

A Contract-based and Trust-aware Collaboration Model

Hien Thi Thu Truong

▶ **To cite this version:**

Hien Thi Thu Truong. A Contract-based and Trust-aware Collaboration Model. Distributed, Parallel, and Cluster Computing [cs.DC]. Université de Lorraine, 2012. English. NNT : 2012LORR0181 . tel-01749599v2

HAL Id: tel-01749599
https://theses.hal.science/tel-01749599v2

Submitted on 28 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE LORRAINE

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
Collegium Sciences & Technologies

Un modèle de collaboration basé sur les contrats et la confiance

A Contract-based and Trust-aware Collaboration Model

THÈSE

présentée et soutenue publiquement le 11th Decembre 2012

pour l'obtention du

Doctorat de l'Université de Lorraine

(spécialité informatique)

par

Hien Thi Thu Truong
(Trương Thị Thu Hiền)

Composition du jury

Président :	Isabelle Chrisment	Professeur, Université de Lorraine
Rapporteurs :	Alexander Pretschner	Professeur, Université de Munich
	Esther Pacitti	Professeur, Université Montpellier 2
Examinateur :	Luc Bouganim	Directeur de recherche, Inria Paris-Rocquencourt
Directeurs de thèse :	Pascal Molli	Professeur, Université de Nantes
	Claudia-Lavinia Ignat	Chargée de recherche, Inria Nancy-Grand Est

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

Inria INVENTEURS DU MONDE NUMÉRIQUE
Loria Laboratoire lorrain de recherche en informatique et ses applications

Mis en page avec la classe thloria.

Abstract

Nowadays, information technologies provide users ability to work with anyone, at any time, from everywhere and with several heterogeneous devices. This evolution fosters a new distributed trustworthy collaboration model where users can work on shared documents with whom they trust. Multi-synchronous collaboration is widely used for supporting collaborative work by maintaining simultaneous streams of user activities which continually diverge and converge. However, this model lacks support on how usage restrictions on data can be expressed and checked within the model. This thesis proposes "C-PPC", a multi-synchronous contract-based and trust-aware collaboration model. In this model, contracts are used as usage rules and users collaborate according to trust levels they have on others computed according to contract compliance. We formalize contracts by using deontic concepts: permission, obligation and prohibition. Contracts are enclosed in logs of operations over shared data. The C-PPC model provides a mechanism for merging data changes and contracts. Any user can audit logs at any time and auditing results are used to update user trust levels based on a trust metric. We propose a solution relying on hash-chain based authenticators that ensures integrity of logs and user accountability. We provide algorithms for constructing authenticators and verifying logs and prove their correctness. A set of experimental results shows the feasibility of the C-PPC model.

To mom and dad.

Acknowledgements

This thesis would not be possible without my advisors Pascal Molli and Claudia-Lavinia Ignat during my Ph.D journey at Université de Lorraine. Claudia was a brilliant advisor who gave me an interesting topic to work on. When I arrived Inria at Nancy without a clue what I was going to do in the domain of distributed collaborative systems, she provided me everything I would need to succeed such as support and guidance with her insight and tremendous patience especially during long nights close to deadlines. Pascal gave me his support by spending his valuable time for insightful discussions from distance. He provided me his superior practical guidance which has enabled me to grow in thinking about problems and to write clearly about solutions. It has been a great privilege to have both of them as my advisors. They gave me the freedom to explore and encouraged me to take the risks required for true innovation. I am thankful for having the chance to work closely with them over the past three years.

I would also like to thank my Ph.D committe members, Isabelle Chrisment, Alexander Pretschner, Esther Pacitti, and Luc Bouganim, for their insights, questions, and advice for my thesis. I additionally want to thank Alexander Pretschner for his invitation for presenting and discussing the work with his research team in Karlsruhe, Germany; and for his perceptive sugesstions regarding obligations in distributed usage control.

I was fortunate to receive Inria CORDI fellowship. I am thankful the finance support which helped me to focus on my Ph.D work without teaching responsibilites over limitted time within three years. I also thank SCORE team that included me in the group, supported my work and gave me finance support for last several months to finish my Ph.D. I refer SCORE as a collection of current and former people that I have had chance to interact with, François Charoy, Claude Godart, Khalid Benali, Nacer Boudjlida, Olivier Perrin, Pascal Urso, Gérald Oster, Samir Youcef, Karim Dahmen, Aymen Baouab, Mehdi Ahmed-Nacer, Luc André, Stéphane Martin, Hyun-Gul Roh, Charbel Rahhal, Mohamed-Rafik Bouguelia and other individuals that helped me along the way. I additionally thank Gérald Oster for his kindly help on technical and many unexpected problems regarding my working life.

Good friends are difficult to come by and I want to thank my lab-mates at Loria center, Thi Oanh Nguyen, Thanh Phuong Nguyen, Thai Hoang, Hong Phuong Le, K.C. Santosh, Makoto

Hasegawa, Laura Perez and many others, for providing me very valuable information regarding working and administrative stuffs. In addition, I would like to thank my Vietnamese friends in Nancy, Phuc Lam Dao, Thanh Tuan Lam, Dieu Linh Le, Hoang Bao Truc Khuat, Duc Trung Nguyen and many others for sharing with me their living-abroad experiences that I cannot learn from the class.

A special thanks to Stéphane Weiss for his unconditional support, care and encouragement. I am thankful that he spent a lot of time for useful discussions around my work and shared with me stress moments that are unavoidable for everyone in Ph.D work. This made my Ph.D journey more manageable.

Last, but definitely not the least, I would also like to thank my entire family for their love and support, most especially to my parents for being patient and understanding for my rare visits during Ph.D time. This thesis is dedicated to them.

Contents

Abstract **i**

List of Figures **xi**

1 Introduction **1**

1.1 Challenges 1

1.2 Thesis Contributions 4

1.2.1 Contract-based Multi-Synchronous Collaboration Model 4

1.2.2 Log Auditing and Trust Assessment 5

1.2.3 Authenticators 6

1.3 Thesis Outline 6

2 Background **9**

2.1 Multi-Synchronous Model 9

2.1.1 Synchronous, Asynchronous and Multi-Synchronous Distinction 10

2.1.2 Example 1: Collaborative Writing 12

2.1.3 Example 2: Source Code Management 13

2.2 Overview of Trust 14

2.2.1 Trust Definition 14

2.2.2 An Example of Trust 16

2.2.3 Trust Model 17

2.3 Overview of Contract 20

2.3.1 Contract Definition 21

2.3.2 Example 1: Free Software Licenses 23

2.3.3 Example 2: Wikipedia Editorial Policies 24

2.3.4 Example 3: Policy in P2P File Sharing Systems 24

2.4 Trust and Contract Interrelationship 24

2.5 A Motivating Example 25

2.5.1 Photo Sharing Issues . . . 26
2.5.2 A User Study . . . 27
2.5.3 A Scenario . . . 28

3 State of the Art **31**
3.1 Contract-based Systems . . . 31
3.2 Access Control Models . . . 33
3.2.1 Role-based Access Control . . . 34
3.2.2 Attribute-based Access Control . . . 35
3.2.3 Optimistic Access Control . . . 38
3.2.4 Computer-Supported Access Control . . . 40
3.3 Usage Control Models . . . 41
3.4 A Posteriori Compliance Checking . . . 43
3.5 Summary and Discussion . . . 45

4 C-PPC Model **47**
4.1 Push-Pull-Clone Paradigm . . . 47
4.2 Optimistic Replication . . . 51
4.2.1 State-based and Operation-based Replication . . . 52
4.2.2 Eventual Consistency Guarantee . . . 52
4.3 C-PPC Model through An Example . . . 53
4.4 The C-PPC Model . . . 56
4.4.1 Model Overview . . . 56
4.4.2 Document, Changes and Logs . . . 58
4.4.3 Contract Formalization . . . 59
4.4.4 Contract Conflict . . . 65
4.4.5 Conflict Resolution . . . 68
4.4.6 Ordering Contracts . . . 69
4.4.7 Repealing Contracts . . . 71
4.5 Collaborative Process . . . 72
4.5.1 Logging Changes . . . 72
4.5.2 Pushing Logs containing Contracts . . . 76
4.5.3 Pulling and Merging Pairwise Logs . . . 77
4.6 Correctness of C-PPC Model . . . 80
4.7 Experiment . . . 81
4.8 Discussion and Summary . . . 83

5 Log Auditing and Trust Assessment **87**
5.1 Auditing Principles . . . 87
5.1.1 Contract Violation . . . 88
5.1.2 Trust Metric . . . 89
5.2 Log Auditing and Trust Assessment . . . 92
5.3 Experiment . . . 94
5.4 Related Work . . . 96
5.5 Summary and Discussion . . . 99

6 Authenticators **101**
6.1 Security Aspects . . . 102
6.1.1 Threat Model . . . 102
6.1.2 Desirable Properties . . . 103
6.2 Authenticator . . . 104
6.2.1 Definition . . . 105
6.2.2 Example . . . 107
6.3 Algorithms . . . 110
6.3.1 Authenticators Construction . . . 110
6.3.2 Authenticators-based Log Verification . . . 112
6.4 Proofs of Correctness . . . 114
6.5 Experiment . . . 117
6.6 Related Work . . . 119
6.7 Summary and Discussion . . . 122

7 Conclusion **123**
7.1 Outcomes . . . 124
7.2 Outlook . . . 126
7.3 Closing Words . . . 129

List of Publications **131**

Bibliography **133**

List of Figures

2.1 Classification of collaboration models 10

2.2 Deontic concepts. 23

2.3 Trust and contract spiral upwards. 25

2.4 Building cooperatively a photo collection 29

4.1 Push-Pull-Clone paradigm between three users. 49

4.2 Push-Pull-Clone paradigm from a single user's view. 50

4.3 A Push-Pull-Clone collaboration scenario of four users over a photo during time interval [*Time 1*,*Time 2*]. 54

4.4 Trust values at two different time points *Time 1* and *Time 2*. 55

4.5 Workflow in C-PPC model. 57

4.6 An example of log with one event in XML format. 59

4.7 An example of log containing contract events. 63

4.8 Three types of inconsistencies. 65

4.9 Deontic square of opposition. 67

4.10 A scenario when a user adopts two contracts that are inconsistent. The user holds Contract 1 and a coming one Contract 2. The conflict between O_{op_2} and F_{op_2} needs to be resolved to proceed further actions. 69

4.11 Causal relations between events (w_i represents for *write* events and c_i represents for *contract* or *communication* events). 74

4.12 An example of history and logs 76

4.13 An example of Summary Vector 79

4.14 Synchronization time with growing of number of *write* events 83

5.1 An example for current trust computed from the variations of number of bad events and unknown events that are found after auditing, $0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$. . 91
5.2 Ability to detect one selected misbehaving user with respect to the total number of interactions in a collaborative network of 200 users with 30% of them are misbehaving users. 95
5.3 Percentage of detected misbehaving users with respect to the number of synchronizations done by selected honest user. 95
5.4 Average percentage of detected misbehaving users with respect to to the total number of interactions in the collaborative network. 96

6.1 Structure of an authenticator. 106
6.2 Example of constructing authenticators. 108
6.3 Time overhead to check authenticators created for Hgview and OpenJDK repositories. 118

Chapter 1

Introduction

Collaboration is undoubtedly a good thing for society. It was addressed earlier by researchers in various fields such as psychologists, anthropologists, sociologists and economists. Research for collaboration between a large number of users has emerged for years in the domain of Computer Supported Cooperative Work (CSCW). Many real-world systems are built from the collaboration of a huge number of users such as the biggest free encyclopedia Wikipedia and open source code projects (e.g. Linux kernel project). Moreover, with the fast-growing of new information technologies, today users can work with everyone, from everywhere, at every time and with heterogeneous devices. These advantages foster a new model of collaboration that is distributed, open and decentralized. Within this model, collaborative works are built and maintained by a large and diverse community of users. Users can work independently with different streams of activity on the same replicated data. These divergent streams converge when users synchronize their streams together. This model of work is called multi-synchronous [58]. The multi-synchronous model relaxes the necessity of having a central authority which maintains collaboration among users. Users can start and maintain the collaborative work themselves.

1.1 Challenges

The goal of this thesis is to address two challenges concerning usage control and trust management in multi-synchronous collaboration. The main challenge is about usage control over personal data after it has been given away. This is a very important aspect for collaborative systems

where many users work together on the same shared data and misuse easily occur.

Nowadays people are losing their control over their personal data when they use centralized systems such as centralized social networks (e.g. Google Plus, Facebook, Twitter). It is very difficult for users to ensure that the content such as photos of family, comments, notes, friend lists they put to those centralized social networks is deleted completely when they want. In order to overcome this Big Brother problem, some recent approaches (e.g. [32, 65, 55]) proposed moving away from a centralized authority based architecture toward a decentralized trusted architecture. With this architecture, users define their network of trust with people they trust and wish to collaborate. Furthermore, the decentralized architecture overcomes disadvantages of the centralized architecture by offering a good scalability, fault-tolerance and possibility of sharing costs of administration [157]. In addition, the decentralized architecture reflects society better than other types of computer architectures [41]. It is being better adapted to the way people think and to user needs for knowledge sharing. Also it provides users more freedom to interact with each other. In a distributed collaboration model, instead of the necessity of having a central authority which has access to all users personal data, control over data is given to individuals. Therefore, the risk of privacy breaches is reduced as only a part of the protected data in distributed network may be exposed at any time.

Multi-synchronous model can be implemented with both centralized and decentralized architectures. With this collaboration model, usage rules over data are understood implicitly in the sense that each user usually knows what she can do on shared data. However, when users have to deal with overwhelming tasks, they should be made aware of usage aspects concerning their tasks in collaboration to avoid unintentional mistake. This main challenge about data usage raises two questions: (i) *how usage rules (contracts) are expressed and managed in multi-synchronous models?* and (ii) *how to detect if users will or will not violate those contracts, especially when users try to hide their misuses?*

We refine two challenging questions into several scientific questions as follows.

- How contracts are expressed and managed for shared data in multi-synchronous model?
- How to merge data when it is associated with contracts?
- How to deal with contract conflicts that easily occur when users synchronize their works?

We made a huge number survey of prior works in the literature with regard to data usage control. We found that traditional access control models such as [164, 174, 59, 182, 75, 72, 42, 16, 98, 163, 36, 151, 33, 143, 174, 54, 53, 179, 138, 61] have not addressed data usage after data was released to users. Usage control, introduced by Park and Sandhu [138] and investigated further by Pretschner [146] and others, appears as an alternative that control how data will be used once accessed. Usage control is regarded as an extension of data protection beyond access control. Usage policies can be enforced by using a detective or a preventive enforcement [61]. Most of existing usage control mechanisms protect data by enforcing users to do right actions [146, 81, 147, 148]. Apart from these approaches, we adopt usage control with detective enforcement. We intend not to prevent users from doing contributions even if their actions might violate usage rules but to help users aware of rules they should respect during collaboration on shared.

Another challenge that we address concerns trust. In collaboration, trust plays role as a part in the initiation and maintenance of cooperative work among participants. Trust can help independent individuals in their decision making process. However, trust is difficult to establish in complex distributed systems involving interactions between a huge number of users. First reason for this difficulty is that in many distributed collaborative systems, people work with others who they might or might not know well. Therefore they need a means to believe information and collaborators are reliable. Issues concerning trust arise when diverse users contribute and share the same data. Second reason is that although a user was considered trustworthy in the past she may not be trustworthy in the future. Trust determination is thus not a static process and trust level for each user is not a static value. Beyond, trust provides an incentive for high-quality contributions so that it should be used in systems where only users with high trust level can contribute to controversial or critical documents. In existing multi-synchronous models, trust is usually implicit. Making trust explicitly presents us a challenge: *how to use, manage and assess trust within multi-synchronous models?* We refine the second challenging question into several scientific questions as follows.

- How to track user behavior?
- How to deal with misuses that are found?

- How to compute and update trustworthiness?
- Which is a possible solution and desirable properties to secure logs (streams of activity) for mutable data in multi-synchronous model? Note that the trust assessment returns correct results only if the logs of past actions are maintained securely.

In this thesis we put trust in an interrelationship with contracts. Trust and contracts are complementary. Trust is assessed from the compliance to contracts. Contracts are given based on trustworthiness. We have surveyed lots of works on existing decentralized computational trust models that can be applied for collaborative environments such as [90, 4, 156, 195, 3]. However, we have no finding of a suitable trust management mechanism that takes contract compliance checking as past experiences into account to compute trust levels for users in collaboration.

1.2 Thesis Contributions

We response to two challenges through a new collaboration model that we call the C-PPC model. As results we bring three following key contributions:

(1) A contract-based multi-synchronous collaboration model

(2) A log auditing mechanism and a novel trust metric to assess user trust levels.

(3) A solution to authenticate logs of user actions in multi-synchronous model.

1.2.1 Contract-based Multi-Synchronous Collaboration Model

In multi-synchronous model, each person has a local workspace. Collaboration can be established among participants without the necessity of having a central authority to maintain user activities or to keep meta data of the collaboration. This model allows all users to be free of being controlled by others. Once they have access to shared data they can contribute as they want without any prevention due to the lack of a central authority.

Push-pull-clone (PPC) paradigm is an instance of multi-synchronous collaboration. In the PPC model, users replicate shared data, modify it and redistribute modified versions of this data by using the primitives push, pull and clone. Users clone shared data and maintain in their local workspaces this data as well as modifications done over it. Users can then push their

changes to different channels at any time they want and other users that have granted rights may pull these changes from these channels. By using pull primitives replicas are synchronized. Optimistic replication mechanisms are used to maintain consistency over replicated data. To deploy successfully the use of contract we have to consider to express contracts as part of replication mechanism. This means the system has to deal with the synchronization of different working streams of user activity, the conflict resolution for contracts and the correct order maintenance between contract and basic writing operations.

We design a contract-extended push-pull-clone model where contracts are objects as parts of replication mechanism. Each user maintains a local workspace that contains local and remote changes done on the shared data and contracts associated with the data. The logged changes and contracts are shared with other users. The merging algorithm and the conflict resolution have to deal not only with merging modifications on data but also with contracts. Contract is used as a means to express usage rules that collaborating users should respect even though it is not be able to prevent users from doing bad actions. From a positive view, we use contracts as a guide to help users know what they should or should not do on the shared data.

The resulting model, C-PPC, is an extension of multi-synchronous model which indirectly supports trust-aware collaboration. A real-world example is given to illustrate the target collaborative model and it will be used throughout this thesis. In order to develop a solution making trust explicit in collaboration, contract is used as a means to express usage rules over data and it is replicated as part of collaboration logs. Contract compliance is then used to assess user trustworthiness. Furthermore, details of the model are described with specific algorithms for each collaborating process. The performance of the proposed C-PPC model is evaluated based on a set of experiments in a simulation environment.

1.2.2 Log Auditing and Trust Assessment

Compliance checking whether user actions in collaborative systems comply with contracts is an important part of our C-PPC model. This issue is done through logging and auditing mechanisms that are frequently used in many systems supporting observation. Log auditing is an approach that adopts *a posteriori* enforcement. It complements *a priori* access control in order to provide a more flexible way of controlling compliance of users after the fact.

Keeping and managing event logs are common solution for ensuring security and accountability. We use this approach to observe user behavior during their collaboration. From recorded logs, we analyze and audit actions that users did on the shared data. For checking if users respect usage restrictions, each user performs a log auditing mechanism. According to auditing results users adjust their trust levels they assign to their collaborators. To our best knowledge there is no prior collaboration model based on contracts which allows to audit and update trust levels of each participant according to auditing results. We propose a novel trust metric to help users to compute trustworthiness.

1.2.3 Authenticators

Another contribution of this thesis is to propose a technique of using hash chain based authenticators $T@site$ to authenticate collaboration logs. The correctness of collaboration outcome is based on the trust of users who should maintain the collaboration log correctly. Unfortunately, a malicious user can always introduce phony updates to forge history or alter the correct order of versions of shared document. This attack raises the threat that honest users might get forged content of shared data. Replicas with corrupted updates might never converge with other valid replicas and this is critical in replication systems.

In the C-PPC model, trust management is mainly based on the auditing results whether people fulfill their contracts. However misbehaving people always try to hide their misbehaviors by tampering logs. Therefore, to ensure the correct auditing result, it is necessary to protect logs from being forged. While tamper-resistance is impossible to be ensured in multi-synchronous collaboration without a central provider, tamper-detection should be guaranteed. In this thesis work, authenticators are constructed in order to ensure integrity and authenticity of logs of operations corresponding to different streams of activity during collaborative process. Our solution for securing logs makes misbehaving users accountable for their misuses.

1.3 Thesis Outline

In Chapter 2, we present our basic terms, i.e. multi-synchronous model, trust and contract that we use in this thesis to design the C-PPC collaboration model. Each concept is illustrated by

means of examples. In addition, we discuss the relationship of trust and contract when we put them together. Furthermore, we present a real world example as a motivation of this thesis work.

In Chapter 3, a brief review of prior works is given. We distinguish the difference of existing works from ours. We highlight their inappropriation to apply them directly to multi-synchronous model. From this review, our work is classified as a mix of optimistic access control approach and usage control approach. The chapter concludes with a discussion about the difference between prior works and the need to support explicit trust and contract in multi-synchronous collaboration.

In Chapter 4, we present our main result, the C-PPC model. We begin this chapter by first presenting background concepts of push-pull-clone model and optimistic replication technique. The emphasis is on the conceptual model with protocols of logging and auditing, and expressing contracts. We present novel algorithms for synchronizing diverse logs embodying contracts in which not only writing operations but also contracts are merged together. The merging algorithms and anti-entropy propagation ensure the causal order of basic operations with basic operations, basic operations with contract primitives and contract primitives with contract primitives. To end this chapter, we discuss the difference of the C-PPC model with related works. The details of log auditing mechanism and log authenticating solution are presented in separated chapters, even though, they are parts of the C-PPC model.

In Chapter 5, we present our log auditing approach to assess trust level in decentralized systems. We first present log auditing principles that we follow. We then define three kinds of contract violation: tampering logs, doing actions that are not allowed and neglecting obligations. We next present procedures for log auditing and trust assessment. Trustworthiness is computed based on log auditing results. We propose a novel flexible trust metric where trust levels are computed with values ranging in interval [0,1]. To complete the C-PPC model, we conduct a set of simulations to show the feasibility our algorithms as well as the performance of the whole C-PPC model.

In Chapter 6, we describe in detail our new proposal to secure collaboration logs by using hash chain based authenticators. Any log tampering will be detected by users when they check authenticators corresponding to events in received logs. We discuss the shortcomings of existing works to show their inappropriation to authenticate collaboration logs in multi-synchronous

model. We provide proofs of correctness and analyze the complexities of our algorithms through a set of experiments based on real histories of open source projects.

In Chapter 7 we conclude with a summary of thesis work, a state of the major contributions of this thesis as well as its limitations. We also outline some possible future directions.

Chapter 2

Background

In this thesis we propose a new collaboration model that provides contract awareness and trust computation in which contract is used as an extension to existing multi-synchronous models. In this chapter, we introduce definitions of main concepts used throughout this thesis: multi-synchronous collaboration, trust and contract. We also present a motivating example for this thesis work.

2.1 Multi-Synchronous Model

Groupware and CSCW represent a paradigm to support the collaboration of several users. Dix et al. presented several ways that groupware can be classified [56], among those ways there is one classification based on where and when individual participants perform the collaborative work summarized in time/space matrix.

The original idea for time/space matrix was first presented by De Sanctis and Gallupe [52] for group support systems and subsequently refined by Johansen [87] which are then presented in other papers and books, e.g. [60, 17]. In our work we adopt the view that users work in different places, therefore we exclude the dimension space. From the time dimension and according to the synchronicity of writing activities in the domain of CSCW, collaboration model is classified into two types that are *synchronous* [60] and *asynchronous*. Synchronous (or real-time) collaboration model allows communication in an instantaneous manner with bounded time and changes performed by one user are transmitted immediately to other group members. Asynchronous or

non-real time model conversely makes no assumption about the time intervals involved between interactions among users.

Dourish [58] argued that there is a model of collaboration that is not synchronous either asynchronous. He coined the term *multi-synchronous* for an area of application that is apart from traditional synchronous and asynchronous models. The term refers to a new model of collaboration which comes from a divergence-based view of distributed data management. It allows users to work in cycles of divergence and convergence.

2.1.1 Synchronous, Asynchronous and Multi-Synchronous Distinction

From time/space matrix with time dimension we classify collaboration models into three types which are presented in Figure 2.1. In synchronous model users work at the same time during certain intervals. For example, when users collaborate on a document, they are allowed to concurrently access to the document. In order to avoid conflicts, in real world applications the document is usually divided either explicitly or implicitly in logical parts such as sections, modules and each collaborator is assigned one part of the document to work on at one certain moment. One practical tool providing synchronous editing is Google Drive (known before as Google Docs editor).

