dependent operations	No	No	No	No	No	Yes	No
Complexity	$O(n)$	$O(2n)$	$O(n^2)$	$O(n)$	$O(e^n)$	$O(e^n)$	$O(n)$
Correct	Yes	Yes	No	Yes	Yes	No	No

Table 5.1: Comparison Between Undo Algorithms

5.4 Conclusion

In this chapter, we presented the most important aspects of the undo principle and most importantly, inverse properties in order to be able to well study this command with the aim of providing a generic solution in the following chapters. Moreover, we focused on the main solutions proposed for undoing operation in collaborative applications mainly collaborative editors and stressed the limits of these solutions.

In Chapter 6, we provide a theoretical study for undo problem and show what are the necessary and sufficient conditions allowing to achieve a correct undo in distributed collaborative applications.

Chapter 6

A Necessary and Sufficient Condition for Undoability

Contents

Maintaining convergence in access control-based collaborative editors requires a selective *undo* mechanism [21]. However, two obstacles may arise when we selectively undo an operation in a collaborative context. On the one hand, convergence of the shared document may be violated since the operations are out-of-order executed at two different sites. On the other hand, there are many properties to be satisfied, which complicates the verification of the undoability correctness.

In this chapter, we present a theoretical study of the undoability problem in collaborative applications by giving a formal proof of necessary and sufficient conditions for achieving correct undo. Yet OT was proposed to go beyond the commutativity, we prove that under some assumptions, it is impossible to design an undoable object without enforcing the natural commutativity. Proving this result was a difficult task. To overcome this difficulty, we formalized the undoability problem as a Constraint Satisfaction Problem (CSP) where CSPs are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.

However, practically it is not possible to always define a set of commutative operations. For instance, in the context of collaborative editors, inserting characters does not commute with deleting characters. Thus, we propose to extend the set of operations with a new form of the idle operation where semantic information about the transformed operations is encapsulated in the idle operation's signature. The enhanced set of operations has to satisfy both inverse and transformation properties. Thus, we investigate transformation rules for idle and inverse operations to prove these rules preserve all the properties.

In the first section of this chapter, we present the formal statement of the undoability problem. In Section 6.2, we investigate a formal analysis of the undoablity problem based on the CSP theory. We provide a necessary and sufficient condition for achieving correct undoability in the context of OT-based collaborative applications. Finally, in Section 6.3, we sketch a preliminary solution that consists in adding an idle operation in order to keep the OT advantages, namely going beyond the commutativity.

6.1 Formal Problem Statement

Collaborative Object. We suppose that there are N sites collaborating on the same shared object replicated at each site. Every site updates its local copy, executes the update immediately then broadcast the generated operation to other sites. Remote sites compute the new form of remote operations by applying the IT function in order to integrate the effect of the local log into the received remote operation. Then the result of the IT function is executed on the local copy of the receiver site. We define formally the collaborative object as follows:

Definition 6.1.1 (Consistent Collaborative Object)**.**

A *Consistent Collaborative Object* (CCO) is a triplet $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ such that:

- $\mathcal{S}t$ is the set of object states (or the space state);
- $\mathcal{O}p$ is the set of primitive operations executed by the user to modify the object state. This set is characterized by the following properties:
  1. for every operation $op \in \mathcal{O}p$ there is *unique inverse* $\overline{op} \in \mathcal{O}p$ such that $op \neq \overline{op}$ and $st \cdot op \cdot \overline{op} = st$ for all states $st \in \mathcal{S}t$;
  2. for every operation $op \in \mathcal{O}p$ there exists a state $st \in \mathcal{S}t$ such that $st \cdot op = st'$ where $st' \neq st$.
- $IT : \mathcal{O}p \times \mathcal{O}p \to \mathcal{O}p$ is a correct transformation function (*i.e.*, IT satisfies properties $\mathbf{TP1}$ and $\mathbf{TP2}$).

A CCO is of order n, denoted n-CCO, if the size of $\mathcal{O}p$ is equal to n.

According to Definition 6.1.1, property (1) means that all operations have to satisfy the undo property $\mathbf{IP1}$. This property should be preserved even though the operation is generated outside an undo process. As for property (2), it discards the use of idle operations (*i.e.*, there is no $op \in \mathcal{O}p$ such that $st \cdot op = st$ for any state st). Indeed, when designing a shared object, a developer provides intuitively only operations that alter the object state. For him, it does not make sense to handle practically idle operations. As seen in Chapter 5, we can devise consistent objects (i.e. $\mathbf{TP1}$ and $\mathbf{TP2}$ are satisfied) without idle operations (see Examples 5.2.2 and 5.2.4). Also we exclude operations having the same inverse (*i.e*, $op = \overline{op}$) since they are not interesting in practice. Note that example 5.2.4 is not a CCO since $\mathbf{IP1}$ is violated while it is easy to prove that example 5.2.2 is a CCO.

Next we define the undoability for a consistent collaborative object.

Definition 6.1.2 (Undoability)**.**

A consistent collaborative object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ is *undoable* iff its transformation function IT satisfies undo properties $\mathbf{IP2}$ and $\mathbf{IP3}$.

Formal Problem Statement. It is possible to show by counterexamples that not all collaborative objects are undoable *w.r.t.* Definition 6.1.2 (see Example 5.2.4). Then, a natural follow-up question is what make a collaborative object undoable? It can be formally stated as:

The Undoability Problem: Given a consistent collaborative object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, what are the necessary and the sufficient conditions that C is undoable?

In the sequel, all used objects are consistent collaborative objects (see Definition 6.1.1).

6.2 Necessary and Sufficient Condition for Undoability

In this section, we present CCO properties related to the Definition 6.1.1. Furthermore, we present the commutativity property since it represents a sufficient condition for undoability as we will prove in the following section. Finally, we show that commutativity property is also necessary to achieve undoability.

6.2.1 CCO Properties

In the following, given a consistent collaborative object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, we present properties derived from $\mathbf{TP1}$ and $\mathbf{IP1}$.

In Lemma 6.2.1, we present Property $C1$ that discards evaluations of $\mathbf{IT}$ leading to equality between two different operations.

Lemma 6.2.1 (Property $C1$)**.**

For every pairwise operations $op_i, op_j \in \mathcal{O}p$, if $op_i \neq op_j$ then $IT(op_i, op_j) \neq IT(op_j, op_i)$.

Proof. By absurd reasoning, suppose that there exist two different operations $op_i \neq op_j$ from $\mathcal{O}p$ such that $IT(op_i, op_j) = IT(op_j, op_i)$. As IT verifies $\mathbf{TP1}$, we have:

$$\begin{aligned} op_i \cdot IT(op_j, op_i) &\equiv op_j \cdot IT(op_i, op_j) && (\mathbf{TP1}) \\ &\equiv op_j \cdot IT(op_j . op_i) \end{aligned}$$

Performing $\overline{IT(op_j, op_i)}$ would lead to:

$$op_i = op_j$$

Clearly, this is absurd since $op_i \neq op_j$. $\square$

The following property $C1X$ extends $C1$ since it also eliminates evaluations of $\mathbf{IT}$ leading to equality between two different operations by transitivity.

Lemma 6.2.2 (Property $C1X$).

For every different pairwise operations op_i, op_j and op_k from $\mathcal{O}p$. If $IT(op_i, op_j) = IT(op_i, op_k)$ then $IT(op_k, op_i) \neq IT(op_j, op_i)$.

Proof. Since C is a CCO then IT verifies $\mathbf{TP1}$ and $\mathbf{IP1}$. Suppose that there exist three operations op_i, op_j and op_k verifying $IT(op_i, op_j) = IT(op_i, op_k)$ and $IT(op_k, op_i) = IT(op_j, op_i)$. According to $\mathbf{TP1}$, we have

$$op_i \cdot IT(op_j, op_i) \equiv op_j \cdot IT(op_i, op_j)$$

Replacing $IT(op_j, op_i)$ by $IT(op_k, op_i)$ leads to the following result:

$$op_i \cdot IT(op_k, op_i) \equiv op_j \cdot IT(op_i, op_j)$$

Since $op_i \cdot IT(op_k, op_i) \equiv op_k \cdot IT(op_i, op_k)$ according to $\mathbf{TP1}$, we obtain: Consequently,

$$op_k \cdot IT(op_i, op_k) \equiv op_j \cdot IT(op_i, op_j)$$

Replacing $IT(op_i, op_k)$ by $IT(op_i, op_j)$ leads to:

$$op_k \cdot IT(op_i, op_j) \equiv op_j \cdot IT(op_i, op_j)$$

Finally, applying $\overline{IT(op_i, op_j)}$ at both sides leads to $op_j = op_k$ (according to $\mathbf{IP1}$) which is false since we supposed that $op_j \neq op_k$. □

Next, property $C2$ (see Lemma 6.2.3) is directly derived from $\mathbf{IP1}$ and $\mathbf{TP1}$. It ensures that if $IT(op_i, op_j) = \overline{op_j}$ then $IT(op_j, op_i) = \overline{op_i}$ and is formalized as follows:

Lemma 6.2.3 (Property $C2$).

For every different pairwise operations $op_i, op_j \in \mathcal{O}p$, if $IT(op_i, op_j) = \overline{op_j}$ then $IT(op_j, op_i) = \overline{op_i}$.

Proof. Since C is a CCO then IT verifies $\mathbf{TP1}$ and $\mathbf{IP1}$. $\mathbf{IP1}$ ensures that $op \cdot \overline{op} \equiv \emptyset$. Consequently, if $IT(op_i, op_j) = \overline{op_j}$ then for every pairwise operations op_i and op_j from $\mathcal{O}p$, we have:

$$\begin{array}{rcl} op_i \cdot IT(op_j, op_i) & \equiv & op_j \cdot IT(op_i, op_j) \\ op_i \cdot IT(op_j, op_i) & \equiv & op_j \cdot \overline{op_j} \\ op_i \cdot IT(op_j, op_i) & \equiv & \emptyset \end{array}$$

Hence, $IT(op_j, op_i) = \overline{op_i}$ according to $\mathbf{IP1}$.

□

Next, we present property $C3$ in Lemma 6.2.4. This property allows to deduce the transformation result of an operation op_j against op_i under some conditions. It is formalized as follows:

Lemma 6.2.4 (Property $C3$)**.**

For every two operations op_i and op_j from $\mathcal{O}p$ such as $IT(op_j, op_i) \notin \{op_j, \overline{op_i}\}$, if $IT(op_i, op_j) = op_i$ and $\overline{op_i}$ commute with $IT(op_j, op_i)$ then $IT(op_j, op_i) = op_j$.

Proof. Since C is a CCO then IT verifies $\mathbf{TP1}$ and $\mathbf{IP1}$. Consequently, for every state $st \in \mathcal{S}t$, $\forall$ $op_i, op_j \in \mathcal{O}p$, according to $\mathbf{TP1}$, we have

$$op_i \cdot IT(op_j, op_i) \equiv op_j \cdot IT(op_i, op_j)$$

Since, $IT(op_i, op_j) = op_i$, the previous equivalence leads to the following one

$$op_i \cdot IT(op_j, op_i) \equiv op_j \cdot op_i$$

Applying $\overline{op_i}$ at both sides leads to

$$\begin{aligned} op_i \cdot IT(op_j, op_i) \cdot \overline{op_i} &\equiv op_j \cdot op_i \cdot \overline{op_i} \\ op_i \cdot IT(op_j, op_i) \cdot \overline{op_i} &\equiv op_j && (\mathbf{IP1}) \end{aligned}$$

Finally, since $\overline{op_i}$ commutes with $IT(op_j, op_i)$), then $op_i \cdot IT(op_j, op_i) \cdot \overline{op_i} \equiv \overline{op_i} \cdot IT(op_j, op_i)$. This lead to

$$op_i \cdot \overline{op_i} \cdot IT(op_j, op_i) \equiv op_j$$

Since $op_i \cdot \overline{op_i} \equiv \emptyset$ (according to $\mathbf{IP1}$), we deduce that $IT(op_j, op_i) = op_j$. Then the result is obtained. Box □

Finally, property $C4$ enforces the difference between an operation and its inverse.

Lemma 6.2.5 (Property $C4$)**.**

For every operations op_i and $op_j \in \mathcal{O}p$, if $IT(op_i, op_j) = \overline{op_i}$ then $IT(op_j, op_i) \neq op_j$.

Proof. If $IT(op_i, op_j) = \overline{op_i}$ for every operations op_i and op_j from $\mathcal{O}p$, then $IT(op_j, op_i) = \overline{op_j} \neq op_j$. Then the result is obtained. □

These properties allow us to well define the constraints to be satisfied by an IT function. The rest of this section is devoted to the commutativity property since it is sufficient to achieve undoability as we will show later.

Definition 6.2.3 (Commutativity)**.**

Two operations op_1 and op_2 commute iff $op_1 \cdot op_2 \equiv op_2 \cdot op_1$.

In the following, we say that a set of operation $\mathcal{O}p$ is commutative if all of its operations are pairwise commutative.

Commutativity property given in Definition 6.2.3 is strong in the sense that it enables us to reorder any pair of operations whatever they are concurrent or causally dependent. Instead, in collaborative applications, we just need to verify whether pairwise concurrent operations commute or not. The impact of commutativity on IT function is shown in the following Theorem:

Theorem 6.2.1.

For any pairwise concurrent operations $op_1, op_2 \in \mathcal{O}p$, op_1 commute with op_2 iff $IT(op_i, op_j) = op_i$, $i = 1, 2$.

Proof. As the transformation function IT is correct (see Definition 6.1.1), then IT satisfies $\mathbf{TP1}$. That is, $op_1 \cdot IT(op_2, op_1) \equiv op_2 \cdot IT(op_1, op_2)$. Since, $IT(op_2, op_1) = op_2$ and $IT(op_1, op_2) = op_1$, we deduce from the previous equivalence that $op_2 \cdot op_1 \equiv op_1 \cdot op_2$. Consequently, op_1 commutes with op_2. Moreover, if $\mathcal{O}p$ is commutative then for every two operations op_1 and op_2 from $\mathcal{O}p$, we have $op_1 \cdot op_2 \equiv op_2 \cdot op_1$. Consequently, $IT(op_1, op_2) = op_1$ and $IT(op_2, op_1) = op_2$ according to $\mathbf{TP1}$. Hence, for any pairwise concurrent operations $op_1, op_2 \in \mathcal{O}p$, op_1 commute with op_2 iff $IT(op_i, op_j) = op_i$, $i = 1, 2$. □

It is worth mentioning that given any 2-CCO, the equivalence between operation sequence and its inverse leads to commutativity. In Lemma 6.2.6, we show that if an operation op verifies $op \cdot op \equiv \overline{op} \cdot \overline{op}$ then op commutes with its inverse $\overline{op}$.

Lemma 6.2.6.

Given an object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ where $\mathcal{O}p = \{op_1, op_2\}$, $op_1 = \overline{op_2}$ and $\overline{op_2} = op_1$. If $op_1 \cdot op_1 \equiv op_2 \cdot op_2$ then $\mathcal{O}p$ is commutative.

Proof. Since $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ is a CCO then $\mathbf{IP1}$ holds. Thus, we can deduce the following sequences when applying op_1 twice on the left and right:

$$\begin{array}{lcl}
op_1 \cdot op_1 & \equiv & op_2 \cdot op_2 \\
op_1 \cdot op_1 \cdot op_1 \cdot op_1 & \equiv & op_2 \cdot op_2 \cdot op_1 \cdot op_1 \\
op_1 \cdot \underbrace{op_1 \cdot op_2}_{IP1} \cdot op_2 & \equiv & op_2 \cdot \underbrace{op_2 \cdot op_1}_{IP1} \cdot op_1 \\
op_1 \cdot op_2 & \equiv & op_2 \cdot op_1
\end{array}$$

consequently, op_1 and op_2 commute. Therefore, the IT function can be defined by $IT(op_i, op_j) = op_i$ for all pairwise concurrent operations from $\mathcal{O}p$. □

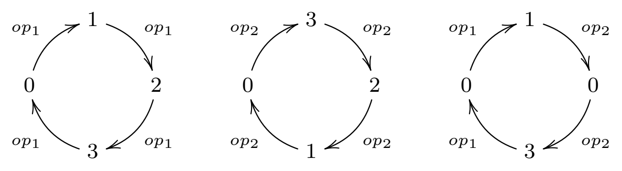
Example 6.2.1. *To illustrate Lemma 6.2.6, consider the following operations*

$op_1 : x \quad \mapsto \quad (x+1) mod\, 4$

$op_2 : x \quad \mapsto \quad (x-1) mod\, 4$

that modifies the state of a natural number. Let 0 be the initial state, the set of states in this case is $\mathcal{S}t = \{0, 1, 2, 3\}$. *Each of these operations is periodic of periodicity 4 since* $\forall x \in \mathcal{S}t$, $x \cdot op_1 \cdot op_1 \cdot op_1 \cdot op_1 = x$

for every state $x \in \mathcal{S}t$ (see Figures 6.1.(1) and 6.1.(2)). Moreover, $op_1 \cdot op_1 \equiv op_2 \cdot op_2$, and obviously, op_1 commutes with op_2 (see Figure 6.1.(3)).



(1) op_1 is 4-periodic. (2) op_2 is 4-periodic (3) op_1 commutes with op_2.

Figure 6.1: Commutative transformation for a 4-periodic set

The question that arises here is whether the commutativity property is sufficient to achieve undoability or not. In the following, we answer this question by showing that for a given consistent object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, if $IT(op_i, op_i) = op_j$ for all op_i and op_j from $\mathcal{O}p$ then C is undoable (see Lemma 6.2.7).

Lemma 6.2.7 (Commutativity implies undoability)**.**

Given an object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, if $\mathcal{O}p$ is commutative then C is undoable.

Proof. To prove that C is undoable we have to verify that **IP2** and **IP3** conditions are preserved. Since $\mathcal{O}p$ is commutative then for all two operations op_i and op_j from $\mathcal{O}p$, we have

$$\begin{aligned} IT(IT(op_i, op_j), \overline{op_j}) &= IT(op_i, \overline{op_j}) \\ &= op_i \end{aligned}$$

Then, **IP2** is preserved. As for **IP3**, it is preserved since,

$$\begin{aligned} IT(\overline{op_i}, IT(op_j, op_i)) &= IT(\overline{op_i}, op_j) && op_j \text{ commutes with } op_i \\ &= \overline{op_i} && \overline{op_i} \text{and } op_j \text{ commute} \\ &= \overline{IT(op_i, op_j)} && op_i \text{ and } op_j \text{ commute} \end{aligned}$$

□

The next step consists in proving that also undoability implies commutativity. For the sake of our results, we formalize our problem as a CSP, taking into account the properties C1-4 presented above.

6.2.2 CSP Model

Since our objective consists in finding necessary and sufficient conditions for achieving correct undoability, we have to find all the evaluations of an IT function that satisfy both transformation and undoability properties, namely **TP1**, **TP2**, **IP1**, **IP2** and **IP3**. Finding such evaluations is achieved by formalizing the undoability problem as a *Constraint Satisfaction Problems* (CSP) problem in order to determine easily all the possible values for any correct IT function satisfying undo properties.

Indeed, CSPs [108] are mathematical problems defined as a set of objects whose state must satisfy a number of constraints. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyse and solve problems of many unrelated families. CSPs aims at exhibiting high complexity since they are solved in a reasonable time thanks to the combination of heuristics and combinatorial search methods. Formally, a CSP is defined as follows[15].

Definition 6.2.4 (CSP).

A CSP is defined as a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of problem variables,
- $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ is set of of domain values for every the variable, *i.e.* for every $k \in [1; n]$, $x_k \in \mathcal{D}_k$, and
- $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ is a set of constraints, where every constraint $\mathcal{C}_i$ is in turn a pair $\langle \mathcal{X}_i, \mathcal{R}_i \rangle$(usually represented as a matrix), where $\mathcal{X}_i$ is a tuple of variables and $\mathcal{R}_i$ is a set of tuples of values.

All these tuples having the same number of elements; as a result R is a relation. An evaluation of the variables is a function from variables to values, $v : \mathcal{X} \rightarrow \mathcal{D}$. Such an evaluation satisfies a constraint $\langle (\mathcal{X}_{i1}, \ldots, \mathcal{X}_{in}), \mathcal{R}_i \rangle$ if $(v(\mathcal{X}_{i1}), \ldots, v(\mathcal{X}_{in})) \in \mathcal{R}$. A solution is an evaluation that satisfies all constraints.

Example 6.2.2. *One of the most famous CSP problems is the eight queen puzzle (see Figure 6.2[16]). It consists in placing eight chess queens on an 8×8 chessboard so that no two queens attack each other. consequently, a solution of this problem requires that no two queens share the same row, column, or diagonal [34].*

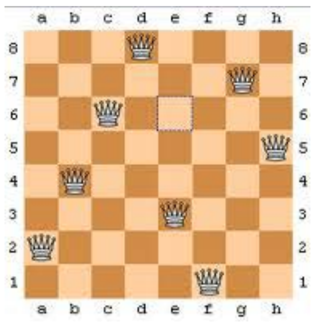


Figure 6.2: One solution to the eight queens puzzle.

In the following, we formalize the undoability problem as a CSP:

The set of variables. In our case $\mathcal{X}$ is the set of different possible values of IT for a given set of operations $\mathcal{O}p = \{op_1, \ldots, op_n\}$. It is formally represented by $\mathcal{X} = \{IT(op_i, op_j) | \; op_i, \, op_j \in \mathcal{O}p\}$. To simplify our model, we consider $\mathbb{N}$ as the domain of operations. Consequently, IT could be represented by a matrix where op_i and op_j refer the row i and the column j while $IT(op_i, op_j)$ refers to the value of $IT[i, j]$ (see table 6.1).

[15]http://en.wikipedia.org/wiki/Constraint_satisfaction_problem.

[16]http://en.wikipedia.org/wiki/Eight_queens_puzzle

The domain. The domain of values is the set of values taken by the IT function. Obviously, we have $\mathcal{D} = \mathcal{O}p$.

Constraints. Finally, our constraints are the conditions to be satisfied by the IT function so that an evaluation is true. More precisely, constraints are $\mathbf{TP2}$, $\mathbf{IP2}$, $\mathbf{IP3}$, and the properties $C1-4$ derived from $\mathbf{TP1}$ and $\mathbf{IP1}$ (see Lemmas 6.2.1-6.2.5) since they could not be expressed with equalities between different variables from $\mathcal{X}$. Thus, $\mathcal{C} = \{\mathbf{TP2}, \mathbf{IP2}, \mathbf{IP3}, C1, C1X, C2, C3, C4\}$.

These constraints could be represented by a matrix IT of size n^2 where $n = |\mathcal{O}p|$ as shown in Table 6.1. Subsequently the set of evaluations has the size n^{n^2}.

IT	op_1	$\dots$	op_i	$\dots$	op_j	$\dots$	op_n
op_1							
$\dots$							
op_i			$IT(op_i, op_i)$		$IT(op_j, op_i)$		
$\dots$							
op_j			$IT(op_i, op_j)$		$IT(op_j, op_j)$		
$\dots$							
op_n							

Table 6.1: Transformation Matrix

The question that arises here is *whether an undoable CCO is commutative or not*? To answer this question, we begin bay studying CCO of order 2 and 4 as the basic cases in order to generalize the result by induction on the CCO's order. To solve the undoability problem for 2-CCO and 4-CCO cases and calculate all the evaluations of IT that respect the mentioned constraints in a reasonable time, we use the Choco solver [17]. The solutions given by the solver are presented in Figures 6.3 and 6.4 where we have considered consistent collaborative objects of order 2 and 4 respectively.

Consider the first case of CCOs with two operations. If a 2-CCO is undoable then the only two IT evaluations are (see Figure 6.3):

1. IT_1^2 verifies $IT(op_i, op_j) = op_i, i = 1,2$ which means op_1 commutes with op_2 according to Definition 6.2.1. For instance, Example 5.2.2 is an example of an undoable CCO of order 2 where its operations set is commutative.

2. IT_2^2 verifies $IT(op_i, op_j) = \overline{op_i}, i = 1,2$, in which case $\mathbf{TP1}$ implies that $op_1 \cdot op_1 \equiv op_2 \cdot op_2$ which means that $\mathcal{O}p$ is commutative according to Lemma 6.2.6.

Accordingly, an undoable 2-CCO is commutative.

(1) Solution 1

IT_1^2	op_1	op_2
op_1	op_1	op_1
op_2	op_2	op_2

(2) Solution 2

IT_2^2	op_1	op_2
op_1	op_2	op_1
op_2	op_1	op_2

Figure 6.3: Solutions of the 2-CCO problem.

As for 4-CCOs, we present the output of our solver in Figure 6.4. We can show easily that the first four evaluations leads to commutativity. Indeed, $\mathbf{TP1}$ enforces equivalence relationship between different sequences of operations according to the evaluation of IT. To more illustrate, we consider every solution and state different equivalences dictated by $\mathbf{TP1}$. Then, we prove these equivalences lead to commutativity.

[17] http://www.emn.fr/z-info/choco-solver

(1) Solution 1

IT_1^4	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

(2) Solution 2

IT_2^4	op_1	op_2	op_3	op_4
op_1	op_1	op_1	op_1	op_1
op_2	op_2	op_3	op_3	op_2
op_3	op_3	op_2	op_2	op_3
op_4	op_4	op_4	op_4	op_4

(3) Solution 3

IT_3^4	op_1	op_2	op_3	op_4
op_1	op_4	op_1	op_1	op_4
op_2	op_2	op_2	op_2	op_2
op_3	op_3	op_3	op_3	op_3
op_4	op_1	op_4	op_4	op_1

(4) Solution 4

IT_4^4	op_1	op_2	op_3	op_4
op_1	op_4	op_1	op_1	op_4
op_2	op_2	op_3	op_3	op_2
op_3	op_3	op_2	op_2	op_3
op_4	op_1	op_4	op_4	op_1

(5) Solution 5

IT_5^4	op_1	op_2	op_3	op_4
op_1	op_2	op_2	op_2	op_2
op_2	op_1	op_1	op_1	op_1
op_3	op_4	op_4	op_4	op_4
op_4	op_3	op_3	op_3	op_4

(6) Solution 6

IT_6^4	op_1	op_2	op_3	op_4
op_1	op_2	op_4	op_2	op_4
op_2	op_3	op_1	op_3	op_1
op_3	op_4	op_2	op_4	op_2
op_4	op_1	op_3	op_1	op_3

(7) Solution 7 (sym to 5)

IT_7^4	op_1	op_2	op_3	op_4
op_1	op_3	op_3	op_3	op_3
op_2	op_4	op_4	op_4	op_4
op_3	op_1	op_1	op_1	op_1
op_4	op_2	op_2	op_2	op_2

(8) Solution 8 (sym to 6)

IT_8^4	op_1	op_2	op_3	op_4
op_1	op_3	op_3	op_2	op_2
op_2	op_4	op_4	op_3	op_3
op_3	op_2	op_2	op_1	op_1
op_4	op_1	op_1	op_2	op_2

(9) Solution 9

IT_9^4	op_1	op_2	op_3	op_4
op_1	op_1	op_4	op_4	op_1
op_2	op_3	op_2	op_2	op_3
op_3	op_2	op_3	op_3	op_2
op_4	op_4	op_1	op_1	op_4

(10) Solution 10

IT_{10}^4	op_1	op_2	op_3	op_4
op_1	op_1	op_4	op_4	op_1
op_2	op_3	op_3	op_3	op_3
op_3	op_2	op_2	op_2	op_2
op_4	op_4	op_1	op_1	op_4

(11) Solution 11

IT_{11}^4	op_1	op_2	op_3	op_4
op_1	op_4	op_2	op_4	op_2
op_2	op_1	op_3	op_1	op_3
op_3	op_2	op_4	op_2	op_4
op_4	op_3	op_1	op_3	op_1

(12) Solution 12 (sym to 11)

IT_{12}^4	op_1	op_2	op_3	op_4
op_1	op_4	op_4	op_3	op_3
op_2	op_3	op_3	op_4	op_4
op_3	op_1	op_1	op_2	op_2
op_4	op_2	op_2	op_1	op_1

(13) Solution 13 (sym to 10)

IT_{13}^4	op_1	op_2	op_3	op_4
op_1	op_4	op_4	op_4	op_4
op_2	op_3	op_2	op_2	op_3
op_3	op_2	op_3	op_3	op_2
op_4	op_1	op_1	op_1	op_1

(14) Solution 14

IT_{14}^4	op_1	op_2	op_3	op_4
op_1	op_4	op_4	op_4	op_4
op_2	op_3	op_3	op_3	op_3
op_3	op_2	op_2	op_2	op_2
op_4	op_1	op_1	op_1	op_1

Figure 6.4: Solutions of the 4-CCO problem.

The transformation function IT_1^4 implies commutativity according to Definition 6.2.1. Additionally, the next three solutions do not verify $IT(op_i, op_j) = op_i \ \forall op_i, op_j \in \mathcal{O}p$ but according to $\mathbf{TP1}$ leads to commutativity. Indeed, IT_2^4 requires the following equivalences dictated by the property $\mathbf{TP1}$:

- $op_2 \cdot op_1 \equiv op_1 \cdot op_2$, $op_4 \cdot op_1 \equiv op_1 \cdot op_4$ and $op_4 \cdot op_2 \equiv op_2 \cdot op_4$ which means that $\{op_1, op_2, op_4\}$ is commutative.
- $op_3 \cdot op_1 \equiv op_1 \cdot op_3$ and $op_4 \cdot op_3 \equiv op_3 \cdot op_4$ which means that op_3 commutes with op_1 and op_4.
- $op_2 \cdot op_2 \equiv op_3 \cdot op_3$ which means that $\{op_2, op_3\}$ is commutative according to Lemma 6.2.6.

Consequently, $\mathcal{O}p$ is commutative.

Similarly, the transformation function IT_3^4 implies:

- $op_2 \cdot op_1 \equiv op_1 \cdot op_2$, $op_3 \cdot op_1 \equiv op_1 \cdot op_3$ and $op_3 \cdot op_2 \equiv op_2 \cdot op_3$ then $\{op_1, op_2, op_3\}$ is commutative.
- $op_4 \cdot op_2 \equiv op_2 \cdot op_4$ and $op_4 \cdot op_3 \equiv op_3 \cdot op_4$ so op_4 commutes with op_2 and op_3.
- $op_4 \cdot op_4 \equiv op_1 \cdot op_1$: which means that op_4 commutes with op_1 according to Lemma 6.2.6.

Thus, $\mathcal{O}p$ is commutative.

Finally, the solution IT_4^4 imposes that:

- $op_2 \cdot op_1 \equiv op_1 \cdot op_2$ and $op_3 \cdot op_1 \equiv op_1 \cdot op_3$ then op_1 commutes with op_2 and op_3.
- $op_4 \cdot op_2 \equiv op_2 \cdot op_4$ and $op_4 \cdot op_3 \equiv op_3 \cdot op_4$ then op_4 commutes with op_2 and op_3.
- $op_2 \cdot op_2 \equiv op_3 \cdot op_3$ which means that $\{op_2, op_3\}$ is commutative according to Lemma 6.2.6.

- $op_4 \cdot op_4 \equiv op_1 \cdot op_1$ which means that $\{op_4, op_1\}$ is commutative according to Lemma 6.2.6.

The remaining solutions (Solutions 5-10) provided by our CSP solver either transform an operation to its inverse or to the inverse of the concurrent operations, *i.e.* there exist $op_i, op_j \in \mathcal{O}p$ verifying $IT(op_i, op_j) = \overline{op_i}$ or $IT(op_i, op_j) = \overline{op_j}$. Even though these solutions do not lead to commutativity ($e.g$ solution 5 detailed in Example 6.2.3) they are not practically interesting since the goal of IT is to integrate the effect of concurrent operations and not to undo their effects. Subsequently, it is preferable to use the first IT evaluation ($IT(op_i, op_j) = op_i, \forall op_i, op_j \in \mathcal{O}p$) as it integrates the operation's effect after transformation.

Example 6.2.3. *Let op_1 and op_2 be two operations over square matrices of order 2 $M_{2,2}$, such that:*

$$\begin{array}{llll} op_1: & M_{2,2} & \to & M_{2,2} \\ & A & \mapsto & 2 \times^t A \end{array}$$

where ${}^tA$ is the transpose of the matrix A and

$$\begin{array}{llll} op_2: & M_{2,2} & \to & M_{2,2} \\ & A & \mapsto & 2 \times \begin{pmatrix} a_{0,1} & a_{0,0} \\ a_{1,0} & a_{1,1} \end{pmatrix} \end{array}$$

Let op_3 and op_4 be the inverse of op_2 and op_1 respectively. Obviously, $op_3 = \frac{1}{2} \times op_2$ and $op_4 = \frac{1}{2} \times op_1$ Consider the transformation function IT_5^4 given by our CSP solver (see Figure 6.4.(5)) summarized as follows:

- $IT(op_1, op) = op_2$
- $IT(op_2, op) = op_1$
- $IT(op_3, op) = op_4$
- $IT(op_4, op) = op_3$

where $op \in \mathcal{O}p = \{op_1, \overline{op_1}, op_2, \overline{op_2}\}$.

According to $\mathbf{TP1}$*, we have:*

$$op_2 \cdot op_2 \equiv op_1 \cdot op_1 \tag{6.1}$$
$$\overline{op_2} \cdot op_1 \equiv op_2 \cdot \overline{op_1} \tag{6.2}$$
$$\overline{op_1} \cdot op_2 \equiv op_1 \cdot \overline{op_2} \tag{6.3}$$
$$\overline{op_1} \cdot \overline{op_1} \equiv \overline{op_2} \cdot \overline{op_2} \tag{6.4}$$

Note that $\mathcal{O}p$ respects these properties for instance for every matrix A, we have $A \cdot op_1 \cdot op_1 = 4 \times A$ and $A \cdot op_2 \cdot op_2 = 4 \times A$. Hence (6.1) is satisfied. Similarly, (6.4) is correct. As for equivalence (6.2), it is satisfied since for every matrix $A \in M_{2,2}$, $\overline{op_2} \cdot op_1 = \frac{1}{2} \times op_2 \cdot op_1$ and $op_2 \cdot \overline{op_1} = op_2 \cdot \frac{1}{2} \times op_1$. Clearly, $\frac{1}{2} \times op_2 \cdot op_1 \equiv op_2 \cdot \frac{1}{2} \times op_1$. Similarly, equivalences (6.3) is correct. It should be pointed out that op_1 does not commute with op_2. Indeed, consider the matrix $A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix}$: Applying op_1 to A produces $A_1 = 2 \times \begin{pmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{pmatrix}$. Then $A_3 = A_1 \cdot op_2 = 4 \times \begin{pmatrix} a_{1,0} & a_{0,1} \\ a_{0,1} & a_{1,1} \end{pmatrix}$. Also $op_2 \cdot A$ produces $A_2 = 2 \times \begin{pmatrix} a_{0,1} & a_{0,0} \\ a_{1,0} & a_{1,1} \end{pmatrix}$. Then applying op_1 to A_2 produces $A_4 = 4 \times \begin{pmatrix} a_{0,1} & a_{1,0} \\ a_{0,0} & a_{1,1} \end{pmatrix}$. Clearly, op_1 does not commute with op_2 since $A_3 \neq A_4$.