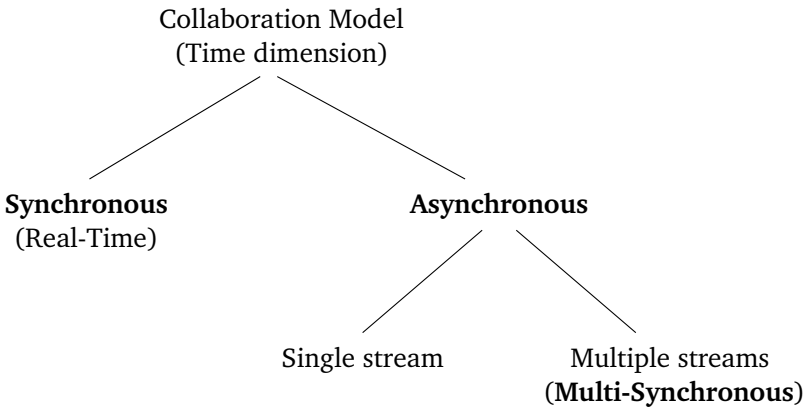


Figure 2.1. Classification of collaboration models

Apart from synchronous model, asynchronous model allows users to work on the document at different times. Asynchronous model does not negate synchronous model, but it could encompass synchronous model. With asynchronous model, times for synchronization of works could vary from one to another. We classify asynchronous work into two subcategories that are: asynchronous

work with a single stream and asynchronous work with multiple streams. The former refers to traditional asynchronous model while the later refers to multi-synchronous model followed Dourish definition.

We define a *stream* is a linear sequence of actions that people performed on the document. In asynchronous collaboration with a single stream, all users see the user changes on document in the same order. For example, in Google Drive, multiple users can edit the document asynchronously at different times. The revision history of documents edited by using Google Drive is kept linearly and all users see the same history at every moment. Another example is the editing online Wiki pages with which users see the same linear history. There is no multiple streams that are synchronized in these applications. As a classic example, editing in turn on one document with locking mechanism, that allows only one person to work on the document at one time, is an intuitive case of asynchronous model.

The multi-synchronous approach allows multiple users to maintain multiple, parallel, simultaneous streams of activity and then manage divergence among these streams. This important characteristic allows users to have private works in cycles of divergence and convergence. Users work simultaneously in their isolated workspaces and user changes are not visible to others before the changes are propagated to them. Shared data diverges when users work in isolation and converges later after the synchronization. The approach encompasses traditional synchronous and asynchronous models. When the period of synchronization of two streams is very small, it characterizes real-time or synchronous model; at the other side when synchronization takes place much less frequently in comparison with user actions, it characterizes asynchronous model.

Another interesting characteristic of multi-synchronous model is that it allows a subgroup of participants to work together privately in the sense that their works are invisible to people remains. All participants will only have the same data when all their streams are synchronized.

The multi-synchronous model refers to a very generic and specialisable model that describes a range of distribution strategies used in CSCW systems. The model therefore can be refined for any particular system or application. With respect to its principle, we can design a specific system that provides multi-synchronous work for collaborators who might either usually or rarely physically meet each other. Since it allows multiple streams of work to diverge, it is necessary to ensure that a consistent view of data can be constructed later. There are solutions for consistency

guarantees in this model. Among of the solutions, rather than using traditional locks, optimistic replication [162] has become a widely used technique that ensures eventual consistency.

Multi-synchronous collaboration model is used not only in research works such as SAMS [128], DSMW [150], but also in practical applications with thousands of real users such as Distributed Version Control Systems (DVCS), e.g. Git [107], Mercurial [132], and Microsoft SharePoint Workspace [126].

In summary, we have distinguished multi-synchronous model from traditional synchronous and asynchronous models. According to this distinction, a model that only supports real-time collaboration is synchronous. A model that supports at least one participant working asynchronously is asynchronous. An asynchronous model supports participants working with multiple streams is multi-synchronous. In next subsections, we present two typical examples that could use multi-synchronous model to support collaborative work among users.

2.1.2 Example 1: Collaborative Writing

Collaborative writing, also known as collaborative editing or collaborative authoring, has been conducted in research works, e.g. [60, 96], and deployed on real world systems, e.g. Wikipedia, Google Drive. In these applications, members of a group are allowed to edit the same document at the same or at a different time even when separated by physical distances. People do not always work simultaneously at a central place (or system) at same time. With multi-synchronous collaboration model, they can take of copy the shared document and work in isolation on their draft at any place they want such as at home, in office, on the air plane, during a meeting, wherever and whenever and with or without a network connection. To obtain the same view on a final common state of the shared document, they meet each other (online or offline) to synchronize their divergent drafts periodically. Right after, they can continue to work in isolation on their draft and synchronize at a later time. The multi-synchronous model relaxes the requirement that users have to work simultaneously at the same time, over the same space or on a unique version of data object.

2.1.3 Example 2: Source Code Management

The working mechanism for source code development is employed by version control systems. Version control systems track the history and the evolution of a single file or a software project. The earliest revision control tools were intended to help a single user to manage revisions of a single file. Then revision control tools were expanded to manage multiple files or projects to help multiple users to collaborate. Most of complex software nowadays are developed collaboratively by many programmers rather by a single one.

Many centralized revision control system were developed as RCS (Revision Control System) since 1980's, CVS (Concurrent Versions System) since 1990's, Subversion, etc. The log of version control systems allows to identify who made what changes and when they made it. The CVS system uses a client-server architecture to store the current version of a project and its history. It allows programmers to connect to the server in order to check out a copy of the whole project, work concurrently and independently on this copy and later check in their changes. The changes from every programmer will be synchronized after conflicts were resolved.

As of today, distributed version control systems (DVCS) such as BitKeeper [27], Mercurial [132], Git [107], Bazaar [22], Darcs [45], etc. have become more and more popular. They are typical examples of multi-synchronous collaboration model. The arrival of distributed revision control enables new ways of collaboration. For a programmer distributed tools are much more efficient than centralized tools. There is no repository which is special or central in distributed version control systems and every clone is equal. A distributed tool keeps all meta data of a project locally rather than on a central server. This helps to reduce unnecessary network overhead. Developers can work on multiple versions completely offline even if the network connection goes down. They work simultaneously on their working streams or branches and branches are merged iteratively. DVCS technology enables new ways of thinking and developing softwares in multi-synchronous manner. It allows people to work with divergent streams that continually converge when developers synchronize their changes to build the final project.

2.2 Overview of Trust

The traditional computer security model is inadequate to deal with the vast change of technologies nowadays. The open, interconnected, decentralized environments blur the distinction of architectures. Blakley points out the reason why security might fail: *"No viable secure system design can be based on the principles of Policy, Integrity, and Secrecy, because in the modern world Integrity and Secrecy are not achievable and Policy is not manageable"* [28]. The author suggested some guiding principles to change security foundations. One of the suggestion is *"make the user ask forgiveness, not permission"*. This can be understood that in an open environment where users trust each other, managing individual permissions may cost more than it is worth.

Trust is a universal relationship we have with people. Most of time we make exchanges with other people without having full knowledge about their intentions. In collaboration, trust is a very essential issue. The people in an organization or a specific community make poor decisions about trust all the time. They often forget the matter of trust and engage in a collaboration relationship based on flawed trust decisions or they only rely on the security of system to trust others. This becomes critically when the collaboration expands to a huge community over large scale and distributed networks. They cannot manage trust and usually forget it. Therefore it is important to understand what trust is and how it influences collaborative behavior. In the article *"A Matter of Trust"* [93], Kaplan refered to a state of professor Cheriton, *"The limit to a distributed system is not performance, it is trust"*.

2.2.1 Trust Definition

Many researchers from different fields give different trust definitions. The concepts of trust in different fields vary in how trust is represented, computed and used. In computer science, trust is not a new research topic. It spans over different research areas such as security and access control, reliability in distributed systems, game theory and agent based systems. We do not intend to discuss about every point of view about trust, rather, we will focus on the view of trust that is close to the collaboration in distributed systems.

We start from the Oxford Advanced Learner's Dictionary on the definition of trust: *"Trust means to believe that somebody is good, honest, sincere, etc and that they will do what you expect*

of them or do the right thing; to believe that something is true or correct.". According to Riedl [155], *"trust is needed due to the impossibility to deal with the world in its full complexity, the impossibility to avoid this complexity completely, and the impossibility to protect oneself completely against all risks of evil behavior from others.".* Gambetta [70] gave a different definition of trust which can be summarized as follows: *"trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action".* The author explains that when we trust someone, it implicitly means that the probability that he will perform an action that is beneficial at least is high enough for us to consider engaging of collaboration with him, and also correspondingly it implies that we refrain from doing so if that probability is low enough. From this definition the author also let us see that trust can be seen as a threshold point located on a probabilistic distribution of general expectations. Mui et al. proposed the concept of trust which is based on past encounters: *"Trust is a subjective expectation an agent has about another's future behavior based on the history of their encounters"* [104]. Based on many existing definitions of trust, as examples cited above, we proceed with the following definition of trust and its related concepts.

- *Trust.* A trust relation can be defined as a belief when a trustee is offered the chance to do actions, he will not behave in a way that damages the truster's expectation.

- *Trustworthiness.* Trustworthiness (a trust value) is a quantitative measure of trust relation assessed subjectively that a truster assigns to a trustee in a particular context. Trustworthiness reflects certain levels of trust relation that a trustee holds.

- *Reputation.* Reputation is a measure that is derived from the past behaviors of a user and it is usually used to assess trustworthiness that a user assigns to another user.

In [88], Josang et al. distinguished trust and reputation in which reputation mentions to what is said or believed about a person's or thing's character or standing. The authors illustrate the difference of trust and reputation through an example: *"I trust you because of your good reputation"* vs. *"I trust you despite your bad reputation".*

The collaboration frequently demands a certain level of trust as a precondition. Trust helps to make *a priori* estimate of the probability that a person will act collaboratively. However with the evolutionary approach as in the nature of human world, trust is understood as a result rather than a precondition of collaboration. For example, this is reinforced by work in game theory [14]. Collaboration could be triggered not by trust but simply by random at first and then selectively retained. Thus even if a person cannot commit himself in advance (since his trust level is very limited) he still is allowed to collaborate but consequently retained due to the success of his contribution. Trust between people exists at certain levels when they collaborate at one time but it will evolve over time. It is required in cooperation as Williams's state *"cooperation requires trust in the sense that dependent parties need some degree of assurance that non-dependent parties will not defect"* [189].

Trust can be seen either implicitly or explicitly in many aspects of human world. Let's consider an example when we invite a person to collaborate on our project, it means we trust him. The trust we place in that collaboration relationship expresses our belief on his competence and his commitment to contribute to the project. We expect that every behavior done by him will follow our rules, standards and norms for working together on the project. Any misbehaving action he does would lead him to become distrusted. Beyond using trust to let people collaborate, the trust placed in that collaboration also expresses the encouragement people to bring positive collaboration since it is important to trust and it might be equally to be trusted.

It is also important to discuss about trust properties. In his work [93], Kaplan summarized that trust is not: transitive, distributive, associative or symmetric, except in certain instances that are very narrowly defined. Trust properties were also mentioned in the work of Vu et al. [184] with a little difference where they are summarized as: autonomy, asymmetry, intransitivity, non-distributivity, non-associativity, mutability. These trust properties need to be properly reflected by trust models in order to avoid ambiguous results to assess the strength of trust relation.

2.2.2 An Example of Trust

As of an example of trust, let's consider the impact of trust in the domain of software development for open source software community. The community includes groups of individuals sharing

common interests to build open projects. They are composed of diverse developer communities. They work together heading to a common goal. In that community developers all share their own work in a trusting manner that others will use it and extend it further to achieve the good product at a final state. This expresses the implicit trust which might usually exist implicitly in many other collaborations as well. Once developers engage to a community and make the decision to collaborate on a project, they trust it will go well and will bring the useful outcome and their objectives will be met. Trust is not only built before the collaboration starts but also is maintained during the collaboration itself between developers, project founders and project maintainers. Often communities will have documented their norms or social contracts or community policies that express the expectations concerning the contributions from community members. One example of this kind of contract is that developers should not do refactoring methods, classes, or do small changes as changing indentation, aligning variables in source code programs. In many free open source softwares where developers are using distributed version control systems, there are many of them working together based on the web of trust. Instead of maintaining complex policies concerning who can write and commit, every developer trusts only a few number of developers from whom they pull new changes. These people at their turn trust a few of others as well. This forms a network of trust between developers. Developers only synchronize their works with others whom they trust.

2.2.3 Trust Model

Trust model is a means used for trust management. It formalizes trust concept in a precise and meaningful manner. It takes into account the information sources to calculate trust. Roughly speaking, trust models define all assumptions on trust properties that they adopt and describe how to calculate trustworthiness. There is no concrete trust model that is applicable to all kinds of applications and contexts. One of the first work to formalize the concept and various aspects of trust to represent them by a mathematical model was introduced by Marsh [118]. The model is based on social properties of trust from the view of sociology and psychology. It is a relatively complex and abstract model and is difficultly used in today's electronic communities.

We will present several aspects related to computational trust models in the sense that in these models, trust is a factor that can be computed, represented and managed. Our work

focuses only on computational trust models. In other words, we do not take into account all trust models built for other fields such as psychology, sociology and economy. A categorization of major areas of trust research can be found in [13].

We can classify trust models into different types depending on the dimension we are looking at. The first dimension to look at trust models is based on the information source that trust models use to compute trustworthiness. Trustworthiness is the value that reflects trust level of users/peers/agents. The information can be collected either directly or indirectly. In the former way, information to calculate trustworthiness comes from user's own experience that she interacts directly with other users or observes the interaction of other members in the community she joins. In the later way, information to calculate trustworthiness comes from observations of other members of the community. That information is gathered by their own experience or from others that they can observe. The indirect information source can be taken from the sociological knowledge. Social information can be understood as social relationships between users and this is only available in systems having rich interactions between users.

Regarding the types of information source, some trust models are classified as either user-driven [153] or content-driven [9]. In the system which is *user driven*, trust is computed based on users rating for other's contributions and behaviors. The example of this type is the eBay reputation system. In the system which is *content driven*, user trustworthiness is computed based on the content that users contributed. The example of this type is the assessment of user trust level when Wikipedia authors contribute to the content of Wikipedia pages [9].

The second dimension to look at trust models is based on the way to manage trustworthiness. With this dimension the trust level of an individual is either seen as a global property that every member shared with each other or as a subjective property that is calculated particularly by each individual. When trust level of an individual is made global, each individual calculates trustworthiness of others based on her past interactions with them. Then her assessment is made public to all other members. When trust level is subjective, each individual assigns a trust value to other members according to her personal experience. In this case we cannot say anything about the trustworthiness of a particular user u; instead we only can say about the trustworthiness of user u from the point of view of another user v.

The models that consider trust as global have a drawback that they lack personalization of each trust value. One thing that is bad to a person might not be that bad to another. So the approach taking trust value as global is useful only in systems where it is possible to assign the same way of thinking to all members. This is only a simplification of the real world community. It cannot be used to deal with more complex communities. However, it can be easily deployed for systems with large number (thousands or millions) of users in which they interact with each other repeatedly. In contrast, the models that consider trust as subjective provide personality of trust levels of particular users, however, it is difficult to establish the link of all members in the community.

Table 2.1: Trust Models Classification

Trust Models		Trustworthiness Calculation	
		Global	Subjective
Information Source	Direct	Marsh [118], EigenTrust [90], NICE [156], Content-Driven Reputation [9]	Global Trust Model [4], PeerTrust [195], Stereotrust [106]
	Witness	non-existing	Trust-X [25], Gutscher's model [78]
	Sociology	eBay and Amazon reputation models	Regret [161], Abdul-Rahman & Hailes 's model [3]

In the Table 2.1 we classify some typical trust models based on dimensions presented above. In this table, we classify information source to distinguish trust models in three types: direct, witness and sociology. The direct sources come from direct experiences (direct transaction with other users, received history, etc) that a user holds so that based on these sources she can compute trust for other users. The witness such as trusted credentials is another source that helps to assess trust. In the system with credential-based trust, users usually have trust on a trusted third party who issues credentials. The sociology source usually comes from social relationship of users that are from external experience between users.

According to the way trustworthiness is calculated, we classify trust models in two types:

global and subjective. With the former, trustworthiness is computed and shared with all users, while with the later, trustworthiness is computed by individual users.

After gathering data from information sources, trustworthiness is computed. Trustworthiness calculation plays an important role in a trust model to represent the strength of trust relation and it is one part of trust management as well. Roughly speaking, trustworthiness can be calculated based on the reputation of peers (e.g. in P2P systems), users (e.g. in collaborative systems), on the probabilistic estimation of social activities (e.g using Bayesian estimation), and on the recommendation of other peers, users in systems.

The trustworthiness can be classified into four types of values that are *single value*, *binary values*, *multiple values*, *continuous values* [184]. For example, user trustworthiness is evaluated as only trust or distrust, or with discrete grading such as {very good, good, bad, very bad}, or scale integer (e.g. from 0 to 10), or finer grain with real numbers from the interval [0, 1] where "0" indicates that a user is absolutely untrustworthy and "1" indicates that the user is absolutely trustworthy. To assess user trustworthiness we need an algorithm that aggregates information from all possible sources and turns it to a meaningful value that can be managed by trust model. To calculate trustworthiness, there are some systems that suggest not to combine every information into one value but to maintain multiple statistics about each user. For example, the TRELLIS system [73] keeps separately ratings for the likelihood a peer cooperates on a transaction and the accuracy of its recommendation.

In our thesis work, users use direct information source from received logs to compute trustworthiness. Trust is considered as subjective since it is assessed by individuals from their personal experiences. Trustworthiness is computed and represented by continuous values ranging in the interval [0,1].

2.3 Overview of Contract

When people join a collaborative system, they are assumed to collaborate positively, but there is no assurance that they will not defect. Therefore many mechanisms have been proposed to ensure correct behavior of people in a form of implicit or explicit contract between people. The purpose of contract is to lay out what is expected from each collaborating individual.

2.3.1 Contract Definition

Contracts compensate for the lack of trust between users for their ongoing actions. We proceed with a contract definition as: *a contract is an agreement on documents for collaborating parties. It is composed of obligation, permission and prohibition. It formalizes what the people are permitted to do, obliged to do and forbidden to do to work together on a task.*

Once a contract is established, it is used as a future reference for user actions. There are two contract levels that are business level which is in a form of natural language and operational level which is in a form of machine-readable representation. The compliance with a given contract of collaborating participants will be used as a past experience for building trust. US Department of Commerce guides that *"the most accurate predictor of future performance is past performance in a similar situation"* [12].

In this thesis, contracts are built on the top of basic deontic logic [194] with normative concepts of the obligatory, the permitted and the forbidden representing what one ought to, may, or must not do. As one part of moral discourse, we use deontic contract for the purpose of directing people action. Deontic logic is one of four main groups of modalities that the philosopher Von Wright mentioned in his articles [194]. First, the alethic models or models of truth that cover concepts as the necessary (the necessarily true), the possible (the possibly true), the contingent (the contingently true). Second, the epistemic models or models of knowing that cover concepts as the verified (that which is known to be true), the undecided, and the falsified (that which is known to be false). Third, the deontic models or models of obligation that cover concepts as the obligatory (that which we ought to do), the permitted (that which we are allowed to do), and the forbidden (that which we must not do). Fourth, the existential models or models of existence that cover concepts as universality, existence, and emptiness.

In deontic models, the first preliminary concepts are the *acts* that are pronounced obligatory, permitted and forbidden. The word "act" is used for properties and not for individuals, for example, "steal" or "smoke" are acts. The negation of a given act performed by an agent, if and only if, he does not perform the act. For example, the negation of the act of answering a question is the act of not answering it.

Von Wright [194] considered concept of permission is true on formal grounds and then he

defined concepts of the permitted and the forbidden. Deontic concepts that are applied to a single act follows.

> "If an act is not permitted, it is called *forbidden*. For instance, theft is not permitted, hence it is forbidden. We are not allowed to steal, hence we must not steal."
>
> "If the negation of an act is forbidden, the act itself is called *obligatory*. For instance: it is forbidden to disobey the law, hence it is obligatory to obey the law. We ought to do that which we are not allowed not to do."
>
> "If an act and its negation are both permitted, the act is called *indifferent*. For instance: in a smoking compartment we may smoke, but we may also not smoke. Hence smoking is here a morally indifferent form of behavior. [..] Indifference is a narrower category than permission. Everything indifferent is permitted but everything permitted is not indifferent. For, what is obligatory is also permitted , but not indifferent."
>
> (Deontic Logic [194], page 3-4).

Norms that come from deontic concepts of the permitted, the obligatory and the forbidden are permission, obligation and prohibition, respectively. We build contracts on the top of these norms and handle them in a distributed manner. It is noticed that a permission can be a strong permission or a weak permission [193]. The permission which is an exception of an obligation or a prohibition, $(P_A \& P_{\sim A})$, is a strong permission. The permission which follows from the absence of a prohibition is a weak permission, $(P_A \& O_A)$. If an act is strongly permitted then also its negation is permitted, whereas if an act is weakly permitted then its negation is forbidden. Hence, with strong permission a subject always has choice to perform the act or not, and it is not the case for weak permission. We illustrate the concepts in deontic logic model in Figure 2.2.

The contractual approach based on deontic logic is useful for a wide range of applications, such as resource management, cooperative task execution, cooperative work in distributed systems and software engineering. Traditionally, a contract is an agreement between two or more persons about actions that are performed. Contracts also regulate behavior when persons cooperate or use shared resources. Generally, contracts are implicit in many systems. It is the case in communication protocols, software licenses, downloading and sharing policies in P2P file-sharing

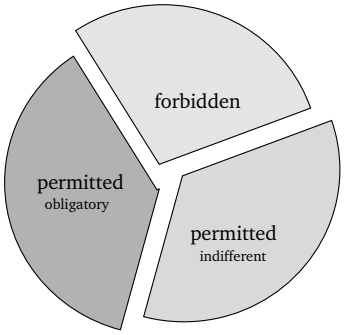


Figure 2.2. Deontic concepts.

systems and paper-based contracts of using network services. In the DVCS for collaborative editing source codes, no contract is specified explicitly but developers maintain a web of trust implicitly. Wikipedia features an informal contract-based model where contracts are checked by crowd sourcing. Anybody can edit according to rules that are checked *a posteriori* by human factor. Differently, our goal is to build a contract-based model in which rules have to be explicitly expressed and checked by system.

We can see that contract ranges either implicitly or explicitly in most of real world systems today and we present some of them below.

2.3.2 Example 1: Free Software Licenses

To make a software free, copyright holders release it under a free software license. The most usually used one for GNU programs is GNU General Public License (GNU GPL). GNU organization uses the *copyleft* method for making all extended versions of a program free software to be free as well. A copylefted program is a copyrighted program with distribution terms. Having a GNU free program software, everyone has rights to use, modify and redistribute the program code under GNU GPL license term. Copyleft prevents uncooperative people from converting a free software by making changes and redistributing the result as a proprietary product. The GNU GPL represents a cooperative contract where programmers have not only rights to use and change free softwares but also obligations to release any extension of softwares as free softwares.

As another example, the *Debian social contract* [49] was designed as a set of commitments and it has been adopted by free software community. The concept "social contract" was suggested by Ean Schuessler. It is then drafted by Bruce Perens, refined by Debian developers and finally

accepted as the publicly stated policy of Debian projects. The contract expresses the commitment of free software community, for example *"Debian is completely free"*, *"developers do not hide bugs"*, *"free software will be widely distributed and used"*.

2.3.3 Example 2: Wikipedia Editorial Policies

Wikipedia is an open and free encyclopedia that welcomes everyone to view and edit freely. However when editors come to edit they should respect its editing policies. They are allowed to give information objectively but not to judge. Also they are not allowed to add copyrighted materials without the permission from copyright holders. Also, they have to cite the sources of information (if any) in their contribution pages. All these policies feature the implicit contract that any editor should respect, otherwise their writing might be retracted.

2.3.4 Example 3: Policy in P2P File Sharing Systems

Many peer-to-peer (P2P) file sharing applications allow users to download and share electronic files freely. These P2P sharing networks allow sharing all types of files from text to image, audio, video files and many of them are copyrighted. Beyond, P2P sharing file networks also concern configuration of network resources such as bandwidth and storage. Therefore, before participating in any P2P sharing activity, users should ensure that such activity complies to policy of the organization that offers the sharing service. The policies are expressed in form of contract for what user activities are permitted or forbidden. For example, users are not allowed to distribute copyrighted files without the explicit permission of the copyright owner. And users are prohibited do activity in ways that jeopardizes the security of sharing network. It is also stated in many P2P file sharing networks that any violation of the contract can lead to service cease with or without notification to users.

2.4 Trust and Contract Interrelationship

How trust and contract are related? Woolthuis et al. [192] enhanced and made a clear clarification for the question of whether trust precedes or follows contracts and shows their impact in collaboration outcome. There are three different views on the role of contracts and their influence

on trust. One view sees contract as a basis for trust since it limits the opportunities of misbehaviors. A different view from social scientists often envisages contract as "in conflict" with trust in which contracts are interpreted as a sign of distrust. Third view of the relationship between trust and contract considers trust preceding contract, thereby decreasing or eliminating the need for contract. The work argues those three views are incomplete since there are possible other views. Trust or trustworthiness and contract are related in cycles that trust and trustworthiness need to begin where contract ends or contract begins where trust ends. In other words, trust is built based on the compliance of contracts and contracts are established based on levels of trust (see Figure 2.3).