Even though the set is not commutative, the transformation function leading to such set of operations (solution 5) imposes that $IT(op_i, op_j) = \overline{op_j}$ with $op_i \neq op_j$. Such a transformation function does not make sense since it just undo the effect of the performed operation when receiving a concurrent remote operation. Hence, after synchronizing all operations, all users will loose their updates.

Consequently, we improve our CSP model with the following two constraints so to avoid IT evaluations that are not interesting in practice and that do not keep the advantages of the OT approach.

In Definition 6.2.5, we present property $C5$ that forbids transforming an operation to its inverse:

Definition 6.2.5 (Property C5)**.**

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ then for every operations op_i and op_j from $\mathcal{O}p$, it must be that

$$IT(op_i, op_j) \neq \overline{op_i}.$$

As for property $C6$, it discards IT functions transforming an operation op_1 against op_2 to the inverse of op_2. Indeed, this is equivalent to undoing the effect of op_2 which is far from being the objective of OT approach leading to integrating the effect of op_1 into op_2.

Definition 6.2.6 (Property C6)**.**

given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ then for every operations op_i and op_j from $\mathcal{O}p$, if $op_j \notin \{op_i, \overline{op_i}\}$ then $IT(op_i, op_j) \neq \overline{op_j}$.

Accordingly, we extend the set of constraints $\mathcal{C}$ with $C5$ and $C6$ in order to keep only the solutions that are practically interesting.The results of the new CSP model using Choco solver for both 2 and 4-CCOs are presented in Figure 6.5. Obviously, we can observe that an undoable CCO of order 2 or 4 implies that its transformation function verifies $IT(op_i, op_j) = op_i$ for every pairwise operations op_i and op_j from $\mathcal{O}p$. Thus, $\mathcal{O}p$ is commutative. In the following, we generalise this result and prove that undoability is equivalent to commutativity.

(1) 2-CCO Solution

IT	op_1	op_2
op_1	o_1	op_1
op_2	op_2	op_2

(2) 4-CCO Solution

IT	op_1	op_2	op_3	op_4
op_1	op_1	op_2	op_3	op_4
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 6.5: Solutions of the 2-CCO problem.

6.2.3 Necessary and Sufficient Condition for Undoability

To generalize the result presented above, we proceed by induction on the size n of the set of operations, where n is even. Indeed, basic cases corresponds to $n = 2$ and $n = 4$, then we can deduce the result for $n > 4$ since we build the set of operation by adding an operation and its inverse to an undoable CCO. Each time we add a new couple of operations $(op, \overline{op})$, it forms a 4-CCO with every couple of operations $(op_i, \overline{op_i})$ from the initial CCO operations set.

Theorem 6.2.2 (Necessary and Sufficient Conditions for Undoabiliy).

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$. C is undoable iff $\mathcal{O}p$ is commutative.

Proof. From Lemma 6.2.7 we deduce that commutativity implies undoability. Now let us prove that if a consistent object C is undoable then its corresponding set of operations is commutative. To do this, we proceed by induction on $2n = |\mathcal{O}p|$, $n \in \mathbb{N}$ (*i.e.* $2n$ is the size of $\mathcal{O}p$).

Basis. As illustrated in Figure 6.5, the statement holds for $n = 2$ and $n = 4$. Indeed, the only solution for the 2-CCO and 4-CCO problems is commutative.

Inductive step. Let us show that if the statement holds for a given $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ of order $2n$, then it holds for the object C' of order $2n + 2$. This can be done as follows. Assume that that if $\mathcal{O}p$ is undoable then $\mathcal{O}p$ is commutative. We also suppose that for every $op_i \in \mathcal{O}p$, where $i \in [1; 2n]$, we have $\overline{op_i} = op_{2\times i}$ if i is odd and $\overline{op_i} = op_{\frac{i}{2}}$ if i is even (see Figure 6.6). Now, consider the object C' with the corresponding set of operations $\mathcal{O}p' = \mathcal{O}p \bigcup \{op_{n+1}, op_{2(n+1)}\}$ of size $2n + 2$ such that the inverse of op_{n+1} is $op_{2(n+1)}$ (the blue set in Figure 6.6). Let us prove that if C' is undoable then $\mathcal{O}p'$ is commutative. According to the initial step ($n = 2$), we deduce that op_{n+1} commutes with $op_{2(n+1)}$ since the set $\{op_{n+1}, op_{2(n+1)}\}$ forms an undoable CCO of order 2.

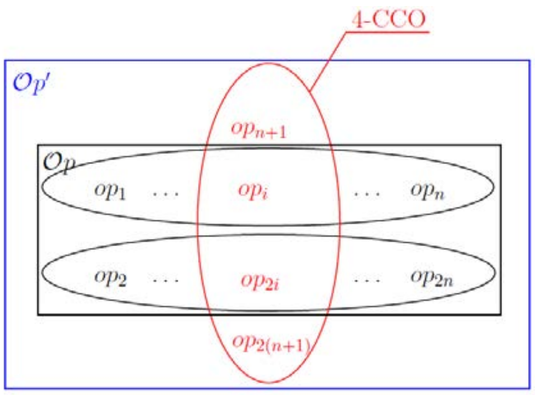


Figure 6.6: Main Theorem.

Furthermore, since C' is undoable, then every CCO formed by subsets $\{op_i, op_{2i}, op_{n+1}, op_{2(n+1)} | op_i \in \mathcal{O}p)\}$ (the red set in Figure 6.6) is undoable. Consequently, using initial step with $n = 4$, we deduce that op_{n+1} and its inverse $op_{2(n+1)}$ commute with every operation op_i and its inverse op_{2i} from the initial set of operation $\mathcal{O}p$. Thereby showing that indeed $\mathcal{O}p'$ is commutative.

Since both the basis and the inductive step have been proved, it has now been proved by mathematical induction that undoability implies commutativity. $\square$

6.2.4 Discussion

From Theorem 6.2.2, we can deduce that commutativity has a direct impact on undoability. Our result shows that non commutative operations do not allow to achieve undoability property while most collaborative operations based on the transformation function approach, use non commutative operations to alter shared objects.

This is a challenging problem because the use of OT approach aims at going beyond commutativity. Subsequently, we believe that it is necessary to extend the set of operations with a new kind of operations namely the *idle* operation yet excluded in Definition 6.1.1.

However, the idle operation as it is defined does not allow to preserve **IP3** in all cases. Mainly, we can face divergence problems when the two concurrent operations have the same effect on the document and thus are transformed into idle operations.

To illustrate this problem, we consider the Example 5.2.4 where the IT function (see Algorithm 12) is extended as follows:

```
1: IT($op_1$,$op_2$):$op'_1$
2: Choice of $op_1$ and $op_2$
3:    Case: $op_1 = Up$ and $op_2 = Up$
4:       $op'_1 \leftarrow I$
5:    Case: $op_1 = Up$ and $op_2 = Down$
6:       $op'_1 \leftarrow Up$
7:    Case:$op_1 = Up$ and $op_2 = I$
8:       $op'_1 \leftarrow Up$
9:    Case: $op_1 = Down$ and $op_2 = Up$
10:      $op'_1 \leftarrow I$
11:   Case:$op_1 = Down$ and $op_2 = Down$
12:      $op'_1 \leftarrow I$
13:   Case:$op_1 = Down$ and $op_2 = I$
14:      $op'_1 \leftarrow Down$
15:   Case:$op_1 = I$ and $op_2 = Up$
16:      $op'_1 \leftarrow I$
17:   Case:$op_1 = I$ and $op_2 = Down$
18:      $op'_1 \leftarrow I$
19:   Case:$op_1 = I$ and $op_2 = I$
20:      $op'_1 \leftarrow I$
21: end choice
```

Algorithm 13: Extension of Algorithm 12 with the Idle Operation

Consider the scenario presented in Figure 6.7 where two sites perform the operation Up concurrently which is transformed to I at both sites. Undoing $op_1 = Up$ at site 1 leads to generating $\overline{op_1} = Down$ and transforming it against I which results in the operation $Down$. Executing $Down$ at state 1 leads to the new state 0. However, undoing op_1 at site 2 consists in applying $\overline{I} = I$ at state 1 which produces the same state 1. Obviously, sites 1 and 2 diverge. The same issue is encountered if $op_1 = op_2 = Down$.

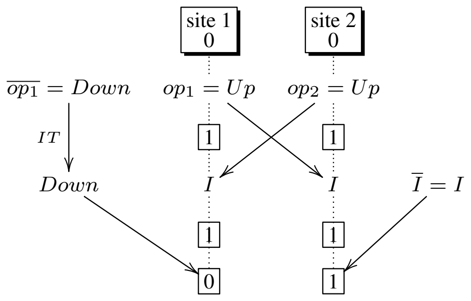
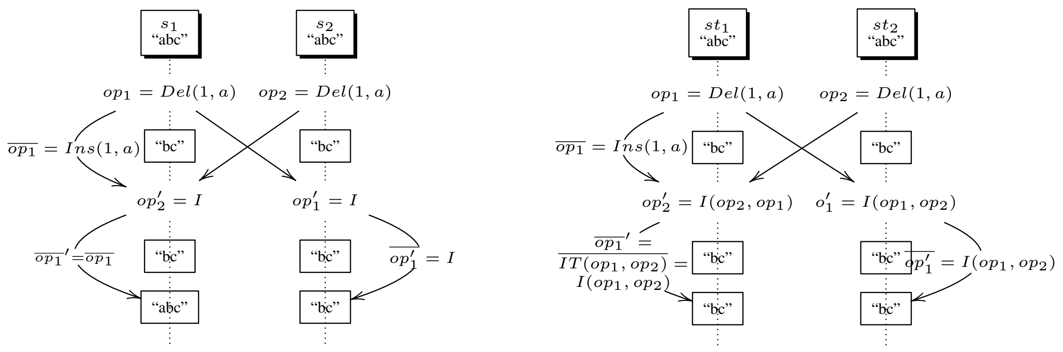


Figure 6.7: Divergence despite the use of the Idle Operation.

It should be pointed out that this problem is faced only if two concurrent operations are both transformed to an idle operation.

Similarly, the idle operation does not allow for undoability in the context of collaborative editors as shown in the scenario of Figure 6.8.(a) where two deletions at the same position are performed concurrently by s_1 and s_2. These operations are transformed into two idle operations I at both sites (the corresponding transformation function is given in Appendix B). Now, if we undo $op_1 = Del(1, a)$ at site s_1, we generate $\overline{op_1} = Ins(1, a)$. Then, we integrate op_2''s effect to obtain $\overline{op_1}' = \overline{op_1}$ whom execution recovers the state "abc". However at s_2, the state remains "bc" since $\overline{op_1'} = \overline{I} = I$. It is obvious that IT fails to preserve IP3 as we have $\overline{op_1}' \neq \overline{op_1'}$.



(a) Undoing op_1 produces a divergence.

(b) Delete-Delete puzzle resolution.

Figure 6.8: Delete-Delete puzzle.

According to the two examples presented above, we deduce that the divergence is caused by the loss of the initial information about the operation to be undone. To overcome this issue, we propose to extend the semantic of the idle operation I as follows:

Definition 6.2.7 (New Semantic of the Idle Operation)**.**

Given two concurrent operations op_1 and op_2 and a correct transformation function IT. If $IT(op_1, op_2)$ is in an idle operation we have:

$$IT(op_1, op_2) = I(op_1, op_2).$$

This new semantic aims at transforming reverse or idle operations correctly but requires to define transformation rules for both reverse and idle operations. For instance, stating that $IT(\overline{op_1}, I(op_2, op_1)) = \overline{IT(op_1, op_2)}$ leads to satisfying **IP3** property. For example, in Figure 6.8, when $\overline{op_1}$ is transformed against $I(op_2, op_1)$ at site 1, the result is equal to $\overline{IT(op_2, op_1)} = \overline{I(op_2, op_1)}$. Since the inverse of an idle operation has no effect on the state, the convergence is achieved since we have to execute two idle operations at both sites.

Basing on the new semantic of the idle operation, we propose a generic undo framework. Our solution consists in extending consistent collaborative objects with idle operations and integrate them correctly to preserve inverse and transformation properties.

6.3 Our Generic Undo Framework

We have proved in Section 6.2 that the commutativity property represents a necessary and sufficient condition to achieve undoability. Accordingly, it is impossible to define a correct undo algorithm for non commutative operations. However, practically it is not possible to always define a set of commutative operations. For instance, in the context of collaborative editors, inserting characters does not commute with deleting characters. Thus, we proposed to extend the set of operations with a new form of the idle operation where semantic information about the transformed operations is encapsulated in the idle operation's signature (see Definition 6.2.7). For this, we relax the Definition 6.1.1 to cover idle operations in order to take into account the idle operation as follows:

Definition 6.3.8 (Relaxed Consistent Collaborative Object)**.**

A *Relaxed Consistent Collaborative Object* (RCCO) is a triplet $Cr = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ such that:

- $\mathcal{S}t$ is the set of object states (or the space state);
- $\mathcal{O}p$ is the set of primitive operations executed by the user to modify the object state such that for every operation $op \in \mathcal{O}p$ there is *unique inverse* $\overline{op} \in \mathcal{O}p$.
- $IT : \mathcal{O}p \times \mathcal{O}p \rightarrow \mathcal{O}p$ is a correct transformation function (*i.e.*, IT satisfies properties **TP1** and **TP2**).

For instance the collaborative object defined in [48] is a RCCO since it includes the idle operation equal to its inverse.

Extended idle operations are generated at the integration process according to the relation between the two operations involved in the transformation function. Indeed, if the effect of the received operation is hidden by the concurrent operation against which it is transformed (*e.g* transforming two deletions or updates at the same position), then the result of the transformation is an idle operation.

In the following, we present the hide relation used to originate semantically extended idle operations and detail the transformation rules used to integrate correctly idle and inverse operations.

6.3.1 The Hide Relation

Given a RCCO $Cr = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, we define the *Hide* relation over $\mathcal{O}p$ as follows:

Definition 6.3.9 (Hide Predicate)**.**

For every two operations op_2 and op_1 from $\mathcal{O}p$ defined on the same state of the shared object, we say that op_1 hides op_2 noted $Hide(op_1, op_2)$ iff $op_1 \cdot IT(op_2, op_1) \equiv op_1$.

According to Definition 6.3.9, we deduce that $Hide(op_1, op_2)$ means $IT(op_2, op_1) \equiv \emptyset$. Thus, also the transformation of $IT(op_2, op_1)$ against any operation op_3 produces an operation with no effect *i.e.* $IT(IT(op_2, op_1), op_3) \equiv \emptyset$.

Lemma 6.3.8.

Given a RCCO $Cr = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, then for any three operations op_1, op_2 and op_3 from $\mathcal{O}p$,

$$Hide(op_1, op_2) \Longrightarrow IT(IT(op_2, op_1), op_3) \equiv \emptyset.$$

Proof. Since $IT(op_2, op_1) \equiv \emptyset$, then its transformation against any operation has no effect on the shared object state. □

Always according to Definition 6.3.9, the following statement holds:

Lemma 6.3.9.

Given three operations op_1, op_2 and op_3, then

$$Hide(op_1, op_2) \Rightarrow IT^*(op_3, op_1 \cdot IT(op_2, op_1)) = IT(op_3, op_1).$$

Proof. Since $Hide(op_1, op_2)$ then $op_1 \cdot IT(op_2, op_1) \equiv op_1$ (see Definition 6.3.9). Consequently:

$$IT^*(op_3, op_1 \cdot IT(op_2, op_1) = IT(op_3, op_1)$$

□

Next, we give some properties of the Hide relation. In the following lemma we state that the Hide relation is transitive.

Lemma 6.3.10 (Hide is Transitive.)**.**

Given three concurrent operations op_1, op_2 and op_3 then

$$Hide(op_3, op_2) \wedge Hide(op_2, op_1) \Rightarrow Hide(op_3, op_1)$$

Proof. We have:

$$\begin{aligned} IT(op_1, op_3) &= IT^*(op_1, op_3 \cdot IT(op_2, op_3)) && \text{(Lemma 6.3.9)} \\ IT(op_1, op_3) &= IT^*(op_1, op_2 \cdot IT(op_3, op_2)) && \textbf{(TP2)} \\ IT(op_1, op_3) &= IT(IT(op_1, op_2), IT(op_3, op_2)) && \\ &\equiv \emptyset && \text{(Lemma 6.3.8)} \end{aligned}$$

Thus, $op_3 \cdot IT(op_1, op_3) \equiv op_3$. Thereby showing that indeed $Hide(op_3, op_1)$ holds. *Q.E.D.* □

Given a RCCO $Cr = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, our solution consists in transforming Cr to $Cr' = \langle \mathcal{S}t, \mathcal{O}p', IT' \rangle$ as follows: $\forall op_1, op_2 \in \mathcal{O}p \times \mathcal{O}p$

- If $Hide(op_1, op_2)$ then $IT'(op_2, op_1) = I(op_2, op_1)$ and $\mathcal{O}p' = \mathcal{O}p \cup \{I(op_2, op_1)\}$;
- Otherwise $IT'(op_2, op_1) = IT(op_2, op_1)$.

Example 6.3.1. *Consider again the example of a shared binary register altered by the two operations Up and $Down$. According to the IT function given in Algorithm 12 (see Chapter 5) related to this example, we have:*

- $Up \cdot IT(Up, Up) \equiv Up \Rightarrow Hide(Up, Up)$
- $Down \cdot IT(Down, Dwon) \equiv Down \Rightarrow Hide(Down, Down)$
- $Up \cdot IT(Up, Down) \equiv Up \Rightarrow Hide(Up, Down)$

- $Down \cdot IT(Down, Up) \not\equiv Up \Rightarrow \neg Hide(Up, Up)$

The final set of operations $\mathcal{O}p = \{Up, Down, I(Up, Up), I(Down, Up), I(Down, Down)\}$

Accordingly, we define three different kinds of operations:

1. Normal operations: an operation $op \in \mathcal{O}p$ is a normal operation if it is originated by a user or is the result of transforming two operations op_1 and op_2 such that neither $Hide(op_1, op_2)$, nor $Hide(op_2, op_1)$.

2. Idle or Hidden operations: a hidden operation is an operation of the form $I(op_1, op_2)$ where both op_1 and op_2 are normal operations verifying $Hide(op_2, op_1)$.

3. Inverse Operations: an inverse operation $\overline{op}$ is an operation generated to undo the normal operation op and verifies $op \cdot \overline{op} \equiv \emptyset$

These different operations must be transformed correctly to ensure inverse and transformation properties. We define in the following a set of transformation rules in order to integrate correctly inverse and idle operations.

6.3.2 Transformation Rules

Given a RCCO, we must ensure that the enhanced set of operations preserves transformation and inverse properties. Otherwise the undoability as well as the convergence of the shared document are lost.

Table 6.2 presents the transformation rules for idle as well as inverse operations. These transformation rules can be integrated by any existing IT according to the Hide relation *w.r.t* Definition 6.3.9. Letters x, y, z and t refer to integrated operations.

$\mathbf{TR1}$. The set of transformation rules $\mathbf{TR1}$ shows how we transform an idle operation against another idle operation where both operations are hidden by the same normal operation. Let two operations x and z are transformed against an operation z such that z hides both x and y. Integrating x against the sequence $y \cdot I(z, y)$ has to take into account the hide relation between x and z. Indeed, if z hides x then according to the transitivity property of the hide relation *w.r.t* Lemma 6.3.10, $I(x, y)$ is transformed against $I(z, y)$ into $I(x, z)$. Otherwise, we integrate the effect of the operation z into x. Subsequently, if the operation y is undone later, the effect of z is yet integrated into x which greatly simplifies the elimination of y's effect from the log.

$\mathbf{TR1I}$. When an idle operation is undone, it should de transformed against the following operations in the log. The set of transformation rules $\mathbf{TR1I}$ shows how we transform an inverse idle operation against another idle operation hidden by the same normal operation. The principle is the same as in $\mathbf{TR1}$. Since, we are integrating an idle inverse operation $\overline{I(x, y)}$, we may face another idle operation hidden by y that already integrates the effect of x *i.e.* $\overline{I(IT(z, x), y)}$ (according to $\mathbf{TR1I}$). In this case, according to whether z hides x or not, we integrate $\overline{I(x, y)}$ into $\overline{I(x, z)}$ (since hide is transitive) or we integrate the effect of z into x to obtain $\overline{I(IT(x, z), y)}$.

$\mathbf{TR2}$. These transformation rules show how to transform an idle operation against a normal operation or a normal inverse operation. According to the transitivity of the hide relation, we update the semantic of the integrated idle operation. So, if an idle operation $I(x, y)$ is integrated against $IT(z, y)$, it is transformed into $I(x, z)$ if z also hides x. Otherwise it is transformed to $I(IT(x, z), y)$. The same principle is used when transforming $I(x, y)$ against a normal inverse operation excepting that the transitivity is not applied since we are integrating the effect of an inverse operation whose effect will be eliminated.

$\mathbf{TR2I}$. These rules transform an idle inverse operation against a normal operation or a idle operation.

TR1	$Hide(z,x) \Rightarrow IT[I(x,y), I(z,y)] = I(x,z)$ $\neg Hide(z,x) \Rightarrow IT[I(x,y), I(z,y)] = I(IT(x,z),y)$
TR1I	$Hide(z,x) \Rightarrow IT[\overline{I(x,y)}, I(z,y)] = \overline{I(x,z)}$ $\neg Hide(z,x) \Rightarrow IT[\overline{I(x,y)}, I(z,y)] = \overline{I(IT(x,z),y)}$ $Hide(z,x) \Rightarrow IT[\overline{I(x,y)}, I(IT(z,x),y)] = \overline{I(x,z)}$ $\neg Hide(z,x) \Rightarrow IT[\overline{I(x,y)}, I(IT(z,x),y)] = \overline{I(IT(x,z),y)}$
TR2	$Hide(z,x) \Rightarrow IT(I(x,y), IT(z,y)) = I(x,z)$ $Hide(z,x) \Rightarrow IT(I(x,y), \overline{IT(z,y)}) = I(x,y)$ $\neg Hide(z,x) \wedge \Rightarrow IT[I(x,y), IT(z,y)] = I(IT(x,z),y)$ $\neg Hide(z,x) \wedge \Rightarrow IT[I(x,y), \overline{IT(z,y)}] = I(IT(x,\overline{z}),y)$
TR2I	$Hide(z,x) \Rightarrow IT(\overline{I(x,y)}, I(z,x)) = \overline{I(x,y)}$ $\neg Hide(z,x) \Rightarrow IT(\overline{I(x,y)}, I(z,x)) = \overline{I(IT(x,z),y)}$ $IT(\overline{I(x,y)}, IT(z,y)) = \overline{I(x,y)}$ $\neg Hide(z,x) \Rightarrow IT(\overline{I(x,y)}, IT(z,x)) = \overline{I(IT(x,z),y)}$
TR3	$IT(\overline{x}, I(y,x)) = \overline{IT(x,y)}$ $\neg Hide(x,y) \Rightarrow IT(\overline{x}, IT(y,x)) = \overline{IT(x,y)}$
TR4	$IT(I(x,y), \overline{y}) = x$ $IT(I(x,y), \overline{I(y,z)}) = I(x,z)$ $Hide(z,y) \Rightarrow IT(I(x,y), \overline{I(z,y)}) = I(x,y)$ $\neg Hide(z,y) \Rightarrow IT(I(x,y), \overline{I(z,y)}) = I(IT(x,\overline{z}),y)$ $\neg Hide(z,x) \Rightarrow IT(I(IT(x,y),z), \overline{I(y,z)}) = I(x,z)$ $\neg Hide(z,x) \wedge Hide(t,y) \Rightarrow IT(I(x,y), I(\overline{z},t)) = I(IT(x,\overline{z}),y)$ $\neg Hide(t,x) \Rightarrow IT(I(IT(x,y),z), \overline{I(IT(y,t),z)}) = I(IT(x,t),z)$ $\neg Hide(x,y) \wedge Hide(z,y) \Rightarrow IT(IT(x,z), \overline{IT(z,y)}) = IT(x,y)$
TR5	$IT(x, I(y,z)) = x$ $IT(x, \overline{I(y,z)}) = x$

Table 6.2: Transformation Rules for Idle and Inverse Operations

TR3. Consists in the transformation rules for a normal inverse operation against an idle or a inverse operation. It enforces the inverse rule **IP3** for normal and idle operations and ensures that the integrated form of the inverse of an operation x is equal to the inverse of the integrated form of the operation x.

TR4. These rules show how to integrate the effect of an inverse operation into different kinds of operations. It allows for updating the log and eliminating the effect of the undone operation from it. For instance, integrating $\overline{y}$ into an operation hidden by y let $I(x, y)$, naturally restores x. Similarly, integrating the effect of $\overline{I(y, z)}$ into $I(x, y)$ results $I(x, z)$ since y is undone and the hide relation is transitive. *etc.*

TR5. Finally, **TR5** states that a normal operation remains the same when transformed against an idle or an idle inverse operation. This stress the fact that an idle operation does not have an effect on normal operations.

The question that arises here is whether transformation and undoability properties are satisfied by these transformation rules. In Appendix A, we illustrate the preservation of these properties by the transformation rules presented above.

6.3.3 Illustrative Example

Consider a collaborative editor based on the transformation function of Algorithm 14 (see Appendix B). Figure 6.9 depicts a scenario where three users edit concurrently the initial string of characters "abc". Each user performs an update operation as follows: user 1 originates the request $u_1 = Upd(1, a, x)$, user 2 performs the request $u_2 = Upd(1, a, y)$ and user 3 performs the request $u_3 = Upd(1, a, z)$. When two concurrent updates target the same position, then the operation with the greater site's identity hides the effect of the other one *w.r.t.* Definition 6.3.9[18]. Accordingly, when u_1 is received at site 2 and 3, it is transformed to $u_1^2 = I(u_1, u_2)$ and $u_1^3 = I(u_1, u_3)$ respectively. When the request u_2 is received at site 3, it is transformed to $u_2^3 = IT(I(u_2, u_3), I(u_1, u_3)) = I(IT(u_2, u_1), u_3)$ according to the rule **TR1** since $\neg Hide(u_1, u_2)$. After the execution of u_1, site 1 receives u_2 which is transformed to $u_2^1 = Upd(1, x, y)$. After a while, it receives u_3 and transforms it to $u_3^1 = Upd(1, y, z)$. Similarly, when site 2 receives u_3 it is transformed to $u_3^2 = Upd(1, z, t)$.

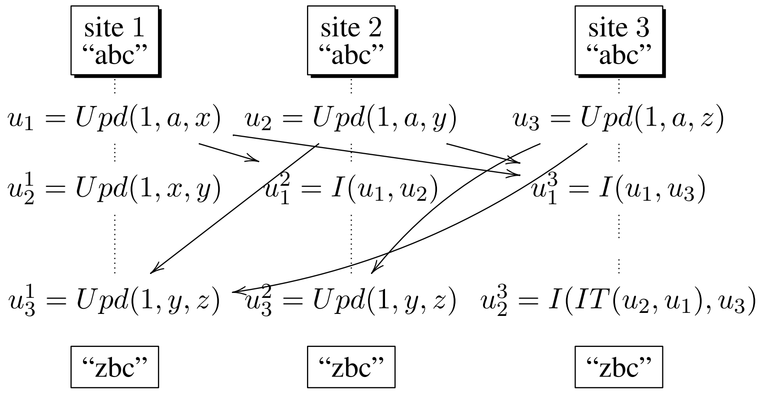
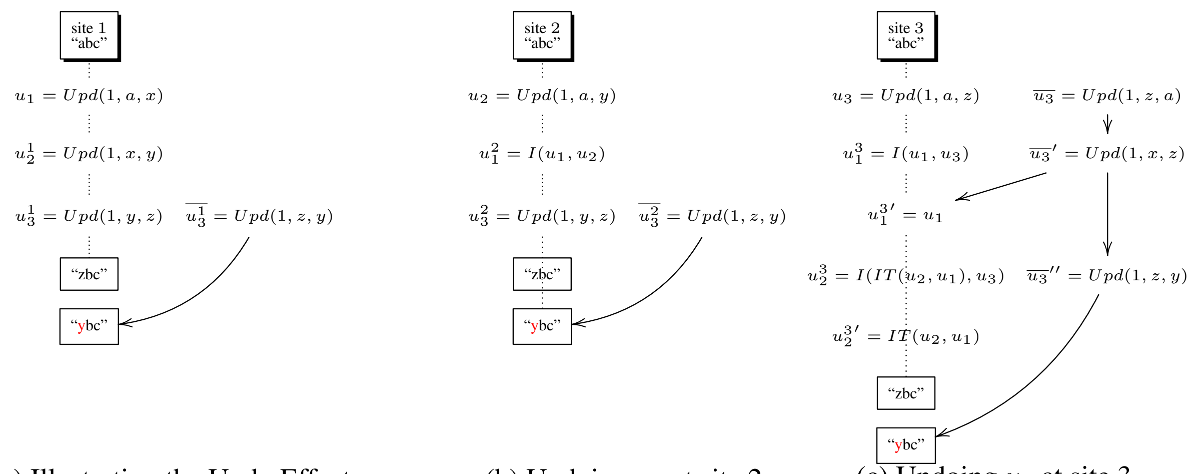


Figure 6.9: An Illustrative Example of our Undo Scheme.

Suppose now, that the request u_3 is undone. Intuitively, the effect of the operation u_2 and not that of u_1 should be recovered (according to their identities order) (*e.g.* the character "y" and not "x" is recovered). Figure 6.10 shows the state and the log of each site after undoing the operation u_3.
As illustrated in Figure 6.10.(a) and (b), undoing u_3 at sites 1 and 2 only requires to generate and execute the operation $\overline{u_3^1} = \overline{u_3^2} = \overline{Upd(1, y, z)} = Upd(1, z, y)$. Thus the state after undo is "ybc" at sites 1 and 2. At site 3 (see Figure 6.10.(c)), we generate $\overline{u_3} = Upd(1, z, a)$, then transform it against $u_1^3 = I(u_1, u_3)$.

[18]To simplify, we suppose that the site's identity refers to its number.



(a) Illustrating the Undo Effect. (b) Undoing u_3 at site 2. (c) Undoing u_3 at site 3.

Figure 6.10: Undoing u_3.

According to the transformation rule **TR3**, this results in the operation $\overline{u_3}' = \overline{IT(u_3,u_1)} = \overline{Upd(1,x,z)}$. Next, we have to integrate the effect of $u_2^3 = I(IT(u_2,u_1),u_3)$ into $\overline{u_3}'$. It comes from rule **TR3** that:

$$\begin{aligned}\overline{u_3}'' =& IT(\overline{u_3}', u_2^3)\\ &= \overline{IT(u_3', IT(u_2,u_1))}\\ &= \overline{IT(Upd(1,x,z), Upd(1,x,y)}\\ &= \overline{Upd(1,y,z)}\\ &= Upd(1,z,y)\end{aligned}$$

Consequently, the final state of site 3 is also "ybc" . The convergence is then obtained.

The following step consists in updating the log of each site by integrating the effect of $\overline{u_3}$ into the the operations that follow u_3 in the log. As depicted in Figure 6.10, integrating the effect of $\overline{u_3}$ int the operation $u_1^3 = I(u_1,u_3)$ results in the operation $u_1^{3\prime} = u_1$ according to the rule **TR4**. As for $u_2^3 = I(IT(u_2,u_1),u_3)$, it is transformed to $u_2^{3\prime} = IT(u_2,u_1)$. Thereby, the effect of $\overline{u_3}$ is correctly eliminated from the log (see Definition 5.2.2).

In the following, we discuss the asymptotic time complexity of our approach and show it has a linear complexity which is of relevance for real time applications.

6.3.4 Asymptotic Time Complexity of the Undo Command

To undo a cooperative request r_{c_i} in the log H, our undo command proceeds as follows:

1. Find r_{c_i} in H;
2. mark r_{c_i} as an undone operation,
3. generate $\overline{r_{c_i}}$,
4. For all operation r_{c_j} $(i+1 \leq j < |H|)$
   - calculate $\overline{r_{c_i}} = IT(\overline{r_{c_i}}, H[j])$ (integrate the effect of $H[j]$ into $\overline{r_{c_i}}$;
   - replace $H[j]$ by $IT(H[j], \overline{r_{c_i}})$ (exclude the effect of r_{c_i} from the log),

5. execute $\overline{r_{c_i}}'$.

Note that we need neither to store transformed forms of r_{c_i} nor original forms of any operation as it is required in [97,101] since puzzles are solved directly by our transformation rules $TR1-5$ (see Table 6.2).

The worst case of the algorithm corresponds to undoing the first operation in the log. The complexity is then linear and equal to $O(|H|)$. To see the impact of our selective undo solution originated by the security layer, we also calculate the complexity of undoing all the log as the worst case faced during the violation of the security policy which is a quadratic complexity $(O(|H|^2))$. Fortunately, we don't face such a case in practice, mainly if communication time is negligible (*e.g.* a user cannot generate an big number of requests before receiving administrative revocations).

6.4 Conclusion

In this Chapter, we defined a necessary and sufficient condition to achieve correct undoability. Indeed, we proved that it is impossible for non commutative sets of operations. Our proof required the use of CSP theory to formalize the undoability problem and find the different IT evaluations leading to undoablilty.

Even though it is negative, this result is interesting for OT-based object developers as well as researchers working on undoability for collaborative applications. Indeed, they can be aware that achieving undoability is impossible for objects without idle operations and based on non commutative operations. To overcome this negative result, we investigate whether or no the idle operation allows to achieve the undoability and extend its semantic. However, convergence requires that the enhanced set of operations satisfy the transformation function properties namely $\mathbf{TP1}$ and $\mathbf{TP2}$.

Given any coordination framework, our solution consists in adding a new form of the idle operation according to the Hide relation deduced from the initial transformation algorithm. This requires to correctly transform idle operations as well as the inverse operations. For this, we defined a set of transformation rules in the first logic order and proved these rules preserve the inverse as well as the transformation properties.

Our solution has a linear complexity and could be integrated to any OT-based coordination framework since it only requires to extend the set of operations by the new semantic of the idle operation captured from the initial transformation function and to enforce the transformation rules $\mathbf{TR1-5}$ that are semantic-independent.

Chapter 7

Experimental Study and Performance Measurements

Contents

Experiments are necessary to understand what the asymptotic complexities mean when interactive constraints are present in the system and to validate our formal access control model presented in Chapter 3.

In order to gain a better understanding of how to use our security protocols, we have implemented both single and multi-administrator approaches. We have developed two prototypes based on the coordination framework OPTIC [48] a collaborative text editor P2PEdit, and a shared calendar application p2pAgenda. Furthermore, we have performed experiments modularizing code on the distributed grid platform GRID'5000.

This chapter is organized as follows. First, we present the coordination framework OPTIC on the top of which we implemented our access control model in Section 7.1. Section 7.2 presents experimental environments used to implement our prototypes. Then, Section 7.3, shows the experimental results of the coordination layer for both prototypes P2PEdit and P2PAgenda. Finally, Section 7.4, highlights experimental results of our access control model.

7.1 Coordination Framework

In this Section, we present the characteristics of the coordination framework OPTIC [50]. Furthermore, we discuss different extensions that we provide to improve this framework and enhance its functioning especially, the garbage collection module for cooperative logs. Indeed, the garbage module allows for cleaning log and extending OPTIC to mobile devices.

7.1.1 OPTIC Characteristics

OPTIC [48, 50] represents a scalable coordination model for real-time collaborative editors. It offers a framework for collaborative editing that addresses the weakness of previous OT works and satisfies all collaborative editing requirements mentioned in Chapter 2. It is characterized by the following aspects:

1. This framework supports an unconstrained collaborative editing work (without the necessity of central coordination). Using optimistic replication scheme, it provides simultaneous access to shared documents.

2. Instead of vector timestamps, it uses a simple technique to preserve causality dependency. This technique is minimal because only direct dependency information between updates is used. It is independent on the number of users and it provides high concurrency in comparison with vector timestamps.

3. Using OT approach, reconciliation of divergent copies is done automatically in a decentralized fashion.

4. The framework can scale naturally thanks to the minimal causality dependency relation. In other words, it may be deployed easily in Peer-to-Peer (P2P) networks.

Initially, OPTIC dealt with linear data-structure using only a set of insert and delete operations.The first extension was to enhance the set of operations by the update cooperative operation [19]. Moreover, the signature of the delete operation were modified in order to support undo. The final set of operations supported by OPTIC is the following:

- $Ins(p, e, \omega)$ where p is the insertion position, e the element to be added at position p and ω is the sequence of positions that contains all different positions occupied by e during the transformation process;

- $Del(p, e)$ which deletes the element e at position p. We add the element in the operation definition since we will need to restore operations during the undo mechanism (see Chapter 5)

- $Upd(p, e, e')$ which replaces the element e at position p by the new element e'.

The novelty of OPTIC was the use of a new form of dependency called *semantic dependency*. This notion allowed for a completely decentralized and scalable algorithm. Indeed, instead of vector time stamps, OPTIC resorts to the semantic dependency defined for the set of operations altering a document.

The semantic dependency notion is a minimal dependency relation which is *independent* of the number of users, and accordingly, allows for dynamic groups. Studying the semantics of linear object (modified by insertion, deletion and update operations) allows to provide dependency relation within a log [47, 48].For instance, when the added elements are adjacent (at positions p and/or $p + 1$), their respective insertion requests are considered as dependent. Also, deleting an element depends on the request that has inserted this element. Thus, there is no dependency between delete requests and they can be executed among them in any order. Likewise, updating an element depends on the request inserting this element. At last, we consider two consecutive update requests as dependent if their update positions are equal and the condition on their site identities is satisfied.

This causal relation builds a *dependency tree* on the requests where each request has only to store the request identity whose it directly depends on. For instance, consider the log presented in Figure 7.1 containing four cooperative operations. According to the semantic dependency relation between these operations the log is equivalent to the tree presented in the same figure.

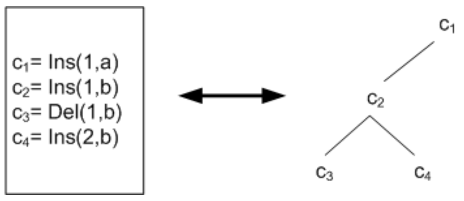


Figure 7.1: Dependency Tree: the cooperative log (left), the corresponding dependency tree (right).

OPTIC also relies on canonical logs, *i.e* a particular class of logs where insertion requests are stored before deletion requests in order to avoid the $TP2$ *puzzle* [48]. This class of logs allows to build transformation paths leading to data convergence. Building canonical logs requires two transformation functions, IT and ET transformations that we discuss in the following.

Inclusive Transformation. IT algorithm is used to execute concurrent requests in any order. When a remote cooperative request is received at a given site, IT function includes the effect of all concurrent requests. The transformation function IT used in OPTIC is given in Appendix B.

Exclusive Transformation. ET algorithm is required to exclude operations effect in order to reorder logs. Let $[r_1; r_2]$ be a request sequence so that r_2 is defined on the state produced by r_1. The *exclusive transformation* $ET(r_2, r_1)$ enables us to *exclude the effect* of r_1 from r_2 as if r_2 had not been executed after r_1. For instance, when r_1 is an insert and r_2 is a delete at a position less than that of r_1, then $ET(r_1, r_2)$ decrements the position of r_1. For more details on the function ET used in OPTIC, the reader is invited to see Appendix B.

Coordination Protocol. The coordination protocol is detailed in Algorithm 16 (for more details, the reader can refer to Appendix B or [19, 48]). It proceeds as follows:

1. When a user manipulates the local copy of the shared document by generating a cooperative operation, we determine its causal dependency by the usage of ComputeBF function (see Algorithm 17 in Appendix B) then broadcast it to other users. Note that after each generation of local cooperative request the log is reorganized (see Algorithm 18 in Appendix B) to keep it canonical.

2. Each site has to determine the new form of every remote request using ComputeFF function (see Algorithm 19 in Appendix B) in order to take into account concurrent requests and integrate their effect in the received request. Once the operation is executed, the log is canonized.

In the following, we briefly present the garbage collector designed for the coordination framework OPTIC in order to maintain logs in a decentralized fashion.

7.1.2 Garbage Collector for Cooperative Logs

Distributed collaborative editors require the storage of document updates in a local buffer. Indeed, it is necessary to store the track of the operations executed locally or received from remote sites to ensure the convergence of the shared document and allow the concurrent execution of operations regardless the reception order.

However, as the document is frequently updated by many users, the size of cooperative logs increases rapidly. This is troublesome especially when deploying a collaborative editor on mobile devices. Indeed, such devices have limited storage capacity which may degrade the performances of the application.

Hence we need to devise a garbage collection technique allowing to delete all operations already seen and executed at collaborating sites at a given time. Thus, many challenges should be taken into consideration since RCEs have specific requirements (see Chapter 2) that may not match with those characterizing mobile devices. Moreover, logs are not identical in different collaborating sites as we allow for the out-of-order execution of operations. Furthermore, the size of the collaborating group is dynamic due to the churn which complicates the garbage task.

In Appendix B, we investigate the issues raised by the garbage collection module and our solutions to avoid them as well as algorithms implementing our solution. This contribution was published in [64] and consists in a decentralized garbage collector that uses the semantic dependency relation for accelerating the garbage process. Indeed, collaborating sites has only to exchange the leaves of their dependency trees in order to establish a consensus about the garbage procedure. Meanwhile, users are still able to receive concurrent updates till the end of the garbage process.

7.2 Developement Frameworks

In this section, we present the different frameworks used in our experimental study.

7.2.1 JXTA

Our prototypes were implemented using the JXTA[19] platform. JXTA (Juxtapose) is a programming language and platform independent Open Source protocol started by Sun Microsystems for peer-to-peer (P2P) networking in 2001. The JXTA protocols are defined as a set of XML messages which allow any device connected to a network to exchange messages and collaborate independently of the underlying network topology (see Figure 7.2).

This technology represents a set of open protocols allowing any connected device on the network, ranging from cell phones and wireless PDAs to PCs and servers (see Figure 7.2), to communicate and collaborate in a P2P manner. Consequently, it enables developers to more quickly and easily create P2P applications.

Since it is based upon a set of open XML protocols, JXTA can be implemented in any modern computer language. Implementations are currently available for Java SE, C/C++, C# and Java ME. In our experiments, we used the Java implementation.

JXTA peers create a virtual overlay network which allows a peer to interact with other peers even when some of the peers and resources are behind firewalls and NATs or use different network transports. In addition, each resource is identified by a unique ID, a 160 bit SHA-1 URN in the Java binding, so that a peer can change its localization address while keeping a constant identification number.

[19]http://jxta.kenai.com/

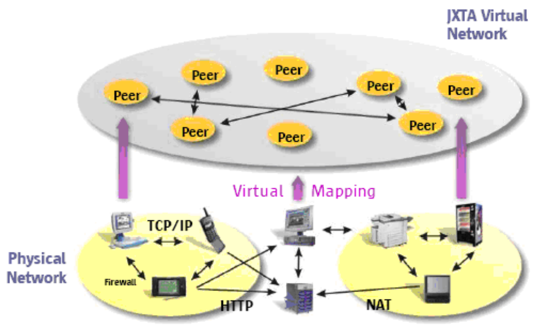


Figure 7.2: JXTA Architecture [1].

7.2.2 NetBeans

The development framework that we used for implementing our algorithms is NetBeans Integrated Development Environment IDE[20] for software developers. It is a free and open-source development tool that provides all the tools needed to create professional desktop, enterprise, web, and mobile applications with the Java platform. It also allows to create professional standards-based user interface with the NetBeans Swing GUI Builder.

7.2.3 Grid'5000

Performance measurements of our algorithms, were done on the Grid'5000 platform since it represents a scientific instrument for the study of large scale parallel and distributed systems. It aims to ease research works on grid infrastructures by providing a highly reconfigurable and controllable experimental platform to its users. It provides resources that could be shared by many users and reserved for specific experiments.

The infrastructure of Grid'5000 is geographically distributed on different sites hosting the instrument, initially 9 sites in France (10 since 2011) as shown in Figure 7.3. Porto Alegre, Brazil is now officially becoming the first site abroad [2].

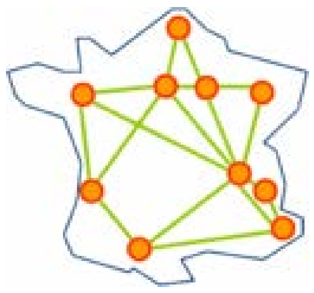


Figure 7.3: Grid'5000 Sites [2].

The Grid'5000 platform is made of 13 clusters, including 1047 nodes for 2094 CPUs. Within each cluster, the nodes are located in the same geographic area and communicate through Gigabyte Ethernet links. Communications between clusters are made through the French academic network (RENATER)[21].

[20]http://netbeans.org/features/index.html
[21]http://www.renater.fr/

Connexion to Grid'5000 requires the usage of the SSH protocol as shown in Figure 7.4 where it is possible to connect to many frontends at the same time.

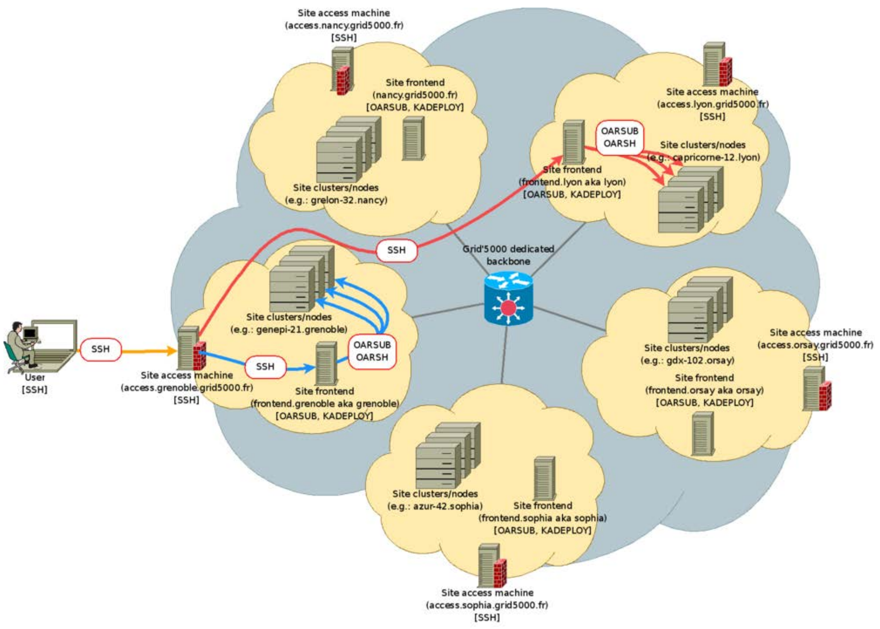
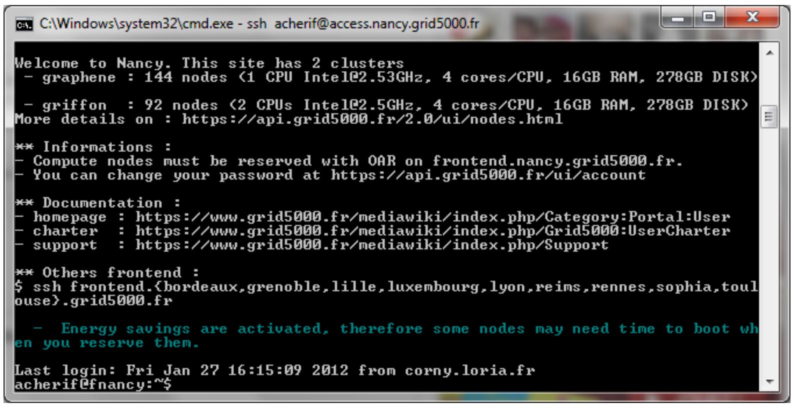


Figure 7.4: Grid'5000 Access with SSH Protocol [2].

In Figure 7.5, we show how we connect to the Nancy frontend. Next step consists in submitting a job by reserving the nodes required as well as the wall-time for the experiment (see Figure 7.6).

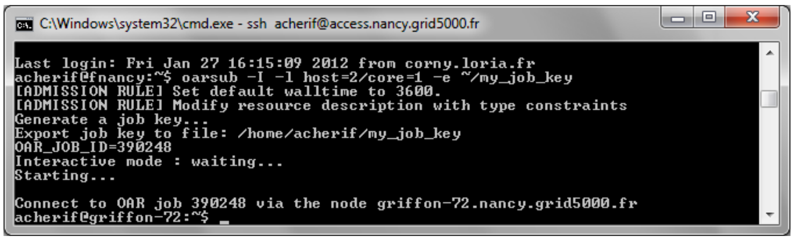

C:\Windows\system32\cmd.exe - ssh acherif@access.nancy.grid5000.fr

```
Welcome to Nancy. This site has 2 clusters
 - graphene : 144 nodes (1 CPU Intel@2.53GHz, 4 cores/CPU, 16GB RAM, 278GB DISK)

 - griffon  : 92 nodes (2 CPUs Intel@2.5GHz, 4 cores/CPU, 16GB RAM, 278GB DISK)
More details on : https://api.grid5000.fr/2.0/ui/nodes.html

** Informations :
- Compute nodes must be reserved with OAR on frontend.nancy.grid5000.fr.
- You can change your password at https://api.grid5000.fr/ui/account

** Documentation :
- homepage  : https://www.grid5000.fr/mediawiki/index.php/Category:Portal:User
- charter   : https://www.grid5000.fr/mediawiki/index.php/Grid5000:UserCharter
- support   : https://www.grid5000.fr/mediawiki/index.php/Support

** Others frontend :
$ ssh frontend.{bordeaux,grenoble,lille,luxembourg,lyon,reims,rennes,sophia,toul
ouse}.grid5000.fr

  -  Energy savings are activated, therefore some nodes may need time to boot wh
en you reserve them.

Last login: Fri Jan 27 16:15:09 2012 from corny.loria.fr
acherif@fnancy:~$
```

Figure 7.5: Connexion to the Grid.

7.3 Experimental Results for the Coordination Layer

Our Java prototypes running on grid'5000 have been developed to forecast the performance of our optimistic access control model on JXTA platform and to assess the feasibility of the approach. Besides this objective, we developed two applications for collaborative editing in computer and mobile devices. The objective of these prototypes is to demonstrate the flexibility of the approach. Toward this goal, two scenarios have been selected, exhibiting two rather different application's profiles regarding the way the


C:\Windows\system32\cmd.exe - ssh acherif@access.nancy.grid5000.fr

```
Last login: Fri Jan 27 16:15:09 2012 from corny.loria.fr
acherif@fnancy:~$ oarsub -I -l host=2/core=1 -e ~/my_job_key
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
Generate a job key...
Export job key to file: /home/acherif/my_job_key
OAR_JOB_ID=390248
Interactive mode : waiting...
Starting...

Connect to OAR job 390248 via the node griffon-72.nancy.grid5000.fr
acherif@griffon-72:~$ _
```

Figure 7.6: Job Creation.

information is administrated (single-administrator versus multi-administrator), the type of this information (html documents versus shared calendar) with the real time response time requirements.

The application scenario deals with collaborative works. A community of users organises a data sharing space to exchange and modify textual data like html documents or agendas, address books, etc. Our approach permits to easily define simple and powerful access control rules on sensitive data (e.g., specific appointments in an agenda or some paragraphs on a working document) while handling rule dynamic (new collaborators join or leave the community).

In the following, we first give details about the experimentation metrics. We then analyse the performance of the coordination layer. Next, we study the performance of our access control layer, and access right checking. Finally, the global performance of the proposed solution is discussed to highlight the response time evaluation in the presence of an access control layer.

7.3.1 Metrics

For our evaluation performance, we consider the following times (see Figure 7.7) used to calculate the response time of our model:

- t_g is the time required to generate a local operation;
- t_i is the time to integrate a remote operation;
- t_c is the time required to communicate an operation to a peer through network;
- t_r is the response time, we can obviously see that $t_r = t_g + t_i + t_c$.

In general, it is established that the OT-based collaborative editors must provide $t_r < 100ms$ [59]. The lower the response time is, the better the collaboration is. As a matter of fact, the user is able to see different updates made on the shared documents instantly.

7.3.2 Performances of the Desktop Application

A prototype P2PEdit (see Figure 7.8) based on our flexible access control model has been implemented in Java. It supports the collaborative editing of HTML pages and it is deployed on P2P JXTA platform. In this prototype, a user can create a HTML page from scratch by opening a new collaboration group. Thus, he is the administrator of this group. Other users may join the group (see Figure 7.9) to participate in HTML page editing, as they may leave this group at any time. Using JXTA platform, users exchange their operations in real-time in order to support WYSIWIS (What You See Is What I See) principle.

Initially, we did three experiences to study the performance of the coordination model OPTIC. Then, we compare these results with the improved version of the coordination algorithm. It should be pointed out that the performance of the coordination layer in OPTIC is mostly determined by the percentage

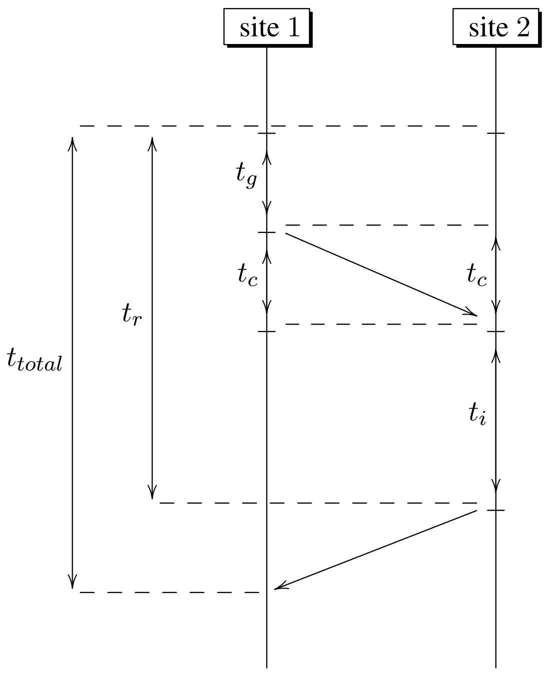
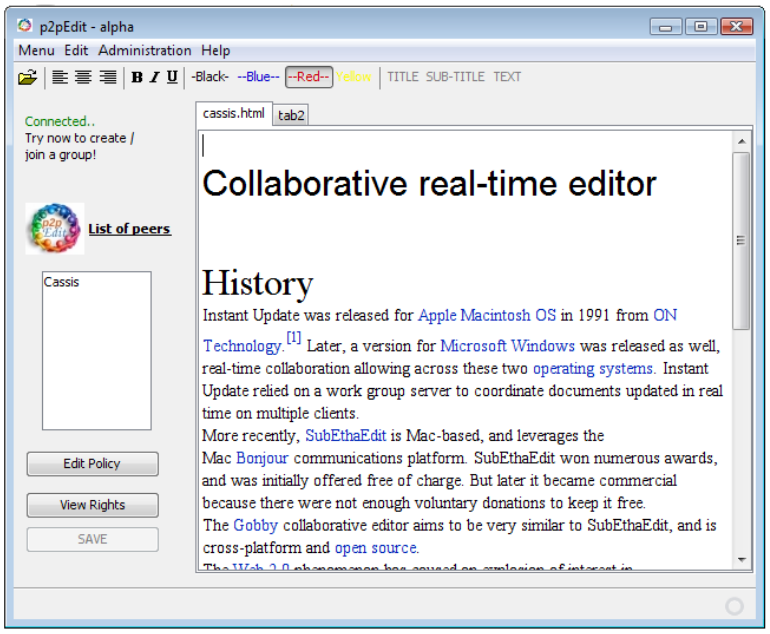


Figure 7.7: Response Time Metrics.


p2pEdit - alpha
Menu Edit Administration Help
-Black- --Blue-- --Red-- Yellow TITLE SUB-TITLE TEXT
Connected..
Try now to create / join a group!
List of peers
Cassis
Edit Policy
View Rights
SAVE
cassis.html
tab2

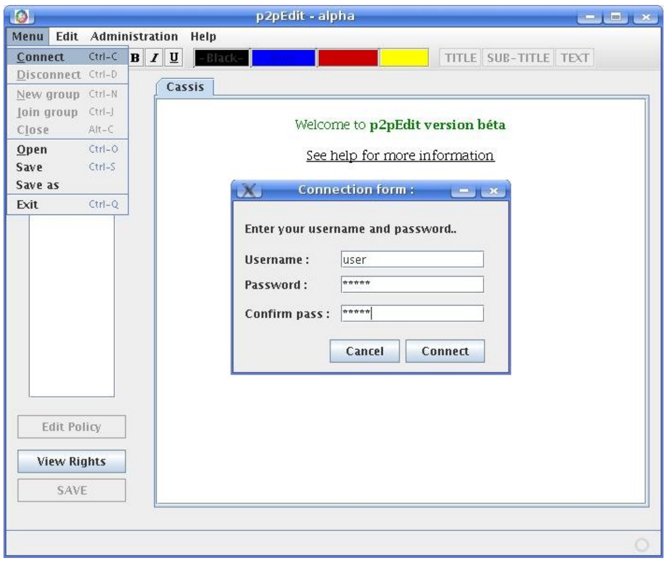
Collaborative real-time editor

History

Instant Update was released for Apple Macintosh OS in 1991 from ON Technology.[1] Later, a version for Microsoft Windows was released as well, real-time collaboration allowing across these two operating systems. Instant Update relied on a work group server to coordinate documents updated in real time on multiple clients.
More recently, SubEthaEdit is Mac-based, and leverages the Mac Bonjour communications platform. SubEthaEdit won numerous awards, and was initially offered free of charge. But later it became commercial because there were not enough voluntary donations to keep it free.
The Gobby collaborative editor aims to be very similar to SubEthaEdit, and is cross-platform and open source.

Figure 7.8: p2pEdit Tool.

of insertion requests inside the cooperative log. Indeed, the worst case corresponds to integrating an insert cooperative request against a cooperative log containing 0% insertions as the canonization process



Welcome to p2pEdit version béta

See help for more information

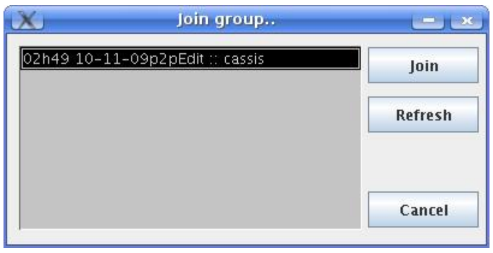
Enter your username and password..

Username : user

Password : *****

Confirm pass : *****

(a) Connect form



02h49 10-11-09p2pEdit :: cassis

(b) Join form

Figure 7.9: Connect and Join Forms.

requires to replace the insertion on the top of the log.

Figure 7.10 depicts the execution time[22] required to integrate a cooperative request for different percentages of insertions inside the cooperative log H. These measurements reflect the times t_g (left), t_i (right) and their sum t_r (middle). This experiment shows that the execution time falls within 100ms for all $|H| \leq 5000$ if H contains 0% INS, $|H| \leq 9000$ if H contains 100% INS which is not achieved in SDT and ABT algorithms [59].

Next, we improved the source code of our algorithm as follows:

- Use of hashed data structures to have direct access to each entry in the cooperative log.

- Implementing the cooperative log as two different sub logs, one for insertions and another for deletions and updates. Thus, we simplify and accelerate the canonization procedure.

The final response time of the coordination layer in the worst case (processing time required to integrate an insert request for a log H containing 0% insertion) is given in the curve of Figure 7.11. We obviously see that the response time falls within 100ms for cooperative logs of size 300000 which is a very promising result.

In the following, we present the response time of the coordination framework OPTIC extended for mobile devices to demonstrate both the generic aspect of this framework and the performance of the garbage collector.

7.3.3 Performances of the Mobile Devices Application

To extend OPTIC for mobile devices, a second prototype was implemented using Java Platform, Micro Edition (Java ME)[23] which provides a robust and flexible environment for the embedded applications,

[22]The experiments have been performed under Ubunto Linux kernel 2.6.24-19 with an Intel Pentium-4 2.60 GHz CPU and 768 Mo RAM.

[23] http://java.sun.com/javame/index.jsp

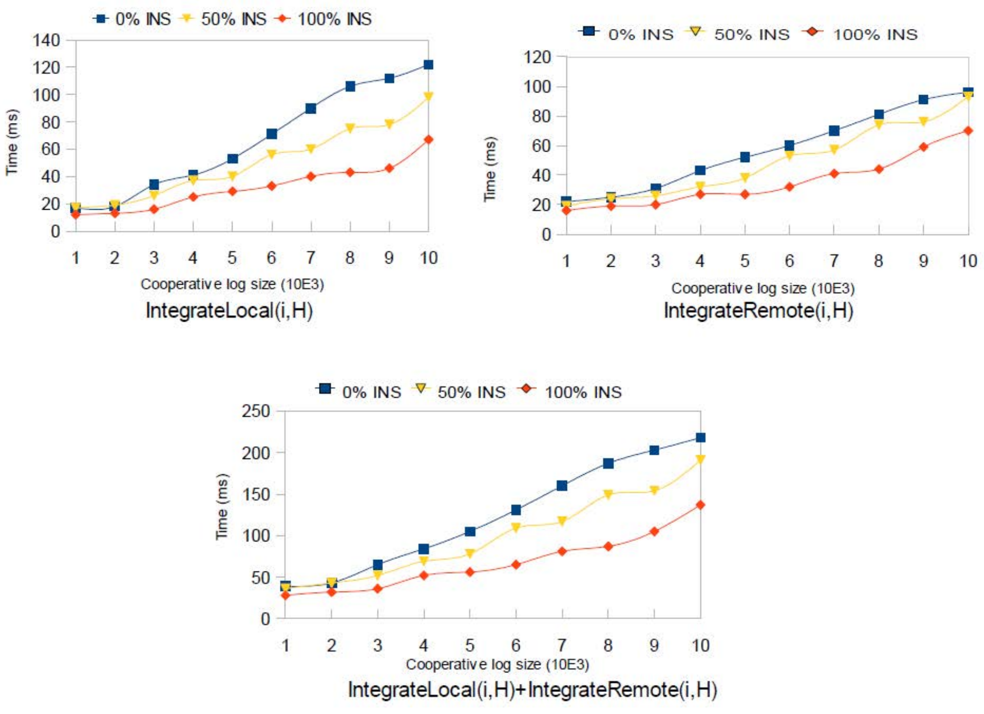


Figure 7.10: Response Time of the OPTIC Framework (worst case).

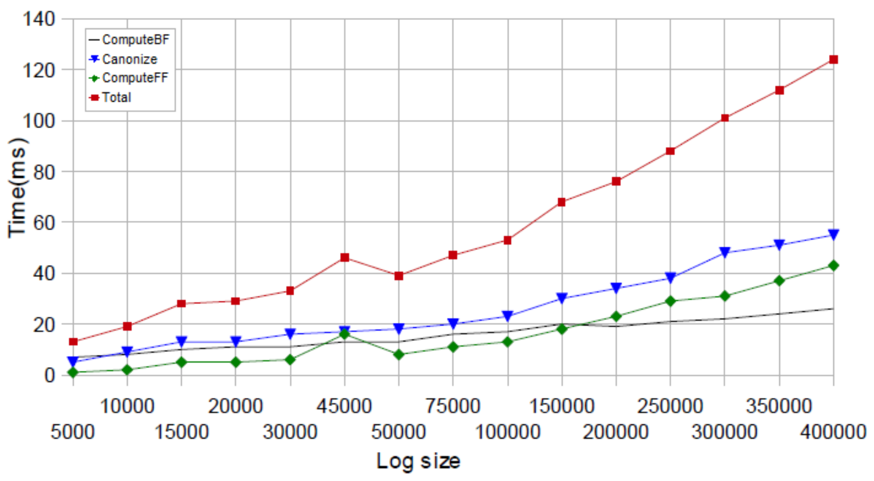
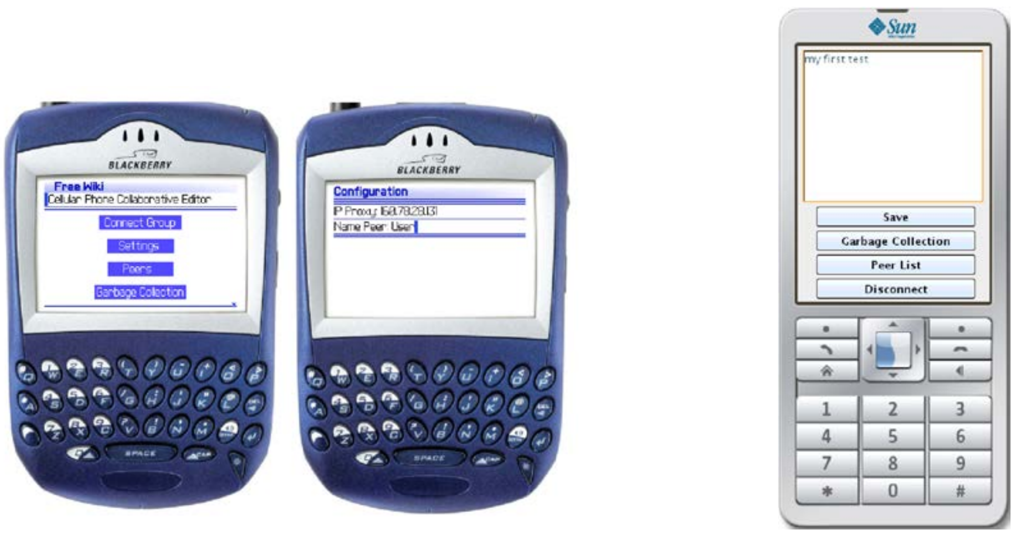


Figure 7.11: Response Time of the Improved Version (worst case).

and proposes two configurations to simulate mobile environments: CDC[24] (Connected Device Configuration) specifies an environment for terminals connected where memory is usually greater than 512 Kb such as tablets screen phones, digital television and cell phones as Nokia 9500, Sony Ericsson P990i and Samsung C6620. CLDC[25] (Connected Limited Device Configuration) target devices with limited or low resources such as mobile phones, PDAs, or light wireless peripheral. For example, BlackBerry, Nokia (6600,E73,N93,...), Motorola (i560,i730,...) and Sumsung (A737,D500,...) offer CLDC envi-

[24]http://java.sun.com/javame/technology/cdc/
[25]http://java.sun.com/products/cldc/

ronment. The realized prototype was developed with netbeans 6.8 under Windows operating system. In Figure 7.12.(a), we see different screen shots of the main windows on the CLDC environment while in Figure 7.12.(b), we illustrate a screen shot of our prototype implemented for CDC environment.



Cellular Phone Collaborative Editor

IP Proxy: 168.78.28.131

Name Peer: User

my first test

(a) Screen shots of the CLDC prototype. (b)Screen shot of the CDC prototype.

Figure 7.12: Screen Shots of the Mobile prototypes.

7.3.3.1 Response Time

To investigate the performances of our prototype realized for mobile phones, we simulate experimental tests for the behaviour of the two developed models (CDC and CLDC). Our experiments depict the time required to generate a cooperative operation and integrate it at a remote site in the worst case.

Figure 7.13 shows the response time for different log size values for the CDC environment. These measurements reflect the times t_g, t_i and their sum t_r. We conclude from this experiment that the execution time falls within $100ms$ for all $|H| \leq 27500$.

Log size	t_g	t_i	t_r
1000	0	0	0
2500	0	0	0
5000	0	15	15
7500	0	15	15
10000	16	15	31
12500	16	15	31
15000	31	15	46
17500	47	31	78
20000	47	32	79
30000	63	47	110

Table 7.1: Response Time of CDC Application.

We also evaluate the performance of the p2pEdit framework for CLDC environment (see Figure 7.14). Experiments show the response time for different log size values. In this case, the execution time falls within $100ms$ for all $|H| \leq 14000$.

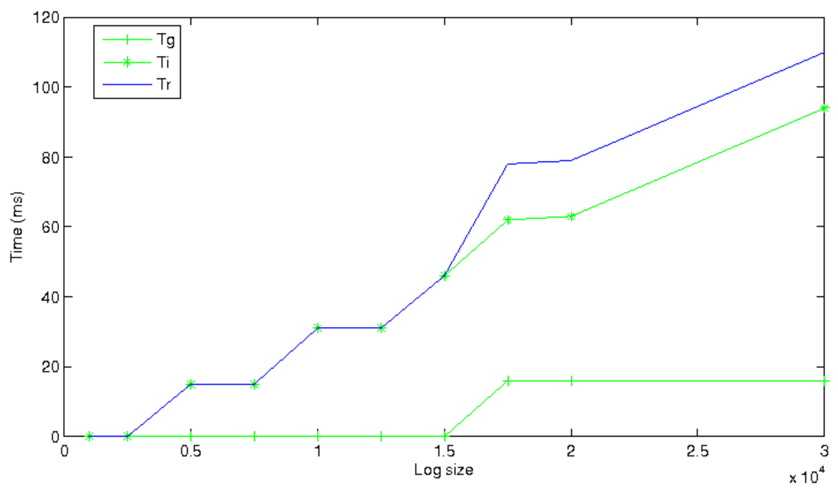


Figure 7.13: Response Time for CDC Environment.