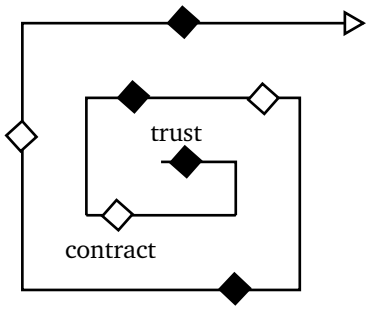


Figure 2.3. Trust and contract spiral upwards.

Trust can both precede and follow contracts. In order to argue how trust and contract can be substitutes or complements, it is shown that trust and contract can be both substitutes and complements. When a contract is not interpreted as a strict legal safeguard, trust and contract will be complementary. When participants trust each other they may decide not to use contract, in that case, trust substitutes contract. Since trust and contract need not be opposing alternatives, trust and contract can well be complements in which contract may have different functions. Contract can be used as a technical aid to remind participants of their commitments that were made. In complex systems, participants may put a detailed contract into place to have a guidance for action and contract should be placed in its social context.

2.5 A Motivating Example

In this section, we present an example for our target model where contract will be used to support trustworthy collaboration.

2.5.1 Photo Sharing Issues

It is very common that people share photos nowadays. Photos help people stay in touch with their family and friends all around the world. They take photos in their daily life experience coming from many contexts such as from a wedding to a vacation or from a local meeting to an international conference. After an event where many people took many photos, people who joined the event want to share those photos with each other. By that way they can remember all the little moments they might have missed during the event. Together users can build a great photo-based story for the event which they have had participated. There are many applications that provide photo sharing service such as Photobucket, Flickr, Picasa, Facebook, Google Plus. Every photo collection on those systems is potentially collaborative. There are three main issues that require to build a new way that people can share and collaborate to build photo album.

Firstly, putting all photos in a single public space which are available for all people might not be an interesting idea. There are some people who keep all photos of the event, but there are some others who just want to pick their favorite photos such as snapshots for the sake of keeping memories since they do not have time to look at over several hundreds of photos. Thus the photos should be kept in local spaces of individuals and different users are given policies and invited to collaborate. From this desire, we proceed with a collaborative work to build together a photo collection in which users keep their photos in their local spaces and share their photos with others. Others can synchronize shared photos with the local ones respectively in their spaces and then collaborate on those photos.

Secondly, even though users have had joined the same event, they might or might not know each other before. Hence, the same photos from the same event are not shared uniformly to all people. Some people just want to share photos only with a subset of the people that attended the event. For example, a user does not want to share the photo or give the right to comment on photos she took during a banquet to everyone. Rather, she just wants to share it only with people she knows well. The sharing decision is made based on social relationship and the trust she has on the others. Furthermore, people do not trust others with the same level and it is still required to have contract as a guidance for what users can do and should do with share photos. People would not only want to share the collection with the people were at the event but also

with non-collaborators under more restriction (e.g. they can only view it).

Thirdly, when photos are stored by a third party, such as Yahoo or Google, then users have no more control over those photos that they have put on third party's server. This is a critical issue as we mentioned in previous chapter about Big Brother problem. This issue raises a requirement that user photos should be kept, managed and shared personally by end-users who own those photos.

On the photo collection they can perform editing (i.e. inserting, editing, deleting) operations to the meta-data of each photo like: location (town, area, city), latitude and longitude information where the photo was taken, date and time, photographer (name and contact information), comments, and other information (keywords, copyright, web link, title, etc).

2.5.2 A User Study

Concerning the real world application of building photography collection collaboratively, we started to have a look on existing photograph sharing softwares, so that, we know how current users are using software for managing their own photos as well as what feature is missing to let users to be able to collaborate on those photos.

We take as an example the Gallery - an open source photo sharing software [67]. This software is different from others that is a website for sharing photos. Gallery is not a website, but it is a software that can be installed and run by individual users. It is a very popular software for creating, displaying and sharing personal photograph collections for more than a decade. It can be connected to other softwares such as blogging software, e.g. WordPress, by using additional plug-in or modules. Since it is a very great photo sharing software, it supports many features for users (e.g. view album, photos, comments, tags, add new comment, subscribe to a photo or an album, edit user information), editor (create album, manage photos, edit meta data, edit permissions).

A survey conducted in 2009 focused on the usage of Gallery. The results of the survey show the important user preferences for a photo sharing software [68]. The survey was done in three weeks with 702 full responses over 1251 participants, from 49 countries and with people speaking 30 different languages. According to the survey, 83% of current Gallery users use their own site as the primary way for sharing photos. Main activities are posting photos of friends

and family (75%). This result indicates that users tend to manage photo data by their own. Further the survey shows that two most important photo-sharing tasks are arranging albums (84%) and managing permissions (70%). On a scale of 1-11, the most important meta-data is copyright/licensing (8.88). Among the most important preferences, the ability to manage access permissions to photos is 76%. Respondents find second most desirable quality is editing options (average raking 7.60 on a scale of 1-11). In managing permissions, 55% respondents create special albums they only share with certain people and 20% respondents create collaborative albums letting others can add photos. The survey result shows that the users are interested in managing their personal photo data. One respondent wrote: *"Make permissions easier to do and understand to set for non-technical folks. I don't want my mom sharing private family photos by mistake."*.

The survey shows that usage policies are very important for users when they share their photos. Gallery is widely used but it lacks features about collaboration between users. This feature could be extended by adding new plug-in so that they can collaborate and manage contracts in current photo sharing systems using Gallery.

2.5.3 A Scenario

Figure 2.4 illustrates a scenario of building collaboratively a photo collection by four users: Sheldon, Penny, Raj and Leo [1]. The collection is not viewed equally at different user workspaces. Photos are kept and shared by user decision which derived from the trust levels they have among them. Due to the use of trust, the same photo might be shared with different users under different contracts.

In the scenario, Sheldon wants to share directly a funny photo with Raj and Penny. Due to the different trust he has with them, Sheldon gives them different contracts. While Raj is allowed to change the photo Penny is not, but both of them are able to share it further. However when Penny receives the photo, she deletes the comment that Sheldon put, inserts a new comment and then shares it with Raj. In this example, Raj receives the same photo from two different people, Sheldon and Penny, and he can discover that Penny did not respect the contract given to her when she received the photo. Therefore Raj changes his belief on the honesty of Penny by adjusting trust level that he assigns to her.

[1] These names used in the example are borrowed from characters in the American sitcom *The Big Bang Theory*

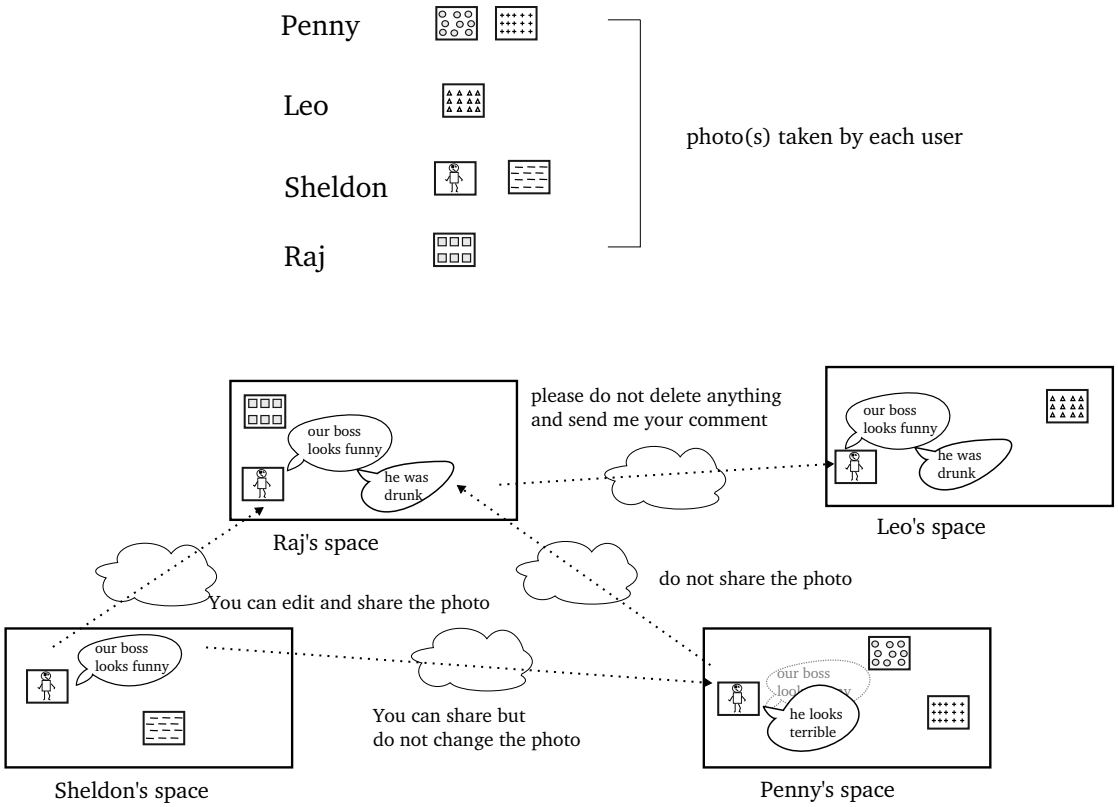


Figure 2.4. Building cooperatively a photo collection

Our ultimate goal is to develop a new model of collaboration for large scale distributed systems. Contracts are given in different forms due to user trust level. A history of collaborative process of building photo collection is used to keep details such as when photos are viewed and whether users have edited them. Then any contract breach will be taken into account to make trust re-assessed in the next cycle. Trust and contract complement each other in the collaboration model.

In later chapters, we will specialize the motivating example in a concrete illustration. The way to specify contract and to check contract compliance will be presented in the C-PPC model. The techniques to protect collaboration logs and to audit logs will be investigated extensively as well.

Chapter 3

State of the Art

Our thesis work addresses the main challenge of data usage control within multi-synchronous collaboration model. We take into account usage control by means of the control over the access to data and the usage of that data. This chapter provides a review of the evolution of usage control with various aspects: contract-based systems, access control and usage control. These topics partly relate to our contract-based model. We will present shortcomings of prior works and identify several issues that those works are inadequate for multi-synchronous model. This exploration gives a coherent understanding of different data control approaches. Furthermore, it helps to indicate the position of our work linked to main streamlines in the research domain.

3.1 Contract-based Systems

In some distributed systems, contract was used to establish agreement framework among system components. Typical contract frameworks can be listed as, for example, the Contract Net protocol [175], Business Contract Architecture [124], Agreement Framework [159, 171], Contract Framework [172]. These frameworks are mostly designed for negotiating and controlling resource usage in a distributed system and engaging to solve the connection problem among nodes with tasks to be executed simultaneously. Contracts express the terms under which nodes in network promise to offer and to get payment with regard to exchanging resources. These contract frameworks deals with contract specification including of the server, the client, the resource, the negotiation and the signature scheme.

Abrahams et al. proposed an asynchronous "Event Condition Obligation" style as an alternative to the conventional synchronous "Event Condition Action" [5]. In the proposal, business contracts, internal policies, legislation, and social conventions are considered as sets of contractual provisions. Each provision specifies a promise, a prohibition, a permission or a power. The provision enforcement relies on a rule-based framework. With a similar idea of putting contract into runtime systems, Gama and Ferreira introduced Heimdall platform providing enforcement mechanism for obligation policies [69]. Chiu at al. presented a meta model with a three-layer architecture (document, business, implementation) for e-contract enforcement in which contract clause includes obligation, permission and prohibition [38]. In the architecture, UML language is used to specify contracts and contracts are then transformed into ECA (Event Condition Action) rules to be checked for any breach. A contract-based framework with ECA enforcement for collaborative work was presented also in [139].

On the topic of languages for representing contracts, Pace and Schneider stated that constructing a widely applicable contract specification language still remains a challenge [133] since contracts concern various aspects of temporal constrains, deontic modalities, conditional commitments. The authors proposed a language for expressing full contracts based on the top of deontic logic, and in combination with temporal logic. Another work on contract specification language was proposed in [84] within a trace-based model for contracts. In this work, the authors present an abstract contract model with the language for contract CSL. The CSL is targeted to represent business contracts and it can be applied to various contracts from different domains. Contract specification language was also presented in many other works [77, 149, 127]. Even though languages for contracts were investigated intensively, all of these works follow the direction of having an independent framework for contracts.

Policies, which are similar to contracts for interactions between participants of different organizations in sharing resources, are being increasingly used to specify rights and obligations inside virtual organizations. In [47] the authors presented a simple architecture for an e-market concerning the provision of automated support for contract performance. The contract terms are represented by obligations, rights, powers and other legal relations. In the architecture, an artificial (controller) makes the judge by contract monitoring based on behavior of partners according to their agreement. The judge is also used to establish trust between partners.

In [125] the authors mentioned the deviation of the agreed contract obligations. Role-based architecture is used for contract establishment. They provide the ability of monitoring contract and notifying of non-compliance events; and also describe a number of control mechanisms for contract management. A finite state machine with five levels from 1 to 5 is used to check the fulfillment of events with respect to contracts. Level 1 reflects a full compliance and level 5 reflects a full non-compliance. Contract enforcement covers the monitoring contract and the notifying about contract breach. It includes both the non-discretionary (or preventive) mechanisms and discretionary mechanisms. The enforcement mechanism is performed by a central system.

In software engineering, Métayer et al. [123, 122] proposed a set of methods and tools to define software liabilities among participants. The framework includes the formal definition of liability and the analysis of log files to verify contractual liability *a posteriori* to make participants accountable.

To conclude, in prior works presented above, some of works [175, 124, 159, 171, 172] focus only on contracts among system components instead of human factors. Some of works [133, 84, 77, 149, 127, 123, 122] focus on contracts for end-users but they require independent framework to express contracts. Some of works need a central authority to enforce contracts [47, 125]. Our work is different from prior works because we integrate contracts together with data and we do not need a central authority to check contract compliance.

3.2 Access Control Models

In this section, we examine existing access control models for collaborative systems which provide opportunities for multiple users to collaborate in heterogeneous environments. We will present each single approach and identify its potential limitations. The presence of multiple users with the vary of goals, experience levels, introduces the potential for uncertainty, unpredictability into collaborative works. Participants are expected to fulfill their obligations and to respect given rights as well as rights of others. Furthermore, each user wants the ability to regulate access to their personal information. However, the goals of collaboration and security contrast in general. While collaboration aims at supporting people to collaborate in a convenient way by providing them information and resources they need, access control mechanisms are designed to limit which

authorized users can access and use over data and resources. Due to this contradiction, designing any authorization mechanism for collaboration needs the balance of these competing goals.

3.2.1 Role-based Access Control

The essence of role based access control (RBAC) is that permissions are assigned to a functional role or a set of roles which are hierarchically organized rather than to individual users. Roles are created for different tasks and users are assigned to roles based on their ability to do those tasks. RBAC was first introduced by Ferraiolo and Kuhn [62] to overcome the limitation of discretionary access control model (DAC). Sandhu et al. [164] presented four reference models to give a systematic approach to understand RBAC model. The NIST standard for RBAC was proposed [63]. RBAC was then extensively investigated in many researches, for example, Shen and Dewan [174], OrBAC [89], OASIS [16]. For collaborative systems, Edwards [59] attempted to make role specification flexible by allowing access control policies to be defined in terms of access control rights on data objects. The author proposed Intermezzo framework as a collaborative infrastructure that provides coordination support to policy control. The policies are used as general guidelines to restrict and define the behavior of the system. A policy specification consists of roles, access rights to resources and corresponding attributes referred by that policy. The roles can be statically defined beforehand or dynamically defined when applications run.

Task-based access control (TBAC) proposed by Thomas and Sandhu [182] is an extension of RBAC. In TBAC, permission is granted in steps that are related to the progress of tasks. TBAC is limited to the context including activities and tasks and these tasks need to be known *a priori* at the time designing system. Team-based access control proposed by Thomas (TMAC) [181] is another extension of RBAC for collaborative environments with groups users in teams rather than in roles. The "team" is an abstraction of a set of users with specific roles and they collaborate with the objective of accomplishing a specific task or goal. Users in teams can have different roles. When compared to the development of role-based access control models, TMAC is a hybrid access control model allowing permissions to across object types as well as fine-grained control on individual users and on individual object instances. The proposed framework TMAC is designed with the need of making it self-administering to reduce security administration overhead. It can be made self-administrating by trapping basic calls issued by host information system

(can be seen as a central authority) to assign and design team members.

RBAC is more scalable than user-based security since it supports to change users assigned from one role to another without changing the underlying structure of security policies. However it has several weaknesses that make it inadequate to multi-synchronous model. The first shortcoming is related with the lack to support individual control given to users. If users cannot be grouped in certain number of roles, it will be difficult to determine every role to give to individuals. The second shortcoming is related to the nature of multi-synchronous model that requires permissions are based not only on individuals but also on data at different granularities. RBAC lacks the ability to define fine-grained policy for individuals and data objects. TMAC overcomes this disadvantage. The third shortcoming is that the model is inadequate for decentralized, open and distributed systems for several reasons. (i) It is difficult (if not want to say impossible) to know beforehand all the users that will request services or data in these systems, therefore assigning appropriate roles to users is harder to manage. For TBAC, collaboration is not always easily to divide in tasks *a priori*. For example in the health-care systems it is difficult to define workflow tasks and their access control requirements in a static way since workflows are usually ad-hoc and users join and leave unpredictably. (ii) There is no central authority for checking access. Neither RBAC, TBAC and TMAC overcome this disadvantage.

3.2.2 Attribute-based Access Control

Attribute-based access control (ABAC) differs from RBAC by replacing the subject by a set of attributes and objects by properties associated with them. Permission is given based on attributes of users. A user needs to prove having necessary attribute to gain access. An example of ABAC is the access control system using X.509 digital certificates which contain attributes such as user role, user identity, citizenship, group membership. The decision to give access to data or system will be made based on these attributes. Access rules specify what attributes are required to access resources or data.

The idea of incorporating attributes to provide more flexible RBAC models was presented in a number of research works. Goh and Baldwin [75] proposed the use of additional role attributes to enrich access control model which allows a full range of roles to be expressed. It is practical to seek role definition by specifying role attributes through standard descriptions, rather than

attempting standardization of role definitions. The distinction of the concept of "constraint of roles" and "role attribute" is drawn in [164]. The constraint of roles applies to the relationship of different RBAC components. The role attributes such as time to activate or deactivate a role, qualification of role, possibility of delegating or transferring describe the inherent aspects of a role without explicit reference to other objects. A relevant set of constraints can be used as a part of a role attribute. Kumar et al. [98] extended RBAC by introducing the notions of role context and context filters. In CS-RBAC (context-sensitive RBAC) model [98], a permission is defined as an authority to perform a specific operation on a class of objects. A role is then defined as a collection of permissions. A user is granted a role context that is a valid role membership within a certain context. The role context can be defined by system administrators at role creation time. The decision to allow or forbid an operation is made at runtime depending on the context at that time. The attributes that constitute users and object contexts are application-dependent. This gives flexibility to application developers to define attributes in particular context according to environment. With the same approach of using contextual information in access control, Georgiadis et al. [72] extended TMAC to C-TMAC (context-based TMAC) by integrating RBAC, TMAC and contextual information such as time of access, the location from which access is requested, the location where the object to be accessed resides. Similar idea of context aware access control was presented by Covington et al. [42] as a further extension of RBAC for security in context-aware applications. The contextual information is an important aspect of collaboration and it includes many types of information such as task, resource, users, time and so on. The access control model is shown to be applied in ubiquitous computing environment. Cuppens and Cuppens-Boulahia [44] proposed a model using the notion of context in OrBAC model. Security rules (permission, prohibition, obligation or dispensation) are specified by administrators to apply to a given context. Provisional context was defined to model permissions, obligations and prohibitions depending on past actions of a user. A given taxonomy of contexts helps to identify different types of context that the data in systems must be managed to deal with those contexts.

Credential-based and trust-based access controls are other ABAC models extended from RBAC for decentralized, open and distributed systems. Credential-based access control models, for example, PolicyMaker [30] (using X.509 digital certificates), Winslett et al. [190], KeyNote [29], DataLog [103], use credentials to give access privileges to users. Credentials contain attributes of

users (e.g. role within organization, membership, delegated permission). Credentials implement the notion of binary trust.

Trust attribute was integrated into RBAC to incorporate the advantages of role based model and credential based models. Sandhu et al. [163] proposed a trusted architecture to enforce access control policies in a peer-to-peer environment. Adams and Davis [8] contributed an alternative access control system that is based on trust and is more suitable to ad-hoc collaborative environments. Access decisions are made based on trust and distrust thresholds without the need of a central authority. Each node in a peer-to-peer network maintains an individual trust profile of other peers. Peers calculate reputation of others based on their observation on others. In TrustBAC model [36], the history of user behaviors is used to assign trust levels which are then mapped to roles of RBAC to determine access privileges. The model does not preclude use of credentials for trustworthiness evaluation. It is well suited for open systems. Frikken et al. [66] proposed a protocol built on hidden credentials to help users gain access rights without compromising privacy of access control policies and credentials. For example, when Alice wants to access to data provided by Bob, she provides to the protocol a subset of her credentials. If the attributes in the credentials satisfy the policy, she gets access. The protocol is a solution ensuring that Alice learns as little as possible about Bob's policy and Bob learns as little as possible about Alice's credentials as well.

Purpose-based access control is another ABAC in which the attribute "purpose" is used to make access decision. Purpose is a central concept in many privacy access control models for database systems [151, 101]. The notion of purpose was defined by Byun et al. [33]. The concept of Hippocratic databases was introduced by Agrawal et al. [151] for privacy protection within relational database systems. In [151], the structure Strawman Architecture consists of privacy policies and privacy authorizations to define usage purposes. Lefevre et al. [101] presented an approach for enforcing privacy policy in database systems to let providers have control on what people are allowed to see their users personal data and for what purpose. Byun et al. [33] presented a model in which purpose is associated with given data element to specify the intentional use of the data in relational databases.

Although ABAC is a valuable candidate for securing collaboration in decentralized, open and distributed environments, it has several drawbacks. ABAC with context focuses on access

rights for systems instead of for human factors. ABAC with credentials does not guarantee the behavior of bearers between the time it was issued and its use. Additionally, credentials do not keep track of user actions and thus users cannot be rewarded for good behavior and punished for misbehavior. In ABAC with purpose, even though privacy policies are defined for which purpose data can be accessed in future time, the real access is only given after the access purpose was checked according to the purposes that have been associated *a prior* to the data item. This is not well suited for collaboration where users need to have information and resources in a more convenient way.

3.2.3 Optimistic Access Control

Apart from standard access control models that are considered as pessimistic models, optimistic access control can be seen as a paradigm for authorization in situations that are unforeseen and the systems cannot be made aware of. The optimistic approach trusts human beings and assumes most accesses will be legitimate or trustworthy. The optimistic access control usually allows users to exceed their privileges in a way that can be securely audited and the system is able to be rolled back. For example, in health-care systems, the doctors might be allowed to exceed their rights accessing to the personal medical record of a particular patient in case of emergence that puts the patient's life at risk. Optimistic access control offers a greater access to personal information and makes it easier to detect and defect after the fact with logs and notifications. Optimistic access control is suitable to a system where the risk of failure and the cost of recovery is less than the cost of not granting access in a particular situation. For instance, it is suitable to give optimistic access to private medical information to save one's life, but it is not suitable to give optimistic access to a financial system because the cost of fraud is high.

Povey introduced the concept of *optimistic security* and presents an optimistic access control scheme with clear requirements for optimistic security [143]. As presented in [143], optimistic access control is based on the following idea: *"Optimistic access control takes the approach of assuming that most accesses will rely on controls external to the system to ensure that the organization's security policy is maintained. [. . .] In an optimistic system, enforcement of the security policy is retrospective, and relies on administrators to detect unreasonable access and takes steps to compensate for the action. Such steps might include: undoing illegitimate modifications,*

taking punitive action (e.g., firing, or prosecuting individuals) or removing privileges". Povey's approach includes five preconditions: constraints, accountability, auditability, recoverability and deterrents. The formal model supports these requirements and a number of applications of the optimistic security model was given also. Stevens and Wulf motivated the use of optimistic access control for an inter-organizational environment [178]. From a privacy standpoint, Hong and Landay identified common features that need to be supported in privacy sensitive computing in which three basic interaction patterns for privacy sensitive applications are mentioned: pessimistic, optimistic, and mixed-initiative [83]. Padayachee and Eloff enhanced optimistic access controls with usage control to ensure that users behave in a trustworthy manner [134]. Motivated by use cases from disaster management systems and medical information systems, the *break-glass* concept was introduced in [1] as a solution for access control. Break-glass is an optimistic mechanism that allows extending person's access right in exceptional cases. While standalone policies express the conditions under which access permissions are granted, break-glass policies express the conditions under which right overrides are allowed. In break-glass based control, the use of emergency account is monitored by audit mechanism. This assures that activities performed using emergency account are done by authorized individuals, hence creates accountability and non-repudiation for the system. Related works on break-glass access control can be found in [31, 117].

In the CSCW domain, Shen and Dewan [174, 54, 53] made a major step overcoming the shortcomings of traditional access control. In [53], the authors distinguish between pessimistic and optimistic access controls derived from optimistic concurrency control. Concerning collaboration rights (write, view, couple objects), pessimistic access control prevents any changes to objects that cannot be later committed whereas optimistic access control checks the local changes on object for access control violation when local changes are committed. Users are allowed to do local changes on objects even they do not have the rights.