The conclusions that can be drawn from this figure are threefold. First, the coordination layer has very good performance compared with [59] , explained by the fact that the semantic dependency decreases the whole response time of the coordination layer. Second, the performance of our CDC prototype performs better than the CLDC one due to the kind of targeted architectures for both environments. Third, we conclude that beyond 27500 (respc. 14000) for the CDC (respc. CLDC) environment, logs should be cleaned through garbage collection mechanism. This result is encouraging since it allows for a large number of operations that users can exchange before reaching the maximal born tolerated for response time (and thus a lower quality for collaboration). When reaching given sizes, all users will start again the collaboration with empty logs which makes the collaboration more and more efficient.

In the following, we investigate the time processing required by our distributed garbage collector designed for the OPTIC framework.

Log size	t_g	t_i	t_r
1000	0	0	0
2500	0	16	16
5000	0	31	31
7500	15	47	62
10000	16	62	78
12500	16	64	80
15000	31	78	109
17500	32	78	110
20000	32	124	156

Table 7.2: Response Time for CLDC Environment.

7.3.3.2 Garbage Collection Time

The following experiment was realized to measure the processing time required to perform garbage collection and collaborate again. The experiment measures this time for different values of the leaves

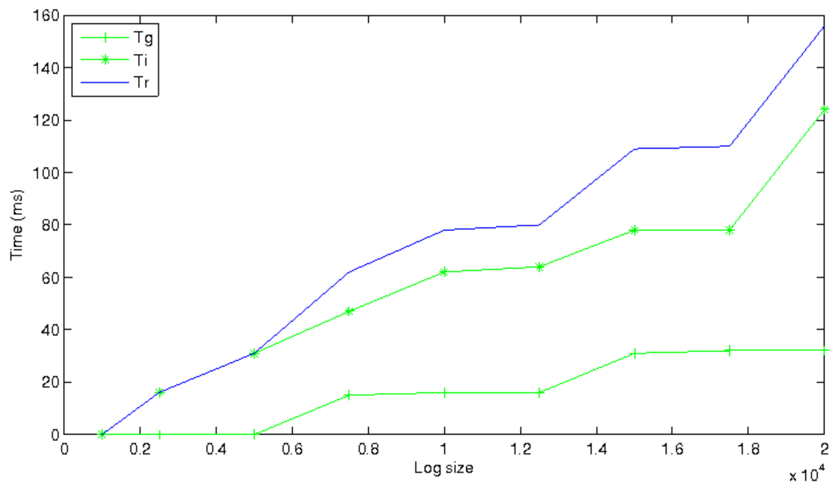


Figure 7.14: Response Time for CLDC Environment.

set (the set of cooperative operations that represent the leaves of the dependency tree of the garbage initiator). According to the results shown in Figure 7.15, we deduce that the blockage time is acceptable till the size 10000, where users have to wait only 5 seconds to start again their collaboration. It should be noted that 10000 does not represent the log size but rather the set of leaves size which means that the log could contains more than this number of operations.

Number of leaves	Garbage time
100	93
250	140
350	188
500	256
1000	515
2000	1031
3000	1469
4000	1907
5000	2359
10000	4719
15000	7234
20000	9406
50000	32813

Table 7.3: Time Required by the Garbage Collector.

7.4 Experimental Results for the Security Layer

In this section we assess the performances of our approach according to the size of the collaborating group. We have developed a second Java prototype p2pAgenda and measured the performance of our prototype on the grid.

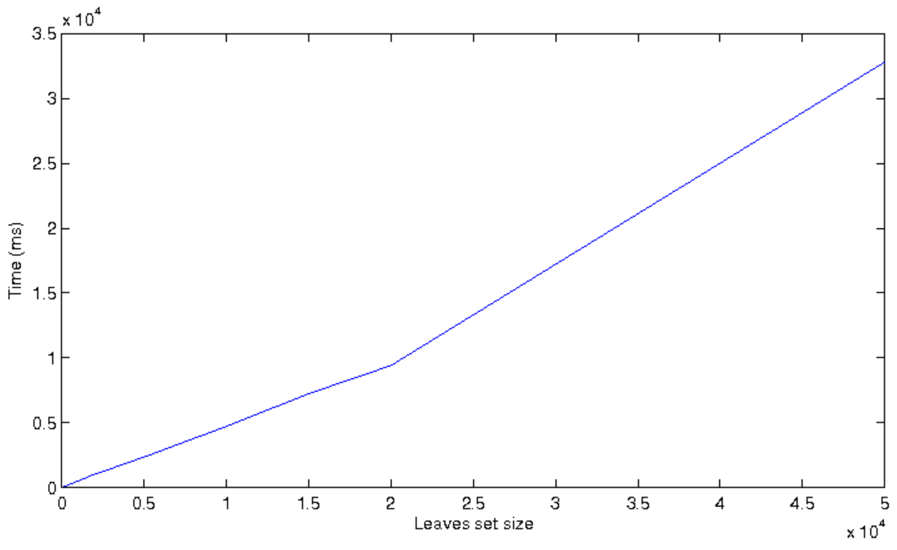


Figure 7.15: Garbage Collection Time Variation with Leaves Set Size.

7.4.1 P2PAgenda Prototype

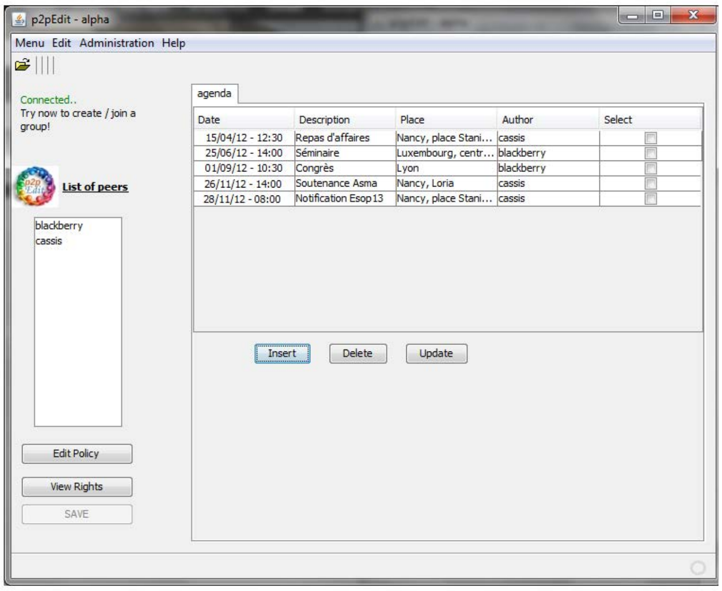
A second prototype P2PAgenda[26] (see Figure 7.16) was implemented and extended by our access control layer to protect shared data in a decentralized fashion. This prototype was published under the APP reference IDDN.FR.001.150007.000.S.P.2010.000.10000. It allows users to concurrently edit a shared agenda as well as access rights in order to protect any part of the shared agenda. Thus each user has a list of access rules that he can update according to the ownership policy. Each user inserts meetings in the agenda and can specify security rules to control the access of other users to his meetings (see Figure 7.17). He can dynamically add and remove different authorizations for accessing to his shared objects according to his needs.

Experiments were done on the distributed platform Grid5000 [2] in order to determine the response time required to see the effect of a cooperative operation in remote sites after they are checked against the policy object.

7.4.2 Processing Time for Checking Local and Remote Requests

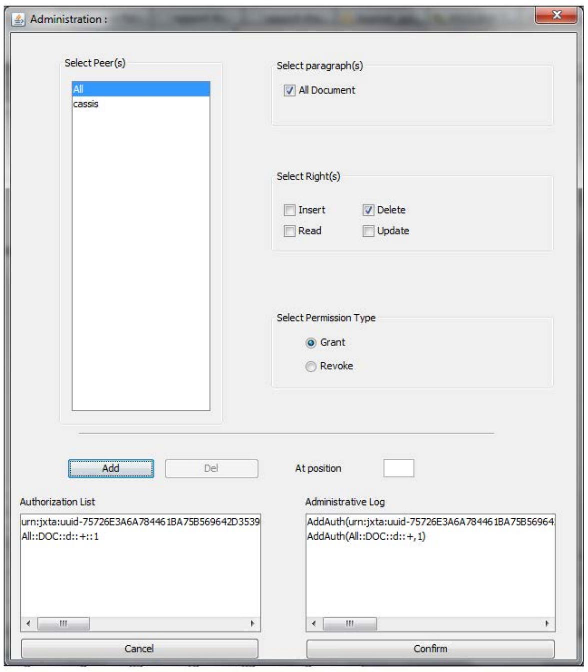
To assess the performance of our approach when different owner policies sizes are faced, we measured the performance of our prototype on randomly generated cooperative requests. For these cooperative requests we generated random access control authorizations. The generated access control policy was simple and not optimized (*i.e.* it contains authorization redundancies). Figure 7.18 reports the results for bench values given in Table 7.4. The figure plots the execution time required for checking both local and remote requests. We show that our method tackles well very large policies and logs and produces a throughput ranging from 1,18 to 16 ms, depending on the authorization-list or administrative log size (in Algorithms 5 and 6, we have to explore either the policy or the administrative log to decide whether a cooperative request is legal or not). These preliminary results are encouraging when compared with current response time recommended for collaborative applications (*100 ms* [59]). Indeed, the experiment results show that the time required to check an operation is low (about 14 ms for a remote operation and 16 ms for a local one with an owner policy containing 10000 authorizations).

[26]http://www.loria.fr/ imine/tools/p2pAgenda/p2pAgenda.htm



Date	Description	Place	Author	Select
15/04/12 - 12:30	Repas d'affaires	Nancy, place Stani…	cassis	☐
25/06/12 - 14:00	Séminaire	Luxembourg, centr…	blackberry	☐
01/09/12 - 10:30	Congrès	Lyon	blackberry	☐
26/11/12 - 14:00	Soutenance Asma	Nancy, Loria	cassis	☐
28/11/12 - 08:00	Notification Esop13	Nancy, place Stani…	cassis	☐

Figure 7.16: Main Screen of p2pAgenda.



Select paragraph(s): [✓ All Document]

Select Right(s): Insert [✓ Delete] Read Update

Select Permission Type: [✓ Grant] Revoke

At position

Authorization List

```
urn:jxta:uuid-75726E3A6A784461BA75B569642D3539
All::DOC::d::+::1
```

Administrative Log

```
AddAuth(urn:jxta:uuid-75726E3A6A784461BA75B56964
AddAuth(All::DOC::d::+,1)
```

Figure 7.17: Administrative Interface of p2pAgenda.

Note that this experiment concerns the worst case where only the last rule in the log grants the right to be performed by the cooperative operation. This good result is explained by the use of hash set data structure to have direct access to our objects in each authorization. Consequently, the check time of one authorization in the owner policy is constant which explains the linear aspect of the curve when varying the authorizations number. For the sake of our results, it would be better to edit non redundant policies to optimize local check. Moreover, administrative logs may be pruned or deleted thanks to the garbage

Owner policy/log size	Local check	Remote check
1000	1,18	2,43
2000	2,66	2,9
3000	4,34	4,44
4000	6,1	6,4
5000	7,91	8,37
6000	9,86	11,01
7000	11,77	11,93
8000	13,1	12,43
9000	14,8	14,09
10000	16	14,13

Table 7.4: Time Processing of Local and Check Remote of a Cooperative Request.

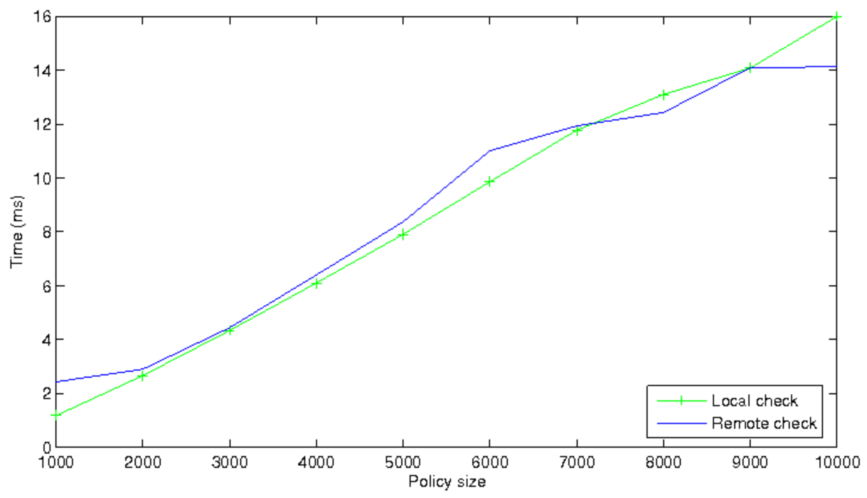


Figure 7.18: Check Time for Local and Remote Requests.

collection mechanism presented in Chapter 3.

7.4.3 Response Time Variation with Peers Number

To validate our experimental setup, we also measured the response time of our security layer for different sizes of collaborating groups of users. The second experiment shows the response time of the application for a cooperative log containing 150000 and a policy formed by many owner policies where each one contains 5000 authorizations. The response time corresponds to the time required by an operation to be seen at a remote site. Two cases are presented in the curve, the first one concerns the case when the policy remains the same *i.e* the context does not change at the sender and the receiver site, and the second one concerns the case when the policy changes during exchanging updates between users.

We can draw several conclusions from Figure 7.19. First, we can obviously see that for a cooperative log containing 150000 operations and a policy containing 80 owner policies where each one contains 5000 authorizations, the response time is less than 100 ms. Second, the number of peers does not degrade the performances of our system since the only parameter that has an impact on the response time when the group size grows is θ (θ corresponds to the maximum bound needed so that a message

Group size	T_r (same context)	T_r (different contexts)
10	83	91
20	84	92
30	88	96
40	87	95
50	89	97
60	89	97
70	90	98
80	90.5	98.5

Table 7.5: Response Time of Integrating a Cooperative Request.

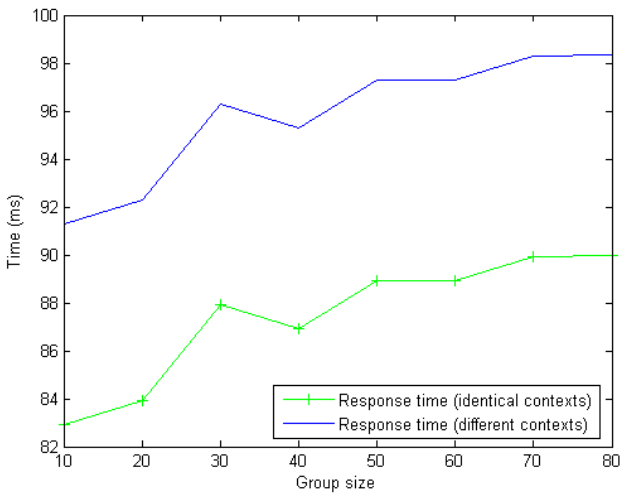


Figure 7.19: Response Time.

traverses the network from any sender to its receiver, see Section B.2) which only depends on the network configuration. Finally, we notice that security policy takes less processing time when the context remains the same which is logical since the check does not produce overhead as the policy does not change concurrently.

7.4.4 Access Control Overhead

To assess the efficiency of our access control layer, we compare the execution time required by the coordination layer, the access control and the communication between peers. In Figure 7.20, integration corresponds to the time required by the underlying coordination layer to execute a cooperative request. Access control, corresponds to the execution time required to evaluate the access control right for a cooperative operation by checking it against the corresponding authorization list or log. To measure the impact of the access control policy, we calculate the relative cost of the access control. Obviously, the relative cost of the access control is around 21% of the total cost. These measurements are promising and demonstrate the applicability of the solution.

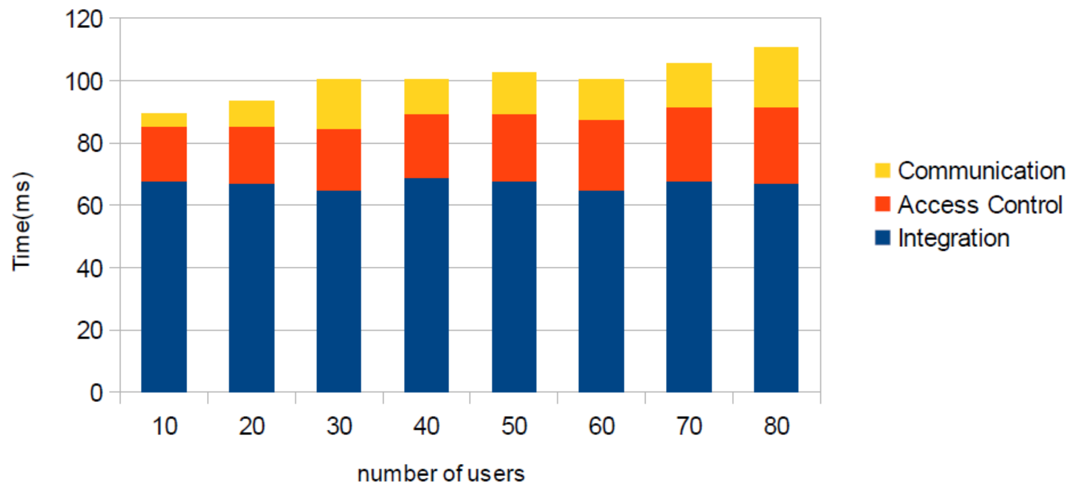


Figure 7.20: Access Control Overhead.

7.5 Conclusion

In this Chapter, we presented the OPTIC framework on the top of which we have implemented our access control model. Moreover, we presented the optimizations that extend this framework mainly, the garbage collection mechanism allowing its deployment on mobile devices.

Performance measurements were done on the distributed platform grid'5000 to see the behaviour of our model for both desktop and mobile devices environments. This study, obviously showed that our model has a good response time which is a very encouraging result.

Chapter 8

General Conclusion

Contents

8.1 Summary

The purpose of this thesis was to develop a generic approach to enabling, understanding and managing access control in collaborative applications in particular RCE.

8.1.1 Why RCE Require Access Control?

Security is a vital component of any application or environment. Indeed, it protects data from unauthorized or improper modifications [80].

Important factors motivate today the access control to be enforced in RCE. With the diversity of possible attacks and the different security properties that a security model should consider, enforcing security is being more and more complex. In fact, the growing interest for different forms of data dissemination, the emergence of the data sharing between many users are different factors that lead to address the access control problem in collaborative applications. Indeed, they provide information and resources characterized by different degrees of sensitivity such as customer data in a financial application or patient data in a healthcare application. Meanwhile, they allow many trusted and untrusted users to share information.

Even though there are several encryption schemes that can be used to secure data, they suffer from a static way of sharing data [16]. As more and more people are attracted by collaborative applications, it becomes increasingly important to provide these applications with new security protocols and models to enforce protection while meeting the requirements of these applications (*e.g.* distribution, human factors, high local responsiveness, *etc.*). These models have to take into account the collaborative aspects in order to prevent the dissemination of sensitive information which may threat the security of both individuals and companies.

8.1.2 Access Control Requirements and Issues

Collaborative editors are specific applications since they must take into account human factors [48]. Consequently, they require:

1. *High local responsiveness* [32,48,99,100].
2. *High concurrency*: [32,48,99]
3. *Consistency* [32,48,99].
4. *Decentralized coordination* [48]
5. *Scalability* [48].

Accordingly, an access control model for RCE must take into account these requirements in order to keep the full potential of such applications. We investigated which problems could be raised by adding an access control layer to a real time collaborative editor. As illustrated in **Chapter 2**, dynamic systems pose new and significant challenges to access control. In such systems, shared documents as well as the collaborating group are subject to many changes. This requires to well manage the real time update of the access control policy. In particular, it is very difficult to meet RCE requirements when deploying an access control layer due to the high communication latencies (*e.g.* Internet).

Our objective is to take advantage of replication techniques in order to overcome the latency issue. Thus, we proposed to replicate not only the shared document at all collaborating sites but also the security data-structure. Consequently, our access control model is capable of evaluating dynamic access rules on a shared document where every update of the shared document is checked against the local copy of the security policy replica.

Nevertheless, maintaining security policy in a decentralized fashion is really a hard task since, policy views may be different from one site to another which may lead to security holes, not to mention convergence issues. Indeed, if security policy can be temporarily inconsistent, any given action may be authorized at one site and yet denied at another. This is troublesome since it may lead to permanent divergent state of the shared document. Subsequently, we addressed the problem of how policies can be updated in real-time in order to meet the real time aspect of collaborative editors. The main contribution focus on reasoning about the real time evolution of access control policies and shared documents concurrently, as well as maintaining the convergence of the shared documents. We proposed in **Chapter 3** an optimistic access control model in the sense that temporary violations of the security policy are permitted. In order to achieve convergence, illegal document updates are undone to restore a correct document view.

8.1.3 Undoing Operations in RCE

Undo feature turns out as an important feature for maintaining convergence of the shared documents while dynamically managing the security policy updates. Since, undoability is still an open issue in distributed collaborative editors, we have dedicated the second part of our thesis to addressing the undoability problem. We have briefly presented the state of the art in **Chapter 5** in order to show the limits of existing solutions.

Based on the constraint problem theory, we demonstrated in **Chapter 6** that it is impossible to reach undoability for non commutative operations. This is a very important result, of relevance for both researcher and developers since it simplifies the undo problematic by giving additional guidelines and necessary and sufficient condition for achieving correctness of the undoability problem.

To overcome this impossibility result, we presented a generic approach to undo operations in OT-based collaborative applications. Our solution consists in an enhanced set of operations with a new semantic of the idle operation. We defined a set of transformation rules in the first logic order in order to take into account the new kind of operations (*i.e.* idle and inverse operations). Finally, we demonstrated these rules indeed preserve both transformation and inverse properties. Clearly, our solution does not produce overhead since it has a linear complexity thus fulfils the high local responsiveness requirement. Moreover, it can be easily applied to any coordination framework since it only extends the set of operations basing on the Hide relation easily captured from the transformation function associated with the initial set of operations.

8.2 Summary of the Contributions

Access Control Requirements. A first contribution of this dissertation is a detailed survey of existing access control classes and models as well as their shortcomings. We also define an enhanced set of requirements that should be fulfilled by an access control model in order to meet existing real time collaborative editors.

Access Control Model. The cornerstone of this thesis is the definition of an optimistic access control approach that could be implemented on the top of any log-based real time collaborative editor. In order to achieve our goals and for the sake of our results, we began by conceiving a simple access control model inspired by the models of [79] since it discusses security policy replication and [89] since it resorts to a policy specification that allows for a dynamic access control management. As in [79], we replicate the access control policy in order to allow for high local responsiveness. We therefore introduce the optimistic fashion to manage access rights. Our access rules specifications resemble to that of [89]. However, we simplify the set of operations and access rights in order to focus on the convergence issues raised by concurrent policy and document updates. Moreover, since our model is based on administrative logs to track policy updates, we provide a garbage collection scheme to well manage these logs in a distributed fashion.

Implementation and Experimental Study. To validate our model, we provided a formal proof of correctness in Chapter 4. We also developed two JAVA prototypes P2PEdit and P2PAgenda build on on the top of the OPTIC collaborative editing framework [48]. Our experimental results have been obtained from these JAVA prototypes running on the distributed platform GRID'5000 to demonstrate the feasibility our approach. The relative cost of the access control is around 21% of the total cost. These measurements are promising and demonstrate the applicability of the solution. This work demonstrates that replicated security solutions give rise to interesting research perspectives. They also may have a large impact on a growing scale of collaborative applications.

A Distributed Garbage Collector for OPTIC. Another important contribution of this thesis is the design of a decentralized garbage collector for the OPTIC framework. Our solution consists in a distributed approach based on the semantic dependency relation defined in [48], to capture a global view on the collaboration state before processing the pruning of cooperative logs in a reasonable time. The model was implemented in an extended version of the OPTIC framework dedicated to mobile frameworks. Our experimental results demonstrate the relative cost of the garbage process and show it offers a good performance. Indeed, the cost of the garbage process is around 5 s for a set of leaves containing 10000 document updates.

Undoability in Collaborative Editors. The fourth contribution of our thesis is a generic undo approach for distributed collaborative applications. Indeed, undoing operations is motivated by the optimistic aspect of our access control model since we allow for temporary policy violations. However, undoability represents a hard task since existing undo solutions either are incorrect or have bad complexity which is to be discarded in RCE. To overcome these difficulties, we first address the undoability problem from a theoretical point of view. This study allowed us to reason about a necessary and sufficient condition for undoability. Yet, OT was proposed to go beyond commutativity, we proved that it is impossible to achieve undoability for non commutative operations. Accordingly, we proposed to extend the set of operations with a new semantic of the idle operation and proved our solution achieves undoability while preserving the convergence of the document.

8.3 Directions for Future Work

Our work can be extended in several directions.

Delegation of Access Rights. Even though being flexible and generic, our access control model does not allow for delegation of rights which would be a very important extension of our proposed access control model. Indeed, there can be many reasons for a user to delegate his rights to one or more users among the collaborating group; for instance, the user may physically disconnect from the collaborative application. A common solution to the disconnection problem is to delegate access rights, whereby collaborators pass on rights to other users in order to perform actions on their behalf. There are different modes of delegations; static delegation refers to situations where actions are known in advance of the delegation while dynamic delegation occurs when tasks are unknown at the time of delegation. In both cases, it would be very interesting to investigate issues raised by the concurrency between delegations and policy as well as document updates in a dynamic environments.

Trust Management. Distributed collaborative applications allow to build virtual communities where many users use and share some resources without being members of the same organization. Thus, it is important for a successful cooperation, that users trust each other. Trust is especially required to well and correctly manage delegations of rights mainly without face to face relationships. Trust is defined as "the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party" [63]. Subsequently, an important direction for future work is to extend our access control model with an additional trust component that allows users to be aware about the reputation of other members in the collaborating group.

Generalization of the Impossibility Result. The impossibility result we provide concerns consistent collaborative objects with every operation different from its inverse. However, there are operations that are equal to their inverse such as the boolean operation Not, or the standard idle operation. Thus, it is worth to generalize the impossibility result in order to cover such operations.

Automated Proof of the Undo Solution. Even though, we proposed a hand proof of our generic solution for undoing operations in collaborative editors, it is desirable to use a theorem prover to automatically prove our lemmas and theorems. This makes the solution more trusted since there are many cases to address in order to prove that all properties are fulfilled by our proposed transformation rules. We have already began the formalization of our solution and intend to use the Spaß prover[27] for demonstrating automatically its correctness.

Implementation of the Undo Module. The undo solution was partially implemented on the OPTIC framework. As a future work, we will complete the implementation of all the transformation rules on the latest version of the OPTIC framework and measure the performances of the our undo procedure.

[27]http://www.spass-prover.org/

List of Figures

List of Tables

List of Algorithms

Appendix A

Correctness Proof of the Undo Approach

In this Appendix, we investigate the correctness proof of our undo solution.We show the transformation rules defined in Chapter 6 satisfy both inverse and transformation properties.

Section A.1 presents the properties of the relation Hide. Next, the preservation of inverse properties is proved in Section A.2 as for that of transformation properties in Section A.3.

A.1 Hide Relation Properties

In this section, we give some properties of the Hide relation.

The following lemma states that the Hide relation is preserved by transformation (see Lemma 8.1.1).

Lemma 8.1.1.

Given three operations op_1, op_2 and op_3, then

$$\left.\begin{array}{l} Hide(op_1, op_3) \\ \wedge\neg Hide(op_2, op_3) \\ \wedge\neg Hide(op_2, op_1) \end{array}\right\} \Rightarrow Hide(IT(op_1, op_2), IT(op_3, op_2))$$

Proof. Since $\neg Hide(op_2, op_3)$ and $\neg Hide(op_2, op_1)$, we deduce that neither $IT(op_3, op_2)$, nor $IT(op_2, op_1)$ is an idle operation. Furthermore, $Hide(op_1, op_3)$ means that both operations op_1 and op_3 alter the same element of the shared object. Consequently, including the effect of the same operation op_2 in op_1 and op_3 respectively produces two operations altering the same element of the shared document. Thereby, $Hide(IT(op_1, op_2), IT(op_3, op_2))$ holds.

Formally, the result is proved by the use of **TP2** property. According to Definition 6.3.9, proving $Hide(IT(op_1, op_2), IT(op_3, op_2))$ is equivalent to proving $IT(op_1, op_2) \cdot IT(IT(op_3, op_2), IT(op_1, op_2)) \equiv IT(op_1, op_2)$. Subsequently, it comes from **TP2** and Lemma 6.3.8 that:

$$\begin{aligned} IT(op_1, op_2) \cdot IT(IT(op_3, op_2), IT(op_1, op_2)) &\equiv IT(op_1, op_2) \cdot IT(IT(op_3, op_1), IT(op_2, op_1)) \\ &\equiv IT(op_1, op_2) \end{aligned}$$

□

The following Lemma states that if two operations hide each other then they are equal.

Lemma 8.1.2.

Given three operations op_1, op_2 and op_3, then

$$Hide(op_1, op_2) \wedge Hide(op_2, op_1) \Rightarrow op_1 = op_2.$$

Proof. According to Definition 6.3.9, we have:

$$Hide(op_1, op_2) \Leftrightarrow op_1 \cdot IT(op_2, op_1) \equiv op_1$$
$$Hide(op_2, op_1) \Leftrightarrow op_2 \cdot IT(op_1, op_2) \equiv op_2$$

According to **TP1** $op_1 \cdot IT(op_2, op_1) \equiv op_2 \cdot IT(op_1, op_2)$ we deduce that $op_1 = op_2$. □

In Lemma 8.1.3, we show that if a given operation op_1 is hidden by two operations op_2 and op_3, then $IT(IT(op_3, op_1), IT(op_2, op_1)) = IT(op_3, op_2)$.

Lemma 8.1.3.

Given three operations op_1, op_2 and op_3, then

$$Hide(op_3, op_2) \wedge Hide(op_2, op_1) \Rightarrow IT(IT(op_3, op_1), IT(op_2, op_1)) = IT(op_3, op_2).$$

Proof. Since $Hide(op_2, op_1)$, then $op_2 \cdot IT(op_1, op_2) \equiv op_2$. According to **TP2**, we have:

$$IT^*(op_3, [op_1 \cdot IT(op_2, op_1)]) = IT^*(op3, [op_2 \cdot IT(op_1, op_2)]) \qquad \textbf{(TP2)}$$
$$IT(IT(op_3, op_1), IT(op_2, op_1)) = IT(op_3, op_2)$$

□

In the following section, we prove that given a transformed RCCO(*e.g* its transformation function and set of operations are extended with idle operations)then **TP2** is preserved.

A.2 Inverse properties Preservation

Since it is possible to avoid **IP2** property by the couple-do-undo pair [97], in the following we focus on **IP3** property and show it is satisfied by our transformation function.

Lemma 8.2.1 shows that our transformation rules allow the preservation of the inverse property **IP3**.

Theorem 8.2.1 (IP3 preservation).

Let op_1 and op_2 be two concurrent operations. Consider $op'_1 = IT(op_1, op_2)$ and $op'_2 = IT(op_2, op_1)$. Then we have:

$$IT(\overline{op_1}, op'_2) = \overline{op'_1}.$$

Proof. Two cases are to be discussed according to the Hide relation between op_1 and op_2

1. $Hide(op_1, op_2)$: according to $\mathbf{TR3}$, we have:

$$\begin{aligned} IT(\overline{op_1}, op_2') &= IT(\overline{op_1}, I(op_2, op_1)) \\ &= \overline{IT(op_1, op_2)} \end{aligned}$$

2. $\neg Hide(op_1, op_2)$: according to $\mathbf{TR3}$, we have:

$$\begin{aligned} IT(\overline{op_1}, op_2') &= \overline{IT(op_1, ET(op_2', op_1)} \\ &= \overline{IT(op_1, op_2)} \end{aligned}$$

Thereby showing that property $\mathbf{IP3}$ is satisfied. □

A.3 Transformation Properties Preservation

In this section, we prove that both $\mathbf{TP1}$ and $\mathbf{TP2}$ are preserved for the enhanced set of operations.

A.3.1 TP1 Preservation

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, then IT satisfies the transformation property $\mathbf{TP1}$ according to the Definition 6.3.8. In Theorem 8.3.2, we show that the relaxed form of C noted $Cr = \langle \mathcal{S}t, \mathcal{O}p', IT' \rangle$ also satisfies $\mathbf{TP1}$.

Theorem 8.3.2 ($\mathbf{TP1}$ Preservation)**.**

Let op_1 and op_2 be two concurrent operations such that $Hide(op_1, op_2)$. Consider $op_1' = IT(op_1, op_2)$ and $op_2' = IT(op_2, op_1)$. Then we have:

$$op_1 \cdot op_2' \equiv op_2 \cdot op_1'.$$

Proof. The proof is trivial since for every operations op_1 and op_2 be two concurrent operations, if $Hide(op_1, op_2)$ then $op_1 \cdot IT(op_2, op_1) \equiv op_1$. Since $\mathbf{TP1}$ is satisfied for the initial set of operations, we also have $op_1 \cdot op_2' \equiv op_1$. Thereby proving that $op_1 \cdot op_2' \equiv op_2 \cdot op_1'$. □

A.3.2 TP2 Preservation

Since two idle operations may be semantically different while having the same hidden operation and the same reflect on the document state, we defining the similarity between two idle operations as follows:

Definition 8.3.1 (Similarity between two Idle operations)**.**

Let three different operations op, op_1 and op_2. Consider the two idle operations $i_1 = I(op, op_1)$ and $i_2 = I(op, op_2$ then i_1 is similar to i_2 (noted $i_1 \approx i_2$) iff $i_1 = I(op, op_1)$.

Even though they are not equal two similar operations have the same effect on the state. For this reason, we relax the **TP2** property to cover idle operations as follows:

Definition 8.3.2 (Relaxed **TP2** (**RTP2**))**.**

Let op_1, op_2 and op_3 be three concurrent operations. Consider $op'_1 = IT(op_1, op_2)$ and $op'_2 = IT(op_2, op_1)$. Suppose that one or more of the these operations is an idle operation. We say that IT preserve **RTP2** iff

1. $IT^*(op_3, [op_1 \cdot op'_2]) = IT^*(op_3, [op_2 \cdot op'_1])$; or
2. $IT^*(op_3, [op_1 \cdot op'_2]) \approx IT^*(op_3, [op_2 \cdot op'_1])$.

Next, we show that the **RTP2** property is satisfied by the relaxed object.

A.3.2.1 RTP2 for Normal Operations Transformed Against Idle Operations

The following Lemma shows that transforming two operations related with the Hide relation against a third one preserves the **RTP2** property.

Lemma 8.3.4 (**RTP2** for Normal Operations)**.**

Given a RCCO $Cr = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, then the relaxed **TP2** property is satisfied for normal operations.

Proof. Let op_1, op_2 and op_3 be three normal concurrent operations from $\mathcal{O}p$, then three cases are to be studied:

1. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: two sub cases are to be discussed:

   (a) $Hide(op_2, op_3)$ and $Hide(op_1, op_3)$: accordingly, we have:

$$\begin{aligned} IT(op_3, op_1) &= I(op_3, op_1) \\ IT^*(op_3, op_1 \cdot op'_2) &= IT(I(op_3, op_1), I(op_2, op_1)) \\ &= I(op_3, op_2) && (\mathbf{TR1} \text{ and } Hide(op_2, op_3)) \end{aligned}$$

   Moreover,

$$\begin{aligned} IT(op_3, op_2) &= I(op_3, op_2) \\ IT^*(op_3, op_2 \cdot op'_1) &= IT(I(op_3, op_2), I(op_1, op_2)) \\ &= I(op_3, op_1) && (\mathbf{TR1} \text{ and } Hide(op_1, op_3)) \end{aligned}$$

   Consequently,

$$IT^*(op_3, op_1 cdot op'_2) \equiv IT^*(op_3, op_2 \cdot op'_1).$$

(b) $\neg Hide(op_2, op_3)$ and $\neg Hide(op_1, op_3)$: int this case, we have:

$$\begin{aligned} IT^*(op_3, op_1 \cdot op_2') &= IT(IT(op_3, op_1), I(op_2, op_1)) \\ &= IT(op_3, op_1) && (\mathbf{TR5}) \end{aligned}$$

Similarly, $IT^*(op_3, op_2 \cdot op_1') = IT(IT(op_3, op_2), I(op_1, op_2)) = IT(op_3, op_1)$ by $\mathbf{TR5}$. According to Lemma 8.1.2, $op_1 = op_2$ then $IT(op_3, op_1) = IT(op_3, op_2)$. Thus,

$$IT^*(op_3, op_1 \cdot op_1') \equiv IT^*(op_3, op_2 \cdot op_1').$$

2. $\neg Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: we discuss the following cases according to the relation between the operations op_1, op_2 and the operation op_3.

   (a) $\neg Hide(op_3, op_1)$ and $\neg Hide(op_2, op_1)$: in this case non of the three operations is transformed to idle operation. Then the result is obtained since $\mathbf{TP2}$ is assumed for the initial set of operation (see Definition 6.3.8).

   (b) $Hide(op_1, op_3)$ and $\neg Hide(op_2, op_3)$: accordingly, we have:

$$\begin{aligned} IT(op_3, op_1) &= I(op_3, op_1) \\ IT^*(op_3, op_1 \cdot op_2') &= IT(I(op_3, op_1), IT(op_2, op_1)) \\ &= I(IT(op_3, op_2), op_1) && (\mathbf{TR2} \text{ and } \neg Hide(op_2, op_3) \end{aligned}$$

   Additionally,

$$\begin{aligned} IT^*(op_3, op_2 \cdot op_1') &= IT(IT(op_3, op_2), IT(op_1, op_2)) \\ &= I(IT(op_3, op_2), op_1) && \text{(Lemma 8.1.1)} \end{aligned}$$

   Hence, $TP2$ is preserved.

3. $Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: the following three cases are faced;

   (a) $Hide(op_1, op_3)$ and $\neg Hide(op_2, op_3)$:

$$\begin{aligned} IT^*(op_3, op_1 \cdot IT(op_2, op_1)) &= IT(IT(op_3, op_1), IT(op_2, op_1)) \\ &= IT(I(op_3, op_1), I(op_2, op_1)) \\ &= I(IT(op_3, op_2), op_1) && (\mathbf{TR1} \text{ and } \neg Hide(op_2, op_3)) \end{aligned}$$

   Moreover,

$$\begin{aligned} IT^*(op_3, op_.; IT(op_1, op_2)) &= IT(IT(op_3, op_2), IT(op_1, op_2)) \\ &= I(IT(op_3, op_2), op_1) && \text{(Lemma 8.1.1)} \end{aligned}$$

   Consequently, the property $\mathbf{RTP2}$ is satisfied.

(b) $Hide(op_1, op_3)$ and $Hide(op_2, op_3)$:

$$\begin{aligned} IT^*(op_3, op_1 \cdot IT(op_2, op_1)) &= IT(IT(op_3, op_1), IT(op_2, op_1)) \\ &= IT(I(op_3, op_1), I(op_2, op_1)) \\ &= I(op_3, op_2) && (\textbf{TR1} \text{ and } Hide(op_2, op_3)) \end{aligned}$$

Moreover,

$$\begin{aligned} IT^*(op_3, op_2 \cdot IT(op_1, op_2)) &= IT(IT(op_3, op_2), IT(op_1, op_2)) \\ &= IT(I(op_3, op_2), IT(op_1, op_2)) \\ &= I(op_3, op_1) && (\textbf{TR2} \text{ and } Hide(op_1, op_3)) \end{aligned}$$

Thereby showing that the property **RTP2** is preserved.

(c) $\neg Hide(op_1, op_3)$ and $\neg Hide(op_2, op_3)$:

$$\begin{aligned} IT^*(op_3, op_1 \cdot IT(op_2, op_1)) &= IT(IT(op_3, op_1), IT(op_2, op_1)) \\ &= IT(IT(op_3, op_1), I(op_2, op_1)) \\ &= IT(op_3, op_1) && (\textbf{TR5}) \end{aligned}$$

Moreover,

$$\begin{aligned} IT^*(op_3, op_2 \cdot IT(op_1, op_2)) &= IT(IT(op_3, op_2), IT(op_1, op_2)) \\ &= IT(IT(op_3, op_2), IT(op_1, op_2)) \\ &= IT(op_3, op_1) && (\text{Lemma 8.1.3}) \end{aligned}$$

Consequently **TP2** is preserved.

Thereby, proving that **TP2** is preserved. □

A.3.2.2 RTP2 for Hidden Operations

To prove that the property **RTP2** is satisfied by idle operations, we have to show that two similar idle operations remains similar after they are transformed against two equivalence sequence of operations. To proceed the proof, we first show that the similarity is preserved by transformation against one operation (see Lemma 8.3.5), then prove it is also preserved by two equal logs (see Lemma 8.3.3) and equivalent logs (see Lemma 8.3.6).

Lemma 8.3.5 (Similarity Preservation by Transformation.).

Let $I(op_1, op_3)$ and $I(op_2, op_3)$ be two similar idle operations, then for every operation *op*, we have:

$$IT(I(op_1, op_2), op) \approx IT(I(op_1, op_3), op)$$

Proof. There are two cases according to the kind of the operation *op* (normal or idle)

1. op is a normal operation: then op integrates the effect of op_2 and op_3, this situation only occurs when $Hide(op_2, op_3)$ and $Hide(op_3, op_2)$ which means that $op_2 = op_3$ according to Lemma 8.1.2. Assume that $op = IT(op_i, op_2) = IT(op_i, op_3)$, op_i being an operation from $\mathcal{O}p$. According to whether op_i hides op_1 or not, we have to discuss the following two cases: If $Hide(op_i, op_1)$, it follows from **TR2** that $IT(I(op_1, op_2), op) = IT(I(op_1, op_3), op) = I(op_1, op)$. Otherwise, $IT(I(op_1, op_2), op) = IT(I(op_1, op_3), op) = I(IT(op_1 op_i), op_3)$.
   Consequently, $IT(I(op_1, op_2), op) = IT(I(op_1, op_3), op)$.

2. op is an idle operation: Let two operations op_i and op_j such that $op = I(op_i, op_j)$. Then, we have to show that
   $$IT(I(op_1, op_2), I(op_i, op_j)) = IT(I(op_1, op_3), I(op_i, op_j)$$
   Two cases are possible:
   (a) $op_2 = op_3$ and $op_j = op_2$: in this case, according to **TR1** if $Hide(op_j, op_1)$ then
   $$IT(I(op_1, op_2), I(op_j, op_2)) = IT(I(op_1, op_3), I(op_j, op_3)) = I(op_1, op_j).$$
   Otherwise, if $\neg Hide(op_j, op_1)$ then $IT(I(op_1, op_2), I(op_j, op_2)) = I(IT(op_1, op_j), op_2)$ and $IT(I(op_1, op_3), I(op_j, op_3)) = I(IT(op_1, op_j), op_3)$ Since $I(IT(op_1, op_j), op_2) \approx I(IT(op_1, op_j), op_2)$, the result is obtained.
   (b) $op_j \notin \{op_2, op_3\}$: we have $IT(I(op_1, op_2), I(op_i, op_j)) = I(op_2, op_i)$.Moreover, $IT(I(op_1, op_3), I(op_i, op_j)) = I(op_1, op_3) \approx I(op_a, op_i)$.

Accordingly, the similarity is preserved by transformation against normal and idle operations. □

Using the precedent Lemma we show that similarity relation is preserved by transformation against two equal logs.

Theorem 8.3.3 (Similarity Preservation by Transformation against two Equal Logs.)**.**

Given a log H and two similar idle operations op_1 and op_2. Then the similarity is preserved after integration against H.
$$IT^*(op_1, H) \approx IT^*(op_2, H)$$

Proof. By induction on H's size and the use of Lemmas 8.3.5. □

Since logs are rather equivalent in a collaborative application, we also have to prove that the similarity is preserved when transforming two similar idle operations against two equivalent sequences of operations.

Lemma 8.3.6 (**RTP2** for Hidden Operations.)**.**

Given two similar operations $I(op, op_i) \approx I(op, op_j)$ and two equivalent sequences of two operations seq_1 and seq_2. Then, we have:
$$IT^*(I(op, op_i), seq_1) \approx IT^*(I(op, op_j), seq_2)$$

Proof. To obtain two similar operations, it must be that $Hide(op_i, op_j)$ and $Hide(op_j, op_i)$ *i.e.* $op_i = op_j$ (see Lemma 8.1.2). We assume that $op'_i = IT(op_i, op_j)$ and $op'_j = IT(op_j, op_i)$. Consider two operations op_1 and op_2 integrated after op_i and op_j. Consider the following transformation forms of op_1 and op_2:

$$\begin{aligned} op'_2 &= IT^*(op_2, op_i \cdot op'_j) \\ op'_1 &= IT^*(op_1, op_i \cdot op'_j \cdot op'_2) \\ op''_1 &= IT^*(op_1, op_j \cdot op'_i) \\ op''_2 &= IT^*(op_2, op_j \cdot op'_i \cdot op''_1) \end{aligned}$$

Without loss f generalization, we assume that $op_i = op_j$ hides op_1. Consider the operation op hidden by both op_i and op_j. Then

$$IT^*(op, [op_i \cdot op'_j]) = I(op, op_j) \approx IT^*(op, [op_j \cdot op'_i]) = I(op, op_i).$$

Now, we have to prove that

$$IT^*(I(op, op_j), [op'_2 \cdot op'_1]) \equiv IT^*(I(op, op_i), [op''_1 \cdot op''_2]).$$

According to the Hide relation between the operations op_1 and op_2 in seq_1 and seq_2, we discuss the following cases:

1. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: The transformation result of op_1 after op_i and op_j is as follows:

$$op'_1 = I(op_1, op_2) \qquad (\mathbf{TR1} \text{ and } Hide(op_2, op_1))$$

As for op_2, it is transformed against $op_i \cdot op'_j$ as follows:

$$op'_2 I(op_2, op_j) \qquad (\mathbf{TR1} \text{ and } Hide(op_j, op_2))$$

In the same way, when op_1 is integrated after $op_j \cdot op'_i$, we obtain:

$$op''_1 = I(op_1, op_i) \qquad (\mathbf{TR1} \text{ and } op_i \mathcal{H} op_1)$$

As for op_2, it is integrated after the same sequence as follows:

$$op''_2 = I(op_2, op_1) \qquad (\mathbf{TR1} \text{ and } op_1 \mathcal{H} op_2)$$

Consequently,

$$\begin{aligned} IT^*(I(op, op_j), [op'_2 \cdot op'_1]) &= IT(IT(I(op, op_j), I(op_2, op_j)), I(op_1, op_2)) \\ &= IT(I(op, op_2), I(op_1, op_2)) && (\mathbf{TR1} \text{ and } Hide(op_2, op)) \\ &= I(op, op_1) && (\mathbf{TR1} \text{ and } Hide(op_1, op)) \end{aligned}$$

Similarly,

$$IT^*(I(op, op_i), [op''_1 \cdot op''_2]) = I(op, op_2)$$

Consequently,

$$IT^*(I(op, op_j), [op'_2 \cdot op'_1]) \approx IT^*(I(op, op_i), [op''_1 \cdot op''_2])$$

2. $Hide(op_2, op_1)$ and $\neg Hide(op_1, op_2)$: The transformation result of these operations after op_i and op_j is as follows:

$$op'_2 = I(op_2, op_j) \qquad (\mathbf{TR1} \text{ and } Hide(op_j, op_2))$$
$$op'_1 = I(op_1, op_2) \qquad (\mathbf{TR1} \text{ and } Hide(op_2, op_1))$$

In the same way,

$$op''_1 = I(op_1, op_i) \qquad (\mathbf{TR1} \text{ and } Hide(op_i, op_1))$$
$$op''_2 = I(IT(op_2, op_1), op_i) \qquad (\mathbf{TR1} \text{ and } Hide(op_1, op_2))$$

According to the relation between *op*, op_1 and op_2, we have to discuss the following cases:

(a) $\neg Hide(op_1, op)$ and $\neg Hide(op_2, op)$: then we have:

$$\begin{aligned} IT(I(op, op_j), op'_2) &= IT(I(op, op_j), I(op_2, op_j)) \\ &= I(IT(op, op_2), op_j) \qquad (\mathbf{TR1} \text{ and } \neg Hide(op_2, op)) \end{aligned}$$

Thus,

$$\begin{aligned} IT(I(op, op_j), [op'_2 \cdot op'_1]) &= IT(I(IT(op, op_2), op_j)), I(op_1, op_2)) \\ &= I(IT(op, op_2), op_j) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(I(op, op_i), op''_1) &= IT(I(op, op_i), I(op_1, op_i)) \\ &= I(IT(op, op_1), op_i) \qquad (\mathbf{TR1} \text{ and } \neg Hide(op_1, op)) \end{aligned}$$

Thus, according to $\mathbf{TR1}$, it follows from $\neg Hide(op_2, op)$ that $IT^*(I(op, op_i), [op''_1 \cdot op''_2]) = IT(I(IT(op, op_1), op_i), I(IT(op_2, op_1), op_i))$. Note that $Hide(op, op_2)$ since op_i hides both *op* and op_2 while op_2 does not hide *op*. Accordingly, $IT(IT(op, op_1), IT(op_2, op_1)) = IT(op_o p_2)$ (see Lemma 8.1.3). Consequently, we have:

$$IT^*(I(op, op_i), [op''_1 \cdot op''_2]) = I(IT(op, op_2), op_i)$$

Which leads to

$$IT^*(I(op, op_j), [op'_2 \cdot op'_1]) \approx IT^*(I(op, op_i), [op''_1 \cdot op''_2])$$

(b) $Hide(op_2, op)$ and $\neg Hide(op_1, op)$:

$$\begin{aligned} IT(I(op, op_j), op'_2) &= IT(I(op, op_j), I(op_2, op_j)) \\ &= I(op, op_2) \qquad (\mathbf{TR1} \text{ and } Hide(op_2, op)) \end{aligned}$$

Then,

$$\begin{aligned} IT^*(I(op, op_j), [op'_2 \cdot op'_1]) &= IT(I(op, op_2), I(op_1, op_2)) \\ &= I(IT(op, op_1), op_2) \qquad (\mathbf{TR1} \text{ and } \neg Hide(op_1, op)) \end{aligned}$$

Similarly,

$$\begin{aligned} IT^*(I(op, op_i), [op''_1 \cdot op''_2]) &= IT(I(op, op_i), I(op_1, op_i)) \\ &= I(IT(op, op_1), op_i) && (\mathbf{TR1} \text{ and } \neg Hide(op_1, op)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(I(op, op_i), [op''_1 \cdot op''_2]) &= IT(I(IT(op, op_1), op_i), I(IT(op_2, op_1), op_i) \\ &= I(IT(op, op_1), op_2) && (\mathbf{TR1} \text{ and } Hide(op_2, op)) \end{aligned}$$

Consequently,

$$IT^*(I(op, op_j), [op'_2 \cdot op'_1]) = IT^*(I(op, op_i), [op''_1 \cdot op''_2])$$

(c) $Hide(op_2, op)$ and $Hide(op_1, op)$: In this case,

$$\begin{aligned} IT(I(op, op_j), op'_2) &= IT(I(op, op_j), I(op_2, op_j)) \\ &= I(op, op_2) && (\mathbf{TR1} \text{ and } Hide(op_2, op)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(I(op, op_j), [op'_2 \cdot op'_1]) &= IT(I(op, op_2), I(op_1, op_2)) \\ &= I(op, op_1) && (\mathbf{TR1} \text{ and } Hide(op_1, op)) \end{aligned}$$

Similarly,

$$\begin{aligned} IT^*(I(op, op_i), op''_1) &= IT(I(op, op_i), I(op_1, op_i)) \\ &= I(op, op_1) && (\mathbf{TR1} \text{ and } Hide(op_1, op)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(I(op, op_i), [op''_1 \cdot op''_2]) &= IT(I(op, op_1), I(IT(op_2, op_1), op_i)) \\ &= I(op, op_1) \end{aligned}$$

Consequently,

$$IT^*(I(op, op_j), [op'_2 \cdot op'_1]) = IT^*(I(op, op_i), [op''_1 \cdot op''_2])$$

We deduce that the similarity is preserved by transformation against two equivalent sequences of two operations.

□

Based on the previous result, we show in Theorem 8.3.4 that the similarity is preserved by transformation against two equivalent logs.

Theorem 8.3.4 (Similarity Preservation by Transformation against two Equivalent Logs.)**.**

Given two equivalent logs $H_1 \equiv H_2$ and two similar idle operations op_1 and op_2. The similarity between op_1 and op_2 is preserved after integration against H_1 and H_2.

$$IT^*(op_1, H_1) \approx IT^*(op_2, H_2)$$

Proof. By induction on H's size and the use of Lemma 8.3.6. □

A.3.2.3 RTP2 Preservation for Inverse Operations

In this section, we illustrate the preservation of the **RTP2** property by inverse operations. We also demonstrate that the resulting logs are equivalent after undoing a given operation.

In the sequel, we use the notation $IT(seq, op)$, where seq is a sequence of operations and op is an operation recursively defined as follows:

- $IT(\emptyset, op) = \emptyset$;
- $IT(op_1 \cdot op_2, op) = IT(op_1, op) \cdot IT(op_2, IT(op, op_1))$
- $IT(op_1 \cdot op_2 \cdot op_3 \ldots op_n, op) = IT(op_1 \cdot op_2, op) \cdot IT(op_3 \ldots op_n, op)$.

Lemma 8.3.7 (Transformation of an Inverse Operation against two Equivalent Logs)**.**

Let $\overline{op}$ be an inverse operation, op_1 and op_2 two concurrent operations. Let $op'_1 = IT(op_1, op)$, $op'_2 = IT(op_2, op)$, $op''_1 = IT(op'_1, op'_2)$ and $op''_2 = IT(op'_2, op'_1)$. Then, we have:

$$IT^*(\overline{op}, op'_1 \cdot op''_2) \approx IT^*(\overline{op}, op'_2 \cdot op''_1)$$
$$IT(op'_1 \cdot op''_2, \overline{op}) \equiv IT(op'_2 \cdot op''_1, \overline{op})$$

Proof. We discuss the following three cases according to the hide relation between op_1 and op_2:o

First case. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: We discuss the following two sub cases:

1. $\neg Hide(op, op_1)$ and so $\neg Hide(op, op_2)$:

$$\begin{aligned} op'_1 &= IT(op_1, op) \\ op''_2 &= I(IT(op_2, op), op_1) && \text{(Lemma 8.1.1)} \\ op'_2 &= IT(op_2, op) \\ op''_1 &= I(IT(op_1, op), op_2) && \text{(Lemma 8.1.1)} \end{aligned}$$

According to the relation between op'_1, op'_2 and $\overline{op}$, we distinguish the following cases:

(a) $\neg Hide(op_1, op)$ then $\neg Hide(op_2, op)$: it follows from **TR3** that:

$$IT(\overline{op}, op'_1) = \overline{IT(op, op_1)}$$

Thus

$$\begin{aligned} IT(\overline{op}, op'_1 \cdot op''_2) &= IT(\overline{IT(op, op_1)}, I(IT(op_2, op), op_1)) \\ &= \overline{IT(op, op_1)} && (\mathbf{TR5}) \end{aligned}$$

Similarly,

$$IT(\overline{op}, op'_2 \cdot op''_1) = \overline{IT(op, op_2)} \qquad (\mathbf{TR5})$$

It follows from $\underline{op_1 = op_2}$ (see Lemma 8.1.2) that $IT(op, op_1) = IT(op, op_2)$. Thereby, $\overline{IT(op, op_1)} = \overline{IT(op, op_2)}$.

(b) $Hide(op_1, op)$ then $Hide(op_2, op)$: in this case,

$$\begin{aligned} IT(\overline{op}, op_1') &= \overline{I(op, op_1)} \\ IT(\overline{op}, op_1' \cdot op_2'') &= IT(\overline{I(op, op_1)}, I(IT(op_2, op), op_1) \\ &= \overline{I(op, op_2)} && (\mathbf{TR1I} \text{ and } Hide(op_2, op))) \end{aligned}$$

In the same way we have,

$$\begin{aligned} IT(\overline{op}, op_2') &= \overline{I(op, op_2)} \\ IT(\overline{op}, op_2' \cdot op_1'') &= IT(\overline{I(op, op_2)}, I(IT(op_1, op), op_2) \\ &= \overline{I(op, op_1)} && (\mathbf{TR1I} \text{ and } Hide(op_1, op)) \end{aligned}$$

Then the result is obtained.

2. $op\mathcal{H}op_1$ and so $op\mathcal{H}op_2$: In this case

$$\begin{aligned} op_1' &= I(op_1, op) \\ op_2'' &= I(op_2, op_1) && (\mathbf{TR1} \text{ and } Hide(op_1, op_2)) \\ op_2' &= I(op_2, op) \\ op_1'' &= I(op_1, op_2) && (\mathbf{TR1} \text{ and } Hide(op_2, op_1) \end{aligned}$$

Two cases are possible according to the hide relation between op_1 and op.

(a) $Hide(op_1, op)$: in this case we also have $Hide(op_2, op)$, which leads to the following sequences:

$$\begin{aligned} IT(\overline{op}, op_1') &= \overline{I(op, op_1)} \\ IT(\overline{I(op, op_1)}, op_2'') &= IT(\overline{I(op, op_1)}, I(op_2, op_1)) \\ &= \overline{I(op, op_2)} && (\mathbf{TR1I} \text{ and } Hide(op_2, op)) \end{aligned}$$

Similarly

$$\begin{aligned} IT(\overline{op}, op_2') &= \overline{I(op, op_2)} \\ IT(\overline{I(op, op_2)}, op_1'') &= IT(\overline{I(op, op_2)}, I(op_1, op_2)) \\ &= \overline{I(op, op_1)} && (\mathbf{TR1I} \text{ and } Hide(op_1, op)) \end{aligned}$$

(b) $\neg Hide(op_1, op)$: in this case we also have $\neg Hide(op_2, op)$, hence

$$\begin{aligned} IT(\overline{op}, op_1') &= \overline{IT(op, op_1)} \\ IT(\overline{IT(op, op_1)}, op_2'') &= IT(\overline{IT(op, op_1)}, I(op_2, op_1)) \\ &= \overline{IT(op, op_1)} && (\mathbf{TR5}) \end{aligned}$$

Similarly

$$\begin{aligned} IT(\overline{op}, op_2') &= \overline{IT(op, op_2)} \\ IT(\overline{IT(op, op_2)}, op_1'') &= IT(\overline{IT(op, op_2)}, I(op_1, op_2)) \\ &= \overline{IT(op, op_2)} && (\mathbf{TR5}) \end{aligned}$$

It follows from $op_1 = op_2$ that $IT(op, op_1) = IT(op, op_2)$. Consequently, we obtain $\overline{IT(op, op_1)} = \overline{IT(op, op_2)}$

Next, we integrate the effect of $\overline{op}$ into the equivalent sequences of operations.

$$\begin{aligned}
IT(op'_1, \overline{op}) &= IT(IT(op_1, op), \overline{op}) \\
&= op_1 && (\mathbf{TR4}) \\
IT(op''_2, IT(\overline{op}, op'_1)) &= IT(I(IT(op_2, op), op_1), \overline{IT(op, op_1)}) \\
&= I(IT(IT(op_2, op), \overline{op}), op_1) && (\mathbf{TR4}) \\
&= I(op_2, op_1) && (\mathbf{TR4})
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op'_2, \overline{op}) &= IT(IT(op_2, op), \overline{op}) \\
&= op_2 && (\mathbf{TR4}) \\
IT(op''_1, IT(\overline{op}, op'_2)) &= IT(I(IT(op_1, op), op_2), \overline{IT(op, op_2)}) \\
&= I(op_1, op_2) && (\mathbf{TR4})
\end{aligned}$$

Since $op_2 = op_1$ we have $op_1 \cdot I(op_2, op_1) \equiv op_2 \cdot I(op_1, op_2)$.
Thus, we deduce that $IT(op'_1 \cdot op''_2, \overline{op}) \equiv IT(op'_2 \cdot op''_1, \overline{op})$.

Second case. $\neg Hide(op_1, op_2)$ and also $\neg Hide(op_2, op_1)$, then the following cases are faced:

1. $Hide(op, op_1)$ and so $\neg Hide(op, op_2)$: we calculate the integrated forms of op_1 and op_2

$$\begin{aligned}
op'_1 &= I(op_1, op) \\
op''_2 &= IT(IT(op_2, op), I(op_1, op)) \\
&= IT(op_2, op) \\
op'_2 &= IT(op_2, op) \\
op''_1 &= IT(I(op_1, op), IT(op_2, op)) \\
&= I(IT(op_1, op_2), op) && (\mathbf{TR2} \text{ and } \neg Hide(op_2, op_1))
\end{aligned}$$

According to whether op_1 hides op or not we discuss the two following cases:

(a) $Hide(op_1, op)$:

$$\begin{aligned}
IT(\overline{op}, op'_1) &= IT(\overline{op}, I(op_1, op)) \\
&= \overline{IT(op, op_1)} && (\mathbf{TR3}) \\
&= \overline{I(op, op_1)}
\end{aligned}$$

Then,

$$\begin{aligned}
IT^*(\overline{op}, op'_1 \cdot op''_2) &= IT(\overline{I(op, op_1)}, IT(op_2, op)) \\
&= \overline{I(IT(op, op_2), op_1)} && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op))
\end{aligned}$$

Moreover,

$$\begin{aligned}
IT(\overline{op}, op'_2) &= IT(\overline{op}, IT(op_2, op)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR3})
\end{aligned}$$

which leads to

$$\begin{aligned} IT^*(\overline{op}, op'_2 \cdot op''_1) &= IT(\overline{IT(op, op_2)}, I(IT(op_1, op_2), op)) \\ &= \overline{IT(IT(op, op_2), IT(op_1, op_2))} && (\mathbf{TR3}) \\ &= \overline{I(IT(op, op_1), op_1)} && \text{(Lemma 8.1.1 and } Hide(op_1, op)) \end{aligned}$$

Thus, we have $IT^*(\overline{op}, op'_1 \cdot op''_2) = IT^*(\overline{op}, op'_2 \cdot op''_1)$

(b) $\neg Hide(op_1, op)$: In this case,

$$\begin{aligned} IT^*(\overline{op}, op'_1 \cdot op'_2) &= IT^*(\overline{op}, [I(op_1, op); IT(op_2, op)]) \\ &= IT(IT(\overline{op}, I(op_1, op)), IT(op_2, op)) \\ &= IT(\overline{IT(op, op_1)}, IT(op_2, op)) && (\mathbf{TR3}) \end{aligned}$$

According to $\mathbf{TP2}$ assumed for the initial set of operations (see Definition 6.3.8), we have

$$\begin{aligned} IT^*(op_2, [op_1; IT(op, op_1)]) &= IT^*(op_2, [op; IT(op_1, op)]) \\ IT[IT(op_2, op_1), IT(op, op_1)] &= IT[IT(op_2, op), I(op_1, op)] \\ IT[IT(op_2, op_1), IT(op, op_1)] &= IT(op_2, op) && \text{(Lemma 6.3.9)} \end{aligned}$$

Replacing $IT(op_2, op)$ by $IT[IT(op_2, op_1), IT(op, op_1)]$ in the precedent equality leads to:

$$\begin{aligned} IT^*(\overline{op}, op'_1 \cdot op'_2) &= IT(\overline{IT(op, op_1)}, IT(op_2, op)) \\ &= IT(\overline{IT(op, op_1)}, IT[IT(op_2, op_1), IT(op, op_1)]) \\ &= IT(IT(op, op_1), IT(op_2, op_1)) && (\mathbf{TR3}) \end{aligned}$$

Moreover,

$$\begin{aligned} IT^*(\overline{op}, [op''_2; op''_1]) &= IT^*(\overline{op}, [I(op_2, op); I(IT(op_1, op_2), op)]) \\ &= IT(IT(\overline{op}, I(op_2, op)), I(IT(op_1, op_2), op)) \\ &= IT(\overline{IT(op, op_2)}, I(IT(op_1, op_2), op)) && (\mathbf{TR3}) \\ &= \overline{IT(IT(op, op_2), IT(op_1, op_2))} && (\mathbf{TR3}) \end{aligned}$$

According $\mathbf{TP2}$ satisfied for normal operations, we have

$$IT(op, op_1), IT(op_2, op_1) = IT(IT(op, op_2), IT(op_1, op_2))).$$

Thereby, the result is obtained.

2. $\neg Hide(op, op_1)$ and $\neg Hide(op, op_2)$: In this case,

$$\begin{aligned} op'_1 &= IT(op_1, op) \\ op'_2 &= IT(IT(op_2, op), IT(op_1, op)) \\ op''_2 &= IT(op_2, op) \\ op''_1 &= IT(IT(op_1, op), IT(op_2, op)) \end{aligned}$$

Consequently, since all operations are normal, and $\mathbf{TP2}$ is satisfied by the initial set of normal operations (see Definition 6.3.8), we have:

$$IT^*(\overline{op}, op'_1 \cdot op'_2) = IT^*(\overline{op}, op''_2 \cdot op'_1)$$

Now, we integrate the effect of $\overline{op}$ into the equivalent sequences of operations.

$$\begin{aligned}
IT(op'_1, \overline{op}) &= IT(IT(op_1, op), \overline{op}) \\
&= op_1 && (\mathbf{TR4}) \\
IT(op''_2, IT(\overline{op}, op'_1)) &= IT(IT(op_2, op), \overline{IT(op, op_1)}) \\
&= IT(op_2, op_1) && \mathbf{TR4} \text{ and } Hide(op, op_1) \text{ and } \neg Hide(op_1, op_2))
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op'_2, \overline{op}) &= IT(IT(op_2, op), \overline{op}) \\
&= op_2 && (\mathbf{TR4}) \\
IT(op''_1, IT(\overline{op}, op'_2)) &= IT(I(IT(op_1, op_2), op), \overline{IT(op, op_2)}) \\
&= IT(op_1, op_2) && (\mathbf{TR4})
\end{aligned}$$

Since $op_2 \cdot IT(op_1, op_2) \equiv op_1 \cdot IT(op_2, op_1)$, we deduce that

$$IT(op'_1 \cdot op''_2, \overline{op}) \equiv IT(op'_2 \cdot op''_1, \overline{op})$$

Thereby, we have shown that indeed undoing two similar inverse operations produces equivalent sequences.

Third case. $Hide(op_2, op_1)$ and $\neg Hide(op_1, op_2)$: according to the relation between op and the two operations op_1 and op_2, the following cases may occur:

1. $op\mathcal{H}op_2$:so by transitivity, we have $op\mathcal{H}op_1$ (see Lemma 6.3.10). Consequently, the transformed forms of op_1 and op_2 are:

$$\begin{aligned}
op'_1 &= I(op_1, op) \\
op''_2 &= IT(I(op_2, op), op'_1) \\
&= IT(I(op_2, op), I(op_1, op)) \\
&= I(IT(op_2, op_1), op) && (\mathbf{TR1} \text{ and } \neg Hide(op_1, op_2)) \\
op'_2 &= I(op_2, op) \\
op''_1 &= IT(I(op_1, op), op'_2) \\
&= I(I(op_1, op), I(op_2, op)) \\
&= I(op_1, op_2) && (\mathbf{TR1} \text{ and } Hide(op_2, op_1))
\end{aligned}$$

Suppose that $Hide(op_1, op)$, it follows from the transitivity of the Hide relation and from $Hide(op, op_2)$ that $Hide(op_1, op_2)$ is true which is false since we supposed that $\neg Hide(op_1, op_2)$. Consequently, $\neg Hide(op_1, op)$ holds. According to whether op_2 hides op or not we discuss the following two sub cases:

(a) $\neg Hide(op_2, op)$:

$$\begin{aligned}
IT(\overline{op}, op_1') &= IT(\overline{op}, I(op_1, op)) \\
&= \overline{IT(op, op_1)} && (\mathbf{TR3}) \\
IT(\overline{op}, op_1' \cdot op_2'') &= IT(\overline{IT(op, op_1)}, op_2'') \\
&= IT(\overline{IT(op, op_1)}, I(IT(op_2, op_1), op)) \\
&= \overline{IT(IT(op, op_1), IT(op_2, op_1))} && (\mathbf{TR3}) \\
&= \overline{IT(IT(op, op_2), IT(op_1, op_2))} && \text{(Lemma 8.3.4)} \\
&= \overline{IT(op, op_2)} && \text{(Lemma 6.3.8)}
\end{aligned}$$

On the other hand, we have:

$$\begin{aligned}
IT(\overline{op}, op_2') &= IT(\overline{op}, I(op_2, op)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR3}) \\
IT(\overline{op}, op_2' \cdot op_1'') &= IT(\overline{IT(op, op_1)}, op_1'') \\
&= IT(\overline{IT(op, op_2)}, I(op_1, op_2)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR5})
\end{aligned}$$

Then, the result is obtained.