The idea of optimistic access control is very suitable for collaboration environments. However, prior works focus on securing systems rather than play a role as solutions for end-users and require external resources such as human factors to audit and perform system recovery if necessary. Maintaining human factor or a central authority is not well suited for multi-synchronous model. Furthermore, the recovery mechanism is not always possible, for example, when secret data has been disclosed, there is no way revoke it.

3.2.4 Computer-Supported Access Control

Following the CSCW research paradigm, Stevens and Wulf argued that access control should be designed as a supporting system rather than the automation paradigm [179]. The authors coined the term *Computer Supported Access Control (CSAC)* for an innovative access control mechanism which is adequate for cooperative work settings. The CSAC allows broadening the perspective on traditional access control on a conceptual level. The issue of access control is reconsidered on a theoretical, methodological, and practical level.

The empirical findings for CSAC indicate temporal constitution of access control distinguished at three points in time: (1) *ex-ante*: the legitimacy is defined before an access is attempted regardless of a specific situation. RBAC adopts this concept. (2) *uno-tempore*: the permission is defined at the moment of the access attempt depending on the context of a concrete situation. Context-based access control adopts uno-tempore control. (3) *ex-post control*: permissions are checked after access was granted. In ex-post control, resource owner sets little or no access restrictions ex-ante. Instead, it relies on the accountability to ensure that resources are used in a responsible way. However, all accesses to resources will be logged and the resource owner can review actions after they have happened. Optimistic access control adopts ex-post control. It allows evaluating access from the perspective of future events.

Beyond the temporal constitution control, in CSAC, the patterns of interaction between three entities that constitute control, i.e. the controller, the accessor and the resource, are taken into account. Three basic types of interactions are pointed out: awareness, protection and negotiation, in which: (i) *awareness:* the access to resource can be observed by others to provide accountability. The possibility that illegitimate access can be detected could have impact on restricting people's access behavior. Awareness is thus useful in uno-tempore settings and in ex-post mechanism as well. Awareness is very important when the cooperative work settings are distributed; (ii) *protection:* controllers regulate access for accessor on resources; (iii) *negotiation:* human factors can negotiate an access to resources. Negotiations are held between controllers and accessors. It can be conducted ex-ante, uno-tempore and ex-post, regarding the access request.

In addition, CSAC needs to fulfill a number of requirements. It should integrate different mechanisms to cover all types of temporal constitution control from ex-ante to ex-post. Also the

CSAC should make human users aware of access attempts and support visibility and traceability. Furthermore, CSAC should provide channels of communication and support an assembly of different mechanisms.

The idea of CSCA is very suitable for usage control in multi-synchronous models. However, CSAC model is too abstract as a conceptual model rather than a practical model that can be applied straightforward. Moreover, CSCA model does not cover obligation factor that we need for contract-based model. Recently, CSCA is adopted in a number of works for collaboration such as access control for home data sharing [119], access control for weakly consistent replication [191] and access control in crisis situations [102]. However, these works have no mention about obligation that is one part of contract.

3.3 Usage Control Models

Usage control can be considered as an evolution of access control in the last decade. The term *usage control* was firstly introduced by Park and Sandhu with the notion UCON [138] for controlling access to and usage of data. At the beginning the term *usage* means usages of rights on digital objects. The UCON model [138] consists of three core components (subjects, objects and rights) and several additional components (authorization rules, conditions and obligations). The core components can be associated with several attributes. These components are involved in authorization process. Obligations have to be fulfilled by a subject to get access. Access is granted when a subject holds certain rights on specific objects.

The family of ABC models [165, 137] derived from the UCON model presents three decision factors, authorizations (A), obligations (B) and conditions (C). Obligations are required to be fulfilled when a human factor gained access, for example, clicking "accept" on a license agreement is obliged before downloading a software. Conditions represent factors when an access is allowed, for example, a specific time period when an access will be accepted. The UCON model presents two aspects related to the usage control, the mutability attributes and the continuity of an access decision. The mutability of attributes implies that attributes can be updated at different times before (pre), during (ongoing), or after (post) the right is exercised. For example, the attribute of how many times a video is allowed to play changes by a decrement each time a user plays

the video. Mutability of attributes as a side-effect of the usage process is the main property of the $UCON_{ABC}$ model. The continuity of decision implies that the UCON model enforces the security policy not only prior to access but also during the time interval that access takes place. In case access attributes are no more valid during the authorization, the system revokes granted rights and stops the usage of accessor on the data or resources. The ABC model is more general than attribute-based access control since it is based on not only the attributes of subjects and objects but also the fulfillment of obligations. A family of $UCON_{ABC}$ models can be found in [137, 198, 199, 109, 200].

Unlike UCON which covers access control that the control to data is granted only if access conditions are satisfied, usage control was extended to what data will be handled after access has been granted. The term "distributed usage control" was first introduced by Pretschner et al. [146] to address specific issues on the protection of digital content between providers and remote consumers. Distributed usage control relaxes the requirement of having a central authority. It allows to formalize post-obligations which must be fulfilled in a specific point of time in future. This is close to digital rights management (DRM) models regarding the protection of digital information when it is shared, copied and distributed in an open environment. In [146] Pretschner et al. distinguished also controllability and observability and enforceability is bound by these notions. Controllability is possible only with a given set of mechanisms. Observability is introduced in case that the full controllability is not achievable. For usage policy and from the obligation aspect, contracts which reflect obligation in the high level policy are classified as controllable contract and observable contract. A set of works in distributed usage control can be found in [146, 81, 147, 148].

Usage control is a suitable approach for open and distributed environments (e.g peer-to-peer, grid, web services, ad-hoc networks, cloud, etc). It is an active area with many proposed frameworks and models in the literature. A survey of usage control models [100] describes usage control in four independent layers starting from the high level specification to the low level of enforcement mechanism and implementation. An idea to adapt usage control as a deterrent to address the inadequacies of access controls can be found in [135]. Interesting applications of usage control for collaborative systems were given in [199, 200, 177].

Existing usage control models are either for *ex-ante* access or *ex-post* access with a fix set

of usage scenarios only. They do not mention about data usage when data is replicated and modified by diverse users over time to time as in multi-synchronous collaboration models. In these models, data usage rules must be as one part of replication mechanism and a requirement to resolve conflicts between usage rules (contracts) needs to be addressed.

3.4 A Posteriori Compliance Checking

Collaborating participants often need a guarantee that their data is used according to established policies (contracts) such as *"this document may be shared further and cannot be modified by the user X"*. Similar to the distinction of controllability and observability [146], the enforcement mechanism of contract was classified by Etalle and Winsborough as either *preventive* or *detective* [61]. In [61], *a posteriori* policy enforcement is mentioned as a mechanism that offers interoperability, flexibility and scalability, which is crucial in collaborative environments.

The approach to preventive enforcement is commonly used in access control or DRM. It ensures that policies will not be violated since unauthorized actions are prevented before occurring. This approach is too pessimistic and it should be used only to prevent actions that must not happen because of the high cost of failure. Furthermore, in collaborative environments where participants are heterogeneous, it is very difficult to put unanticipated circumstances into preventive policies and to ensure polices can be enforced across such different systems as well. In [61] Etalle and Winsborough pointed out the main requirements that the collaborative environments impose on policy enforcement systems. Let's assume participants collaborate on a document. The first requirement is that the policy will be attached to the document and will be enforced during its whole life time in the system. Then users can modify, merge and share the document in a dynamic cooperative system. When the document is owned by different users, the original owner should be able to define policy freely and any further owner is able to restrict this policy. The last requirement is that users should be allowed to perform actions that are normally not permitted, but they will be made accountable for their actions later on. This requirement reflects the shortcoming of the strictly preventive approach.

The *detective enforcement* is a more flexible approach than the *preventive enforcement*. Let us take an example from the real life. There is nothing preventing a person from passing a red

light or exceeding the limit speed in a city, even both actions are prohibited by law. Yet this person knows doing these actions breaches the law. Using GPS with speed limit warning might remind her to control her action to comply the law, but it cannot prevent her from breaching the law. Moreover, we should not prevent this action *a priori* because in some case, it is allowed to do that. For example, in case of emergency, the exceeding speed limit of an ambulance to arrive at hospital quickly is legal. In these above examples, what the system enforces the person is actually the reaction (e.g punishing, blaming, reducing trust) after the fact when it can determine that she broke the law. This is clearly not done in a preventive manner but in a detective manner.

Detective enforcement is an approach to enforce policy not by preventing unauthorized use but rather by deterring it. In a system where participants can be made accountable, deterring illegitimate action is almost effective as preventing it. Individuals are encouraged to act carefully to avoid potential vulnerabilities. For example, the policy *"this document is protected by AAA security license"* makes a user deterred when she intends to behave badly because she will have to take responsibility for her misbehavior according to the security law afterward if the misbehavior is exposed. In term of temporal checking we can consider preventive enforcement is *a priori* checking while detective enforcement is *a posteriori* checking. To make users accountable, the log auditing mechanism can be adopted to check whether users behave in accordance with the applicable policies or contracts. When being audited, a user is checked if she holds policies that allow her to do the actions that she has carried out, and if she fulfills the obligations according to the policies she was given. Using logs to verify users actions are common in many works [130, 35, 79, 94, 86]. To provide a deterrence to users, a trust management can be used together with log auditing and user accountability. This encourages users to behave well in order to gain high trust level in collaboration with others.

Even though *a posteriori* check presents great advantages and is more suitable for collaborative environments, it does not offer an absolute guarantee that data will not be misused. If it is important to protect data from any misuse, preventive enforcement is more appropriate. The two approaches *a priori* and *a posteriori* check, however, are mutually complementary, hence they can be integrated together in systems to provide flexibility while still ensuring that some important data will not be used illegitimately.

3.5 Summary and Discussion

From what we have surveyed, we can see that existing works are inadequate to our requirements for multi-synchronous collaboration. The traditional access control is too pessimistic in considering that users are not trusted in requesting data for the right purpose. Moreover, most of approaches were mainly applied for centralized systems where policies can be verified by a central authority. However, in multi-synchronous collaboration models, there is no central authority that can audit users. The optimistic model is well-suited to collaborative environments where users need a certain level of mutual trust to collaborate with each other. However, they were not designed to deal with issues in multi-synchronous collaboration where users can work concurrently on shared documents with replication mechanism and data synchronization. They do not deal with merging policies and resolving conflicts among contracts.

We aim at a solution for end-users. Our target model integrates the ideas of optimistic access control and usage control. A user can proceed even if either the pre-conditions, pre-obligations, ongoing conditions or ongoing obligations are invalid. However, the system will notify the user that the action is invalid and the user has to take responsibility for the misuse since all actions are logged. The user is made accountable for her misuse in next assessment cycle. In term of enforcement, usage control policies can be enforced by using a detective enforcement or a preventive enforcement. Our work adopts the detective enforcement. We are not preventing users from violating contracts; instead we make each user aware of her given contracts and contract violations of others. The auditing result will be used to evaluate user trustworthiness. Although our model deals with handling data after it was released, the model can be used with access control mechanism for granting access to data before its usage.

Chapter 4

C-PPC Model

The main issue in designing a contract-based multi-synchronous model is that contracts are objects of replication mechanism. In our contract-based model each user maintains a local workspace that contains local data as well as modifications done on the shared data and contracts related to the usage of shared data. The logged changes and contracts are shared with other users. The merging algorithm and the conflict resolution have to deal not only with modifications on data but also with contracts. For checking if users respect contracts, each user performs a log auditing mechanism. According to auditing results users adjust their trust levels they assign to their collaborators. In this chapter, we present main parts of our C-PPC model: formalization of log and contract, collaborative process, merging algorithm and conflict resolution.

4.1 Push-Pull-Clone Paradigm

As stated in previous chapter, multi-synchronous is a very generic model of collaboration. In this thesis, we instantiate it by a push-pull-clone (PPC) paradigm.

In PPC model people can work together even they are geographically distributed with different computing resources. In this system there is no need of a central server to be set up and run. We mention here the basic concepts of push and pull as basic data dissemination methods in a peer-to-peer network.

- Push is a communication when data transfer initiated by a sender while pull conversely is a communication when data transfer initiated by a receiver.

- Push-pull communications are not necessary to be synchronous but also can be asynchronous among heterogeneous working sites. By "asynchronous" we mean that pushing and pulling do not need to be done synchronously. Instead, a sender can push her data to a channel and a receiver can come and pull at a later point of time. In this push-pull communication a sender has complete control what data is delivered and when it is delivered.

In some existing collaborative systems such as DVCS, PPC model is widely used. It follows multi-synchronous model which supports users working simultaneously on different streams of activity on the shared data. In the PPC model, users replicate shared data, modify it and redistribute modified versions of this data by using the communication primitives push, pull and clone. These primitives are used for managing divergence and convergence of different streams of activity. We model push-pull-clone for a wider range of collaboration application rather than DCVS only, where:

- **Clone.** In multi-synchronous style, there is no point that plays central role of the whole system. The same data object (e.g project, database, document, and so on) that people are collaborating on is replicated as many instances. After the object is created at first time and put at a particular repository, it can be copied as a new instance by others by doing *clone* operation. This is a communication primitive in PPC model. Users clone shared data and maintain it in their local workspaces. Every user has her own private repository instance. By this way, most of operations are done locally and maintained in local repository instance. They are only propagated through networking to be synchronized with other repository instances. In the PPC-based system, there are multiple repository instances and their roles are equal. It provides more flexibility than in systems that rely on a central server, however, it does not negate central server-based systems. It means that in PPC model it is possible to maintain a central server for a team as optional decision.

- **Push.** The push primitive is used when users want to send their local changes to different repository instances (usually remote repositories). Pushing can be done at any time, however, users can only push changes to repository that they have grant to push. The user who manages a remote repository can give some rules about who can push changes to the

repository. Since pushing is done to particular repository instances, it is thus not necessary that all repository instances are identical after a push.

- **Pull.** Pull operations are usually used to synchronize two different repositories, often between a local one and a remote one. Specifically a user pulls change sets from a remote repository to her local one. Similar to push, after pulling it is not necessary that two repositories are identical, for example, a local one can contain some change sets that are not in the remote one.

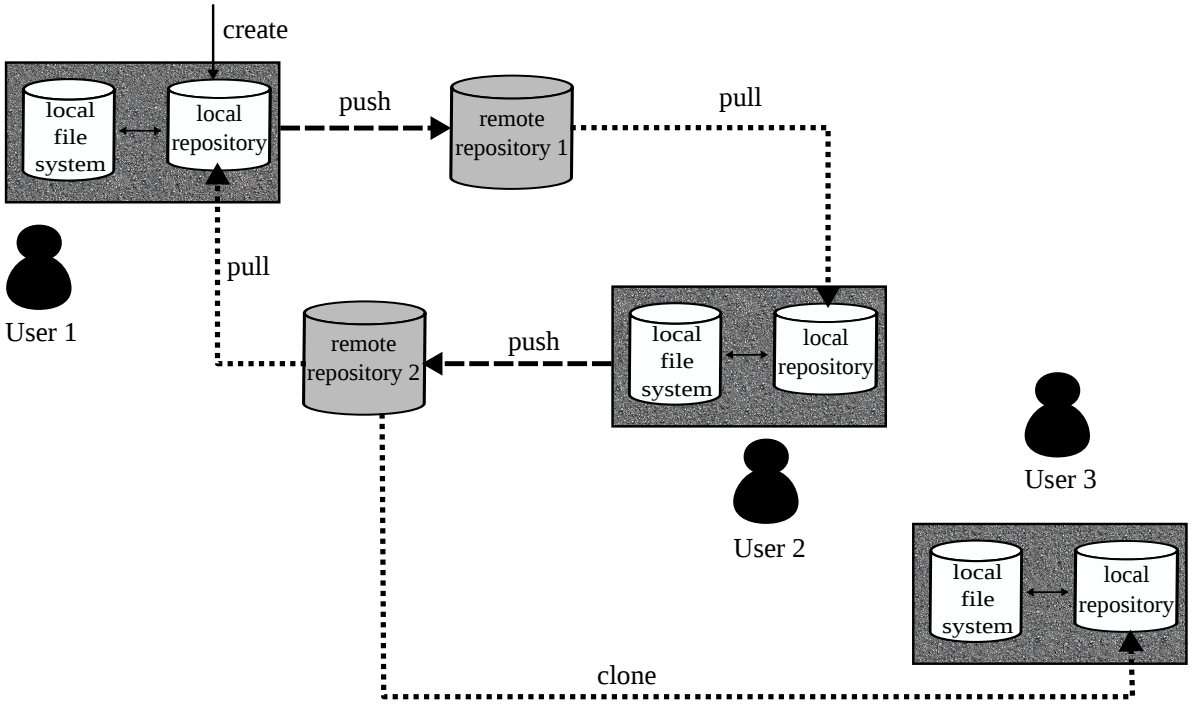


Figure 4.1. Push-Pull-Clone paradigm between three users.

In Figure 4.1 we give an example of the PPC model with three users and each one has a local workspace that she maintains at local site. Every operation the user performs on local file system is updated only to local repository before it is sent to other repositories. In this figure, two users *user1* and *user2* interact with each other by first creating and cloning repositories, and then using push and pull primitives to exchange changes. They do not need to interact over a central remote repository, instead they can push and pull from different remote repositories. We take an example of using different repositories to show the decentralization supported by the PPC model. While *user1* only pushes to repository 1 and pulls from repository 2, *user2* does

the converse process by pulling from repository 1 and pushing to repository 2. This shows the flexibility of PPC model that does not need to maintain any central server. In the figure we can see also that *user3* performs a clone from *user2* at a later point of time to join the collaboration.

Push, pull, and clone primitives are used for efficient distributed collaboration and they were already implemented in distributed version control systems such as Git and Mercurial. We assume the system uses a pairwise FIFO channel between two users for changes propagation to guarantee that messages are received in the order they were sent. This order can be preserved by using logical timestamps to sort messages into chronological order. Push, pull and clone communication primitives are operated on such ordered channel.

In Figure 4.2 we present a workflow in PPC model from a single user view point. Most of writing operations such as commit, update, merge, on the data are done locally by each user while communicating push, pull, and clone are done in interacting with remote repositories. Modifications are done on local file systems and then committed to local repository. This can be repeated in unlimited rounds without the need to have connection with a remote repository. However, user can push the changes to remote repository and pull new changes from it at anytime as she wants.

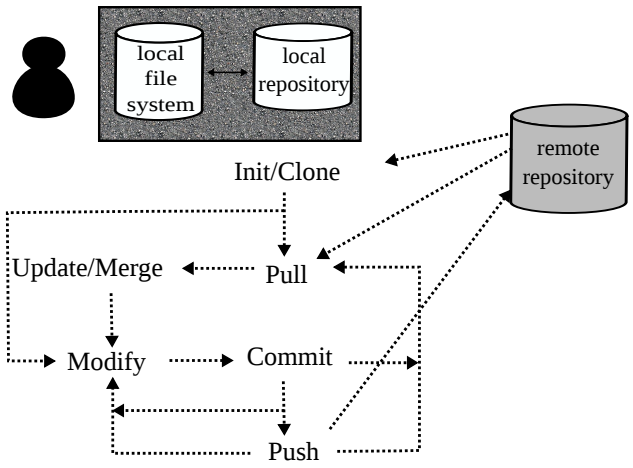


Figure 4.2. Push-Pull-Clone paradigm from a single user's view.

The PPC model is a very general collaboration model without a collaboration provider where users share their data only with people whom they trust. It generalizes the collaboration model with a service provider where users interact only with a server that forwards the changes to the other users.

The PPC model offers some advantages over traditional model with a central provider. First, it allows users working in private workspaces with offline mode. The model gives users the independence and an active role in collaboration rather than depending on the network connection or the aliveness of a remote server. Therefore, it provides the high performance and high availability, especially in environments where high networks latencies and disconnections make on-the-fly updates difficult. Second, it provides a very flexible workflow with which users can work whenever and whatever as they want. Third, it provides an implicit back up mechanism. Rather than relying heavily mechanism for back up data on a central server, in PPC model the chance to lose data is much reduced when data is replicated in multiple repositories.

4.2 Optimistic Replication

We have seen that multi-synchronous collaboration allows users working isolately in private workspaces. However, as the final goal is to achieve the same outcome, individual works need to be synchronized. Replication appears as a technique that is used commonly in large-scale or mobile systems. The term *optimistic replication* was first introduced by Saito and Shapiro [162]. Apart from pessimistic replication that needs synchronous coordination of replicas, optimistic replication allows replicated data to be accessed and updated separately before the replicas are synchronized. This is based on the optimistic assumption that conflicts of updates on divergent replicas can be fixed later during the synchronization. It lets users share a common data object through replication and update their local replicas independently. It provides weak consistency to ensure availability to each user. The main elements of replication are as belows.

- *Replica*: a copy of a data object that is stored at single sites. A site can store multiple replicas of different data objects. To simplify the description of replication mechanism, we use the terms *replica* and *site* interchangeably in some replication algorithms.
- *Operation*: an update a site performs to a replica. Replication systems can offer updates over the whole replica or more fine-grain updates to parts of the replica.

Optimistic replication has a wide spectrum of applications in different domains as cloud computing (e.g Dynamo [50], OceanStore [97]), databases (e.g Bayou [140]), file systems (e.g Coda

[166], Ficus [142], Microsoft WinFS [114]), directory services (e.g Grapevine [26], Microsoft Active Directory), and distributed version control system (e.g Git [107], Mercurial [132], Pastwatch [197]).

4.2.1 State-based and Operation-based Replication

Optimistic replication can be classified into state-based and operation-based. In state-based replication, each site applies updates to its replica without maintaining a change log file and the replication is driven by the current state of the source replica. Techniques for constructing synchronized system such as state-machine replication [169] were well understood. The state-based approach is in wide use in file system such as NFS or Coda [166] and Active Directory in Windows Server. Each site executes an update to modify the state of a single replica. Every site sends its local state to some other replicas that can merge the received state with their own state.

In operation-based replication, each site keeps a log of the updates and the site communicates its log to other sites. After a log has arrived at a replica, the replica applies the log to obtain a more up-to-date state. Operation-based approaches are used when the cost to transfer state is high such as in databases, mobile systems and when operation semantics are important such as in Bayou [140] or IceCube [95]. Unlike maintaining consistency in state-based replication which simply involves only sending the newest version to other sites, in operation-based replication, the system must maintain a history of operations. Operation-based approach is more efficient when replicated objects are large and operations are at high granularity or when network traffic is low such as in mobile environment. Moreover, operation-based approach allows more flexible conflict resolution [162, 105]. However, operation-based replication has some limitations such as overhead of replaying all operations. A hybrid approach would reduce these limitations as a system may keep a recent history of operations and convert too old operations to state. Since hybrid approach is considered as an optimization of operation-based approach, we do not consider this approach in this thesis.

4.2.2 Eventual Consistency Guarantee

Data consistency are essential feature of large scale distributed systems. In this thesis, we consider the consistency guarantees related to data replication. The consistency of replication

means if two replicas of an object start from the same initial state, and have the same sequence of operations applied to them, the two replicas will reach the same final state. Gillbert and Lynch proved in [74] a conjecture of professor Brewer: *"it is impossible for a web service to provide three guarantees: consistency, availability and partition-tolerance for the asynchronous network model"*. Vogels in [183] claimed that in large distributed systems (partitioned) there is always a trade-off between high availability and data consistency and it is not possible to achieve both consistency and availability at the same time. The systems that need to provide high performance and high availability are said to offer weak/eventual consistency (in contrast with strong consistency). That means the state of replicas will converge only eventually. Sites are allowed to update local changes and continue working on the new updates without requiring the update to be synchronized with other sites. The local updates of sites are exchanged and applied in the background at each site. Later, these updates will be sent to other sites and the consistency is achieved eventually. The eventual consistency are presented formally in [162, 173].

Optimistic replication adopts eventual consistency. Unfortunately, in weak consistency systems, conflict arises when replicas are synchronized and handling conflict is often hard. To overcome this problem, conflict-free replicated data type (CRDT) (former it is known as commutative replicated data type) was firstly introduced in [145]. The basic idea of CRDT approaches is to design a replicated data type that allows concurrent operations pair-wise to commute from the start. This property tolerates conflict when concurrent operations performed at local sites are synchronized regardless replay order. There are different proposals for CRDTs that have been published recently [145, 188, 158].

4.3 C-PPC Model through An Example

In this section we present an overview of our proposal that extends PPC model by using contracts (so-called the contract extended PPC model or C-PPC). Our target is a collaboration model that requires high level of respect and trust among users. To create a trustful and respectful collaborative environment, collaboration contract will be used in all collaborative interactions.

We turn back with the real world example of building collaboratively a photo collection in previous chapter. At the beginning, the network is built based on social trust between

users and connections are established only between users who trust each other. Users trust their collaborators with different trust levels that are updated according to their collaboration experience. For instance, Sheldon trusts both Raj and Penny, however, with different trust levels, and thus he gives them different contracts over the shared document. For example, Sheldon gives Raj the permission to edit, while he gives Penny only the permission to share the document. Receivers are expected to follow these contracts; otherwise, their trust levels will be adjusted once misbehavior is detected. We log changes that users do on the photo collection with contracts that they receive from others when they receive their changes. Assume that in the time interval [*Time 1*, *Time 2*], Leo, Raj, Penny, and Sheldon perform their local changes on the collection.