(b) $Hide(op_2, op)$:

$$\begin{aligned}
IT(\overline{op}, op_1') &= IT(\overline{op}, I(op_1, op)) \\
&= \overline{IT(op, op_1)} && (\mathbf{TR3}) \\
IT^*(\overline{op}, op_1' \cdot op_2'') &= IT(\overline{IT(op, op_1)}, op_2'') \\
&= IT(\overline{IT(op, op_1)}, I(IT(op_2, op_1), op)) \\
&= \overline{IT(IT(op, op_1), IT(op_2, op_1))} && (\mathbf{TR3}) \\
&= \overline{I(IT(op, op_1), op_2)} && \text{(Lemma 8.1.1)}
\end{aligned}$$

In the same way,

$$\begin{aligned}
IT(\overline{op}, op_2') &= IT(\overline{op}, I(op_2, op)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR3}) \\
&= \overline{I(op, op_2)} \\
IT^*(\overline{op}, op_2' \cdot op_1'') &= IT(\overline{I(op, op_2)}, I(op_1, op_2)) \\
&= \overline{I(IT(op, op_1), op_2)} && (\mathbf{TR1I} \text{ and } \neg Hide(op_1, op))
\end{aligned}$$

Thereby proving that $IT^*(\overline{op}, op_2' \cdot op_1'') = IT^*(\overline{op}, op_1' \cdot op_2'')$ *i.e.* $\mathbf{RTP2}$ is satisfied.
In the following, we integrate the effect of $\overline{op}$ into the equivalent sequences of operations $op_2' \cdot op_1'' \equiv op_1' \cdot op_2''$.

$$\begin{aligned}
IT(op_1', \overline{op}) &= IT(IT(op_1, op), \overline{op}) \\
&= op_1 && (\mathbf{TR4}) \\
IT(op_2'', IT(\overline{op}, op_1')) &= IT(I(IT(op_2, op_1), op), \overline{IT(op, op_1)}) \\
&= IT(op_2, op_1) && (\mathbf{TR4})
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op_2', \overline{op}) &= IT(I(op_2, op), \overline{op}) \\
&= op_2 && (\mathbf{TR4}) \\
IT(op_1'', IT(\overline{op}, op_2')) &= IT(I(op_1, op_2), \overline{IT(op, op_2)}) \\
&= I(op_1, op_2) && (\mathbf{TR2} \text{ and } Hide(op, op_1))
\end{aligned}$$

Since $op_2 \cdot I(op_1, op_2) \equiv op_1 \cdot IT(op_2, op_1)$, we deduce that

$$IT(op_1' \cdot op_2'', \overline{op}) \equiv IT(op_2' \cdot op_1'', \overline{op})$$

2. $\neg Hide(op, op_2)$ and $Hide(op, op_1)$: the integrated forms of op_1 and op_2 in this case are:

$$\begin{aligned}
op_1' &= I(op_1, op) \\
op_2'' &= IT(IT(op_2, op), I(op_1, op)) \\
&= IT(op_2, op) && (\mathbf{TR5}) \\
op_2' &= IT(op_2, op) \\
op_1'' &= IT(I(op_1, op), op_2') \\
&= I(op_1, op_2) && (\mathbf{TR2} \text{ and } Hide(op_2, op_1))
\end{aligned}$$

Since $Hide(op_2, op_1)$, we deduce that $Hide(op_2, op)$. According to whether op_1 hides op or not, we distinguish the following two case:

(a) $Hide(op_1, op)$: the integration of $\overline{op}$ after $op_2' \cdot op_1''$ and $op_1' \cdot op_2''$ leads to:

$$\begin{aligned}
IT(\overline{op}, op_1') &= IT(\overline{op}, I(op_1, op)) \\
&= \overline{I(op, op_1)} && (\mathbf{TR3} \text{ and } Hide(op_1, op)) \\
IT^*(\overline{op}, op_1' \cdot op_2'') &= IT(\overline{I(op, op_1)}, IT(op_2, op)) \\
&= \overline{I(op, op_2)} && (\mathbf{TR2} \text{ and } Hide(op_2, op))
\end{aligned}$$

Similarly, we have:

$$\begin{aligned}
IT(\overline{op}, op_2') &= IT(\overline{op}, IT(op_2, op)) \\
&= \overline{I(op, op_2)} && (\mathbf{TR3} \text{ and } Hide(op_2, op)) \\
IT^*(\overline{op}, op_2' \cdot op_1'') &= IT(\overline{I(op, op_2)}, I(op_1, op_2)) \\
&= \overline{I(op, op_1)} && (\mathbf{TR1I} \text{ and } Hide(op_1, op))
\end{aligned}$$

Since $\overline{I(op, op_2)} \approx \overline{I(op, op_1)}$, we have

$$IT^*(\overline{op}, op_1' \cdot op_2'') \approx IT^*(\overline{op}, op_2' \cdot op_1'')$$

It has been shown that indeed, $\mathbf{RTP2}$ holds.

(b) $\neg Hide(op_1, op)$: transforming $\overline{op}$ against $op'_1 \cdot op''_2$ leads to:

$$\begin{aligned}
IT(\overline{op}, op'_1) &= IT(\overline{op}, I(op_1, op)) \\
&= \overline{IT(op, op_1)} && (\mathbf{TR3}) \\
IT^*(\overline{op}, op'_1 \cdot op''_2) &= IT(\overline{IT(op, op_1)}, IT(op_2, op)) \\
&= IT(\overline{IT(op, op_1)}, IT(IT(op_2, op), IT(op_1, op))) \\
&= IT(\overline{IT(op, op_1)}, IT(IT(op_2, op_1), IT(op, op_1))) && \text{(Lemma 8.3.4)} \\
&= \overline{IT(IT(op, op_1), IT(op_2, op_1))} && (\mathbf{TR3}) \\
&= \overline{I(IT(op, op_1), op_2)} && \text{(Lemma 8.1.1)}
\end{aligned}$$

As for transforming $\overline{op}$ against $op'_2 \cdot op''_1$, it leads to:

$$\begin{aligned}
IT(\overline{op}, op'_2) &= IT(\overline{op}, IT(op_2, op)) \\
&= \overline{I(op, op_2)} && (\mathbf{TR3}) \\
IT^*(\overline{op}, op'_2 \cdot op''_1) &= IT(\overline{I(op, op_2)}, I(op_1, op_2)) \\
&= \overline{I(IT(op, op_1), op_2)} && (\mathbf{TR1} \text{ and } \neg Hide(op_1, op))
\end{aligned}$$

Thus, the result is obtained.

Accordingly, we have proven that

$$IT^*(\overline{op}, op'_1 \cdot op''_2) \approx IT^*(\overline{op}, op'_2 \cdot op''_1)$$

In the following, we integrate the effect of $\overline{op}$ into the equivalent sequences of operations and prove resulting logs are equivalent. On one hand, we have:

$$\begin{aligned}
IT(op'_1, \overline{op}) &= IT(IT(op_1, op), \overline{op}) \\
&= op_1 && (\mathbf{TR4}) \\
IT(op''_2, IT(\overline{op}, op'_1)) &= IT(IT(op_2, op), \overline{IT(op, op_1)}) \\
&= IT(op_2, op_1) && \mathbf{TR4} \text{ and } Hide(op, op_1) \text{ and } \neg Hide(op_1, op_2))
\end{aligned}$$

On the other hand, we have:

$$\begin{aligned}
IT(op'_2, \overline{op}) &= IT(IT(op_2, op), \overline{op}) \\
&= op_2 && (\mathbf{TR4}) \\
IT(op''_1, IT(\overline{op}, op'_2)) &= IT(I(op_1, op_2), \overline{I(op, op_2)}) \\
&= I(op_1, op_2) && (\mathbf{TR4} \text{ and } Hide(op, op_1))
\end{aligned}$$

Since $op_2 \cdot I(op_1, op_2) \equiv op_1 \cdot IT(op_2, op_1)$, we deduce that

$$IT(op'_1 \cdot op''_2, \overline{op}) \equiv IT(op'_2 \cdot op''_1, \overline{op})$$

3. $\neg Hide(op, op_2)$ and $\neg Hide(op, op_1)$: in this case,

$$\begin{aligned} op'_1 &= IT(op_1, op) \\ op''_2 &= IT^*(op_2, [op; op'_1]) \\ op'_2 &= IT(op_2, op) \\ op''_1 &= IT(IT(op_1, op), op'_2) \\ &= I(IT(op_1, op), op_2) && \text{(Lemma 8.1.3)} \end{aligned}$$

The following two cases are faced:

(a) $Hide(op_1, op)$: so by transitivity $Hide(op_2, op)$, and $\overline{op}$ is transformed against $op'_1 \cdot op''_2$ as follows: Here we have,

$$\begin{aligned} IT(\overline{op}, op'_1) &= IT(\overline{op}, IT(op_1, op)) \\ &= \overline{I(op, op_1)} && (\mathbf{TR3}) \end{aligned}$$

Consequently,

$$\begin{aligned} IT^*(\overline{op}, op'_1 \cdot op''_2) &= IT(\overline{I(op, op_1)}, IT^*(op_2, op \cdot op'_1)) \\ &= IT(\overline{I(op, op_1)}, IT^*(op_2, op_1 \cdot IT(op, op_1))) && \text{(Lemma 8.3.4)} \\ &= IT(\overline{I(op, op_1)}, IT(op_2, op_1)) && \text{(Lemma 6.3.8 and } Hide(op_1, op)\text{)} \\ &= \overline{I(op, op_1)} && (\mathbf{TR2I} \text{ and } Hide(op_2, op)) \end{aligned}$$

Furthermore, $\overline{op}$ is transformed against $op'_2 \cdot op''_1$ as follows:

$$\begin{aligned} IT(\overline{op}, op'_2) &= IT(\overline{op}, IT(op_2, op)) \\ &= \overline{I(op, op_2)} && (\mathbf{TR3}) \\ IT^*(\overline{op}, op'_2 \cdot op''_1) &= IT(\overline{I(op, op_2)}, op''_1) \\ &= IT(\overline{I(op, op_2)}, I(IT(op_1, op), op_2)) \\ &= \overline{I(op, op_1)} && (\mathbf{TR1I} \text{ and } Hide(op_1, op)) \end{aligned}$$

Consequently,

$$IT^*(\overline{op}, op'_1 \cdot op''_2) = IT^*(\overline{op}, op'_2 \cdot op''_1)$$

(b) $\neg Hide(op_1, op)$ and $\neg Hide(op_2, op)$: on the one hand $\overline{op}$ is transformed as follows:

$$\begin{aligned} IT(\overline{op}, op'_1) &= IT(\overline{op}, IT(op_1, op)) \\ &= \overline{IT(op, op_1)} && (\mathbf{TR3}) \\ IT^*(\overline{op}, op'_1 \cdot op''_2) &= IT(\overline{IT(op, op_1)}, op''_2) \\ &= IT(\overline{IT(op, op_1)}, IT(IT(op_2, op), IT(op_1, op))) \\ &= \overline{IT(IT(op, op_1), IT(IT(op_2, op_1), IT(op, op_1)))} && \text{(Lemma 8.3.4)} \\ &= \overline{IT(IT(op, op_1), IT(op_2, op_1)))} && (\mathbf{TR3}) \\ &= \overline{IT(IT(op, op_2), IT(op_1, op_2)))} && \text{(Lemma 8.3.4)} \\ &= \overline{IT(op, op_2)} && \text{(Lemma 6.3.8)} \end{aligned}$$

On the other hand $\overline{op}$ is transformed as follows:

$$\begin{aligned}
IT(\overline{op}, op'_2) &= IT(\overline{op}, IT(op_2, op)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR3}) \\
IT^*(\overline{op}, op'_2 \cdot op''_1) &= IT(\overline{IT(op, op_2)}, op''_1) \\
&= IT(\overline{IT(op, op_2)}, I(IT(op_1, op), op_2)) \\
&= \overline{IT(op, op_2)} && (\mathbf{TR5})
\end{aligned}$$

Consequently, property $\mathbf{RTP2}$ is satisfied *i.e.*

$$IT^*(\overline{op}, op'_1 \cdot op''_2) = IT^*(\overline{op}, op'_2 \cdot op''_1)$$

Now, we integrate the effect of $\overline{op}$ into the equivalent sequences of operations $op'_1 \cdot op''_2$ and $op'_2 \cdot op''_1$ as follows:

$$\begin{aligned}
IT(op'_1, \overline{op}) &= IT(IT(op_1, op), \overline{op}) \\
&= op_1 && (\mathbf{TR4}) \\
IT(op''_2, IT(\overline{op}, op'_1)) &= IT(IT(IT(op_2, op), IT(op_1, op)), \overline{IT(op, op_1)}) \\
&= IT(IT(IT(op_2, op_1), IT(op, op_1)), \overline{IT(op, op_1)}) && \text{(Lemma 8.3.4)} \\
&= IT(op_2, op_1)
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op'_2, \overline{op}) &= IT(IT(op_2, op), \overline{op}) \\
&= op_2 && (\mathbf{TR4}) \\
IT(op''_1, IT(\overline{op}, op'_2)) &= IT(I(IT(op_1, op), op_2), \overline{IT(op, op_2)}) \\
&= I(IT(IT(op_1, op), \overline{op}), op_2) && (\mathbf{TR4}) \\
&= I(op_1, op_2)
\end{aligned}$$

Since $op_2 \cdot I(op_1, op_2) \equiv op_1 \cdot IT(op_2, op_1)$, we deduce that

$$IT(op'_1 \cdot op''_2, \overline{op}) \equiv IT(op'_2 \cdot op''_1, \overline{op})$$

$\square$

In the following, we show that the the similarity is preserved for inverse operations.

A.3.2.4 TP2 for Hidden Inverse Operations

To prove that $\mathbf{RTP2}$ is satisfied by the integrated forms of an idle operation during the undo procedure, we show that the similarity between two hidden inverse operations is preserved after they are transformed against two equivalent sequences of operations. The following lemma shows that undoing a hidden inverse operation followed by a sequence of size 2 satisfies $\mathbf{RTP2}$ and produces two equivalent sequences of operations.

Lemma 8.3.8 (Undoing Hidden Inverse Operations)**.**

Given two similar inverse operations $\overline{I(op_1, op_i)} \equiv \overline{I(op_1, op_j)}$ and two equivalent sequences seq_1 and seq_2 where $|seq_1| = |seq_2| = 2$. Then, we have:

$$IT^*(\overline{I(op_1, op_i)}, seq_1) \approx IT^*(\overline{I(op_1, op_j)}, seq_2)$$

$$IT(seq_1, \overline{I(op_1, op_i)}) \equiv IT(seq_2, \overline{I(op_1, op_j)})$$

Proof. Let $op'_i = IT(op_i, op_j)$ and $op'_j = IT(op_j, op_i)$. To have two similar operations, it must be that $Hide(op_i, op_j)$, $Hide(op_j, op_i)$ and $Hide(op_i, op_1)$ in which case, we have

$$\begin{aligned}
op'_i &= I(op_i, op_j) \\
op'_j &= I(op_j, op_i) \\
op'_1 &= IT^*(op_1, [op_i \cdot op'_j]) \\
&= I(op_1, op_j) && (\mathbf{TR1}) \\
op''_1 &= IT^*(op_1, [op_j \cdot op'_i]) \\
&= I(op_1, op_i) && (\mathbf{TR1})
\end{aligned}$$

Consider two operations op_2 and op_3 such that:

$$\begin{aligned}
op'_2 &= IT^*(op_2, [op_i \cdot op'_j \cdot op'_1]) \\
op'_3 &= IT^*(op_3, [op_i \cdot op'_j \cdot op'_2]) \\
op''_3 &= IT^*(op_2, [op_j \cdot op'_i \cdot op''_1]) \\
op''_2 &= IT^*(op_3, [op_j \cdot op'_i \cdot op''_2])
\end{aligned}$$

Let the sequences $seq_1 = [op'_2 \cdot op'_3]$ and $seq_2 = [op''_3, op''_2]$ against which $\overline{op'_1}$ and $\overline{op''_1}$ will be transformed. According to the hide relation between different operations, we discuss the following cases:

First case. $Hide(op_2, op_3)$ and $Hide(op_3, op_2)$: according to the hide relation between operations op_1, op_2 and op_3, we discuss the following cases:

1. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: this means $op_1 = op_2 = op_3$ (see Lemma 8.1.2). Thus, the transitivity of the Hide relation implies that both op_i and op_j hide op_2 and op_3. According to $\mathbf{TR1}$, the transformation results of op_1, op_2 and op_3 after op_i and op_j are:

$$\begin{aligned}
op'_2 &= I(op_2, op_1) \\
op'_3 &= I(op_3, op_2) \\
op''_3 &= I(op_3, op_1) \\
op''_2 &= I(op_2, op_3)
\end{aligned}$$

Now, let us prove that the similarity relation between $\overline{I(op_1, op_i)}$ and $\overline{I(op_1, op_j)}$ is preserved by transformation *i.e*

$$IT^*(\overline{I(op_1, op_j)}, seq_1) \approx IT^*(\overline{I(op_1, op_i)}, seq_2).$$

First, we have:

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(IT(\overline{I(op_1, op_j)}, I(op_2, op_1)), I(op_3, op_2)) \\ &= IT(\overline{I(op_1, op_j)}, I(op_3, op_2)) && (\textbf{TR2I} \text{ and } Hide(op_2, op_1)) \\ &= \overline{I(op_1, op_j)} \end{aligned}$$

Similarly,

$$IT^*(\overline{I(op_1, op_i)}, seq_2) = \overline{I(op_1, op_i)}$$

Then the result is obtained.
Now, let us integrate the effect of $\overline{I(op_1, op_j)}$ and that of $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$respectively and show resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(op_2, op_1), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\textbf{TR4}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(op_3, op_2), \overline{I(op_1, op_j)}) \\ &= I(op_3, op_2) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\textbf{TR4}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(op_2, op_3), \overline{I(op_1, op_i)}) \\ &= I(op_2, op_3) \end{aligned}$$

Since $I(op_2, op_j) \cdot I(op_3, op_2) \equiv I(op_3, op_i) \cdot I(op_2, op_3) \equiv \emptyset$, we deduce that

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Consequently, resulting logs are equivalent after undoing the hidden inverse operations.

2. $\neg Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: in this case, it follows from $op_2 = op_3$ that $\neg Hide(op_1, op_3)$ and $\neg Hide(op_3, op_1)$. Moreover, $\neg Hide(op_2, op_i)$, otherwise by transitivity, $Hide(op_2, op_1)$. Finally, $\neg Hide(op_i, op_2)$ since neither op_2 hides op_1, nor op_1 hides op_2. Accordingly, we have:

$$\begin{aligned} op'_2 &= IT(op_2, op_i) && (\textbf{TR5}) \\ op'_3 &= I(IT(op_3, op_i), op_2) && (\textbf{TR5} \text{ and Lemma 6.3.10}) \end{aligned}$$

In the same way,

$$\begin{aligned} op''_3 &= IT(op_3, op_j) && (\textbf{TR5}) \\ op''_2 &= I(IT(op_2, op_j), op_3) && (\textbf{TR5} \text{ and Lemma 6.3.10}) \end{aligned}$$

Then,

$$\begin{aligned}
IT(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, IT(op_2, op_i)) \\
&= \overline{I(op_1, op_j)} && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op_1)) \\
IT^*(I(\overline{op_1}, op_j), seq_1) &= IT(\overline{I(op_1, op_j)}, I(IT(op_3, op_i), op_2)) \\
&= \overline{I(op_1, op_j)}
\end{aligned}$$

Similarly,

$$IT^*(I(\overline{op_1}, op_j), seq_2) = \overline{I(op_1, op_i)}$$

Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and the effect of $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ respectively in order to demonstrate that the resulting sequences are equivalent.

$$\begin{aligned}
IT(op'_2, \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_i), \overline{I(op_1, op_j)}) \\
&= IT(op_2, op_i) && (\mathbf{TR4}) \\
IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(IT(op_3, op_i), op_2), \overline{I(op_1, op_j)}) \\
&= I(IT(op_3, op_i), op_2)
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op''_3, \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\
&= IT(op_3, op_j) && (\mathbf{TR4}) \\
IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_j), op_3), \overline{I(op_1, op_i)}) \\
&= I(IT(op_2, op_j), op_3)
\end{aligned}$$

From $op_2 = op_3$, we deduce that $IT(op_2, op_i) \cdot I(IT(op_3, op_i), op_2) \equiv IT(op_3, op_j) \cdot I(IT(op_2, op_j), op_3)$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Thereby, showing that undoing two similar hidden inverse operations produces equivalent logs.

3. $Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: so by transitivity, both op_i and op_j hide op_2 and op_3. This case is similar to case 1.

4. $Hide(op_2, op_1)$ and $\neg Hide(op_1, op_2)$: so neither op_i nor op_j hides op_2. This case is similar to case 2.

Second case. $\neg Hide(op_2, op_3)$ and $\neg Hide(op_3, op_2)$: According to the relation between op_1 and both op_2 and op_3 we discuss the following cases:

1. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: then op_i and op_j hide op_2 and do not hide op_3. Also, $\neg Hide(op_3, op_1)$ otherwise, we have $Hide(op_3, op_2)$ by transitivity of hide relation. In this case we have:

$$\begin{aligned}
op'_2 &= I(op_2, op_1) && (\mathbf{TR1}) \\
op'_3 &= IT(op_3, op_i) && (\mathbf{TR5}) \\
op''_3 &= IT(op_3, op_j) && (\mathbf{TR5}) \\
op''_2 &= I(op_2, op_1) && (\mathbf{TR1})
\end{aligned}$$

Accordingly, we obtain the following sequences:

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, I(op_2, op_1)) \\ &= \overline{I(op_1, op_j)} && (\textbf{TR2I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_j)}, IT(op_3, op_i)) \\ &= \overline{I(op_1, op_j)} \end{aligned}$$

Similarly,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, IT(op_3, op_j)) \\ &= \overline{I(op_1, op_i)} \end{aligned}$$

Consequently,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(op_1, op_i)}, I(op_2, op_1)) \\ &= \overline{I(op_1, op_i)} && (\textbf{TR2I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Then the similarity is preserved by transformation.

Next, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op'_2 \cdot op'_3$ and effect of $\overline{I(op_1, op_i)}$ into $op''_2 \cdot op''_3$ and show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(op_2, op_1), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\textbf{TR4} \text{ and } Hide(op_j, op_2)) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(IT(op_3, op_i), \overline{I(op_1, op_j)}) \\ &= IT(op_3, op_i) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\ &= IT(op_3, op_j) && (\textbf{TR4}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(op_2, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_2, op_i) && (\textbf{TR4} \text{ and } Hide(op_j, op_2)) \end{aligned}$$

From $op_i = op_j$, we deduce that $I(op_2, op_j) \cdot IT(op_3, op_i) \equiv IT(op_3, op_j) \cdot I(op_2, op_i)$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Thus, we have demonstrated that the sequences that follows similar hidden inverse operations after they are undone are equivalent.

2. $\neg Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: so op_i and op_j do not hide op_2. Two cases are to be discussed according to the hide relation between op_i and op_3:

(a) $\neg Hide(op_i, op_3)$:

$$\begin{aligned}
op_2' &= IT(op_2, op_i)\\
op_3' &= IT(IT(op_3, op_i), IT(op_2, op_i))\\
op_3'' &= IT(op_3, op_j)\\
op_2'' &= IT(IT(op_2, op_j), IT(op_3, op_j))
\end{aligned}$$

Then,

$$\begin{aligned}
IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(IT(\overline{I(op_1, op_j)}, IT(op_2, op_i)), op_3')\\
&= IT(\overline{I(op_1, op_j)}, op_3') && (\mathbf{TR2I})\\
&= \overline{I(op_1, op_j)} && (\mathbf{TR2I})
\end{aligned}$$

and

$$\begin{aligned}
IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(IT(\overline{I(op_1, op_i)}, IT(op_3, op_j)), op_2'')\\
&= IT(\overline{I(op_1, op_i)}, IT(IT(op_2, op_j), IT(op_3, op_j))) && (\mathbf{TR2I})\\
&= \overline{I(op_1, op_i)} && (\mathbf{TR2I})
\end{aligned}$$

Thereby showing that indeed $\mathbf{RTP2}$ is satisfied.
Next, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op_2' \cdot op_3'$ and effect of $\overline{I(op_1, op_i)}$ into $op_2'' \cdot op_3''$ and show that the resulting sequences are equivalent.

$$\begin{aligned}
IT(op_2', \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_1), \overline{I(op_1, op_j)})\\
&= IT(op_2, op_j)\\
IT(op_3', IT(\overline{I(op_1, op_j)}, op_2')) &= IT(IT(IT(op_3, op_i), IT(op_2, op_i)), \overline{I(op_1, op_j)})\\
&= IT(IT(op_3, op_i), IT(op_2, op_i))
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op_3'', \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)})\\
&= IT(op_3, op_j) && (\mathbf{TR4})\\
IT(op_2'', IT(\overline{I(op_1, op_i)}, op_3')) &= IT(IT(IT(op_2, op_j), IT(op_3, op_j)), \overline{I(op_1, op_i)})\\
&= IT(IT(op_2, op_j), IT(op_3, op_j))
\end{aligned}$$

From $op_i = op_j$ and $\mathbf{TP1}$, we deduce that $IT(op_2, op_i) \cdot IT(IT(op_3, op_i), IT(op_2, op_i)) \equiv IT(op_3, op_j) \cdot IT(IT(op_2, op_j), IT(op_3, op_j))$. Hence,

$$IT(op_2' \cdot op_3', \overline{I(op_1, op_j)}) \equiv IT(op_3'' \cdot op_2'', \overline{I(op_1, op_i)})$$

(b) $Hide(op_i, op_3)$: since op_i hide op_1 then according to the relation between op_1 and op_3, we discuss the following cases:

i. $Hide(op_1, op_3)$ and $Hide(op_3, op_1)$: in this case we have:

$$\begin{aligned}
op_2' &= IT(op_2, op_i)\\
op_3' &= I(op_3, op_1)\\
op_3'' &= I(op_3, op_1)\\
op_2'' &= IT(op_2, op_j)
\end{aligned}$$

Thus,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, IT(op_2, op_i)) \\ &= \overline{I(op_1, op_j)} && (\mathbf{TR2I}) \end{aligned}$$

Which leads to:

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_j)}, I(op_3, op_1)) \\ &= \overline{I(op_1, op_j)} && (\mathbf{TR2I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Similarly,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, I(op_3, op_1)) \\ &= \overline{I(op_1, op_i)} && (\mathbf{TR2I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(op_1, op_i)}, IT(op_2, op_j)) \\ &= \overline{I(op_1, op_i)} && (\mathbf{TR2I}) \end{aligned}$$

Thereby showing that indeed, property $\mathbf{RTP2}$ is satisfied by hidden inverse operations. Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and that of $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ respectively and show that the resulting sequences are equivalent (the proof is similar to the two precedent sub cases).

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_i), \overline{I(op_1, op_j)}) \\ &= IT(op_2, op_i) && (\mathbf{TR5}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(op_3, op_1), \overline{I(op_1, op_j)}) \\ &= I(op_3, op_j) && (\mathbf{TR4} \text{ and } Hide(op_j, op_3)) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\mathbf{TR4} \text{ and } Hide(op_i, op_3)) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(IT(op_2, op_j), \overline{I(op_1, op_i)}) \\ &= IT(op_2, op_j) \end{aligned}$$

From $op_i = op_j$, we deduce that $IT(op_2, op_i) \cdot I(op_3, op_j) \equiv I(op_3, op_i) \cdot IT(op_2, op_j)$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

ii. $Hide(op_3, op_1)$ and $\neg Hide(op_1, op_3)$: in this case we have:

$$\begin{aligned} op'_2 &= IT(op_2, op_i) \\ op'_3 &= IT(I(IT(op_3, op_1), op_j), IT(op_2, op_i)) \\ &= I(IT(op_3, op_1), op_j) \\ op''_3 &= I(IT(op_3, op_1), op_i) \\ op''_2 &= IT(op_2, op_j) \end{aligned}$$

To integrate the effect of seq_1 into $\overline{I(op_1, op_j)}$, we proceed as follows:

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, IT(op_2, op_i))) \\ &= \overline{I(op_1, op_j)} && (\mathbf{TR2I}) \end{aligned}$$

Then,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_j)}, I(IT(op_3, op_1), op_j)) \\ &= \overline{I(op_1, op_3)} && (\mathbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Similarly, integrating seq_2 into $\overline{I(op_1, op_i)}$ leads to the following sequences:

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, I(IT(op_3, op_1), op_i)) \\ &= \overline{I(op_1, op_3)} && (\mathbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(op_1, op_3)}, IT(op_2, op_j)) \\ &= \overline{I(op_1, op_i)} && (\mathbf{TR2I}) \end{aligned}$$

The result is then obtained.
Next, we integrate the effect of $\overline{I(op_1, op_j)}$ $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ respectively in order to show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_i), \overline{I(op_1, op_j)}) \\ &= IT(op_2, op_i) && (\mathbf{TR5}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(IT(op_3, op_1), op_j), \overline{I(op_1, op_j)}) \\ &= I(op_3, op_j) && (\mathbf{TR4}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(IT(op_3, op_1), op_i), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\mathbf{TR5}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\ &= IT(op_2, op_j) && (\mathbf{TR5}) \end{aligned}$$

From $op_i = op_j$, we deduce that $IT(op_2, op_i) \cdot I(op_3, op_j) \equiv I(op_3, op_i) \cdot IT(op_2, op_j)$.
Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

3. $Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: by transitivity, it comes from $Hide(op_i, op_1)$ and $Hide(op_j, op_1)$ that $Hide(op_i, op_2)$ and $Hide(op_j, op_2)$. This leads to the following results:

$$\begin{aligned} op'_2 &= I(op_2, op_1) \\ op'_3 &= IT(op_3, op_i) \\ op''_3 &= IT(op_3, op_j) \\ op''_2 &= I(op_2, op_1) \end{aligned}$$

Accordingly, undoing $I(op_1, op_j)$ leads to:

$$\begin{aligned}
IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(IT(\overline{I(op_1, op_j)}, I(op_2, op_1)), IT(op_3, op_i)) \\
&= IT(\overline{I(IT(op_1, op_2), op_j)}, IT(op_3, op_i)) && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op_1) \\
&= \overline{I(op_1, op_j)} && (\mathbf{TR2I})
\end{aligned}$$

Furthermore,

$$\begin{aligned}
IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(IT(\overline{I(op_1, op_j)}, IT(op_3, op_j)), op_2'') \\
&= IT(\overline{I(op_1, op_i)}, I(op_2, op_1)) && (\mathbf{TR2I}) \\
&= \overline{I(IT(op_1, op_2), op_i)} && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op_1)
\end{aligned}$$

Consequently, the similarity is preserved by transformation. Thereby showing that indeed $\mathbf{RTP2}$ is satisfied.

Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and $\overline{I(op_1, op_i)}$ into $op_2' \cdot op_3'$ and $op_2'' \cdot op_3''$ respectively. Then we demonstrate that the resulting sequences are equivalent.

$$\begin{aligned}
IT(op_2', \overline{I(op_1, op_j)}) &= IT(I(op_2, op_i), \overline{I(op_1, op_j)}) \\
&= I(op_2, op_i) \\
IT(op_3', IT(\overline{I(op_1, op_j)}, op_2')) &= IT(IT(op_3, op_i), \overline{I(IT(op_1, op_2), op_j)}) \\
&= IT(op_3, op_i) && (\mathbf{TR5})
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op_3'', \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\
&= IT(op_3, op_j) && (\mathbf{TR5}) \\
IT(op_2'', IT(\overline{I(op_1, op_i)}, op_3')) &= IT(I(op_2, op_1), \overline{I(op_1, op_i)}) \\
&= I(op_2, op_i) && (\mathbf{TR4})
\end{aligned}$$

From $op_i = op_j$, we deduce that $I(op_2, op_i) \cdot IT(op_3, op_i) \equiv IT(op_3, op_j) \cdot I(op_2, op_i)$. Hence,

$$IT(op_2' \cdot op_3', \overline{I(op_1, op_j)}) \equiv IT(op_3'' \cdot op_2'', \overline{I(op_1, op_i)})$$

4. $\neg Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: in this case, $\neg Hide(op_1, op_3)$ otherwise by transitivity of the relation Hide we obtain $Hide(op_2, op_3)$ which is false. Moreover, $\neg Hide(op_3, op_1)$ because if we suppose that $\neg Hide(op_3, op_1)$, it implies that there is a hide relation between op_2 and op_3 which is not the case. Consequently, we discuss two cases according to the relation between $op_i = op_j$ and op_2.

   (a) $Hide(op_i, op_2)$ and $\neg Hide(op_2, op_i)$: this leads to $\neg Hide(op_i, op_3)$ since there is no relation between op_2 and op_3. The following transformations are obtained:

$$\begin{aligned}
op_2' &= I(IT(op_2, op_1), op_j) \\
op_3' &= IT(op_3, op_i) \\
op_3'' &= IT(op_3, op_j) \\
op_2'' &= I(IT(op_2, op_1), op_i)
\end{aligned}$$

Then,

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, I(IT(op_2, op_1), op_j)) \\ &= \overline{I(op_1, op_2)} && (\textbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

So,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_2)}, IT(op_3, op_i)) \\ &= \overline{I(op_1, op_2)} && (\textbf{TR2I}) \end{aligned}$$

and

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(IT(\overline{I(op_1, op_i)}, IT(op_3, op_j)), op''_2) \\ &= IT(\overline{I(op_1, op_i)}, I(IT(op_2, op_1), op_i)) && (\textbf{TR2I}) \\ &= \overline{I(op_1, op_2)} && (\textbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

The result is then obtained.
Next, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op'_2 \cdot op'_3$ and effect of $\overline{I(op_1, op_i)}$ into $op''_2 \cdot op''_3$ and show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(IT(op_2, op_1), op_j), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\textbf{TR4}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(IT(op_3, op_i), \overline{I(op_1, op_j)}) \\ &= IT(op_3, op_i) && (\textbf{TR5}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\ &= IT(op_3, op_j) && (\textbf{TR5}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_1), op_i), \overline{I(op_1, op_i)}) \\ &= I(op_2, op_i) && (\textbf{TR4}) \end{aligned}$$

From $op_i = op_j$, we deduce that $I(op_2, op_j) \cdot IT(op_3, op_i) \equiv IT(op_3, op_j) \cdot I(op_2, op_i)$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

(b) $Hide(op_i, op_2)$ and $Hide(op_2, op_i)$: we obtain the same transformations as in the precedent case.

(c) $\neg Hide(op_i, op_2)$: in this case we have $\neg Hide(op_i, op_3)$ otherwise there is a hide relation between op_3 and op_1 which is not the case.

$$\begin{aligned} op'_2 &= IT(op_2, op_i) \\ op'_3 &= IT(IT(op_3, op_i), IT(op_2, op_i)) \\ &= IT^*(op_3, op_i \cdot op_2) \end{aligned}$$

In the same way,

$$\begin{aligned} op''_3 &= IT(op_3, op_j) \\ op''_2 &= IT(IT(op_2, op_j), IT(op_3, op_j)) \\ &= IT^*(op_2, op_j \cdot op_3) \end{aligned}$$

Then,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(IT(\overline{I(op_1, op_j)}, IT(op_2, op_i))), op'_3) \\ &= IT(\overline{I(op_1, op_j)}, IT^*(op_3, op_i \cdot op_2)) && (\mathbf{TR2I}) \\ &= \overline{I(op_1, op_j))} && (\mathbf{TR2I}) \end{aligned}$$

Similarly,

$$IT^*(\overline{I(op_1, op_i)}, seq_2) = \overline{I(op_1, op_i))}$$

The result is then obtained.
In the following, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op'_2 \cdot op'_3$ and the effect of $\overline{I(op_1, op_i)}$ into $op''_2 \cdot op''_3$. Then we show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_i), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\mathbf{TR4}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(IT(IT(op_3, op_i), IT(op_2, op_i)), \overline{I(op_1, op_j)}) \\ &= IT(IT(op_3 op_i), IT(op_2, op_i)) && (\mathbf{TR5}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\ &= IT(op_3, op_j) && (\mathbf{TR5}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(IT(IT(op_2, op_j), IT(op_3, op_j)), \overline{I(op_1, op_i)}) \\ &= IT(IT(op_2, op_j), IT(op_3, op_j)) && (\mathbf{TR4}) \end{aligned}$$

From $op_i = op_j$ and $\mathbf{TP1}$, we deduce that $I(op_2, op_j) \cdot IT(IT(op_3 op_i), IT(op_2, op_i)) \equiv IT(op_3, op_j) \cdot IT(IT(op_2, op_j), IT(op_3, op_j))$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Thereby showing that indeed, undoing two similar hidden inverse operations produces equivalent sequences of operations. *Q.E.D.*

Third case. $Hide(op_2, op_3)$ and $\neg Hide(op_3, op_2)$: According to the relation between op_1 and both op_2 and op_3 we discuss the following cases:

1. $Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: suppose that $Hide(op_3, op_1)$, this leads to $Hide(op_2, op_3)$ (see Lemma 6.3.10) which is false. Accordingly, we have:

$$\begin{aligned} op'_2 &= I(op_2, op_1) \\ op'_3 &= I(op_3, op_2) \\ op''_3 &= I(op_3, op_1) \\ op''_2 &= I(IT(op_2, op_3), op_1) \end{aligned}$$

We first integrate the effect of seq_1 into $\overline{I(op_1, op_j)}$ as follows:

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, I(op_2, op_1)) \\ &= \overline{I(op_1, op_j)} && (\textbf{TR2I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Thus,

$$\begin{aligned} IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_j)}, I(op_3, op_2)) \\ &= \overline{I(IT(op_1, op_3), op_j)} && (\textbf{TR2I} \text{ and } \neg Hide(op_3, op_1)) \end{aligned}$$

In the same way,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, I(op_3, op_1)) \\ &= \overline{I(IT(op_1, op_3), op_i)} \end{aligned}$$

Consequently,

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(IT(op_1, op_3), op_i)}, I(IT(op_2, op_3), op_1)) \\ &= \overline{I(IT(op_1, op_3), op_i)} && (\textbf{TR2I} \text{ and Lemma 8.1.1}) \end{aligned}$$

Thereby showing that indeed **RTP2** is satisfied.

Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ respectively and show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(op_2, op_1), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\textbf{TR4} \text{ and } Hide(op_j, op_2)) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(op_3, op_2), \overline{I(op_1, op_j)}) \\ &= I(op_3, op_j) && (\textbf{TR5}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_j) && (\textbf{TR4} \text{ and } Hide(op_j, op_3)) \end{aligned}$$

$$\begin{aligned} IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_3), op_1), \overline{I(IT(op_1, op_3), op_i)}) \\ &= I(IT(op_2, op_3), op_i) && (\textbf{TR4}) \end{aligned}$$

Consequently $I(op_2, op_j) \cdot I(op_3 op_j) \equiv I(op_3, op_j) \cdot I(IT(op_2, op_3), op_i) \equiv \emptyset$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

2. $\neg Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: in this case we have:

$$\begin{aligned} op'_2 &= IT(op_2, op_i) \\ op'_3 &= I(IT(op_3, op_i), IT(op_2, op_i)) \\ op''_3 &= IT(op_3, op_j) \\ op''_2 &= IT(IT(op_2, op_j), IT(op_3, op_j)) \end{aligned}$$

Then, according to $\mathbf{TR2I}$,

$$IT(\overline{I(op_1, op_j)}, seq_1) = \overline{I(op_1, op_j)}$$
$$IT^*(\overline{I(op_1, op_i)}, seq_2) = \overline{I(op_1, op_i)}$$

Since $IT(\overline{I(op_1, op_j)}, seq_1) = IT^*(\overline{I(op_1, op_i)}, seq_2)$, it has now been proved that property $\mathbf{RTP2}$ is satisfied by hidden inverse operations. Next, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op'_2 \cdot op'_3$ and effect of $\overline{I(op_1, op_i)}$ into $op''_2 \cdot op''_3$ and show that the resulting sequences are equivalent.

$$\begin{aligned}
IT(op'_2, \overline{I(op_1, op_j)}) &= IT(IT(op_2, op_i), \overline{I(op_1, op_j)}) \\
&= IT(op_2, op_i) && (\mathbf{TR5}) \\
IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(IT(op_3, op_i), IT(op_2, op_i)), \overline{I(op_1, op_j)}) \\
&= I(IT(op_3, op_i), IT(op_2, op_i)) && (\mathbf{TR5})
\end{aligned}$$

Similarly,

$$\begin{aligned}
IT(op''_3, \overline{I(op_1, op_i)}) &= IT(IT(op_3, op_j), \overline{I(op_1, op_i)}) \\
&= IT(op_3, op_j) && (\mathbf{TR5}) \\
IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(IT(IT(op_2, op_j), IT(op_3, op_j)), \overline{I(op_1, op_i)}) \\
&= IT(IT(op_2, op_j), IT(op_3, op_j)) && (\mathbf{TR5})
\end{aligned}$$

Since $Hide(IT(op_2, op_j), IT(op_3, op_j))$ and $\mathbf{TP2}$ is assumed for the initial set, we deduce that $IT(op_2, op_i) \cdot I(IT(op_3, op_i), IT(op_2, op_i)) \equiv IT(op_3, op_j) \cdot IT(IT(op_2, op_j), IT(op_3, op_j))$. Hence,

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Consequently, it has been proved that undoing two similar hidden inverse produces equivalent sequences of operations.

3. $Hide(op_1, op_2)$ and $\neg Hide(op_2, op_1)$: in this case we have:

$$\begin{aligned}
op'_2 &= I(op_2, op_1) \\
op'_3 &= I(op_3, op_2) \\
op''_3 &= I(op_3, op_1) \\
op''_2 &= I(IT(op_2, op_3), op_1)
\end{aligned}$$

Then,

$$\begin{aligned}
IT^*(\overline{I(op_1, op_j)}, seq_1) &= IT(IT(\overline{I(op_1, op_j)}, I(op_2, op_1)), I(op_3, op_2)) \\
&= IT(\overline{I(IT(op_1, op_2), op_j}, I(op_3, op_2)) && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op_1) \\
&= \overline{I(IT(op_1, op_2), op_j)}
\end{aligned}$$

We also have:

$$\begin{aligned}
IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(IT(\overline{I(op_1, op_i)}, I(op_3, op_1)), I(IT(op_2, op_3), op_1)) \\
&= IT(\overline{I(IT(op_1, op_3), op_i)}, I(IT(op_2, op_3), op_1)) && (\mathbf{TR2I} \text{ and } \neg Hide(op_3, op_1)) \\
&= \overline{I(IT(IT(op_1, op_3), IT(op_2, op_3)), op_i)} && (\mathbf{TR2I} \text{ and } \neg Hide(op_2, op_1)) \\
&= \overline{I(IT(IT(op_1, op_2), IT(op_3, op_2)), op_i)} && \text{(Lemma 8.3.4)} \\
&= \overline{I(IT(op_1, op_2), op_i)} && \text{(Lemma 6.3.8)}
\end{aligned}$$

Thereby showing that indeed $\mathbf{RTP2}$ is satisfied.
Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ and show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(op_2, op_1), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\mathbf{TR4} \text{ and } Hide(op_j, op_2)) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(op_3, op_2), \overline{I(IT(op_1, op_2), op_j)}) \\ &= I(op_3, op_2) && (\mathbf{TR5}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\mathbf{TR4} \text{ and } Hide(op_j, op_3)) \end{aligned}$$

$$\begin{aligned} IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_3), op_1), \overline{I(IT(op_1, op_3), op_i)}) \\ &= I(IT(op_2, op_3), op_i) && (\mathbf{TR4}) \end{aligned}$$

Since $I(op_2, op_j) \cdot I(op_3, op_2) \equiv I(op_3, op_j) \cdot I(IT(op_2, op_3), op_i) \equiv \emptyset$, we deduce that

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

4. $\neg Hide(op_1, op_2)$ and $Hide(op_2, op_1)$: without loss of generalisation, we suppose that $Hide(op_i, op_2)$. Subsequently, op_2 and op_3 are transformed as follows according to the relation between op_1 and op_3:

   (a) $Hide(op_1, op_3)$ and $Hide(op_3, op_1)$:

$$\begin{aligned} op'_2 &= I(IT(op_2, op_1), op_j) \\ op'_3 &= I(IT(op_3, op_1), op_2) && \text{(Lemma 6.3.10)} \\ op''_3 &= I(op_3, op_1) \\ op''_2 &= I(IT(op_2, op_1), op_i) \end{aligned}$$

   Accordingly, we have

$$\begin{aligned} IT(\overline{I(op_1, op_i)}, op'_2) &= IT(\overline{I(op_1, op_j)}, I(op_3, op_1)) \\ &= \overline{I(op_1, op_2)} && (\mathbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

   Thus,

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_2)}, I(IT(op_3, op_1), op_2)) \\ &= \overline{I(op_1, op_2)} && (\mathbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

   Similarly, we have:

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, op'_3) &= IT(\overline{I(op_1, op_i)}, I(op_3, op_1)) \\ &= \overline{I(op_1, op_i)} && (\mathbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Thus,

$$\begin{aligned} IT(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(op_1, op_i)}, I(IT(op_2, op_1), op_i)) \\ &= \overline{I(op_1, op_2)} && (\textbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Then, the integrated forms of both $\overline{I(op_1, op_i)}$ and $\overline{I(op_1, op_j)}$ are similar. Thereby showing that indeed $\mathbf{RTP2}$ is satisfied.
In the following, we integrate the effect of $\overline{I(op_1, op_j)}$ and $\overline{I(op_1, op_i)}$ into $op'_2 \cdot op'_3$ and $op''_2 \cdot op''_3$ respectively. then we show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(IT(op_2, op_1), op_j), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\mathbf{TR4}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(IT(op_3, op_1), op_2), \overline{I(op_1, op_2)}) \\ &= I(op_3, op_2) && (\mathbf{TR4}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\mathbf{TR5}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_1), op_i), \overline{I(op_1, op_i)}) \\ &= I(op_2, op_i) && (\mathbf{TR5}) \end{aligned}$$

$I(op_2, op_j) \cdot I(op_3, op_2) \equiv IT(op_3, op_i) \cdot I(op_2, op_i) \equiv \emptyset$, we deduce that

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

Q.E.D.

(b) $Hide(op_1, op_3)$ and $\neg Hide(op_3, op_1)$:

$$\begin{aligned} op'_2 &= I(IT(op_2, op_1), op_j) \\ op'_3 &= I(op_3, op_1) && (\mathbf{TR5}) \\ op''_3 &= I(op_3, op_1) \\ op''_2 &= I(IT(op_2, op_1), op_i) \end{aligned}$$

Thus, we transform $\overline{I(op_1, op_j)}$ as follows:

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, op'_2) &= IT(\overline{I(op_1, op_j)}, I(IT(op_2, op_1), op_j) \\ &= \overline{I(op_1, op_2)} && (\textbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Consequently,

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_2)}, I(op_3, op_1)) \\ &= I(IT(op_1, op_3), op_2) && (\textbf{TR1I} \text{ and } \neg Hide(op_3, op_1)) \end{aligned}$$

Furthermore, we transform $\overline{I(op_1, op_i)}$ as follows:

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, I(op_3, op_1)) \\ &= \overline{I(IT(op_1, op_3), op_i)} && (\textbf{TR2I} \text{ and } \neg Hide(op_3, op_1)) \end{aligned}$$

Since $Hide(op_2, op_1)$, then according to $\mathbf{TR1I}$, we have:

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(IT(op_1, op_3), op_i)}, I(IT(op_2, op_1), op_i)) \\ &= \overline{I(IT(op_1, op_3), op_2)} \end{aligned}$$

Then, the similarity is preserved by transformation which means that the property $\mathbf{RTP2}$ is satisfied.

Next, we integrate the effect of $\overline{I(op_1, op_j)}$ and $\overline{I(op_1, op_i)}$ into $op_2' \cdot op_3'$ and $op_2'' \cdot op_3''$ respectively. We then show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op_2', \overline{I(op_1, op_j)}) &= IT(I(IT(op_2, op_1), op_j), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\mathbf{TR4}) \\ IT(op_3', IT(\overline{I(op_1, op_j)}, op_2')) &= IT(I(op_3, op_1), \overline{I(op_1, op_2)}) \\ &= I(op_3, op_2) && (\mathbf{TR5}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op_3'', \overline{I(op_1, op_i)}) &= IT(I(op_3, op_1), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\mathbf{TR4}) \end{aligned}$$

It comes from $\mathbf{TR4}$ and $\neg Hide(op_3, op_2)$ that

$$\begin{aligned} IT(op_2'', IT(\overline{I(op_1, op_i)}, op_3')) &= IT(I(IT(op_2, op_1), op_i), \overline{I(IT(op_1, op_3), op_i)}) \\ &= I(IT(op_2, op_3), op_i) \end{aligned}$$

Since $I(op_2, op_j) \cdot I(op_3, op_2) \equiv I(op_3, op_i) \cdot I(IT(op_2, op_3), op_i) \equiv \emptyset$, we deduce that

$$IT(op_2' \cdot op_3', \overline{I(op_1, op_j)}) \equiv IT(op_3'' \cdot op_2'', \overline{I(op_1, op_i)})$$

It has now been proved that undoing two similar hidden inverse operations leads to equivalent sequences of operations.

(c) $\neg Hide(op_1, op_3)$ and $Hide(op_3, op_1)$:

$$\begin{aligned} op_2' &= I(IT(op_2, op_1), op_j) \\ op_3' &= I(IT(op_3, op_1), op_2) \\ op_3'' &= I(IT(op_3, op_1), op_i) \\ op_2'' &= I(IT(IT(op_2, op_1), IT(op_3, op_1)), op_i) \\ &= I(IT(IT(op_2, op_3), IT(op_1, op_3)), op_i) \\ &= I(IT(op_2, op_3), op_i) \end{aligned}$$

First, we transform $\overline{I(op_1, op_j)}$ against op_2' as follows:

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, op_2') &= IT(\overline{I(op_1, op_j)}, I(IT(op_2, op_1), op_j)) \\ &= \overline{I(op_1, op_2)} && (\mathbf{TR1I} \text{ and } Hide(op_2, op_1)) \end{aligned}$$

Thus, transforming $\overline{I(op_1, op_j)}$ against seq_1 leads to the following sequences:

$$\begin{aligned} IT(\overline{I(op_1, op_j)}, seq_1) &= IT(\overline{I(op_1, op_2)}, op_3') \\ &= IT(\overline{I(op_1, op_2)}, I(IT(op_3, op_1), op_2)) \\ &= \overline{I(op_1, op_3)} && (\mathbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Now, we transform $\overline{I(op_1, op_i)}$ against op''_3 as follows:

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, op''_3) &= IT(\overline{I(op_1, op_i)}, I(IT(op_3, op_1), op_i)) \\ &= \overline{I(op_1, op_3)} && (\textbf{TR1I} \text{ and } Hide(op_3, op_1)) \end{aligned}$$

Consequently, the transformation of $\overline{I(op_1, op_i)}$ against seq_2 is:

$$\begin{aligned} IT^*(\overline{I(op_1, op_i)}, seq_2) &= IT(\overline{I(op_1, op_3)}, I(IT(op_2, op_3), op_i)) \\ &= \overline{I(op_1, op_3)} \end{aligned}$$

Thereby proving that the similarity is preserved by transformation.
Next, we integrate the effect of $\overline{I(op_1, op_j)}$ into $op'_2 \cdot op'_3$ and the effect of $\overline{I(op_1, op_i)}$ into $op''_2 \cdot op''_3$ and show that the resulting sequences are equivalent.

$$\begin{aligned} IT(op'_2, \overline{I(op_1, op_j)}) &= IT(I(IT(op_2, op_1), op_j), \overline{I(op_1, op_j)}) \\ &= I(op_2, op_j) && (\textbf{TR4}) \\ IT(op'_3, IT(\overline{I(op_1, op_j)}, op'_2)) &= IT(I(IT(op_3, op_1), op_2), \overline{I(op_1, op_2)}) \\ &= I(op_3, op_2) && (\textbf{TR4}) \end{aligned}$$

Similarly,

$$\begin{aligned} IT(op''_3, \overline{I(op_1, op_i)}) &= IT(I(IT(op_3, op_1), op_i), \overline{I(op_1, op_i)}) \\ &= I(op_3, op_i) && (\textbf{TR4}) \\ IT(op''_2, IT(\overline{I(op_1, op_i)}, op'_3)) &= IT(I(IT(op_2, op_3), op_i), \overline{I(op_1, op_3)}) \\ &= I(IT(op_2, op_3), op_i) \end{aligned}$$

Since $I(op_2, op_j) \cdot I(op_3, op_2) \equiv I(op_3, op_i) \cdot I(IT(op_2, op_3), op_i) \equiv \emptyset$, we deduce that

$$IT(op'_2 \cdot op'_3, \overline{I(op_1, op_j)}) \equiv IT(op''_3 \cdot op''_2, \overline{I(op_1, op_i)})$$

It has now been proved that undoing two similar hidden inverse operations leads to equivalent sequences of operations.

□

According to precedent proof, we demonstrate in Theorem 8.3.5 that property **TP2** is preserved for the enhanced set of operations that includes the new semantic of the idle operation. As for resulting logs after undoing a hidden inverse operation, they are equivalent. Thereby showing that indeed our undo approach is correct.

Theorem 8.3.5 (Undoing Hidden Inverse Operations)**.**

Given two similar inverse operations $I(\overline{op_1}, op_i) \equiv I(\overline{op_1}, op_j)$ and two equivalent logs H_1 and H_2. Then, the following statements are true:

$$IT^*(\overline{I(op, op_i)}, H_1) \equiv IT^*(\overline{I(op, op_j)}, H_2)$$

$$IT(H_1, \overline{I(op, op_i)}) \equiv IT(H_2, \overline{I(op, op_j)})$$

Proof. By induction on $|H|$ and the use of the precedent lemma. □

Appendix B

A Distributed Garbage Collector for Cooperative Logs

Collaborative editors are generally based on log usage, as it is necessary to store the track of the operations received to ensure the convergence of the shared data. Hence, every site s, has to maintain a log containing the requests of which site s is informed $i.e.$ the requests originated at site s or originated at other sites and then communicated to s.

The motivation for maintaining a log at each site is that a remote operation must integrate the effect of all concurrent operations to be executed on the receiver site. However, not all operations received by a site must be kept in its log. In particular, an operation can be safely deleted from a site's log if: (i) it is already received in all other sites and (ii) all operations that depend on it are received by all sites.

Consequently, it is possible and even recommend to use a garbage collection mechanism in order to clean logs and makes possible to deploy a log-based collaborative editor on a mobile application.

Indeed, the mobile phone technologies are becoming pervasive in recent years. These items such as IPhones, IPad and Androïds are very attractive since they provide relatively good resources for a mobile device. Several works aim at integrating desktop applications in these tools to make them closer to the real computer. However, adapting desktop applications to these tools is a challenging problem as they do not have the same features.

We stress that even though mobile phones offer ad-hoc communications and are suitable to host distributed applications, they are battery-powered so less powerful and have limited storage capacities compared to those offered by computers.

Deploying RCE based on logs on mobile devices can be costly. Indeed, they require lot of memory to manage the increasing log size. Consequently, a mechanism of garbage collection must be set up to allow for log reset at a given time and then to start again the collaboration with empty logs what improves the performances and response time of the collaborative application.

The motivation for garbage collection is twofold. Firstly, storage capacity is not infinite as in practice memory always has limited size. Secondly, with the continuous increase of log size, the performances of the system degrade. Hence we need to devise a garbage collection technique allowing to delete all operations already seen and executed at collaborating sites at a given time knowing that logs are not identical in different sites as we allow out-of-order execution of operations. Moreover, group size in collaboration model is dynamic (churn) which complicates the garbage task.

However, many challenges should be taken considered since RCEs have specific requirements that may not match with those characterizing mobile devices (see Chapter 1):

- *High local responsiveness*

- *High concurrency*
- *Consistency*
- *Decentralized coordination*
- *Scalability*

In the following, we present the OPTIC framework. We illustrate garbage collection issues and show how we can avoid them. Finally, we present our successful garbage collection scheme that optimally manages mobile devices resources for the OPTIC framework.

B.1 The Coordination Framework OPTIC

In this section, we present the main features and algorithms of the collaborative framework OPTIC [47, 48]. This framework is characterized by its distribution and high performance. It resorts to a novel technique called semantic dependency for defining the causality relation between document updates and resorts to the integration functions IT and ET.

Inclusive Transformation. IT algorithm is used to execute concurrent requests in any order.

The transformation function IT used in OPTIC is defined as follows:

```
$IT(q_1, q_2) = q'_1$
$q'_1 \leftarrow q_1$
Choice of $q_1$ and $q_2$
  Case: $q_1 = i_1$ and $q_2 = i_2$
    if ($p_2 < p_1$ or ($p_2 = p_1$ and $q_2.c < q_1.c$)) then $q'_1.o \leftarrow Ins(p_1 + 1, e_1, p_1\omega_1)$
  Case: $q_1 = i_1$ and $q_2 = d_2$
    if ($p_2 < p_1$) then $q'_1.o \leftarrow Ins(p_1 - 1, e_1, p_1\omega_1)$
    else if ($p_2 = p_1$) then $q'_1.o \leftarrow Ins(p_1, e_1, p_1\omega_1)$
  Case: $q_1 = d_1$ and $q_2 = i_2$
    if ($p_2 \leq p_1$) then $q'_1.o \leftarrow Del(p_1 + 1)$
  Case: $q_1 = d_1$ and $q_2 = d_2$
    if ($p_2 < p_1$) then $q'_1.o \leftarrow Del(p_1 - 1)$
    else if ($p_2 = p_1$) then $q'_1.o \leftarrow Nop()$
  Case: $q_1 = u_1$ and $q_2 = i_2$
    if ($p_2 \leq p_1$) then $q'_1.o \leftarrow Up(p_1 + 1, old(u_1), new(u_1))$
  Case: $q_1 = u_1$ and $q_2 = u_2$
    if ($p_1 = p_2$) and ($q_2.c < q_1.c$) then $q'_1.o \leftarrow Up(p_1, new(u_2), new(u_1))$
    else if ($p_1 = p_2$) and ($q_2.c > q_1.c$) then $q'_1.o \leftarrow Nop()$
  Case: $q_1 = u_1$ and $q_2 = d_2$
    if ($p_2 < p_1$) then $q'_1.o \leftarrow Upd(p_1 - 1, old(u_1), new(u_1))$
    else if ($p_2 = p_1$) then $q'_1.o \leftarrow Nop()$
end choice
return $q'_1$
```

Algorithm 14: Inclusive transformation.

Exclusive Transformation ET algorithm is used to exclude operations effect in order to reorder logs. It is given in Algorithm 15.

The coordination protocol proceeds as follows (see Algorithm 16):

```
$ET(q_1, q_2) = q'_1$
$q'_1 \leftarrow q_1$
Choice of $q_1$ and $q_2$
  Case: $q_1 = i_1$ and $q_2 = i_2$
    if ($p_1 = p_2$ and $q_1.c \geq q_2.c$) or ($p_1 = p_2 + 1$ and $q_1.c \leq q_2.c$)) then return "Undefined"
    else $q'_1.o \leftarrow Ins(p_1 - 1, e_1, \omega_1)$
  Case: $q_1 = i_1$ and $q_2 = d_2$
    if ($p_1 = p_2$) then $q'_1.o \leftarrow Ins(p_1, e_1, Tl(\omega_1))$ /* Function $Tl$ returns the resulting stack without the top position */
    else if ($p_1 > p_2$) then $q'_1.o \leftarrow Ins(p_1 + 1, e_1, Tl(\omega_1))$
  Case: $q_1 = d_1$ and $q_2 = i_2$
    if ($p_1 \geq p_2 + 1$) then $q'_1.o \leftarrow Del(p_1 - 1)$
    else if ($p_1 = p_2$) then return "Undefined"
  Case : $q_1 = d_1$ and $q_2 = d_2$
    if ($p_1 \geq p_2$) then $q'_1.o \leftarrow Del(p_1 + 1)$
  Case : $q_1 = u_1$ and $q_2 = i_2$
    if ($p_1 > p_2$) then $q'_1.o \leftarrow Up(p_1 - 1, old(u_1), new(u_1))$
    else if ($p_1 = p_2$) then return "Undefined"
  Case : $q_1 = u_1$ and $q_2 = d_2$
    if ($p_1 > p_2$) then $q'_1.o \leftarrow Up(p_1 + 1, old(u_1), new(u_1))$
  Case : $q_1 = u_1$ and $q_2 = u_2$
    if ($p_1 = p_2$ and $q_1.c > q_2.c$) then $q'_1.o \leftarrow Up(p_1, old(u_2), new(u_1))$
    else if ($p_1 = p_2$) return "Undefined"
end choice
return $q'_1$
```

Algorithm 15: Exclusive transformation.

```
Main:
INITIALIZATION
while not aborted do
    if there is an input $o$ then
        GENERATE_REQUEST($o$)
    else
        RECEIVE_REQUEST
        INTEGRATE_REMOTE_REQUESTS
    end if
end while

INITIALIZATION:
$Q \leftarrow []$
$L \leftarrow []$
$l \leftarrow l_0$
$r \leftarrow 1$
$c \leftarrow$ Identification of local user

GENERATE_REQUEST($o$):
$l \leftarrow Do(o, l)$
$q \leftarrow (c, r, null, o)$
$q' \leftarrow$ COMPUTEBF($q$,$L$)
$L \leftarrow$ CANONIZE($q$,$L$)
broadcast $q'$ to other users

RECEIVE_REQUEST:
if there is a request $q$ from a network then
    $Q \leftarrow Q + q$
end if

INTEGRATE_REMOTE_REQUEST:
if there is $q$ in $Q$ that is causally-ready then
    $Q \leftarrow Q - q$
    $q' \leftarrow$ COMPUTEFF($q$,$L$)
    $l \leftarrow Do(q'.o, l)$
    $L \leftarrow$ CANONIZE($q'$,$L$)
end if
```

Algorithm 16: Control Concurrency Algorithm

```
COMPUTEBF($q$,$L$) : $q'$
$q' \leftarrow q$
for ($i = |L| - 1; i \geq 0; i - -$) do
    if $q'$ is not dependent of $L[i]$ then
        $q' \leftarrow ET(q', L[i])$
    else
        $q'.a = (L[i].p, L[i].k, \mathbf{df_j})$
        {$df_j$ with $j = 1, 2, 3, 4$ or 5 according to the dependency form}
        return $q'$
    end if
end for
return $q'$
```

Algorithm 17: Detection of Causal Dependency

```
CANONIZE($q$,$L$) : $L'$
$L' \leftarrow [L; q]$
$i \leftarrow |L'| - 1$
while $L'$ is not canonical do
    $< L'[i-1], L'[i] > \leftarrow PERM(L'[i], L'[i-1])$
    $i \leftarrow i - 1$
end while
return $L'$
```

Algorithm 18: Canonizing Logs

```
COMPUTEFF($q$,$L$) : $q'$
$q' \leftarrow q$
$j \leftarrow -1$
if $q'.a \neq null$ then
    Let $L[j]$ be the request whose $q'$ depends on ($j \in \{0, \ldots, |L| - 1\}$)
    Modify $q'.o$ with respect to $L[j].o$ and the dependency form
end if
for ($i = j + 1; i \leq |L| - 1; i + +$) do
    $q' \leftarrow IT(q', L[i])$
end for
return $q'$
```

Algorithm 19: Transforming a Request Against a Log

1. When a user manipulates the local copy of the shared document by generating a cooperative operation, we determine its causal dependency by the usage of ComputeBF function (see Algorithm 17) then broadcast it to other users. Note that after each generation of local cooperative request the log is reorganized (see Algorithm 18) to keep it canonical.

2. Each site has to determine the new form of every remote request using ComputeFF function (see Algorithm 19) in order to take into account concurrent requests and integrate their effect in the received request. Once the operation is executed, the log is canonized.

Given the OPTIC framework, our objective is to extend it with a garbage protocol. In the following, we investigate the issues raised by the garbage process and our solutions to overcome these issues.

B.2 Garbage Collection Issues

The garbage collection represents a real challenge for collaborative editors especially those based on peer-to-peer communication. In fact, existing solution only fit fixed group size applications having the same view of logs in all collaborating sites. It should be noted that in dynamic applications, the garbage collection faces several issues:

First Issue: Out of order execution between garbage and update operations Since logs are equivalent and not identical from one site to another due to out of order execution of operations, the log removal operation may be executed on different contexts from one site to another. In such situation, collaborative editing after the garbage collection procedure inevitably leads to divergence cases as it is shown in the scenario of Figure B.1. In this scenario, we consider two users that begin the collaboration with the same initial state "eact". Site 1 generates $o_1 = ins(1, r)$ to insert the character r at position 1 and gets the state "react". The second site removes the character "e" at position 1 with the operation $o_2 = del(1)$ and then gets the state "act". Concurrently, the site 1 initiates garbage collection, cleans his log and propagates the removal order to the site 2. The latter receives the garbage collection order before the operation o_1. Hence, when he receives $o_1 = ins(1, r)$, he executes it in its received form since IT function has no impact on o_1 (the log is empty). Consequently, the final state at site 2 is "ract". Now, site 1 receives $o_2 = del(1)$ and executes it also in its reception form as his log is empty and then derives the state "eact" which is different from the state of site 2.

It should be pointed out that an operation may also loose its dependency and remains eternally unready if logs are removed in arbitrary fashion.

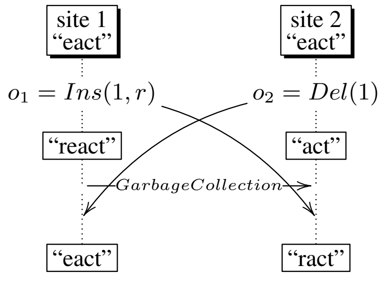


Figure B.1: Divergence caused by garbage collection applied on different contexts

Accordingly, cleaning process requires a global view of the collaboration state in order to ensure that log removal will be executed at the same context in all collaborating sites. In other words, we must be able to deduce whether an operation was received by all users or not, and whether it is needed by an other operation for an integration process. If it is the case, then the operation could be safely removed from the log. Otherwise, replicas will diverge.

To overcome this problem, it is necessary to draw a global view on the state of each user in order to decide about the portion of log that could be deleted without leading to divergence. This is possible to achieve through request-response messages before log removal. The question that arises here: is global view possible in a dynamic and distributed collaboration?

Second Issue: Scalability Problem Suppose that the garbage collection only begins when all users are ready to clean their logs. To know if a user is able to remove his log or not, we must wait for his answer. However in a peer-to-peer context, a user may leave the group at any time and thus never responds to the garbage request. A user can also crash down and be prevented to respond. In both cases, it is difficult to take the correct garbage decision while preserving consistency criteria. Because of message asynchrony, a user has no safe means to know whether another user has or has not crash. Even the timeout approach has many inconveniences. For instance, a user may respond just after the timeout expiration. Moreover, even the timeout itself is difficult to define.

The solution proposed in [99] relies on state vectors to draw a state view for each user. Every silent user has to send periodically the value of its state vector to other users. State vectors of active sites are deduced from the operations they perform. These vectors are used to calculate a minimal state vector allowing for garbage decisions. However, this approach is limited to static groups and could not be used in the case of dynamic groups because of failure impact on the garbage procedure $i.e.$ when a site is absent or silent, its state vector remains unchanged, then other users could no more delete operations from their logs.

Third Issue: Joint Problem Another issue that could be faced when trying to garbage logs in RCE is when the garbage process is disturbed by the join of a new user. It is known that in peer-to-peer context, a new user can join the group at any time. When a new peer tries to join the group while its members are processing a garbage collection procedure, we may diverge if that user receives the current log before its deletion by all the members of the group. Hence, the new user is able to generate new operations based on a context completely different from other user contexts. Consequently, we inevitably diverge according to the scenario presented in Figure B.1.

To illustrate this, let us consider the scenario in Figure B.2 in which we have 3 sites collaborating together in order to edit the same document. The initial group is only composed of users s_1 and s_2. Then site s_3 decides to join the collaboration. Site s_3 requests the log and the state from site s_2. Concurrently, the site s_1 initiates a garbage collection and propagates the request to site s_2. The latter sends the log and the state to s_3 before receiving the request of the former. Then he decides to empty his log as well as site s_1 does. When s_3 is ready to join the group, he has a non empty log while others have empty ones. It is obvious that the context of s_3 is different from that of s_1 and s_2 which leads to divergence.