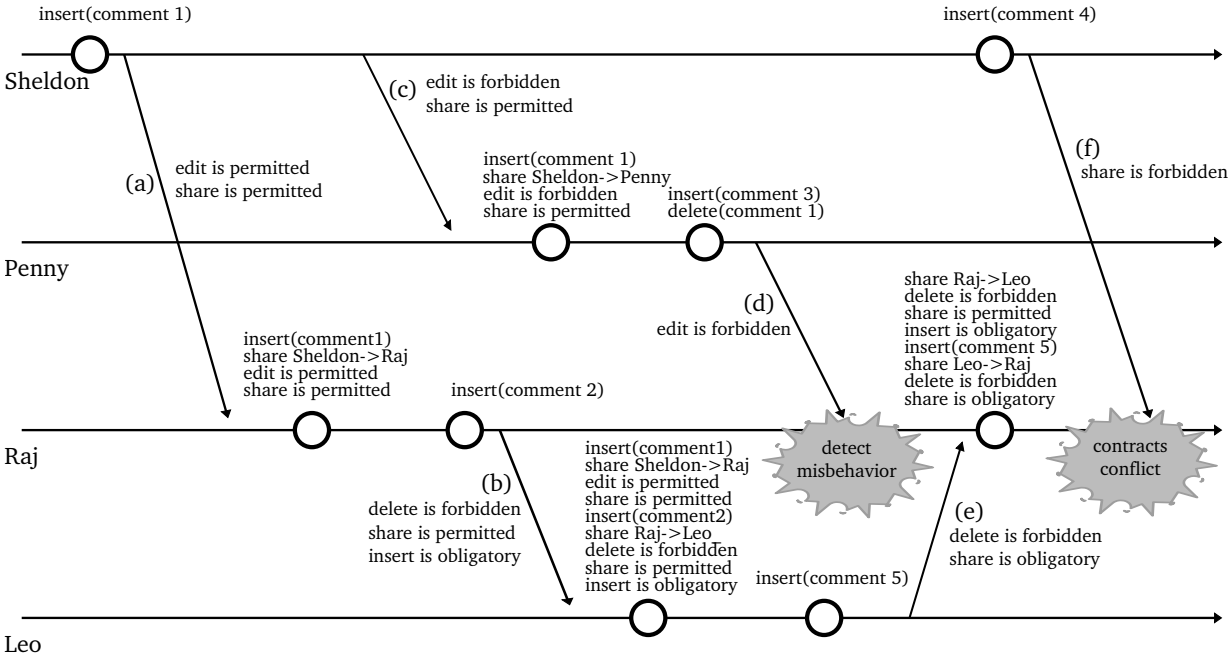


Figure 4.3. A Push-Pull-Clone collaboration scenario of four users over a photo during time interval [*Time 1*,*Time 2*].

Let us assume that trust values at a time instance *Time 1* are shown in Figure 4.4 in which values are real numbers ranged in the interval [0, 1]. However, trust values change over time based on user's assessment. The values we take in this example show the personal trust each user has on others. A user has no global knowledge of the trust values that each user assigns to others, but knows directly only the trust values she has assigned to other users.

In Figure 4.3 and Figure 4.4, after *Time 1*, Sheldon trusts Raj with a trust value 0.6. Sheldon makes a change on photo A by adding a new comment *comment 1*. He shares his change on

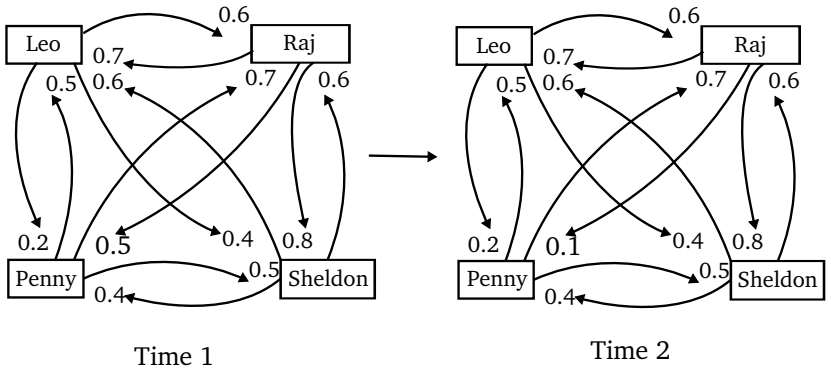


Figure 4.4. Trust values at two different time points *Time 1* and *Time 2*.

photo A with Raj with the contract {*edit is permitted*, *share is permitted*} (step a). The change and the contract are pushed to a communication channel with Raj. As it is the first time that Raj initiates a communication with Sheldon, he has to clone photo A with changes from Sheldon. Raj now has in his local workspace a clone of the photo collection from Sheldon, on which he can work in isolation. In the example, Raj adds another comment to the photo, *comment 2*. Since he has the right to distribute further the data, he then shares it with his trusted friend, Leo (step b). Raj wants Leo to redistribute the collection of photos only after adding his own comment on photo A. He therefore shares his data with the contract {*delete is forbidden*, *share is permitted*, *insert is obligatory*}. Concurrently, Sheldon collaborates also with Penny. He trusts her less than Raj and he wants to forbid her from editing the photo while allowing her to share it. Sheldon specifies the contract {*edit is forbidden, share is permitted*}. Penny thus has no right to edit photo A after she receives it from Sheldon (step c).

Penny violates the contract she received from Sheldon by deleting his comment and adding her comment, *comment 3*, to photo A. She continues to collaborate with Raj by specifying the contract that he is forbidden to delete any comment on the photo (step d). As soon as Raj receives the changes from Penny, he discovers that she misbehaved. He thus updates the trust value on Penny. The Figure 4.4 shows that her trust level at *Time 2* is decreased to 0.1.

In parallel, Sheldon adds a new comment, *comment 4*, to photo A. He wants the comment is kept private except for Raj. He thus shares the photo with this new comment with Raj under the restriction of not sharing it further, {*share is forbidden*} (step f). Raj receives the data with the new contract from Sheldon. This contract conflicts with the contracts he holds after

synchronizing with Leo (step g) which is *share is obligatory*. As Raj wants to be able to further modify and share the photo, he decides to resolve the conflict, for example, by discarding the changes from Sheldon, or by negotiating to relax the prohibition of sharing.

In the example, Leo behaves well by always respecting contracts he has received. He adds *comment 5* to photo A and shares it with Raj by restricting him not to delete the added comments with the contract {*delete is forbidden*, *share is obligatory*} (step g). After these interactions in the time interval [*Time 1*, *Time 2*], user trust values are illustrated as at *Time 2* in Figure 4.4.

We have given an illustration of how the C-PPC model can be used for the collaboration between four users, Leo, Raj, Penny, and Sheldon with the assumption that users trust each other at certain levels. We move next to the formal representation of the C-PPC model.

4.4 The C-PPC Model

In this section, we describe main elements of the C-PPC model, i.e. users, logs of operations on shared document and contracts between users.

4.4.1 Model Overview

The general idea can be described as follows. In PPC model, without trust management, users might share and reconcile replicas with all group members at the same implicit trust level without taking into account if they will behave well or not. In the C-PPC model users are not uniformly trusted and a user generally collaborates only with trusted users at certain trust levels. Users trust their friends with different trust levels that are updated according to their collaboration experience. In this collaboration receivers are expected to follow given contracts; otherwise, their trust levels will be adjusted once misbehavior is detected. Our model is adaptable to the case when a user changes his behavior from good to bad as his trust is reduced and other users will reconsider to collaborate with that user. We log user modifications on the shared documents as well as contracts users give to other users when they share their modifications. Users trust levels are adjusted mainly based on their past behavior. A misbehavior detection mechanism and trust model to manage trust levels of collaborators are mandatory to maintain correctly such trust-aware collaborative work.

Without loosing the generality, we consider data object that users are collaborating as document regardless it can be a single document, a structured data as file system, or a photo collection. For the sake of simplicity, we focus on the C-PPC model for a single file. The issue how it is used for collaborating on multiple files will be discussed at the last section of this chapter.

The C-PPC model uses push, pull and clone as native direct pair-wise communication primitives among users. To work with others a user simply sets up a local workspace for his own work, and uses trusted channels to *push* his document to trusted friends. Other users can then get the document by cloning it from the user's workspace. In this way they have independent local workspaces for the shared document, do their changes locally on their replicas and publish them by executing a *push* primitive. Another user then executes a *pull* primitive to get the changes into his local workspace.

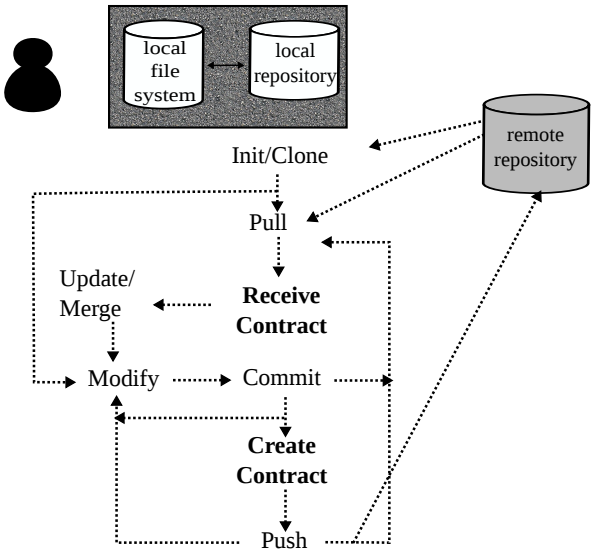


Figure 4.5. Workflow in C-PPC model.

Figure 4.5 gives a workflow of the proposed C-PPC model by extending the existing PPC model. Users are main participants in the C-PPC model. They are connected in a collaborative network based on the trust they have in other users. However, users are not uniformly trusted by others. Each user keeps at her local site the replica of shared document and works locally on the replica. After receiving or before sending local changes to remote repositories, the user has to manage contracts.

4.4.2 Document, Changes and Logs

The system keeps a document as a log of operations that have been done during the collaborative editing process. The log maintains information about user contributions to parts of the document and when these contributions were performed. The outcome of collaboration is a document that could be obtained by replaying the *write* operations such as *insert*, *delete*, *update* from the log. Two users can write independently on the shared document. Changes are propagated in weakly consistent manner that a user can decide when, with whom and what data is sent and synchronized. Push, pull and clone communication primitives are operated on FIFO channels for an ordered exchange of operations done on document replicas. A replica log contains all operations that have been generated locally or received from other users. Logs are created and updated at user sites. The log structure is defined in the following definitions.

Definition 4.4.1. *(event) Let* $\mathbb{P}$ *be a set of operations* {insert, delete, edit, share} *that users can generate; and let* $\mathbb{T}$ *be a set of event types* {write, communication, contract}*. An event e is defined as a triplet of* ⟨evt $\in \mathbb{T}$, op $\in \mathbb{P}$, attr⟩*, in which* attr *includes attributes which are in form of* {attr_name, attr_value} *to present additional information for each event.*

Each event is parameterised with an operation and its attributes varying from applications. A shared document can be as large as a database (e.g. Bayou) or as small as a single file. Basic operations range from characters, lines, paragraphs to deltas between revisions.

Definition 4.4.2. *(log) A document log L is defined as an append-only ordered list of events in the form* $[e_1, e_2, \ldots, e_n]$.

Users store operations in their logs in an order that is consistent with the generated order. The event corresponding to a *share* operation of type *communication* is issued when a user pushes her changes and it is logged at the site of receiver when this one performs a pull. This *share* event can be followed in the log by an event of type *contract* representing usage policies for the shared data.

In Figure 4.6 we give an example of a log containing a single event that has three attributes. The log is presented in XML format. The *write* event refers to insert operation and belongs to

```
<log> <!-- at local site of Sheldon -->
  <event>
   <evt>write</evt>
   <op>insert</op>
   <attr>
      <by>Sheldon</by>
      <content>comment 1</content>
      <gsn>1</gsn> <!--generation timestamp -->
   </attr>
  </event>
</log>
```

Figure 4.6. An example of log with one event in XML format.

write type. The event has attribute $\{by, Sheldon\}$ (done by Sheldon), $\{content, comment1\}$ and $\{GSN, 1\}$ for the sequence number when the event is generated.

The event attribute *GSN* (*generate sequence number*) of an event is assigned at the site where the event was generated. The event attribute *RSN* (*receive sequence number*) of either a *communication* or a *contract* event is assigned at reception of this event by receiving site. We will discuss how to use these sequence numbers later in Section 4.5.

4.4.3 Contract Formalization

A contract is an usage expression which one user expects others to respect when they receive and use shared data. Contracts are built on the top of basic deontic logic [194]. Contract in C-PPC model is different from traditional usage policy that is presented accompanied with application systems. For example, W3C platform for privacy preference P3P [186], which uses preference exchange language, APPEL [185], is an industry standard that provides a method for users to gain control over the use of personal information collected by websites they visited. In the systems, obligations are enforced by access control mechanism at the time a request is checked. Our approach does not require additional platform to express contract. Instead, contract is a part of replication system and it is built over operations within application domain. Moreover, obligations are fulfilled by users.

Symbolism of deontic concepts

Deontic logic is used as the logic of rights and duties. The philosopher Von Wright builds deontic logic around *"permission"*, defining *"forbidden"* as *"not permitted"* and an act is obligatory if the negation of it is forbidden. The act which is neither forbidden nor obligatory is then indifferent [194]. We will summarize how these deontic modalities are symbolized as a formal logic as in Von Wright model.

We use A to denote a name of an act and $\sim A$ is used as a name of its negation. The proposition that the act named by A is permitted is expressed in symbols by P_A. The proposition that the act named by A is forbidden, which is the negation of the proposition that it is permitted, is symbolized by $\sim P_A$ or F_A. The proposition that the act named by A is obligatory, which is the negation of the proposition that the negation of the act is permitted, is symbolized by $\sim P_{\sim A}$. We use simpler symbol for the obligatory, O_A. The proposition that the act named by A, which is called indifferent, is symbolized by $(P_A)\&(P_{\sim A})$. In this symbolism, P, O and F are called the deontic operators. Sentences of the type "P *name of act(s)*" are called P-sentences. Similarly, we might have O-sentences, F-sentences. Also we have *"permitted"*, *"obligatory"*, *"forbidden"* as deontic values.

The deontic operators apply to a single act with what we call a atomic name. Since we can define the conjunction, disjunction, implication of two given acts to be what we call a molecular name, we can apply deontic operators to pairs of acts as well. If A and B denote acts, then $A\&B$ is used as a name for their conjunction, $A \vee B$ as a name for their disjunction, $A \rightarrow B$ as a name of their implication.

Considering the distribution property of deontic operators, wrong conclusions might be taken with respect to the application of these operators. We first consider negation operation. If the act A is permitted, we can conclude nothing to the permitted, forbidden or obligatory as character of $\sim A$ since A might be obligatory or indifferent. Sometimes $\sim A$ is permitted, sometimes not. If A is obligatory as well as permitted, then $\sim A$ would be forbidden. If A is what we call indifferent, then $\sim A$ is also permitted. For example, in smoking compartment, smoking and not-smoking is permitted. But in the non-smoking compartment, not-smoking is permitted but smoking is forbidden.

We next consider distribution property in the conjunction of two acts. If both A and B are permitted, it does not mean $A\&B$ is permitted because doing either of them may commit us not to do other. For example, it is permitted to promise to give a thing and it is also permitted not to give a thing, but it is forbidden to promise to give thing and then not give it.

We finally consider distribution property in the disjunction of two acts. If at least one of the two acts A and B is permitted, it follows that their disjunction $A \vee B$ is permitted. When both acts are forbidden, their disjunction is forbidden.

The Von Wright deontic model includes also laws of deontic logic. A true proposition, that a certain molecular P-/O-sentence expresses a deontic tautology, is called a law of deontic logic. We summarize below these laws from notions of permission and obligation. It should be noticed that the combining force order follows that "$\sim$" is stronger than "&", "&" is stronger than "$\vee$", and "$\vee$" is stronger than "$\rightarrow$".

- Two laws on the relation of permission and obligation:

  (1) P_A is identical with $\sim O_{\sim A}$.

  (2) O_A entails P_A.

- Four laws for the dissolution of deontic operators:

  (1) $O_{A\&B}$ is identical with $O_A \& O_B$.

  (2) $P_{A\&B}$ is identical with $P_A \vee P_B$.

  (3) $O_A \vee O_B$ entails $O_{A \vee B}$.

  (4) $P_{A\&B}$ entails $P_A \& P_B$.

- Seven laws on commitment (doing an act commits to do another act if the implication of one to another is obligatory):

  (1) $O_A \& O_{A \rightarrow B}$ entails O_B. This law is intuitively obvious. If doing an act that is obligatory commits us to do another act, then this act is obligatory also.

  (2) $P_A \& O_{A \rightarrow B}$ entails P_B. If doing what we are free to do commits us to do another act, then this act is permitted to do also.

(3) $\sim P_B \& O_{A \to B}$ entails $\sim P_A$. This is a vice versa law of the law above. If doing an act commits us to a forbidden, then this act is forbidden also.

(4) $O_{A \to B \vee C} \& \sim P_B \& \sim P_C$ entails $\sim P_A$. This law is an extension of above law. Doing an act that commits us to a choice of forbidden alternatives, then this act is forbidden also.

(5) $\sim(O_{A \vee B} \& \sim P_A \& \sim P_B)$. This law means it is impossible to oblige to choose between forbidden alternatives.

(6) $O_A \& O_{(A \& B) \to C}$ entails $O_{B \to C}$. This law means that if doing two acts, one of which being obligatory, commits us to do a third act, then doing the second act commits us to do the third act.

(7) $O_{\sim A \to A}$ entails O_A. If it is failure to perform an act commits us to perform it, then this act is obligatory.

Combining two acts into a molecular act might lead to the incompatible. Two acts are incompatible if their conjunction is forbidden. For example, reading and smoking both are not permitted in library, so they are incompatible.

From deontic modalities to contracts

Based on deontic concepts, we formalize contract in the C-PPC model as follows.

Definition 4.4.3. *(contract primitive). For a set of n possible operations* $\mathbb{P} = \{op_1, op_2, \ldots, op_n\}$*, a contract primitive is denoted by a deontic operator followed by a write or a communication operation. A contract primitive is an event in log that takes deontic operators P (the permitted), O (the obligatory), F (the forbidden) as modality attributes (so-called modal). If op is an operation in* $\mathbb{P}$ *then the contract primitive* c_{op} *based on op is denoted as:* F_{op} *(doing op is forbidden),* O_{op} *(doing op is obligatory), and* P_{op} *(doing op is permitted). When we use the generic notation c it means that the contract c can refer to any operation.*

Definition 4.4.4. *(contract). A contract C is a collection or a set of contract primitive(s) which are built on operations of* $\mathbb{P}$*. It is denoted as* $C_{\mathbb{P}} = \{c_{op_1}, c_{op_2}, \ldots, c_{op_n}\}$*. Alternatively, we can use the notation* $C = \{c_{op_1}, c_{op_2}, \ldots, c_{op_n}\}$ *for a contract.*

```
<log> <!-- at local site of Raj -->
  <event>
   <evt>write</evt><op>insert</op>
   <attr>
      <by>Sheldon</by>
      <content>comment 1</content>
      <gsn>1</gsn>
   </attr>
  </event>
  <event>
   <evt>share</evt><op>share</op>
   <attr>
      <by>Sheldon</by>
      <to>Raj</to>
      <gsn>2</gsn>
      <rsn>1</rsn> <!-- receipt timestamp -->
   </attr>
  </event>
  <event>
   <evt>contract</evt><op>edit</op>
   <attr>
      <by>Sheldon</by><to>Raj</to>
      <modal>P</modal>
      <gsn>3</gsn>
      <rsn>2</rsn>
   </attr>
  </event>
  <event>
   <evt>contract</evt><op>share</op>
   <attr>
      <by>Sheldon</by><to>Raj</to>
      <modal>P</modal>
      <gsn>4</gsn>
      <rsn>3</rsn>
   </attr>
  </event>
</log>
```

Figure 4.7. An example of log containing contract events.

For example, in Figure 4.3, Sheldon inserts a comment into photo A and gives it to Raj with a contract $C_{\{edit,\ share\}} = \{P_{edit},\ P_{share}\}$ (edit and share are permitted) with two single contract primitives P_{edit} and P_{share}. When a user shares data by means of a push primitive, at the site of the receiver, a share event is logged with attributes representing users who sent and received the changes. Moreover, contract events are logged describing the contracts received. In Figure 4.7, we illustrate the representation of the log at site of Raj after he cloned *photo A* from Sheldon.

If we have n contract primitives, we can obtain a contract by merging these contract primitives. For instance, if we have two contract primitives $c_1 = P_{op_1}$ (op_1 is permitted) and $c_2 = O_{op_2}$ (op_2 is obligatory), then we can build the contract $C_{\{op_1,op_2\}} = \{P_{op_1},\ O_{op_2}\}$. Concerning merging contract primitives to obtain a contract, we consider two following axioms:

(A1) $C = \{c_1\}$ & $(c_1 \rightarrow c_2) \longrightarrow C = \{c_1, c_2\}$ (*deducibility*)

(A2) $C = \{c^1_{op}, ..., c^n_{op}\}$ & $(c^1_{op} \succ c^2_{op} \succ \ldots \succ c^n_{op}) \longrightarrow C = \{c^1_{op}\}$ (*priority*) ("$\succ$" denotes a higher priority relation between two contract primitives or two operations).

The axiom (A1) shows the consequent deducibility. We assume a set of inference rules can be defined among the contract primitives in the system. Following this axiom, if $c_1 \in C$ and $c_1 \rightarrow c_2$ then $c_2 \in C$. At a certain time if the user u has a contract C that contains c_1 and respecting c_1 commits to respecting c_2 then even if c_2 is not explicitly given to user u, c_2 is added to C. This is helpful to reduce the number of contract primitive given at a certain sharing time since users do not have to specify contracts that can be inferred from other contract primitive based on deducible rules of system settings. For instance, if we suppose that $O_{edit} \rightarrow P_{insert}$, then $C = \{O_{edit}\}$ & $(O_{edit} \rightarrow P_{insert}) \longrightarrow C = \{O_{edit}, P_{insert}\}$. This means if a user receives an obligation to edit, she will have the permission to insert automatically since the setting of inference rule $O_{edit} \rightarrow P_{insert}$ holds. Another example, if system allows $P_{edit} \rightarrow P_{insert}$ then $C = \{P_{edit}\}$ & $(P_{edit} \rightarrow P_{insert}) \longrightarrow C = \{P_{edit}, P_{insert}\}$.

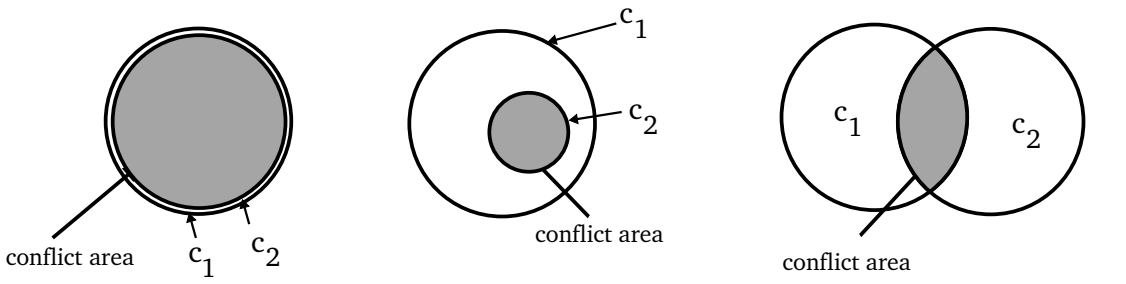
The axiom (A2) rules the merging process of different contract primitives referring to the same operation. If we have n contract primitives referring to an operation op with priority order $c^1_{op} \succ \cdots \succ c^n_{op}$ then these contract primitives can be merged and the resulting contract is deducible as $C = \{c^1_{op}\}$. For instance, if $C = \{F_{op}, P_{op}\}$ and $F_{op} \succ P_{op}$ then it is deducible to $C = \{F_{op}\}$. This means that even though op is permissive with the contract primitive P_{op}, it is forbidden to perform op if $C = \{F_{op}\}$ and $C = \{P_{op}\}$ are merged with the condition $F_{op} \succ P_{op}$.

4.4.4 Contract Conflict

When multiple users work on the same shared data and share their changes to one another under different contracts, it is not possible to ensure that the system will be conflict-free regarding these contracts. Therefore, it is necessary to identify conflicts, to detect conflicts and to propose conflict resolution strategies.

The term *deontic conflict* and *deontic inconsistency* have been used interchangeably in the literature. In his book *On Law and Justice*, Ross [160] identified three types in which inconsistency in law arises: *total-total*, *total-partial* and *partial-partial*.

(1) Total-total inconsistency: this means neither of a pair of norms is applicable without conflicting with the other. If the conditional facts of each norm are symbolized by a circle, a total-total inconsistency occurs when the two circles coincide (Figure 4.8a). In total-total inconsistency two norms are absolutely incompatible. This is thus said strong inconsistency since no norm can be performed without causing norm violations. For example, the total-total inconsistency arises when an action is simultaneously obligatory and forbidden.



(a) Total-Total inconsistency (b) Total-Partial inconsistency (c) Partial-Partial inconsistency

Figure 4.8. Three types of inconsistencies.