B.3 Garbage Collection Algorithm

In this section, we will discuss the concept of garbage and devise solutions for garbage collection. Each site maintains its own log which can be seen as a dependency tree using the semantic dependency relations (see Figure 7.1). The leaves of this tree represent a summary about what a site has received since an operation is causally ready only when its dependency is already executed. Let $\mathcal{L}$ be the set of leaves maintained by each site s. For instance, in the example of Figure 7.1, $\mathcal{L} = \{o_3, o_4\}$. Consequently, two sites having the same set of leaves means that they have executed the same set of operations and hence they have equivalent logs. This set of leaves is updated each time a local operation is generated or a distant one is received by replacing old leaves by new ones. The root of the dependency tree noted R

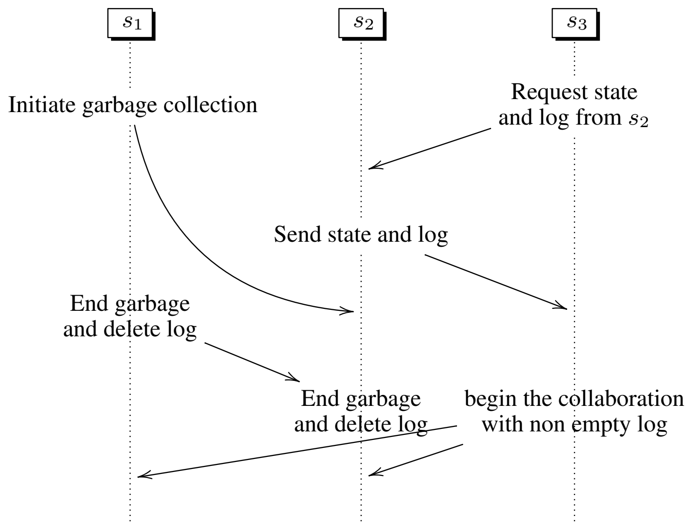

s_1
s_2
s_3
Initiate garbage collection
Request state and log from s_2
Send state and log
End garbage and delete log
End garbage and delete log
begin the collaboration with non empty log

Figure B.2: Divergence caused by a new user joining the group.

represents the identity of the site that performed the last the garbage collection procedure. The root has no effect on the state but rather serves as a break point indicating a garbage collection initiation. We assume that initially every site begins with an empty root.

It is obvious that the tree structure helps to draw the set of leaves and thus facilitates the comparison between different user logs. Consequently we reduce the size of parsed operations to decide whether the garbage procedure is allowed or not since we only compare the set of leaves rather than the entire logs. The tree structure allowing for leaves comparison is used in order to check whether all users have the same context or not thus we overcome the **First Issue** presented in Section B.2.

To proceed garbage collection, the collaborating sites exchange some garbage messages to decide whether it is possible to clean logs or not. In fact, it is not always possible to delete operations from the log. For instance, two sites having different sets of leaves are unable to clean their logs since we can deduce that there are operations in network not yet received by all sites. To do so, collaborating sites have to exchange the following messages in order to decide about the garbage:

- Garbage collection initiation (GCI): is a message sent by the site initiating the garbage collection procedure; the GCI contains the site identity as well as the set of leaves $\mathcal{L}$.
- Acquirement (ACK): when a site receives the GCI message, he computes the difference between his set of leaves $\mathcal{L}$ and that received in the GCI, then sends the result to the initiator.
- Garbage collection order (GCO): this message contains the root of the new empty tree that will replace the old one. Thus, when it is received from the initiator, it leads to the removal of the log.

A user can have different three states according to the collaboration state: *blocked*, *active* and *passive*. He is blocked when a garbage collection procedure is processing and as soon as the garbage ends he turns to the active state. Otherwise, the user is passive (this is the state when a user joins the group).

In Algorithm 16, we give all the steps of the control concurrency algorithm taking into account the garbage collection messages.

When a user generates a local operation, he invokes the function that will integrate remote operations (INTEGRATE_LOCAL_OPERATION). When a user decides to initiate a garbage collection, he processes the LUNCH_GC procedure. Now, when he receives a garbage collection message, he calls the RECEIVE_GCI_MESSAGE (m) procedure detailed in Algorithm 21 where we can see that according to the type of the received message (GCI, ACK or GCO) the user will apply the corresponding processing. If the message received is a GCI, the user invokes the RECEIVE_GCI_MESSAGE (m) procedure. To summarize, the garbage collection scheme proceeds in five steps:

1. The garbage collection initiating site stops the local generation of the operations to turn into blocked state, and sends the GCI message to the rest of the group. At the mean time, the initiator continues the integration of remote operations.

2. When receiving a GCI message, each site stops the local generation of operations, and checks if the operations contained in the GCI leaves have already been executed or not. To check it, we simply compute the difference between the receiver set of leaves and the GCI one. The resulting set is returned to the initiator in an ACK message. The difference represents the leaves executed by the site and not yet seen by the initiator site.

3. Each time the initiator receives an ACK message, he stores the difference locally in his own list of waited operations $\mathcal{W}$. This list is updated every time he receives one of the waited operations by extracting this operation from the set. In other words, an ACK message is causally ready when all leaves it contains are locally executed.

4. The initiator remains blocked until all ACK are received and all waited operations are executed locally. Note that waited ACK concerns only connected peers that were discovered initially by the initiator and are continuing the collaboration. New peers or disconnected peers are ignored. Thus we ensure that the initiator will not wait indefinitely for ACKs and overcome the **Second Issue**. When all waited ACK are received and $\mathcal{W} = \emptyset$, the initiator destroys his log and sends the garbage order GCO to all the group (see Algorithm 22).

5. When receiving the GCO message, a site executes il when it is causally ready. A GCO message is causally ready when all operations in its leaves set are already executed at the receiver site. Otherwise, the GCO is not ready and thus the log removal could not be executed. When the GCO message is causally ready, we verify whether the set of leaves that it contains is equal to the local set of leaves. If not, the user is ignored and considered as a new user in the group (he must request the log from other users). Otherwise, the site deletes its local log and start again the local generation with an empty log containing the same root received in the GCO.

Note that each collaborating site can generate a garbage collection at any time. Hence, it is possible, that two or more users initiate two garbage procedures concurrently. To address this case, we associate a unique identifier that is randomly generated in order to ensure fairness and offer to all users the same opportunities to initiate a garbage collection procedure. We assume that these identifiers are totally ordered according to their priorities. The initiator identifier is sent in the GCI. If an initiator receives a remote GCI, he just compares his identifier to that received in the GCI message (see Algorithm 21). If he has the high priority, he continues his garbage otherwise, he stops the local garbage procedure and respond to the initiator by an ACK messages as normal users (see Algorithm 22).

As our algorithm meets peer-to peer networks, users can join the group at any time. The question that arises here is: what to do when a new user joins the group during a garbage collection procedure? In fact, as discussed in Section B.2, this situation can lead to replicas divergence if the user that receives the new one request for log and state has not yet received the GCI message. To avoid the **Third Issue** mentioned in section B.2, we simply force the new peer to wait θ time before requesting the log from the nearest peer. The time θ corresponds to the maximal bound needed by a message to traverse the network from any sender to its receiver. In our model, we consider that θ is a parameter depending on network configurations. Consequently, the new users should follow the following steps. First, he sets his state to passive and waits θ time then sends a request to the nearest peer in order to get the log and the state. If the user who receives this request is in a blocked state, then the requestor must wait until the requested user is unblocked. If it is not the case ($i.e$ the requested peer is in active state) he sends his log and state to the requestor. Otherwise, he waits the reception of the GCO message and then responds (see Algorithm 20).

```
 1: Main:
 2: JOIN
 3: INITIALIZATION
 4: while not aborted do
 5:     if there is an input message $m$ then
 6:         GENERATE_MESSAGE($m$)
 7:     else
 8:         RECEIVE_MESSAGE
 9:     end if
10: end while
11: INITIALIZATION:
12: $state \leftarrow active$
13: $R \leftarrow$ " "
14: $\mathcal{W} \leftarrow \emptyset$
15: $\mathcal{L} \leftarrow \emptyset$
16: $s \leftarrow$ Identification of local user
17: $initiator \leftarrow false$
18: GENERATE_MESSAGE($m$):
19: if m is an operation then
20:     INTEGRATE_LOCAL_OPERATION
21: else
22:     if m is a GCI then
23:         LUNCH_GC()
24:     end if
25: end if
26: RECEIVE_MESSAGE:
27: if $m$ is an operation then
28:     if $m \in \mathcal{W}$ then
29:         $\mathcal{W} \leftarrow \mathcal{W} - m$
30:     end if
31:     INTEGRATE_REMOTE_OPERATION
32: else
33:     if $m$ is a log request then
34:         if state=active then
35:             SEND_LOG
36:         end if
37:     end if
38: else
39:     RECEIVE_GCI_MESSAGE ($m$)
40: end if
41: JOIN:
42: $state \leftarrow passive$
43: wait $\theta$
44: send a request log message
45: wait until receiving log
46: for all operation in the log do
47:     INTEGRATE_REMOTE_OPERATION
48: end for
```

Algorithm 20: Control Concurrency Algorithm with Garbage Collection Scheme

```
 1: RECEIVE_GC_MESSAGE ($m$)
 2: if $m = GCI(s', \mathcal{L}'_s)$ then
 3:     if $s < s'$ and $initiator = true$ then
 4:         $initiator \leftarrow false$ {Abort garbage collection initiation}
 5:     end if
 6:     $state \leftarrow blocked$
 7:     $\mathcal{L}_r \leftarrow \mathcal{L} \setminus \mathcal{L}_s$
 8:     send $ACK(\mathcal{L}_r)$
 9: else
10:     if $m = ACK(\mathcal{L}', s')$ and $initiator = true$ then
11:         $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{L}'$
12:         $\mathcal{D} \leftarrow \mathcal{D} \setminus \{s'\}$
13:     end if
14: else
15:     if $m = GCO(s', \mathcal{L}')$ then
16:         if GCO is causally ready and $\mathcal{L}' = \mathcal{L}$
17:         clean log
18:         $R \leftarrow s'$
19:         $state \leftarrow active$
20:     end if
21: end if
```

Algorithm 21: Receive Garbage Message Procedure

```
 1: LUNCH_GC()
 2: $initiator \leftarrow true$
 3: $state \leftarrow blocked$
 4: $\mathcal{D} \leftarrow OnlinePeers()$
 5: send GCI($s$,$\mathcal{L}$)
 6: wait until $\mathcal{D} \cap OnlinePeers() = \emptyset$ and $\mathcal{W} = \emptyset$
 7: send GCO($s$,$\mathcal{L}$)
 8: $initiator \leftarrow false$
 9: clean log
10: $state \leftarrow active$
```

Algorithm 22: Lunch Garbage Collection Procedure

B.4 Illustrative Examples

Example B.4.1. *To illustrate the garbage collection scheme, consider the scenario in Figure B.3. In this Figure, we consider three collaborating sites s_1, s_2 and s_3 where s_1 decides to initiate a garbage collection. The set of leaves of each site are referred to as $\mathcal{L}_{s_1}$, $\mathcal{L}_{s_2}$ and $\mathcal{L}_{s_3}$ respectively. To initiate the* *garbage collection, site s_1 stops the local generation of operations, and sends to other sites a garbage collection message $GCI(s_1, \mathcal{L}_{s_1})$ containing his set of leaves $\mathcal{L}_{s_1}$. When the GCI message arrives at sites s_2 and s_3, each site computes the difference between the received set of leaves $\mathcal{L}_{s_1}$ and the local one ($\mathcal{L}_{s_2}$ for s_2 and $\mathcal{L}_{s_3}$ for s_3). Then the resulting set is sent to the initiator of GCI (s_1) through the acquirement message ACK. The set of leaves sent in ACK message by s_2 and s_3 are added to the set of waited operations ($\mathcal{W}$) by site s_1. This list is updated every time a remote operation is received (by removing the received operation from $\mathcal{W}$). The group still blocked until all ACK messages are received by s_1 and his $\mathcal{W}$ is empty ($i.e.$ all waited operations are locally executed). Then s_1 cleans the log, adds his identity as the root the new root of the dependency tree, and sends the GCO containing the cleaning order to other sites of the group (s_1 and s_2). After the reception of the GCO message, the three sites starts again the local generation with empty logs.*

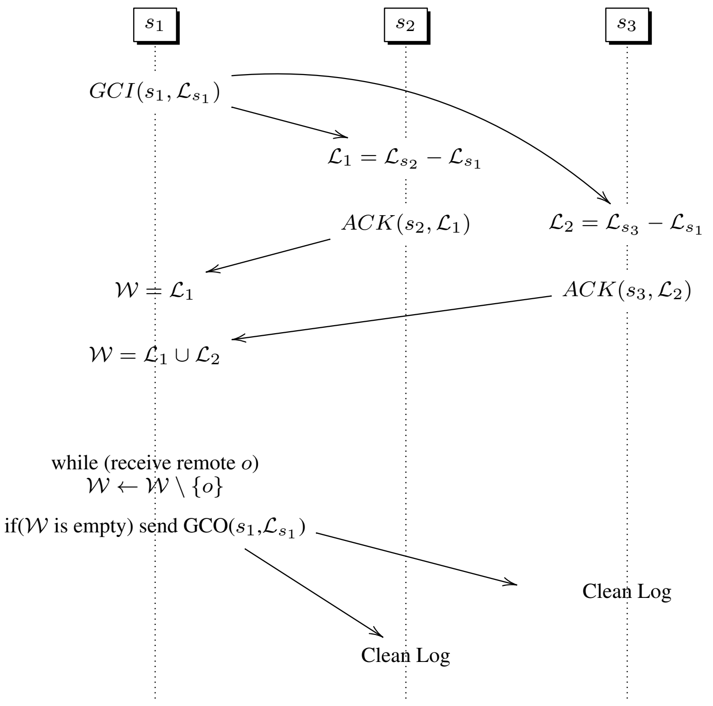

s_1
s_2
s_3
$GCI(s_1, \mathcal{L}_{s_1})$
$\mathcal{L}_1 = \mathcal{L}_{s_2} - \mathcal{L}_{s_1}$
$ACK(s_2, \mathcal{L}_1)$
$\mathcal{L}_2 = \mathcal{L}_{s_3} - \mathcal{L}_{s_1}$
$\mathcal{W} = \mathcal{L}_1$
$ACK(s_3, \mathcal{L}_2)$
$\mathcal{W} = \mathcal{L}_1 \cup \mathcal{L}_2$
while (receive remote o)
$\mathcal{W} \leftarrow \mathcal{W} \setminus \{o\}$
if($\mathcal{W}$ is empty) send GCO(s_1,$\mathcal{L}_{s_1}$)
Clean Log
Clean Log

Figure B.3: Garbage collection scenario 1.

Example B.4.2. *To illustrate how we proceed garbage collection in the case of slow or non responding peers, let us consider the scenario illustrated in Figure B.4 where we have three collaborating sites s_1, s_2 and s_3. Let $\mathcal{L}_{s_1}$, $\mathcal{L}_{s_2}$ and $\mathcal{L}_{s_3}$ be the set of leaves of s_1, s_2 and s_3 respectively. Suppose that site s_1 initiates the garbage collection. So, he stops the local generation of operations, and sends to other sites a garbage collection message $GCI(s_1, \mathcal{L}_{s_1})$ containing his set of leaves $\mathcal{L}_{s_1}$. When the GCI message arrives at sites s_2 and s_3, each site computes the difference between the received set of leaves $\mathcal{L}_{s_1}$ and the local one ($\mathcal{L}_{s_2}$ for s_2 and $\mathcal{L}_{s_3}$ for s_3). Suppose that $\mathcal{L}_{s_1} = \emptyset$ which means that s_2 has the same set of leaves as s_1 and that $\mathcal{L}_{s_3} \neq \emptyset$. Moreover, s_3 is a slow peer, thus his ACK is not received by s_1. When s_1 receives s_2's ACK, he rediscovers connected peers and finds that s_3 does not respond. Consequently, he only consider the ACK of s_2. Since $\mathcal{W} = \emptyset$, s_1 sends the GCO message. When received by s_2, he deletes his log since $\mathcal{L}_1 = \mathcal{L}_2$. However, at site s_3, the log removal is not executed until all s_1 leaves are executed at site s_3 and $\mathcal{L}_{s_3} = \mathcal{L}_{s_1}$ to be sure that all operations seen at site s_1 when initiating the GC are received by s_3. It should be noted that if site s_3 has generated an operation concurrently to the* *garbage collection procedure ($\mathcal{L}_{s_3} \neq \mathcal{L}_{s_1}$), this operation will be lost since the set of leaves is different from that of the initiator. Consequently, the user will be rejected from the garbage collection procedure and considered as a new user who joins the group thus needing to request the log and the state after the garbage ends. This issue does not concern local networks since the loss of messages is not frequent but may be faced in large networks such as Internet.*

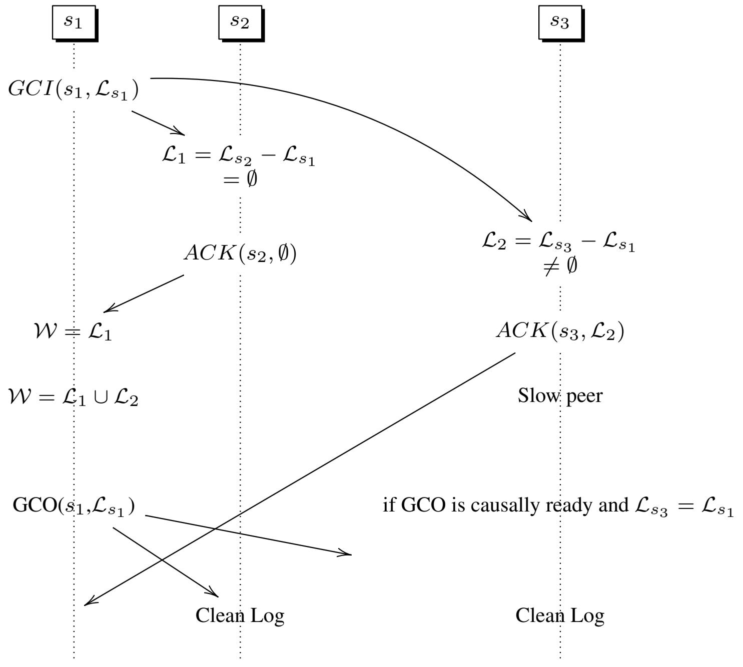

s_1
s_2
s_3
$GCI(s_1, \mathcal{L}_{s_1})$
$\mathcal{L}_1 = \mathcal{L}_{s_2} - \mathcal{L}_{s_1} = \emptyset$
$\mathcal{L}_2 = \mathcal{L}_{s_3} - \mathcal{L}_{s_1} \neq \emptyset$
$ACK(s_2, \emptyset)$
$\mathcal{W} = \mathcal{L}_1$
$ACK(s_3, \mathcal{L}_2)$
$\mathcal{W} = \mathcal{L}_1 \cup \mathcal{L}_2$
Slow peer
GCO(s_1,$\mathcal{L}_{s_1}$)
if GCO is causally ready and $\mathcal{L}_{s_3} = \mathcal{L}_{s_1}$
Clean Log
Clean Log

Figure B.4: Garbage collection scenario 2 (case of a slow peer).

Bibliography

[1] http://medianet.kent.edu/surveys/iad04f-p2papplications-amit/p2pa.html.

[2] www.grid5000.fr.

[3] *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), 22-26 June 2009, Montreal, Québec, Canada*. IEEE Computer Society, 2009.

[4] B. A. and J. F. Barkley. Space: Spacial access control for collaborative virtual envoronments, 1998.

[5] G. D. Abowd and A. J. Dix. Giving undo attention. *Interact. Comput.*, 4(3):317–342, Dec. 1992.

[6] G.-J. Ahn and R. S. Sandhu. The rsl99 language for role-based separation of duty constraints. In *ACM Workshop on Role-Based Access Control*, pages 43–54, 1999.

[7] J. E. Archer, Jr. and R. W. Conway. Cope: A cooperative programming environment. Technical report, Ithaca, NY, USA, 1981.

[8] V. Atluri and W.-k. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security*, ESORICS '96, pages 44–64, London, UK, 1996. Springer-Verlag.

[9] R. M. Baecker, J. Grudin, W. Buxton, and S. Greenberg. *Readings in Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann, Jan. 1995.

[10] M. Y. Becker, C. Fournet, and A. D. Gordon. Secpal: Design and semantics of a decentralized authorization language. Technical report, In Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF, 2006.

[11] K. Berket, A. Essiari, and A. Muratas. Pki-based security for peer-to-peer information sharing, 2004.

[12] T. Berlage. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Trans. Comput.-Hum. Interact.*, 1(3):269–294, Sept. 1994.

[13] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. A decentralized temporal autoritzation model. In *SEC*, pages 271–280, 1996.

[14] E. Bertino, P. A. Bonatti, and E. Ferrari. A generalized temporal role-based access control model. *Ieee Transactions on Knowledge and Data Engineering*, pages 4–23, 2005S.

[15] E. Bertino, S. Castano, and E. Ferrari. Securing xml documents: the author-x project demonstration. In *SIGMOD Conference*, page 605, 2001.

[16] L. Bouganim, F. D. Ngoc, and P. Pucheral. Client-based access control management for xml documents. In *VLDB*, pages 84–95, 2004.

[17] A. Bullock and S. Benford. An access control framework for multi-user collaborative environments. In *GROUP '99*, pages 140–149, New York, NY, USA, 1999. ACM.

[18] P. Cederqvist. *Version Management with CVS*. Network Theory Ltd., Decembre 2002.

[19] A. Cherif. Du Contrôle d'Accès Dynamique pour les Editeurs Collaboratifs. Mémoire de Master Recherche, LORIA, Université Henri Poincaré, Nancy, 2008.

[20] A. Cherif and A. Imine. Undo-based access control for distributed collaborative editors. In *CDVE*, pages 101–108, 2009.

[21] A. Cherif, A. Imine, and M. Rusinowitch. Optimistic access control for distributed collaborative editors. In *SAC*, pages 861–868, 2011.

[22] R. Choudhary and P. Dewan. A general multi-user undo/redo model. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work*, pages 231–246, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[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*, SACMAT '01, pages 10–20, New York, NY, USA, 2001. ACM.

[24] B. Crispo, S. Sivasubramanian, P. Mazzoleni, and E. Bertino. P-hera: Scalable fine-grained access control for p2p infrastructures. *Parallel and Distributed Systems, International Conference on*, 1:585–591, 2005.

[25] J. F. da Silva, L. P. Gaspary, M. P. Barcellos, and A. Detsch. Policy-based access control in peer-to-peer grid systems. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 107–113, Washington, DC, USA, 2005. IEEE Computer Society.

[26] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

[27] P. Dewan and R. Choudhary. Primitives for programming multi-user interfaces. In *UIST*, pages 69–78, 1991.

[28] A. Dix, R. Mancini, and S. Levialdi. The cube - extending systems for undo. In *In Proc. of DSVIS'97*, pages 473–495. Eurographics, 1997.

[29] J. Douceur and J. S. Donath. The sybil attack. pages 251–260, 2002.

[30] W. K. Edwards. Policies and roles in collaborative applications. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, CSCW '96, pages 11–20, New York, NY, USA, 1996. ACM.

[31] W. K. Edwards. Coordination infrastructure in collaborative systems. In *Ph.D. dissertation*, November 22, 1995.

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

[33] Y. Elrakaiby, F. Cuppens, and N. Cuppens-Boulahia. Interactivity for reactive access control. In *SECRYPT*, pages 57–64, 2008.

[34] C. Erbas, S. Sarkeshik, and M. M. Tanik. Different perspectives of the n-queens problem. In *Proceedings of the 1992 ACM annual conference on Communications*, CSC '92, pages 99–108, New York, NY, USA, 1992. ACM.

[35] D. F. Ferraiolo and J. F. Barkley. Specifying and managing role-based access control within a corporate intranet. In *ACM Workshop on Role-Based Access Control*, pages 77–82, 1997.

[36] 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.

[37] 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, New York, NY, USA, 2001. ACM.

[38] G. S. Graham and P. J. Denning. Protection: principles and practice. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, AFIPS '72 (Spring), pages 417–429, New York, NY, USA, 1972. ACM.

[39] I. Greif and S. Sarin. Data sharing in group work. In *Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, CSCW '86, pages 175–183, New York, NY, USA, 1986. ACM.

[40] R. Guerraoui and C. Hari. On the consistency problem in mobile distributed computing. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 51–57. ACM Press, 2002.

[41] E. Halepovic and R. Deters. The jxta performance model and evaluation. *Future Generation Comp. Syst.*, 21(3):377–390, 2005.

[42] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19:461–471, August 1976.

[43] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The f accrual failure detector. In *SRDS*, pages 66–78, 2004.

[44] C.-T. V. Hu. *The policy machine for universal access control*. PhD thesis, Moscow, ID, USA, 2002. AAI3055379.

[45] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn. Assessment of access control systems. Technical Report 7316, 2006.

[46] C.-L. Ignat and M. C. Norrie. Customizable collaborative editor relying on treeopt algorithm. In *ECSCW*, pages 315–334, 2003.

[47] A. Imine. *Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair*. Phd thesis, University of Henri Poincaré, Nancy, France,, December 2006.

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

[49] A. Imine, A. Cherif, and M. Rusinowitch. A flexible access control model for distributed collaborative editors. In *Secure Data Management*, pages 89–106, 2009.

[50] 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.

[51] T. Jaeger and A. Prakash. Requirements of role-based access control for collaborative systems. In *ACM Workshop on Role-Based Access Control*, 1995.

[52] T. Jaeger and A. Prakash. Requirements of role-based access control for collaborative systems. In *RBAC '95*, page 16, New York, NY, USA, 1996. ACM.

[53] T. Jaeger and A. Prakash. Requirements of role-based access control for collaborative systems. In *Proceedings of the first ACM Workshop on Role-based access control*, RBAC '95, New York, NY, USA, 1996. ACM.

[54] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, SACMAT '01, pages 66–74, New York, NY, USA, 2001. ACM.

[55] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 113–122, New York, NY, USA, 2008. ACM.

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

[57] B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8:18–24, January 1974.

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

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

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

[61] 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.

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

[63] R. C. Mayer, J. H. Davis, and F. D. Schoorman. An Integrative Model of Organizational Trust. *The Academy of Management Review*, 20(3):709–734, 1995.

[64] M. D. Mechaoui, A. Cherif, A. Imine, and F. Bendella. Log garbage collector-based real time collaborative editor for mobile devices. In *CollaborateCom*, pages 1–10, 2010.

[65] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 898–909. VLDB Endowment, 2003.

[66] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the transformational approach to build a safe and generic data synchronizer. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 212–220. ACM Press, 2003.

[67] S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Comput. Supported Coop. Work*, 13(1):63–89, Jan. 2004.

[68] D. A. Norman and S. W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1986.

[69] S. L. Osborn, R. S. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[70] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *CSCW*, pages 259–268, 2006.

[71] J. S. Park, R. Sandhu, and G.-J. Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, 4:37–71, February 2001.

[72] B. C. Pierce and J. Vouillon. What's in unison? a formal specification and reference implementation of a file synchronizer. Technical report, 2004.

[73] 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.

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

[75] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans. Database Syst.*, 16(1):88–131, 1991.

[76] I. Ray and T. Xin. Concurrent and real-time update of access control policies. In *DEXA*, pages 330–339, 2003.

[77] 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.

[78] 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.

[79] P. Samarati, P. Ammann, and S. Jajodia. Maintaining replicated authorizations in distributed database systems. *Data Knowl. Eng.*, 18(1):55–84, 1996.

[80] P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In *FOSAD*, pages 137–196, 2000.

[81] P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In *FOSAD*, pages 137–196, 2000.

[82] R. Sandhu and P. Samarati. Access control: Principles and practice. In *IEEE Communications*, pages 40–48, 1994.

[83] R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 147–158, New York, NY, USA, 2005. ACM.

[84] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[85] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[86] R. S. Sandhu, D. F. Ferraiolo, and D. R. Kuhn. The nist model for role-based access control: towards a unified standard. In *ACM Workshop on Role-Based Access Control*, pages 47–63, 2000.

[87] R. S. Sandhu and R. K. Thomas. Conceptual foundations for a model of task-based authorizations. In *CSFW*, pages 66–79, 1994.

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

[89] H. Shen and P. Dewan. Access control for collaborative environments. In *CSCW '92*, pages 51–58, New York, NY, USA, 1992. ACM.

[90] B. SHNEIDERMAN. The future of interactive systems and the emergence of direct manipulation? *Behaviour & Information Technology*, 1(3):237–256, 1982.

[91] M. Sohlenkamp and G. Chwelos. Integrating communication, cooperation, and awareness: the diva virtual office environment. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 331–343, New York, NY, USA, 1994. ACM.

[92] R. Stallman. Gnu emacs manual. Technical report, 2007.

[93] C. Sturm, K. R. Dittrich, and P. Ziegler. An access control mechanism for p2p collaborations. In *Proceedings of the 2008 international workshop on Data management in peer-to-peer systems*, DaMaP '08, pages 51–58, New York, NY, USA, 2008. ACM.

[94] 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.

[95] M. Suleiman, M. Cart, and J. Ferrié. Concurrent operations in a distributed and mobile collaborative environment. In *IEEE ICDE'98*, pages 36–45, 1998.

[96] C. Sun. Undo any operation at any time in group editors. In *In Computer-Supported Cooperative Work (CSCW*, pages 191–200, 2000.

[97] C. Sun. Undo as concurrent inverse in group editors. *ACM Trans. Comput.-Hum. Interact.*, 9(4):309–361, 2002.

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

[99] 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.

[100] 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.

[101] D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(10):1454–1470, 2009.

[102] D. Sun, S. Xia, C. Sun, and D. Chen. Operational transformation for collaborative word processing. In *CSCW*, pages 437–446, 2004.

[103] S. G. Tammaro, J. N. Mosier, N. C. Goodwin, and G. Spitz. Collaborative writing is hard to support: A field study of collaborative writing. *Comput. Supported Coop. Work*, 6(1):19–51, apr 1997.

[104] W. Teitelman. Interlisp reference manual, 1978.

[105] 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, New York, NY, USA, 1997. ACM.

[106] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented authorization management. In *In Proceedings of the IFIP WG11.3 Workshop on Database Security, Lake Tahoe*, pages 166–181, 1997.

[107] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37:29–41, March 2005.

[108] E. Tsang. Foundations of constraint satisfaction, 1993.

[109] 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.

[110] J. S. Vitter. User: A new framework for redoing. In *Software Development Environments (SDE)*, pages 168–176, 1984.

[111] W. Wang. Team-and-role-based organizational context and access control for cooperative hypermedia environments. In *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots: returning to our diverse roots*, HYPERTEXT '99, pages 37–46, New York, NY, USA, 1999. ACM.

[112] R. W.Baldwin. Naming and grouping privileges to simplify security management in large database. In *IEEE Computer Society Syposium on Research in Security and Privacy*, pages 71–60, 1990.

[113] S. Weiss, P. Urso, and P. Molli. Compensation in Collaborative Editing. Rapport de recherche RR-6160, INRIA, 2007.

[114] S. Weiss, P. Urso, and P. Molli. An Undo Framework for P2P Collaborative Editing. In Elisa Bertino and James B.D. Joshi, editors, *4th International Conference on Collaborative Computing : Networking, Applications and Worksharing - CollaborateCom*, volume 10, pages 529–544, Orlando États-Unis, 11 2008. Springer Berlin Heidelberg.

[115] T. Wobber, T. L. Rodeheffer, and D. B. Terry. Policy-based access control for weakly consistent replication. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 293–306, New York, NY, USA, 2010. ACM.

[116] T. Xin and I. Ray. A lattice-based approach for updating access control policies in real-time. *Inf. Syst.*, 32(5):755–772, 2007.

[117] T. Xin and I. Ray. A lattice-based approach for updating access control policies in real-time. *Inf. Syst.*, 32:755–772, July 2007.

[118] G. Zhang and M. Parashar. Context-aware dynamic access control for pervasive applications. In *In Communication Networks and Distributed Systems Modeling and Simulation Conference.*, 2004.

Abstract

The importance of collaborative systems in real-world applications has grown significantly over the recent years. The majority of new applications are designed in a distributed fashion to meet collaborative work requirements. Among these applications, we focus on Real-Time Collaborative Editors (RCE) that provide computer support for modifying simultaneously shared documents, such as articles, wiki pages and programming source code by dispersed users. Although such applications are more and more used into many fields, the lack of an adequate access control concept is still limiting their full potential. In fact, controlling access in a decentralized fashion for such systems is a challenging problem, as they need dynamic access changes and low latency access to shared documents. In this thesis, we propose a generic access control model based on replicating the shared document and its authorization policy at the local memory of each user. We consider the propagation of authorizations and their interactions. We propose a optimistic approach to enforce access control in existing collaborative editing solutions in the sense that a user can temporarily violate the access control policy. To enforce the policy, we resort to the selective undo approach in order to eliminate the effect of illegal document updates. Since, the safe undo is an open issue in collaborative applications. We investigate a theoretical study of the undo problem and propose a generic solution for selectively undoing operations. Finally, we apply our framework on a collaboration prototype and measure its performance in the distributed grid GRID'5000 to highlight the scalability of our solution.

Keywords: Access Control, Collaborative Editors, Operational Transformation Approach (OT), Selective Undo.

Résumé

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. Parmi ces applications, nous nous intéressons aux éditeurs collaboratifs temps-réel (RCE) qui permettent la manipulation de divers objets partagés, tels que les pages wiki ou les articles scientifiques par plusieurs personnes réparties dans le temps et dans l'espace. Bien que ces applications sont de plus en plus utilisées dans de nombreux domaines, l'absence d'un modèle de contrôle d'accès adéquat limite l'exploitation de leur plein potentiel. En effet, contrôler les accès aux documents partagés de façon décentralisée et sans alourdir les performances du système collaboratif représente un vrai challenge, surtout que les droits d'accès peuvent changer fréquemment et de façon dynamique au cours du temps. Dans cette thèse, nous proposons un modèle de contrôle d'accès générique basé sur l'approche de réplication optimiste du document partagé ainsi que sa politique de contrôle d'accès. Pour cela, nous proposons une approche optimiste de contrôle d'accès dans la mesure où un utilisateur peut violer temporairement la politique de sécurité. Pour assurer la convergence, nous faisons recours à l'annulation sélective pour éliminer l'effet des mises à jour illégales. Vu l'absence d'une solution d'annulation générique et correcte, nous proposons une étude théorique du problème d'annulation et nous concevons une solution générique basée sur une nouvelle sémantique de l'opération identité. Afin de valider notre approche tous nos algorithmes ont été implémentés en Java et testés sur la plateforme distribuée Grid'5000.

Mots-clés: Contrôle d'Accès, Editeurs Collaboratifs, Transformée Opérationnelle, Annulation Sélective.