(2) Total-partial inconsistency: this means one of the two norms is not applicable in any case without coming into conflict with the other, whereas the other norm does not conflict in all cases with the first one. Such inconsistency occurs where one circle lies inside the other (Figure 4.8b). As an example, the total-partial inconsistency arises when an action is simultaneously permitted and forbidden.

(3) Partial-partial inconsistency: this means each of the two norms has cases that conflict with the other but also cases in which no conflict arises. This inconsistency exists when two

circles intersect (Figure 4.8c). We can see this inconsistency in an example which a person is obliged to attend a concert but the entering to the theatre without ticket is forbidden. The partial-partial inconsistency arises between two norms which are the obligation to attend and the prohibition to enter without ticket. If the person has a ticket, then she can fulfill one of two norms or both without causing violation to the other. In this case, she can enter the theatre, also by attending the concert, she fulfills the obligation requiring her to attend it. By this, no conflict arises. However, if she does not have a ticket, then she cannot act following one norm without violating the other. Without having a ticket, when she respects the prohibition to enter by staying outside, she violates the obligation to attend the concert. In contrast, when she fulfills the requirement to attend the concert, she will violate the prohibition not allowing her to enter without ticket. Through the example we see two norms have one case that they are incompatible, and also another case they are compatible.

Ross [160] also figured out that in judging inconsistencies an important part is the relationship between statutes where conflict occurs. Inconsistency is drawn (a) within the same statute or (b) between older and more recent statutes.

Concerning inconsistencies, in the normative discourse, the unrealizability is mentioned with two conditions: (i) the norm belonging to a set of norms must be individually realizable. This condition means each single norm should not be impossible to conform to. (ii) however, the norms in that a set of norms are not jointly realizable. This means what is prescribed by a set of norms cannot be performed simultaneously.

Definition 4.4.5. *(contract consistency) A contract which is a collection (or a set) of contract primitives (norms of obligations, permissions, and prohibitions), is consistent, if and only if, its contract primitives are simultaneously jointly realizable.*

Inconsistencies arise due to the incompatibility of the deontic operators. The deontic square of opposition (Figure 4.9), which is based on Aristotle's philosophy (as stated by Moretti [129]) about logic square and first used in deontic logic by Bentham [24], depicts the relationship of norms. It shows four types of inconsistencies of four deontic modalities.

(1) *Contraries*: the pair of obligation and prohibition forms this opposition. An action cannot be obligatory and forbidden simultaneously. This is a total-total inconsistency since both may

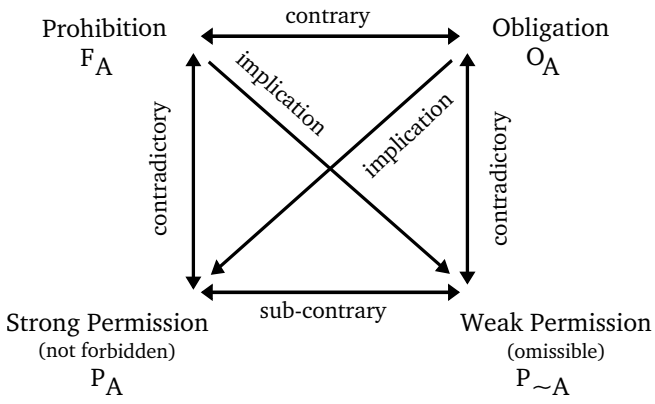


Figure 4.9. Deontic square of opposition.

be false.

(2) *Contradictories*: the pairs of strong permission and prohibition, obligation and weak permission, form this opposition. One norm in each pair of norms is true. An action is either permitted or forbidden as well as an action is either obligatory or omissible.

(3) *Implication*: obligation implies strong permission, prohibition implies weak permission. If an action is obligatory, then it cannot be forbidden, thus its permission is possible. Also, if an action is forbidden, then it cannot be obligatory, and thus its omission is possible.

(4) *Sub-contrary*: from the contrary of obligation and prohibition, strong permission (P_A) and weak permission ($P_{\sim A}$) are contrary to each other. An action may be performed if it is not forbidden, as well as it may be omitted if it is not obligatory.

This square recalls three possibilities for an action that are either forbidden, obligatory or indifferent (permitted but not obliged). From this square we can observe that the contrary relationship between prohibition and obligation raises the real conflict (total-total inconsistency). The situation when an action is simultaneously obliged and forbidden influences behaviors in conflicting fashion in the sense that it is impossible to do the action that is compliant with one norm without conflicting the other. For the pair of permission and prohibition, we adopt the view that their contradictory is an inconsistency but not a real total-total conflict. This comes from the fact that a permission may not be acted on, so no real conflict occurs between permission and prohibition. Therefore from our view, real conflict rather than normal inconsistency arises only between obligation and prohibition. In our model, contracts must be consistent or they must not contain any inconsistency or conflict between their contract primitives.

Even though each contract is conflict free, conflicts may arise when two contracts are merged together. It is easy to realize the fact that two users assert two contract primitives that are inconsistent is certainly possible and extremely frequent. It is even possible for one and the same user to assert two inconsistent contract primitives. If we want that users can collaborate in contract-compliant manner, we must resolve the conflict. The important thing is to identify inconsistencies and to examine the techniques used to remove them.

Definition 4.4.6. *(contract conflict) Two contracts C_1 and C_2 conflict if at least one contract primitive $c_i \in C_1$ conflicts with one another $c_j \in C_2$. Contract primitives are conflicting between O_{op} and F_{op}. Besides, two contracts are inconsistent to each other if at least one contract primitive $c_i \in C_1$ is inconsistent with one another $c_j \in C_2$.*

In next section we present our solution to deal with inconsistencies of contracts.

4.4.5 Conflict Resolution

We present in this section our solution to deal with inconsistencies of contracts. There is no fixed principle for conflict resolution. In order to ensure the consistency of contracts in the system, conflicts are resolved based on several criterias. Contracts, among which a real conflict arises, cannot co-exist in the contractual system, hence they must be avoided. One way to do this is by means of negotiation between users. In this case, the system should inform contracting partners about their contractual situation and what are the conflicting contracts on which operations. Then contracting partners decide how obligations and prohibitions can be "relaxed" in order to allow additional options for further actions. For example, Figure 4.10 depicts the case that conflict needs to be negotiated to precede further work of contracting partners. In the figure we can see even the conjunction of two obligations might create conflict. Say, *a night club is obliged to close emergency exit to prevent crimes quit for drugs* (O_{op_1} from police department), and *a night club is obliged to open emergency exit* (O_{op_2} from fire safety). In this case $O_{op_1 \& op_2}$ is inconsistent and two obligations are not realizable at the same time. In current work, we consider only the inconsistency between deontic operators (P, F and O) and not yet between semantic content of actions under those operators.

In case of partial inconsistencies only, inconsistent contracts can co-exist. For example, in

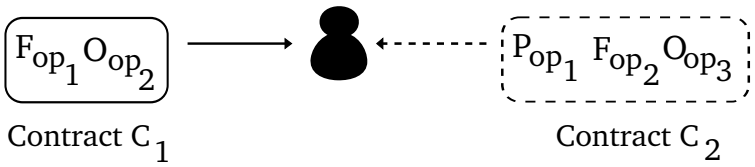


Figure 4.10. A scenario when a user adopts two contracts that are inconsistent. The user holds Contract 1 and a coming one Contract 2. The conflict between O_{op_2} and F_{op_2} needs to be resolved to proceed further actions.

Figure 4.10, if we eliminate the conflict between O_{op_2} and F_{op_2}, there still remains the partial inconsistency between F_{op_1} and P_{op_1}. In this case if the user is not performing op_1 then F_{op_1} and P_{op_1} are both respected and therefore no real conflict occurs. Similarly, if F_{op_1} and P_{op_1} are both given by the same user and P_{op_1} overrides F_{op_1} (we discuss overriding rules later) then no real conflict occurs. Even though this type of inconsistency can exist in the system, it is better to reduce inconsistency possibilities. Users should be informed about their contractual situation when a synchronization is performed. This helps them choosing between different contracts the ones that give them better benefit. In what follows we describe a method for ordering contracts based on the order of operations associated with them.

4.4.6 Ordering Contracts

Contract primitives are naturally interrelated and interdependent and there is no hierarchy between them. The set of contract primitives is not a normal set but a partially ordered set and the ordering relations are intrinsic to the contracts in the system. One way to select a contract in case of conflict is to assign orders to various contract primitives, and sort them in ascending or descending order and then compare contracts composed of them. Some priorities can be established in terms of particular objectives and they vary depending on particular applications. The hierarchical ordering of contracts enables users to give preferences to some contracts over others.

Depending on the operation types, the order of contract primitives is given as follows. Operations are categorized into different groups according to their types. Given two operations op_1 and op_2 with the priority order $op_1 \succ op_2$, then the order of contract primitives with the same deontic operator is assigned according to the order of operations, $c_{op_1} \succ c_{op_2}$. If the contract primitives associated with operations belong to different groups, then we determine a combined

order for each, based on the order of the group and then the order within group.

We formalize the method to order single operations as well as sets of operations. Our method is inspired from the work of Cholvya and Hunterb [39]. We present below the ordering of operations within single category and across multiple categories, and next is our solution to compare contracts.

- Ordering categories of operations: Each category includes a set of operations referring to a specific kind of action. Let us consider two categories $\Lambda_1 = [\alpha_1, ..., \alpha_m]$ and $\Lambda_2 = [\beta_1, ..., \beta_n]$. The ordering of Λ_1 and Λ_2 is given based on the priority of them in a particular system. In addition, the order of Λ_1 and Λ_2 implies the order of every operation of Λ_1 and Λ_2.

  For example, $\Lambda_1 > \Lambda_2 \Rightarrow \alpha_i > \beta_j \ \forall \alpha_i \in \Lambda_1, \beta_j \in \Lambda_2$.

- Ordering operations within a single category: Operations in a single category of actions can be put in a hierarchical order specific to a particular system. For two operations α_1 and α_2, their order should be either $\alpha_1 > \alpha_2$ or $\alpha_1 < \alpha_2$.

  For example, (*add-comment* > *read*) in category *edit*.

To compare two contracts, let $\mathbb{P}$ be a set of n operations that could be ordered as [op_1, op_2, ..., op_n] from the highest to the lowest priority conforming to the orders of operations within each category (if any) and between categories as presented above. Let $\mathbb{S}$ be a set of n-digit ternary numbers from 0 to $3^n - 1$ and a contract C composed of m contract primitives built over operations of $\mathbb{P}$, $C = \{c_1, c_2, \ldots, c_m\}$, $c_i = P_{op_j}|O_{op_j}|F_{op_j}$, $c_i \in C$, $op_j \in \mathbb{P}$, $1 \leq i \leq m$, $1 \leq j \leq n$ as a list of contract primitives which are ordered following the order of operations.

To order contract primitives, we set norms in some kind of hierarchy, some is regarded as of higher than others. Without losing generality, let us assume that deontic operators are ordered as $P \succ O \succ F$; also operations in $\mathbb{P}$ are put in order $op_n \succ op_{n-1} \succ \ldots \succ op_1$. A mapping from C to $\mathbb{S}$ results in $s \in \mathbb{S}$. For each $op_j \in \mathbb{P}$, $1 \leq j \leq n$, we set:

if $\exists c_i = P_{op_j}|O_{op_j}|F_{op_j} \ \in C$ then:

(1) if $c_i = P_{op_j}$ then $s[j]_3 = 2$,

(2) if $c_i = O_{op_j}$ then $s[j]_3 = 1$,

(3) if $c_i = F_{op_j}$ then $s[j]_3 = 0$,

where $1 \leq i \leq m$;

if $\nexists c_i \in C$, $1 \leq i \leq m$, then $s[j]_3 = 2$. This case presents the absence of any contract primitive based on operation op_j. We adopt the positive view that the absence of obligations and prohibitions implies the permission. Therefore, $s[j]$ is set value as 2, as same as in case that op_j is permitted.

The comparison of two contracts C_1 and C_2 is based on the comparison of their corresponding numbers $[s_1]_3$ (mapped from C_1) and $[s_2]_3$ (mapped from C_2). We have $(C_1 > C_2) \Leftrightarrow (s_1 > s_2)$ and vice versa.

For instance, given a set $\mathbb{P}$ of two operations (n=2) in the order $op_2 \succ op_1$, say we want to compare two contracts $C_1 = \{O_{op_1}, P_{op_2}\}$ and $C_2 = \{O_{op_2}\}$. The 2-digit ternary numbers $s_1 = [12]_3$ and $s_2 = [21]_3$ are mapped from C_1, C_2 to $\mathbb{S}$. Since $s_2 > s_1$, so that $\{O_{op_2}\} > \{O_{op_1}, P_{op_2}\}$, hence, $C_2 > C_1$.

This ordering mechanism helps users to make a decision in case of inconsistencies to choose more benefic contracts. It is important to make users aware of what is added to the system might introduce inconsistency. Furthermore, in a peer-to-peer network without a central authority that maintains the consistency of contracts, once conflicts are detected, they should be resolved or adapted by users.

4.4.7 Repealing Contracts

In addition to adding contracts to data when it is shared with collaborators, our approach supports removal of given contracts. We consider the overriding rule to repeal contracts issued in the past.

Overriding rule allows an old contract to be replaced by a new one. In this case the new contract overrides the old one. The contract primitive c_2 overrides c_1 if both c_1 and c_2 are given by the same sender to the same receiver and c_2 was sent (received) later than c_1. We can express this by c_2 overrides $c_1 \Longleftrightarrow (c_1.op = c_2.op)$ and $(c_1.attr.by = c_2.attr.by)$ and $(c_1.attr.to = c_2.attr.to)$ and (c_2 was sent (received) after c_1).

Let us present an example of contract overriding when Raj realizes that the operation *op* under the contract primitive $c_{op} = F_{op}$ he gave to Leo some time ago should not be forbidden any longer because conditions that made the prohibition of performing *op* have changed. He wants to permit Leo to do *op*. Since previous changes performed together with given contracts were logged and shared with many users, the only solution for removing the prohibition is by compensation. Raj can override the prohibition by giving a new contract to Leo. When the new contract is accepted by Leo, the prohibition is removed for him.

With this compensation solution, the addition of new contract might introduce new inconsistencies or lead to wrong conclusion as mentioned in previous section of deontic symbolism. We notice that inconsistencies would arise at receiver side when a sender tries to repeal an old contract primitive by asserting its negation. Therefore overriding a contract is not just simply adding its negation. This could make contracts in system inconsistent. An ideal system thus must help users to be aware of any conflict when they repeal a contract, for example, by providing awareness mechanism about conflict.

There is an alternative to remove old contract without introducing a new contract. Rather than negating a contract users might reject its validation [11]. This helps to avoid inconsistency (notice that rejecting is not the same as negating, with negation we assert another contract primitive to the system for a negation while with rejection we just simple add an event to confirm the revocation of an old contract primitive). However, we did not adopt this solution in our current work as the compensation solution is more appropriate to our logging mechanism.

4.5 Collaborative Process

This section describes the basic protocols of collaborative process over C-PPC model: logging changes, pushing logs containing document changes and contracts, and merging pairwise logs.

4.5.1 Logging Changes

Each site maintains a local *clock* to count events (*write*, *communication*, and *contract*) generated locally or received from remote sites. When changes are made or received, they are added to log in the following manner:

- When a site generates a new *write* event e, it adds e to the end of its local log in the order of occurrence and augments its clock. The *clock* value is assigned to attribute *GSN* (*generate sequence number*) of event e (i.e. $e.attr.GSN = clock$).

- When a site receives and accepts (from now and afterward we simply say a site receives a remote log since we do not proceed further in case users reject the remote log) a log from another site, events from the remote log that are new to its local log are appended at the end of the local log in the same order as in the remote log.

- When a user shares a document with another user, she sends a *communication* event followed by some contracts, which are logged by receiving user. We denote by e one of these events (communication or contract). At time of reception, receiver assigns his local clock to attribute *RSN* (*receive sequence number*) of e, (i.e. $e.attr.RSN = clock$).

- We assume that a user is unwilling to disclose to other collaborating users all *communication* and *contract* events that she has given to a certain user. Thus *communication* and *contract* events are not kept in the log of the sender. Moreover, even if a site sends those events to other sites, receiving sites could refuse integration of remote changes. In this way, sending sites would contain events that are not accepted by receivers. Therefore, *communication* and *contract* events are not logged by sending site.

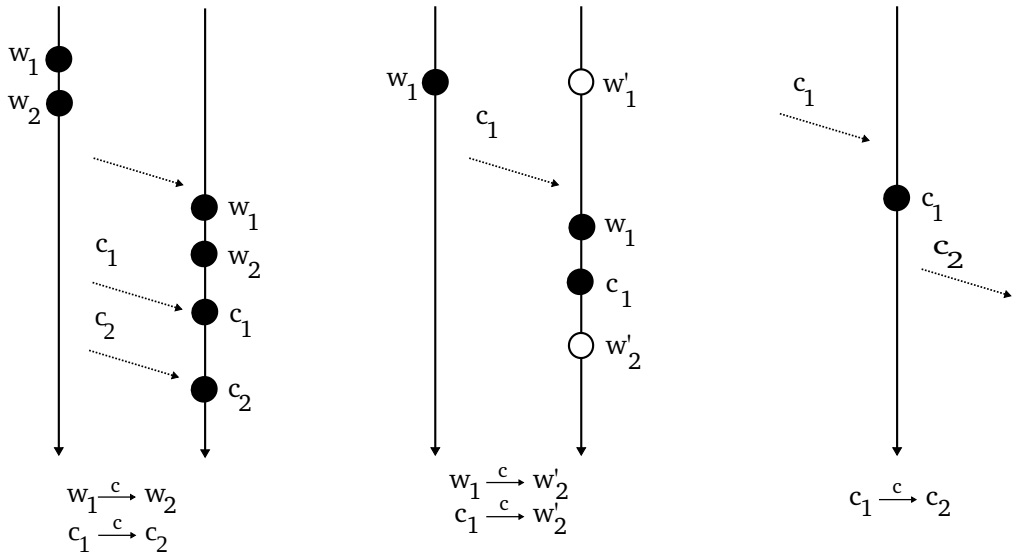
- An event e is said *committed* by site u when it is added (logged) to local log of u in one of the following cases:

  (1) e is a *write* event generated and saved (i.e. kept in log) by u.

  (2) e is a *contract* or *communication* event given to u by another site v. Recall that sending site v does not keep *contract* or *communication* events in its local log.

An important feature of the C-PPC model is that changes of one site are not propagated to all other sites since user trust levels are different and sites might receive different contracts for the same document state. We discuss the consistency of proposed model based on the CCI consistency model [37] which requires preserving *causality*, ensuring document *convergence* and preserving user *intention*.

Concerning causality preservation, our model deals with two causal relations (denoted as $\xrightarrow{c}$): *causal* relation (based on *happened-before* defined by Lamport [99]) and *semantic causal* relation.



(a) two write (contract) events generated by one site

(b) two events generated by two different sites

(c) two contracts received and sent by one site

Figure 4.11. Causal relations between events (w_i represents for *write* events and c_i represents for *contract* or *communication* events).

- *Causal relation:* two events e_1 and e_2 are in a causal relation, denoted as $e_1 \xrightarrow{c} e_2$, if:

  (1) for two events of the same type (i.e. two *write* events, two *contract* events or two *communication* events) e_1 and e_2 generated by the same site, if e_1 was committed before e_2 then $e_1 \xrightarrow{c} e_2$. For example, for two write events we have ($e_1.attr.by = e_2.attr.by$) and ($e_1.attr.GSN < e_2.attr.GSN$) and ($e_1.evt = e_2.evt = write$) $\Longrightarrow$ $e_1 \xrightarrow{c} e_2$ (see e.g. Figure 4.11.(a)). For two contract events we have ($e_1.attr.to = e_2.attr.to$) and ($e_1.attr.by = e_2.attr.by$) and ($e_1.attr.RSN < e_2.attr.RSN$) and ($e_1.evt = e_2.evt = contract$) $\Longrightarrow$ $e_1 \xrightarrow{c} e_2$

  (2) for two events generated by different sites, e_1 generated by site u and e_2 generated by site v, $e_1 \xrightarrow{c} e_2$ if e_2 is committed after e_1 has been received (or committed) at site v (see e.g. Figure 4.11.(b))

- *Semantic causal relation*: Two contract events e_1 and e_2 are said to be in a semantic causal relation if e_1 is received by a site before that site sends e_2 to another site. The contract event one site gives to other sites should depend on her current contracts: $(e_1.evt = e_2.evt = contract)$ and $(e_1.attr.to = e_2.attr.by)$ and $(e_1.attr.RSN < e_2.attr.GSN) \Longrightarrow (e_1 \xrightarrow{c} e_2)$ (see e.g. Figure 4.11.(c)).

The above causal relations between events are used in the auditing mechanism for detection of users that did not respect the given contracts.

In the C-PPC model, logs are propagated by using anti-entropy [51] which ensures the *happened-before* relation between events as defined by Lamport [99] without using state vectors [64] or causal barriers [152]. We say that event e_1 *happened-before* e_2, denoted as $e_1 \xrightarrow{hb} e_2$, if e_2 was generated at some site after e_1 was either generated or received by that site. The *happened-before* relation is transitive, irreflexive and antisymmetric. Two events e_1 and e_2 are said concurrent if neither $e_1 \xrightarrow{hb} e_2$ nor $e_2 \xrightarrow{hb} e_1$.

Two events that are in a causal or semantic causal relation are also in a happened-before relation.

We define a *partially ordered set* (poset) $H = (E, \xrightarrow{hb})$ where E is a ground set of events and $\xrightarrow{hb}$ is the *happened-before* relation between two events of E, in which $\xrightarrow{hb}$ is irreflexive and transitive. We call H as an event-based history in our context. Given a partial order $\xrightarrow{hb}$ over a poset H, we can extend it to a total order "$<_t$" with which "$<_t$" is a linear order and for every x and y in H, if $x \xrightarrow{hb} y$ then $x <_t y$. A linear extension L of H is a relation $(E, <_t)$ such that: (1) for all e_1, e_2 in E, either $e_1 <_t e_2$ or $e_2 <_t e_1$; and (2) if $e_1 \xrightarrow{hb} e_2$ then $e_1 <_t e_2$. This total order preserves the order of operations from a partially ordered set H to the linear extensions on the same ground set E.

We call these linear extensions as individual logs observed by different sites. The Figure 4.12 shows an example of a history and its congruent linear extensions.

In collaborative systems, where multiple sites collaborate on the same shared data, we can consider that the global stream of activity of all sites is defined by a partially ordered set of events. Each site, however, maintains a single log as its local observation and synchronization. It can see only events in local workspace that it generated locally or received from other sites. The

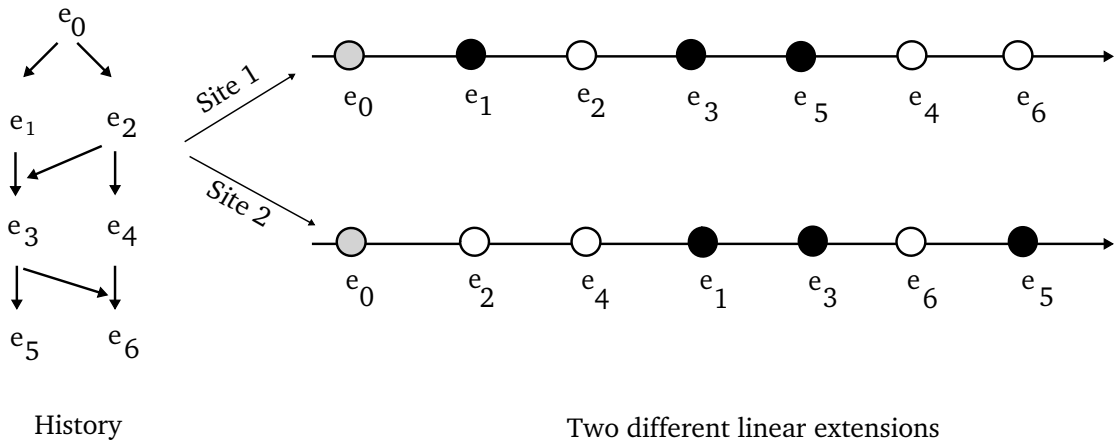


Figure 4.12. An example of history and logs

site keeps therefore an individual log as a linearization of history built on a subset of a ground set of evens. There are remaining events of global history built on entire ground set of events that are not visible for the site.

4.5.2 Pushing Logs containing Contracts

A key advantage of weakly consistent replication by relaxing data consistency is that the protocol for data propagation can accommodate contracts to let users decide with whom to reconcile. Anti-entropy [51, 140] is an important mechanism to achieve eventual consistency among a set of replicas. Basic anti-entropy allows two replicas to become updated by sending updates generated at one replica to other replica. Anti-entropy guarantees causal order of events which specifies that if an event is known to a site then any event preceding that event is already known to the site. When a pull is performed by *user 1* from the channel where *user 2* pushed his changes, only the log of *user 2* that *user 1* did not see from the last time when two users synchronized, is sent to *user 1*. The remote log is synchronized with the local log by using a merging algorithm. The synchronization requires to detect the concurrency and the causality between changes from multiple sites, and to resolve conflicts between concurrent changes. Replicas are consistent if their states are identical when they have applied the same set of operations.

In addition to propagation of changes, since in C-PPC model sites may have different levels of trust in other sites and the trust relationship may change during the collaboration, contracts are given to restrict usage on the shared document when a user shares a document with another

user. The user pushes her log as follows:

- Since a document is shared as a log of events, therefore, to send a contract as usage expression, the contract is attached to the end of the log.
- In sharing, a user specifies a new contract; however, she cannot specify a higher contract than what she currently holds. For instance, if a user u currently holds a contract C on the document d, she only can share d with another user with a contract C' that $C' \leq C$ (contracts are compared as presented in section 4.4.6).
- A user cannot specify a new contract which conflicts with her current contract. For instance, if a user u has a contract $C = \{O_{op}\}$, then she cannot add F_{op} to C.
- The contracts a user specifies to two distinguished users might be different. These two users do not not know the contract of the other user as far as they do not collaborate with each other.

During the collaborative process the log of each site grows and the document and contracts are updated each time a user synchronizes with other users.

4.5.3 Pulling and Merging Pairwise Logs

In the collaboration of two sites, at initial state, their copies are identical. Afterward, users change independently their copies by executing a set of operations and therefore their copies diverge. Document operations are synchronized among sites to achieve convergence of different document copies. The methods for merging different copies of document were introduced in many works [180, 154, 34, 85]. When users collaborate on same document under certain contracts, the integration depends on the right at which operations are permitted or not.

The collaboration involves logs reconciliation. Consider that a user u receives a remote log L' from a remote user v through anti-entropy propagation consisting of events from site v that site u did not see since their last synchronization. u has to elect new events from L' to append to her log L.

A site might receive a remote log with conflicting contracts. In case of unresolvable conflicts, the user decides either to reject the remote document version or to leave the local version to

Function isMerged(u, v, L, L', CT, CT')

```
1  if Trust(u, v) is low then
       // v is distrustful
2      result ← Reject;
3  else
4      if ct' ∈ CT' conflicts ct ∈ CT then
5          if conflict is resolvable then
6              result ← Merge;
7          else  result ← Reject;
8      else
9          result ← Merge;
10 return result;
```

accept new one. The function *isMerged* checks for conflict before merging. It checks if remote log *L'* sent by user v can be merged with local log L of user u. The function takes as arguments the log L of the local site u and the remote log L' of the remote site v containing new events since their last synchronization. Given these logs, the current contract holded by u and the contract that v gives to u when L' is sent can be computed. We denote these contracts by CT and CT' respectively (CT is the contract holded by u and *CT'* is the contract given to u by v). A site neither merges nor creates a new branch if the sender is distrustful. We consider a dominance of contract if a user revokes an old contract and replaces it by a new one. For instance, the old contract F_{share} received by site v from site u can be replaced by a new one P_{share}. Two logs can be merged if no conflict is found.

If the result returned by *isMerged* function is *Merge* (merging can be performed), we perform synchronization by using our proposed merging algorithm. We assume the merging algorithm ensures causality not only between *write* events but also between *communication* events and *contract* events. We next discuss in detail how to ensure causality.

To determine the total order of events committed by one site, we use the *"commit sequence number" CSN*. In merging function, commit sequence number *CSN* is used to track the last event committed by one site.

As we mentioned before, the attributes of event e, $e.attr.GSN$ and $e.attr.RSN$ record the values of the clock of its generation and its receipt, respectively. Note that every event has *GSN* attribute assigned before log is propagated, but *RSN* attribute is assigned to *communication*

and *contract* events at the receiving site during the synchronization.

The value of *CSN* of an event e committed by site u is computed as follows.

- If e is a *write* event generated by u, the commit sequence number *CSN* is assigned by the value of attribute *e.atrr.GSN*. The site who committed e is extracted from e's attribute *e.attr.by*.
- If e is a *communication* event or a *contract* event given by a site u to a site v and accepted by site v, the commit sequence number *CSN* is assigned by the value of attribute *e.attr.RSN*. The site who commits e is extracted from attribute *e.attr.to*.

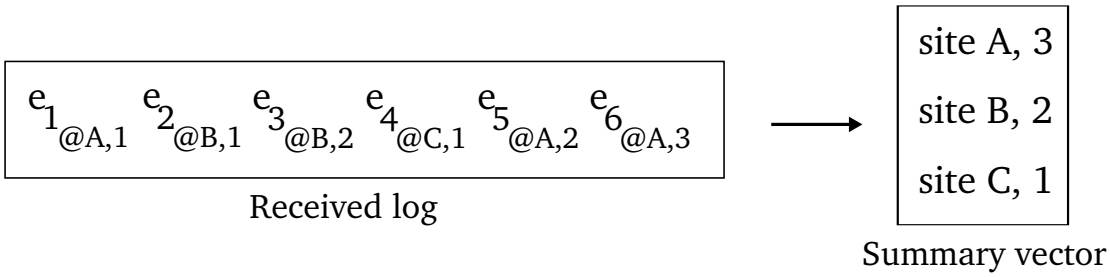


Figure 4.13. An example of Summary Vector

The merging function with which new events are added only to the end of log enables that if the log of a site u contains an event e committed by v with a commit sequence number *CSN*, then it contains all the events committed by v prior to e. In order to avoid merging events that have been already integrated, we use a summary vector SV which has the maximum size equal to the number of users. The summary vector of site u (SV_u) keeps the highest commit sequence number *CSN* of each site $v \neq u$ known by u in its components $SV_u[v]$ (see the example in Figure 4.13). A summary vector is a set of time-stamp of commit sequence numbers, each from a different user indexed by site identifiers. This allows a site u to correctly determine that an event from site v should be merged into local log if its *CSN* is higher than the current entry value of SV corresponding to its belonging site. The summary vector used here is different from the version vector used in weakly consistent replication to maintain causal relationship between events where the size is the number of sites and the version vector needs to be exchanged together with the corresponding operation. Instead, summary vector is maintained locally at sites and its size is the number of other sites whose events are known to the site.

Function merge(L, L', clock)

```
1  for i = 1 to sizeof(L') do
2      e ← L'[i] ;
3      if e.evt = write then
4          CSN ← e.attr.GSN;
5          site ← e.attr.by;
6      else
7          if e.attr.RSN = null then
8              e.attr.RSN ← clock;
9              clock ← clock + 1;
10         site ← e.attr.to;
11         CSN ← e.attr.RSN;
12     if CSN > SV[site] then
13         append e to the end of L;
14         SV[site] ← CSN;
15 return L;
```

It is possible to replay *write* events from log to get document state. We can use existing approaches of the CRDT family [144, 188, 158] for synchronization, in which concurrent operations can be replayed in any causal order as they are designed to commute in order to ensure consistency of document. CRDT algorithms design operations commuted from the start, so that making the synchronization simple as operations from the remote log that have not been previously integrated into the local log are simply appended to the end of the log.

The complexity of function *merge* is $O(n)$ where n is the size of the remote log L'.

4.6 Correctness of C-PPC Model

The C-PPC model uses operation-based optimistic replication. The core data structure used in the model is a partially order log. Events (write, communication and contract) are communicated using anti-entropy protocol which ensures causality. The document is achieved correctly if and only if the log was not tampered. This is an assumption of our model. Our solution about the construction and verification of authenticators to secure log is presented in chapter 6 of this manuscript. Authenticators prevent re-ordering of log events and therefore causality is preserved. If log was tampered, receiving site might discard it and the trust level of the site that misbehaved

would be decremented.

Concerning the document convergence, as C-PPC model uses CRDT for commutative operations, it ensures that different contracts are received by different sites when the same set of *write* operations was executed at those sites, their copies of the shared document are identical. However, the shared document might be in different states at two sites since the shared document is not uniformly distributed due to the use of contracts and the trust levels of users. And finally, concerning the property of intention preservation of C-PPC model, it is ensured by causality preservation and CRDT algorithm.

The C-PPC model supports multi-synchronous collaboration which allows simultaneous work in isolation workspace even when network is disconnected and user changes are propagated and synchronized with reconnection. The extension of using contract for PPC model made the condition that logs are synchronized more complex due to the arising of contract conflict. However, users can use log auditing mechanism to detect any conflict of contracts and logs are synchronized together if and only if all conflicts are resolvable. Conflicts can be resolved by the rejection of the owner or by the overriding rules.

4.7 Experiment

Due to the unavailability of real data traces of collaboration including contract, we evaluate the feasibility of the C-PPC model through simulation using PeerSim simulator [10]. We setup the simulation with a network of 200 users in which some of users are set as honest users, and the remain ones are set as misbehaving users.

For simulating process, we generate randomly the data flow of collaboration during the simulation. The data flow includes operations, contracts and users with whom to share. The network topology with which users share log with their neighbors are built randomly by the simulator. One interaction is defined as a process of sharing a log with a specified contract, from one user to another one. Since the total number of interactions generated should be pseudo uniformly distributed over all users, we let one user perform sharing with not more than 3 other users at each step. Similarly, the number of operations and contracts generated by one user each time is at most 10 operations and 3 contracts (if we consider only 3 types of actions in our

system: insertion, deletion and sharing).

Each node in the network represents for one user. Between two interactions, nodes generate local operations randomly but must follow its current contract. Nodes keep their contractual state temporarily to generate correctly operations. However not every node respects its contracts. While honest nodes generate allowed operations, misbehaving nodes generate operations that violate their contracts. This data is used to evaluate our algorithms of detecting misbehavior.

Since contracts are generated randomly with only limited condition that they should not bigger than node's current contract (contracts are ordered as in previous section), conflict certainly arises in simulation between contracts of different nodes. As nodes in simulator cannot behave human acts, we omit negotiation protocol for contracts. Furthermore, to simplify we do not allow neither total-total inconsistencies nor partial-total inconsistencies for contracts hold by nodes. Contract conflict thus are detected before logs are synchronized. Once conflict are found, logs are rejected to be merged and the node which detected conflict waits for next cycle or for other nodes which send log without conflicting (see our algorithm isMerged). With this restriction, logs are always maintained under consistent contracts.

We conduct this experiment to evaluate the time overhead generated by using contract for the synchronization and auditing mechanism. We compare two models: with and without contract. To be able to make the comparison between these two models, we follow the same data flow. In the model without contract, the synchronization mechanism requires merging logs of write events only. In the model with contract the synchronization mechanism requires merging logs of write events and contract events. Additionally, an auditing mechanism for user misbehavior detection has to be applied.

We compute for each model the total time (T) of all the synchronizations performed by a given user to build the same state of document, $T = \sum t_i$, where t_i is the time required for the i^{th} synchronization. Figure 4.14 shows the result according to the number of write events in the local log. From these results we can see that the time overhead generated by using contract is reasonable since the difference of time overhead computed for two models increases slowly with an increasing of number of events.

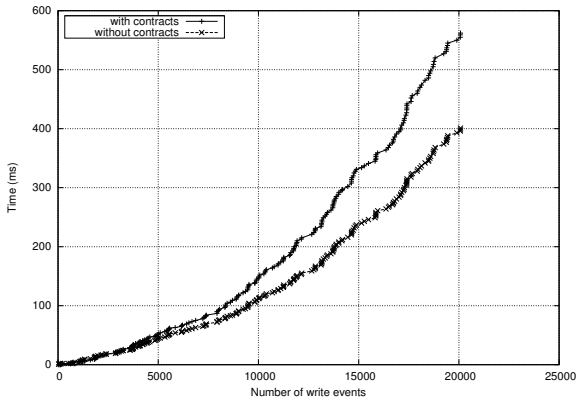


Figure 4.14. Synchronization time with growing of number of *write* events

4.8 Discussion and Summary

Our collaboration model is closely related to the approach proposed in [191] in terms of ensuring security and privacy in a weakly consistent replication system where users are not uniformly trusted. Access control policy claims are treated as data items. The guards added to replication protocol enforce specified policies at the synchronization step. A replica must check whether the requested action is allowed by the policy and then decide whether to accept or deny updates. In this approach each replica is a local authority that maintains current policies. This is similar to our approach where we let each user perform self-auditing based on local view of other users actions. However, unlike access control policy-based model where an enforcing policy is integrated in the replication protocol, there is no entity in our system that enforces users to follow contracts. Instead, each user performs a self-auditing on collaboration logs to analyze whether users respected the given contracts. Once any misbehavior not conforming to given contracts is detected, the trust level of the user who misbehaved will be decremented. Moreover, the approach described in [191] only expresses rights but not obligations that each replica should follow. Also, only the author of an item can define the policy associated to it and hence no solution for resolving conflicts between policies was discussed. In our approach we need to deal with policy conflicts as multiple contributors can specify different contracts on the shared document.

We discuss about some potential limitations in our work. First, contract-based collaboration does not offer a solution for plagiarism and violation of contracts outside of the system. Beyond

write, *communication* and *contract* events that a computer system could log, there are always side channels that can work around the logging. For example, a malicious user could replay *write* events from a log to create a new document and then shares it and claims herself as being the owner. Or a malicious user could reveal the content of the document outside of the system by using other communication means such as email, telephone call and chat where these actions are not logged by the system. These violations can be detected by humans or by using plagiarism techniques, however, this is out of the scope of this thesis work. The proposed model uses contract as a means to express data usage restrictions that helps to protect data privacy and to build a trustworthy collaborative environment.

A second limitation of our approach is how to deal with the growing size of the log during collaborative process. The log should be ultimately truncated so that it does not grow without bound. That requires some additional constraints and consensus of collaborators. When the cycle of collaboration grows big after long period, log of operations can be converted to state of document; and the log will be started from beginning. In such case, all contracts recorded might not be kept any more and this is reasonable since the behavior kept a long time ago might not be suitable to evaluate trust level at present. At the moment we do not consider log truncation in the proposed model.

Third limitation is that we have not fully explored a wide range of contract that can be specified in our collaboration model. In C-PPC model, contracts are based on a basic deontic logic including permission, prohibition and obligation. They can be combined with operators from temporal logic to express time dimension of contracts, however, we will consider this in our further work.

Next, we discuss the ability to apply the C-PPC model to multiple documents rather than a single one. Our approach is a general solution and thus it is applicable to multiple documents. We can keep a single log for operations over multiple documents. As an example, the single log is kept for operations of different files in a source code project using Mercurial distributed version control system. Each file has its unique identifier in the project, so we can keep operations with additional attribute of file identifier to distinguish them. In each interaction of sharing, contract can include multiple contract primitives that refer to rights and duties of user on different files at the same time. For editing operations on documents, we use CRDT approaches, so concurrent

operations can be replayed in any order without making document content diverged. When our approach is considered to apply to multiple documents, the approach works without considering some particular file operations such as moving a file, renaming a file. We can allow these file operations in our approach if a solution of CRDT for file systems is proposed. However, this is out of scope of our work.

To summarize, we have presented a contract extended push-pull-clone model (C-PPC) for multi-synchronous collaboration where users share their private data by specifying some contracts that receivers should follow. Trust levels are adapted according to users' past behavior regarding conformance to given contracts. Changes done by users on the shared data and contracts given when data is shared are logged in a distributed manner. We proposed a merging algorithm that deals not only with changes on data but also with contracts. Solutions for resolving conflicts among contracts specified in parallel by multiple contributors on a shared document have been proposed. We conducted an addition experiment by using PeerSim simulator to compare the overhead of the C-PPC model in case of using and not using contracts. The obtained result is reasonable. A log auditing mechanism is applied during collaboration and users who did not conform to given contracts are detected and their trust levels are updated. Log auditing and trust assessment procedures will be presented in next chapter together with a number of experiment results. The solution to secure logged changes is trivial. We present a solution for this issue in a chapter afterward.

Chapter 5

Log Auditing and Trust Assessment

Compliance checking whether user actions in a collaborative system conform with contracts is an important part of our C-PPC model. This question is done through logging and auditing mechanisms that are frequently used in many systems supporting observation. Log auditing is an approach that adopts *a posteriori* enforcement. It complements *a priori* access control in order to provide a more flexible way of controlling compliance of users after the fact. Log auditing algorithms depend mostly on the way logs and contracts are represented such as by first order logic or by a policy description language (i.e. P3P) as well as the prior assumptions of the systems. For example, systems either allow the presence of central authority who performs auditing or users do self-auditing; and enforcement mechanisms of policies are either preventive or detective. This chapter describes our auditing algorithm for trust assessment which is a part of the C-PPC collaboration model.

5.1 Auditing Principles

Before presenting our auditing algorithm proposed for C-PPC model, we clarify some principles concerning our auditing mechanism.

(1) Users can perform log auditing in order to make misbehaving users accountable for their actions without the need of any central authority. In this way, the dependence on an online entity that provides auditing logs is overcome. However, the disadvantage of the mechanism is that users have no knowledge about global actions done by all other users in order to

completely assess if a particular user behaved well or not. Our auditing mechanism is therefore based on incomplete evidences. Though this assumption could be claimed as a drawback, it is suitable to human society where a person is assessed only based on some of her noticed behaviors.

(2) Logs that reflect actions actually done by users and that are input to the auditing mechanism must be maintained correctly. Even though avoiding log tampering is impractical in distributed environments, tampering detection is possible. We have proposed using authenticators for detecting log tampering (see chapter 5). Log tampering is detected at time of synchronization before the log is accepted by receivers.

(3) How to use log auditing result and treat data resulted from misused actions? When a user discovers other users who misbehaved, she does not blame them publicly. However, she personally updates their trust levels. Users use trust models to manage their friend reputation. We use a trust metric to update user trust levels. The trust levels obtained from auditing result are used as input data to compute user trustworthiness. We focus only on using auditing result to update trust levels. We exclude any further aspects of trust models such as how to propagate personal view of trust among users or how to use external resources to assess trust levels or how to aggregate trust values.

These above principles distinguish our auditing mechanism from other prior approaches. In following subsections, we identify situations when contracts are violated and then provide the log auditing mechanism.

5.1.1 Contract Violation

In this subsection, we specify three types of attacks that might lead to contract violation.

- Malicious users tamper logs to eliminate or modify contracts or other events in a log. We consider that a user u is *malicious* if she re-orders, inserts or deletes events in a log that consequently affects auditing result. For instance, u removes some obligations that she wants to neglect. The log auditing mechanism assumes that logs are authenticated. Any tampering should be detected by a log authentication mechanism.

- When users do actions that are forbidden by specific contracts, these action events are labeled as *bad*. For example, a user violates prohibition contract that she holds over shared document.

- Users neglect obligations that need to be fulfilled. For instance, a user receives an obligation *"insert is obligatory"* but she never fulfills this obligation. Even if at a given moment when a log auditing mechanism is performed and no event that fulfills the obligation is found then we still not be able to claim that the user misbehaved. She might fulfill the obligation at a later time. The given obligation is labeled as *unknown* meaning that the obligation has not yet been fulfilled. Once the obligation is fulfilled, the *unknown* label is removed.

Users are expected to respect given contracts. If a user respects all given contracts, then she will get a good trust value assessed by others. Ideally, if a user misbehaves in one of the three ways mentioned above, her misbehavior should be detected by other users. The auditing mechanism returns a trust value that is computed from the number of events labeled with *good*, *unknown* and *bad*. Note that this manner of computing trust values does not distinguish an accidental attack from an intentional attack. In order to make users aware of unintentional misuses, the system might remind them the contract that they are holding.

5.1.2 Trust Metric

With C-PPC model users first bring social trust into the system. However trust is not immutable and it changes over time. Thus trust should be managed by using a trust model. In our work, trust is computed based on auditing results with contract compliance checking. Violation of contracts might not be avoidable, however once it is detected, misbehaving users should be identified and made responsible. Various trust models for decentralized systems exist such as [156, 90, 4, 46]. We explored trust metrics used in these trust models. Most of existing P2P trust models propose mechanisms to update trust values based on direct interactions between peers while we use log auditing to help one user evaluate others either through direct or indirect interactions. Our mechanism for discovering misbehaving users can be coupled with any existing trust model in order to manage user trust values. To our best knowledge, there is no trust model and trust metric dealing with local direct trust computed from auditing results.

We use a trust metric to compute trust values based on auditing results. Each user evaluates a direct trust value from direct and indirect log evidences. We call *local direct trust value* the trust value computed directly by users from local log they have and not from indirect source such as recommendation trust. Recall that writing events that violate contracts are defined as *bad*. Contract events that are not fulfilled or neglected are defined as *unknown* since they are unknown to be fulfilled in the future. Remaining events that are not bad or unknown, are defined as *good* events.

We propose a novel metric to compute the current trust value for a user based on these auditing results. Let's assume trust values range in the interval [0..1], and if all events audited are good, then the trust values reaches 1. Otherwise, the *unknown* events and *bad* events make trust values decreased. We denote x_1 is the number of *good* events, x_2 is the number of *unknown* events and x_3 is the number of *bad* events, and $y = x_1 + x_2 + x_3$ is the total number of audited events. We assume the number of *bad* events, x_3, is k stronger times than x_2 in making trust decreased. Applying weighted mean of x_2 and x_3 to compute the decrement they cause on trust values over y audited events, we have malicious rate is computed as in equation 5.1.

$$malicious_rate = \frac{\frac{x_2 + k * x_3}{1 + k}}{y} = \frac{x_2 + k * x_3}{y + k * y} \tag{5.1}$$

x_2 : number of unknown events
x_3: number of bad events
y: number of total audited events
k: weight for average mean of x_3 over x_2

In the C-PPC model we assume that users perform actions mostly according to their given contracts in the sense that when bad events occur, the trust will decrease quickly. This assumption is realistic as in social life, when one bad action might strongly decrease the reputation of a trusted person. Derived from this hypothesis, we design a function to compute trust value that decreased exponentially with the amount of bad events or unknown events found.

In equation 5.2, λ is the decreasing rate for trust up on the number of malicious events are found. If in a system that violations are assumed happening rarely then the trust should be decreased quickly if violations are found. In this case λ is set big enough to reflect correctly the

quick decrement. Otherwise in a system that violations are assumed easily happening then the trust should not be decreased strongly. In this case λ is set small.

$$current_trust(u,v) = exp(-\lambda * malicious_rate) \tag{5.2}$$

λ: decreasing coefficient
u: auditor; v: auditee

Figure. 5.1 depicts an example of trust values computed from parameters: number of events $y = 100$ including *good*, *bad* and *unknown* events. The *bad* events are three times more dangerous than unknown event, $k = 3$. The decreasing rate for trust in case of malicious events founded is five times exponential, $\lambda = 5$. The graph plots function $exp(-5 * \frac{x+3*y}{400})$. From the figure we can see that trust is at highest value 1 when no malicious events (bad and unknown) are found. Otherwise, trust decreases exponential, however, it decreases more slowly on the dimension of *unknown* than the dimension of *bad* since bad events cause trust decremented more quickly (in this case it is three times).

Let us consider an example with $y = 100$, number of *unknown* events $x_2 = 5$, number of *bad* events $x_3 = 2$. With a setting of parameters $k = 3$ and $\lambda = 5$, we compute the current trust value as: $current_trust = 0.87$ following equation 5.2.

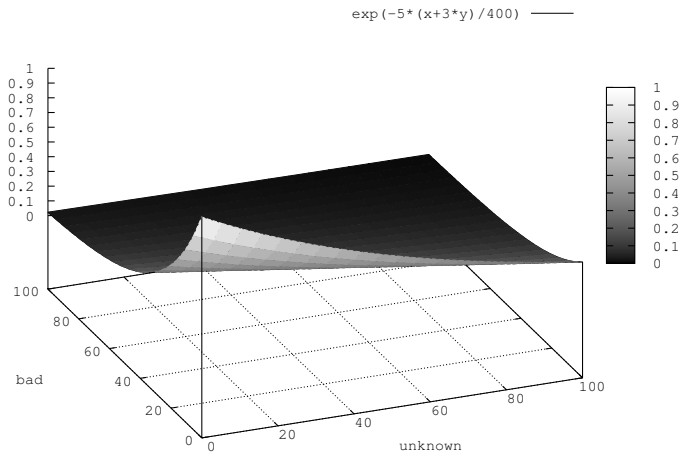


Figure 5.1. An example for current trust computed from the variations of number of bad events and unknown events that are found after auditing, $0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$.

Given the last local direct trust value of user v assessed by user u, $trust_{n-1}(u,v)$. The new

$$trust_n(u,v) = \alpha * current_trust(u,v) + (1-\alpha) * trust_{n-1}(u,v). \quad (5.3)$$

α: coefficient of the important of current trust over old trust
n: the step when u assessed trust put on v

aggregation for new trust value for user v is computed by u as in equation 5.3 where α stands for the importance of the current trust against the old trust value computed from last auditing time. Here α is the weight standing for the strength of current trust, $0 < \alpha < 1$. Note that $trust_0(u,v)$ is the initial social trust between u and v and it is assigned by u for the trust she puts on v.

5.2 Log Auditing and Trust Assessment

Our auditing procedure aims at detecting contract violations and making users accountable for their actions by adjusting their trust levels following a trust metric computed as in equation 5.3. The general idea of the auditing procedure is to browse the log and check each event appearing in the log whether it conforms given contracts. For each violation of a particular user, we increase the number of *bad* events counted for the user. Similarly, for each obligation that is not yet fulfilled, we increase the number of *unknown* events. This statistic of contract violation by a user overall the total events that are audited is used to compute trust level of this user.

Procedures *updateAuditState* and *audit* present auditing protocol and trust computation when a user u audits actions of all other users, say v, who appears in the log. In these procedures, G_v and Q_v are used to keep a set of contracts and a set of obligations which user v holds, respectively ($Q_v \in G_v$). At the initial step, $G_v = \emptyset$ and $Q_v = \emptyset$. For each event e in the log L, the procedure *updateAuditState* checks its event type, contract, communication or write event. If e is a contract given to user v then it is added to G_v. Moreover, if e is an obligation, it is counted as *unknown* event until an event that fulfills it will be found. If e is a write or a communication event performed by user v, it is checked if it complies with or violates contracts in G_v. In the procedure for updating auditing state, for each user v, $numberOfBadEvents[v]$ and $numberOfUnknownEvents[v]$ are used to count the number of *bad* and *unknown* events that are audited, respectively (remaining events are considered *good*). The entry $auditedEvents[v]$ is used to store the total number of audited events. All users v audited by u are inserted in

the set V. At the initial step of *audit* procedure, these variables: $numberOfBadEvents[v]$, $numberOfUnknownEvents[v]$ and $auditedEvents[v]$ are set equal to 0, and the set V is set empty.

Procedure updateAuditState(e)

```
1  if (e.evt =' contract') then
2      v ← e.to;
3      G_v ← G_v ∪ {e};
4      if e overrides c in G_v then
5          G_v ← G_v \ {c};
6      if (e.attr.modal =' O') then
7          Q_v ← Q_v ∪ {e};
8          numberOfUnknownEvents[v] + +;
9  else
10     v ← e.by;
11     if e violates G_v then
12         numberOfBadEvents[v] + +;
13     if e fulfills c in Q_v then
14         Q_v ← Q_v \ {c};
15         numberOfUnknownEvents[v] − −;
16 V ← V ∪ {v};
17 numberOfAuditedEvents[v] + +;
```

The auditing procedure browses the local log that a user who performs auditing holds. Procedure *audit* takes as input the local log L of user u and the position in the log *lastCheckedPos* identifying the last event checked in the previous auditing mechanism. L is browsed to check whether behavior of other users is correct. When log analysis is finished, trust values of all audited users v in V are recomputed based on auditing results. By doing this, their accountability is made through updating their trustworthiness.

A user can perform log auditing at any time at local site and trust values are updated personally. Log analysis has polynomial order of n time complexity $O(n)$ with n is the number of events that are audited. In case auditing creates significant overhead, users might skip auditing some parts of log which were done by highly trusted users. However, in case these users behave badly, they are discovered only in a next auditing phase.

Procedure audit($L, lastCheckedPos$)

1 **for** $i = lastCheckedPos + 1$ to $length(L)$ **do**
2 $e \leftarrow i^{th}$ event in L;
3 $updateAuditState(e)$;
4 **foreach** v in V **do**
5 $current_trust(u,v) \leftarrow exp(-\lambda * (numberOfUnknownEvents_v + k * numberOfBadEvents_v)/((1+k) * auditedEvents_v))$;
6 $trust(u,v) = \alpha * current_trust(u,v) + (1-\alpha) * trust(u,v)$;

5.3 Experiment

We use the simulation environment set up with PeerSim simulator as in experiment presented in previous chapter. In the experiment of this chapter, we focus on the ability of detecting misbehaving users. The portion of honest/misbehaving users in different experiments varies depending on the purpose of evaluation.

To evaluate the ability of misbehavior detection, we check first the ability to detect a selected misbehaving user according to the total number of interactions performed by all users. The estimation is performed on a collaborative network of 200 users with 60 misbehaving users (30% of users are misbehaving users). The auditing process is performed after each synchronization with another user. We select randomly one misbehaving user to be audited and we analyze the percentage of other users that can detect him. Figure 5.2 shows the result recorded in cycles. We can see that the misbehaving user is detected by a few of users at the beginning and then the number of users that detect his misbehavior increases along with the increasing of number of interactions.

Second, we check the percentage of misbehaving users that can be detected. We select randomly one honest user from the network to observe the percentage of misbehaving users that she can detect. Figure 5.3 shows the result according to the number of synchronizations done by the selected user with others. We can see from the graph that up to 20% of misbehaving users are detected after the first four synchronizations (auditing is done four times), and after the fifth synchronization more than 80% of misbehaving users are detected. We can see a drastic change in the figure between the fourth and the fifth synchronization. That change is due to a synchronization of the log of selected user with a remote log that contains misbehavior of most

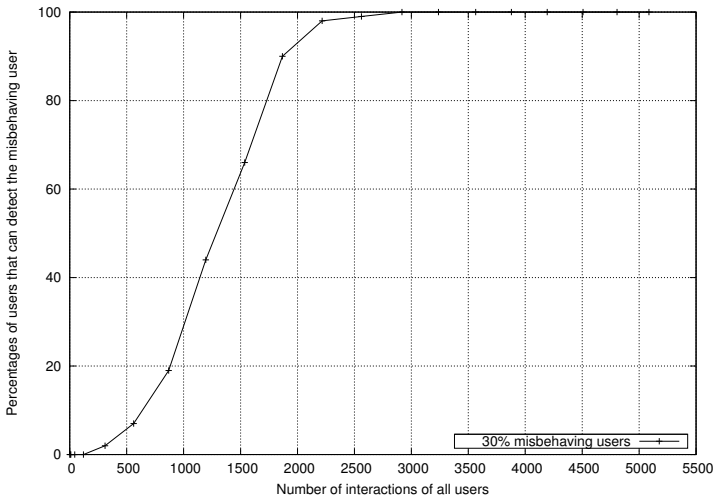


Figure 5.2. Ability to detect one selected misbehaving user with respect to the total number of interactions in a collaborative network of 200 users with 30% of them are misbehaving users.

remaining misbehaving users. This can occur in distributed networks of random topology where clusters of collaborating users exist. Once an interaction occurs between two users belonging to such clusters, misbehaving users of the two clusters are discovered. Only about 10% of misbehaving users may require more interactions to be detected. From results in Figure 5.3 we can see that the ability to detect misbehaving users depends also on the topology of the collaborative network. In the future work we will perform more experiments to evaluate how topology would affect the detection.

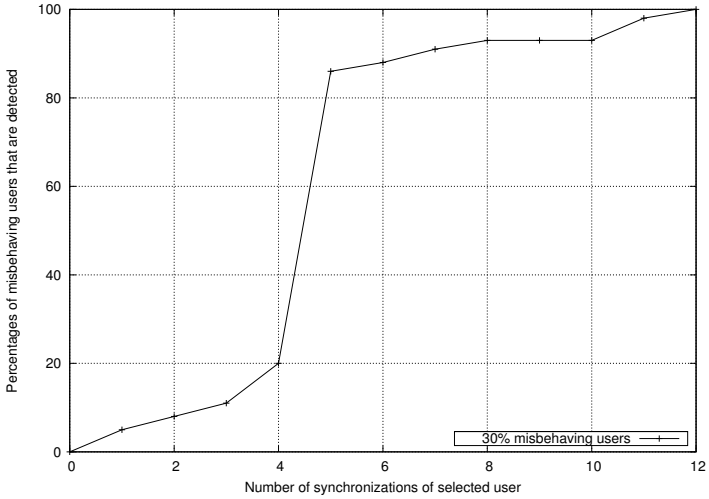


Figure 5.3. Percentage of detected misbehaving users with respect to the number of synchronizations done by selected honest user.

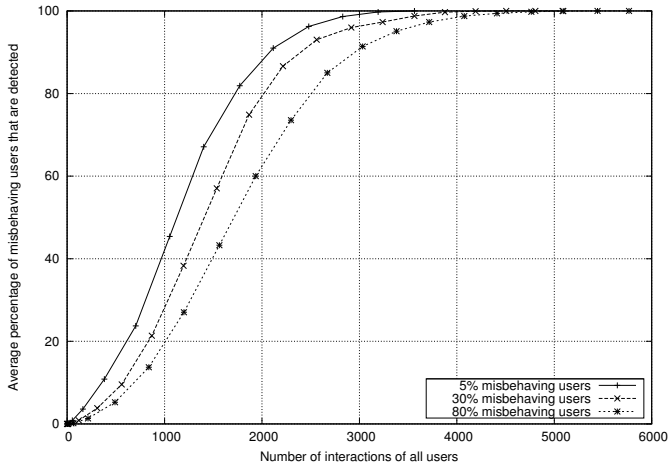


Figure 5.4. Average percentage of detected misbehaving users with respect to to the total number of interactions in the collaborative network.

In order to have a global view about the evolution of the percentage of detected misbehaving users, we compute the average value of detected misbehaving users over all users of the collaborative network. Figure 5.4 shows, on average, the percentage of misbehaving users that are detected by one user. We perform the experiment in case of a low, medium and high population of misbehaving users in the network (i.e. 5%, 30%, 80% of misbehaving users). The results show that the system still functions well in case of a high/low population of misbehaving users.

5.4 Related Work

A lot of prior work addresses the log auditing problem over logs expressed by using linear temporal logic (LTL) [21, 15, 18, 35, 19, 71]. Some of these works deal with logs based on partially ordered set, e.g. [21, 18], which is similar to our logs. This line of work encodes polices as temporal logic rules. In deciding compliance, based on history of past communications and logical rules, the system determines whether a possible next communication will violate the policies. Barth et al. [18] classify compliance into weak and strong. Weak compliance ensures actions taken by agents locally satisfy the privacy policy while strong compliance ensures agents actually meet future requirements. On this view of compliance, log auditing mechanism for our C-PPC model can ensure weak compliance only since there is no way to ensure people always respect contracts for their future actions. In [19], Barth et al. proposed an abstract model for evaluating privacy

policy and supporting policy compliance and audit. The model includes agents that are not only mechanical agents but also human agents. Agents send each other messages containing personal information about each other. Each message carries a tag that needs not bear any relation to the contents of the message. Mechanical agents maintain workflow based on message tags. Events appear suspicious if they indicate that some message tags were incorrect. Auditing algorithms are not fully automatic but by using an oracle (such as a human auditor) who can read message content. A given content is compared with its tag by looking over a causally graph-based workflow as audit log. Finding accountable agents for policy violations is performed with breath-first search. To detect misbehaviors that lead to policy violations, the algorithm samples communication at random and check. The more the oracle is called the more accurate the auditing result is. With a similar idea of using first order logic, Garg et al. [71] proposed an algorithm that checks audit logs iteratively for compliance with privacy policies. The iterativeness is applied to adapt to the incompleteness of audit logs since logs might not be able to be collected at a certain point of time. The idea of dealing with incomplete logs is very suitable to distributed environments where logs evolve over time by adding more information. Though we apply a simple logging method in C-PPC model without using temporal logic, iteratively checking is useful in our auditing mechanism since in some case at a certain time it is difficult to determine whether an obligation (a part of contracts) will be fulfilled or neglected.

Auditing is naturally used for detecting violations. In general, the violations are either controllable or only observable [20]. Basin et al. pointed out that not all actions can be controlled but only observed, for example, the passage of time. This distinction makes the enforceability divided in preventive or detective [61]. Beyond, auditing can be performed online by run-time monitoring at the time actions occur [21] or offline after actions have been done [7]. From this distinction, preventive enforcement should be done online while detective enforcement can be done both online and offline. We do not assume having a strict system that enforces every user action. Instead, we offer a flexible system that lets users acting freely but they are expected to respect given contracts and they are made accountable for their actions. Thus our log auditing proceed with detective enforcement that are mainly carried out offline.

There are numerous works assuming the presence of a central authority that is responsible for auditing [19, 35]. In [35], Cederquist et al. give a log auditing approach for detecting misbehavior

in closed collaborative environments, where a small group of users share a large number of documents and policies. In their work, first order logic is used to express policies, and auditing protocol is performed by a central auditing authority. The authority uses evidence trace to audit agent's actions, and agents are asked to provide valid proofs for those actions. Users trust others based on their accountability with the assumptions that logs are unable to forge and users will not vanish after they join the system. These assumptions are different from the issues addressed by our C-PPC model, since we do not assume the presence of a central authority neither the commitment of users to stay indefinitely online in the systems. In our work, users make auditing themselves. The case that users leave system with either good or bad reputation will not affect the system since they are not involved any more in the collaborations among users. In [94, 130], authors present a logical policy-centric framework for behavior-based decision making. The framework consists of a formal model of past behaviors of participants which is based on event structures. However, these models require a central authority that has the ability to observe all actions of all users. This assumption is not valid for a purely distributed collaboration.

Not only maintaining a central authority in distributed systems is infeasible, but also maintaining a central system log is impractical since there is no central entity which is trusted by all participants to maintain a log faithfully. In our C-PPC model, each participant keeps its own log of events generated locally, or of events received remotely. This makes participants maintaining different logs of the same history of the system and logs evolve with the contributions of distributed users. The complexity of our log auditing mechanism compared to centralized solutions comes from the fact that each user has only a partial overview of the global collaboration log and he can audit only users with whom he collaborates. Therefore, a user can take decisions only from the information he possesses from those users.

Logging and auditing are used toward accountability according to user behaviors. In PeerReview [79], nodes are classified as *correct* and *faulty*. Each node maintains a tamper-evident log which records messages it sent and received, and also inputs and outputs of the application. Each node is also connected to a set of other nodes acting as its witness. Any node (auditor) can ask another node (auditee) for the log. It then replays actions in the log to compare result with output recorded before. Also it checks actions in log if they conform node's reference implementation. If these checking results are positive, then auditing node can determine the

audited node is correct; otherwise audited node is faulty. PeerReview system is similar to our C-PPC model in the sense that log auditing is done by users or peers rather than by a central authority. We check actions according to contracts that users received and then to make users accountable, audit result is used to adapt their trust values.

5.5 Summary and Discussion

Log auditing is used as a means to identify misuses that may impact final result. It can provide an awareness to warn users for their behavior and lead them to decent actions. It also fosters a positive relationship between people (auditor and auditee) that leads to significant values for the collaboration outcome. In this chapter, we proposed a method for log auditing in decentralized environments where no central authority neither a central log exists. The key idea is to detect misbehavior and update trust level to make users accountable for their actions. In addition, we presented a novel trust metric to compute trust level which is based on the output of audit log. Furthermore, we conducted a number of experiments to evaluate the auditing procedures.

We identified two cases of contract violation: violation of given prohibitions and neglect of given obligations. We hypothesized that these violations have different impacts on trust relation. Violating a permission strongly damages trust more than neglecting obligations at the auditing time since the user might fulfill them later on. This is reflected in the equation of trust metric to compute current trust in which the former is considered k times stronger than the later. Furthermore, we hypothesized that current trust has different impact, α times, from previous one computed from last auditing procedure. In addition we assume that users behave well most of time, therefore we set the trust decreased exponentially with λ as significantly degrading parameter.

We implemented some simulations with PeerSim simulator to evaluate the feasibility of the auditing procedure as one part of the C-PPC model. Through experiments we showed the auditing results with different percentages of misbehavior with 5%, 30% and 80% of misbehaving users. Though these results are done in simulating environment only, we believe that the auditing does not create significant overhead problem for the C-PPC model. The ability to detect misbehaving users mostly depends on the interactions between users as well as the topology of collaborative

network (for this aspect we have not fully evaluated by current implementation).

We hope to address several issues in our future work. A concrete trust model for multi-synchronous model (C-PPC model in our context) is necessary to manage trust levels of users. The trust model (if possible) will take into account the trust metric that we have proposed. An other work that we might do in the future is to evaluate how the topology of collaborative work affects the auditing result. In our current experiments we tried to generate actions that are uniformly distributed among users. In a completely random network, the auditing result could be different. For example, malicious users might act wrongly inside an isolation subgroup, and all other people outside of this group will trust him as a honest collaborator. When we know the impact of network topology we could find more exact parameters for trust metric to compute trust values.

Chapter 6

Authenticators

The C-PPC model is an extension of multi-synchronous collaboration model. Instead of having only a single stream for all users activity, C-PPC model allows to maintain multiple, simultaneous streams of activity, and then manages divergence between these streams. Different users maintain therefore different streams of history containing activity of all users. In this thesis, we address an issue of how to secure logs in operation-based multi-synchronous collaboration. To our best knowledge, no existing work addressed this issue.

In operation-based replication systems users can misbehave by tampering history for their convenience. For instance, they can remove some contents of the history or change the order of some events from the history. This might be critical for some collaborative systems such as version control systems. It is vitally important to be able to retrieve and run different versions of a software. If the history can be modified, revisions do not correspond to the expected behavior of the software. Moreover, developers cannot be made responsible for the revisions for which they contributed. Furthermore, history modification may introduce security holes in the system. As an example, in C-PPC model, if users can modify logs then they will remove some contracts that they want to neglect. Therefore, there is a need to ensure integrity of log and in case log was tampered the misbehaving user should be detected.

In multi-synchronous collaboration, the correctness of collaboration outcome is based on the trustworthiness of users who maintain history of versions. Unfortunately, a malicious user can always introduce phony updates to forge history or alter the correct order of versions. This attack raises the threat that honest users might get forged content of shared data. Replicas with

corrupted updates might never converge with other valid replicas and this is critical in replication systems.

Solutions for securing logs can be classified into two main families: non-cryptographic secure logging and cryptographic secure logging. The former approach is based on secure logging machine such as write-only medium (e.g CD/DVD) or tamper resistant hardware, trusted hardware to prevent adversary from modifying logs [40]. However, in real world applications in large scale distributed environments, it is impractical to assume the presence of such devices. The later approach has been investigated deeply with numerously extensive research (namely, [43, 48, 80, 82, 112, 115, 196, 170]). However, existing solutions are adapted only for collaboration based on a single global stream of activity over data space. For instance, floor control policies [167] and locking mechanisms [168] ensure a single global stream of activity by allowing a single user at a time to access objects in shared workspace.

In this chapter, we propose a solution relying on hash-chain based authenticators for securing logs in multi-synchronous collaboration. The proposed authenticators ensure the authenticity and integrity of the logs, i.e. any log tampering is detectable. Moreover, the proposed authenticators provide user accountability, i.e. any user can be made accountable of her misbehavior of log tampering.

6.1 Security Aspects

In operation-based collaborative systems, authentication of data items for collaborative workflows has gained increasing importance. Say for example, if Raj receives a document from Penny and processes a part of it and then forwards to Sheldon, the operation-based document should include the chronological log of actions that each user, Denholm, Penny and Sheldon, performed on the document. This section presents a threat model followed by desirable properties for dealing with security requirements.

6.1.1 Threat Model

A threat model, which models the capabilities of attackers, is necessary to analyze the threats that will be addressed by our solution.

There are two types of malicious users: insiders and outsiders. We consider in this thesis an inside adversary who has full rights to access a replicated object. Such an adversary might want to alter the history including actions performed on data by authorized contributors. For example, when Raj provides a document to Penny who can perform and contribute new updates to the document, she should not be able to modify actions that Raj performed which were recorded in log. Our work assumes only adversaries who act inside of the system. We cannot prevent outsider attacks where an adversary copies data to create a new document and claims at a later time as being an owner. This might be possible in our system if an adversary removes completely the log of events, which corresponds to a document removal. We could deal with outsider attacks with the support of a trusted platform, however, we exclude this assumption in our collaborative system.

We assume users trust each other with their social-based relationship when they start to collaborate. However, trust it not immutable and trusted users once they gained access to the log can always misbehave. Such an active attacker can read, (over)write, delete and change order of log entries. In doing so, an attacker alters existing records or adds forged information to history.

6.1.2 Desirable Properties

The following properties are addressed to authenticate operation-based history:

Integrity

Adversaries are infeasible to forge a log, such as modify its entries or put new forged events into log, without being detected. Integrity is the most important property required for securing logs in operation-based replication. Ensuring the integrity of a single document can be done easily by using cryptographic signatures or checksums. However, ensuring the integrity of a replicated document as a log of events crossing multiple contributors is more challenging.

Concurrency-collision-freeness

In a history H, some events might be concurrent, while some others might be in a happened-before relation. If L_i and L_j are different linearizations of the same history H then any authentication mechanism applied to L_i and L_j should yield the same result. The "yielding the same result" is

expressed by the concurrency-collision-freeness property: the authentication mechanism holds a function f that $f(L_i) = f(L_j)\ \forall L_i, L_j \in H$. The "concurrency-collision-freeness" property should be guaranteed in authenticating logs.

We give an example of a history which is linearized into two logs by two sites. The history H, which is built on the ground set of events $P = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6\}$ with "$\xrightarrow{hb}$" relation of events, i.e. $e_0 \xrightarrow{hb} e_1$, $e_0 \xrightarrow{hb} e_2$, $e_1 \xrightarrow{hb} e_3$, $e_2 \xrightarrow{hb} e_4$, $e_2 \xrightarrow{hb} e_3$, $e_3 \xrightarrow{hb} e_5$, $e_3 \xrightarrow{hb} e_6$, $e_4 \xrightarrow{hb} e_6$, is recorded in logs, $L_1 = [e_0, e_1, e_2, e_3, e_5, e_4, e_6]$ and $L_2 = [e_0, e_2, e_4, e_1, e_3, e_6, e_5]$. They both preserve orders of all events of the history H. In order to fulfill concurrency-collision-free property, any authentication mechanism applied to L_1 and L_2 should yield the same result. If the authentication results are different, then it means that one of the logs was tampered.

Forward-aggregated authenticity

While logs grow, log verifiers can skip verification of log entries which have been already authenticated. The authentication mechanism should allow accumulation of log verification for a time interval. Not only the integrity of individual log entries but also the integrity of the whole log stream should be preserved. This forward-aggregated authenticity property is similar to forward security and append-only property investigated in many existing works of secure log audit [2, 111, 196].

Public verifiability

This property allows any user in a collaborative system to verify the integrity of logs. Adversaries are made accountable for unauthorized actions. This property can be ensured by using digital signatures such as RSA or DSA signature scheme. Public verifiability is especially desirable in distributed collaborative systems where logs need to be audited by any collaborator without relying on a trusted central authority.

6.2 Authenticator

In this section, we present our approach to construct authenticators $T@site$ to deter users from log tampering while preserving the above mentioned properties.

When a sending site sends a document to a receiving site, it creates an authenticator for its log. The authenticator is attached to the sent document. The receiving site creates a new authenticator when it receives the document. We assume each site involved in this push-pull communication possesses a cryptographic public/private key pair that is assigned to a unique site identifier and that all users can retrieve the public key of each other. This assumption is reasonable in practice [187, 120]. The private key of the key pair is used to sign entries of log to prevent malicious sites modifying events on behalf of other sites. Though sites can choose a public key pair on their-own, to limit Sybil attacks [57] we can require that each site possesses a digital certificate from trusted certification authority or has an offline channel (such as email) to identify the owner of public keys. In either case the certification authority plays no role in the process of authenticator creation, and it is used only during initial phase when a site joins the system. We also use cryptographic hash function with properties collision-resistant (it should be difficult to find two different messages m_1 and m_2 such that $hash(m_1) = hash(m_2)$) and preimage-resistant (with a given hash value h, it should be difficult to find any message m such that $hash(m) = h$). The collision-resistant property can be used to establish the uniqueness of logs at a certain moment when an authenticator is created.

6.2.1 Definition

An authenticator is a log tamper-evident which captures a sub-sequence of operation(s) of a log that were generated in one updating session. An updating session at one user's site is the session between two subsequent push/pull primitives to/from other sites. For example, consider that during a working session, user u generates a log $[e_1,\ e_2]$ where $e_1 \xrightarrow{hb} e_2$. When user u pushes his changes, he creates an authenticator for the sequence of events in the log that their orders should not be tampered by any other user. For instance, a receiver of this log should not be able to re-order e_1 and e_2 to change the happened-before order of e_1 and e_2.

Definition 6.2.1. *An authenticator, denoted as $T_{@site}$, is defined as a tuple $\langle ID,\ SIG,\ IDE,\ PRE,\ SYN\rangle$ where:*

ID: identifier of authenticator which is a tuple $< siteID, opID >$ where $siteID$ is the identifier of the site which creates the authenticator and $opID$ is the operation identifier(s) that

the authenticator is linked to;

SIG: the value of signature signed by the private key of the site;

IDE: a list of operation identifiers used to compute SIG;

PRE, SYN: identifiers of preceding and receiving authenticators.

Definition 6.2.2. *The SIG of an authenticator $T_{@site}$ at a certain update is computed as a signature of a cumulative hash by a sender S or a receiver R, where the sender computes SIG of the most recent authenticator $T_{m@S}.SIG = \sigma_S(hash(T_{m-1@S}.SIG \mid\mid E))$ with condition that $E \neq \varnothing$; and the receiver computes $T_{n@R} = \sigma_R(hash(T_{n-1@R}.SIG \mid\mid E \mid\mid T_{m@S}.SIG)$ with the condition that there exists new update(s) from S appended to log of R, where:*

$T_{m@S}$: the most recent authenticator committed by sender S;

$T_{n@R}$: the most recent authenticator committed by receiver R;

$T_{m-1@S}$: the preceding authenticator of $T_{m@S}$;

$T_{n-1@R}$: the preceding authenticator of $T_{n@R}$;

$E = [e_{i_1}, e_{i_2}, ..., e_{i_r}]$: subsequent changes generated after preceding authenticator;

$\sigma_{site}(\cdot)$ denotes the signature of site *and $\mid\mid$ denotes the concatenation of arguments used in hashing, where hashing can be done using any traditional hash function such as SHA-256.*

The structure of an authenticator is illustrated in Figure 6.1.

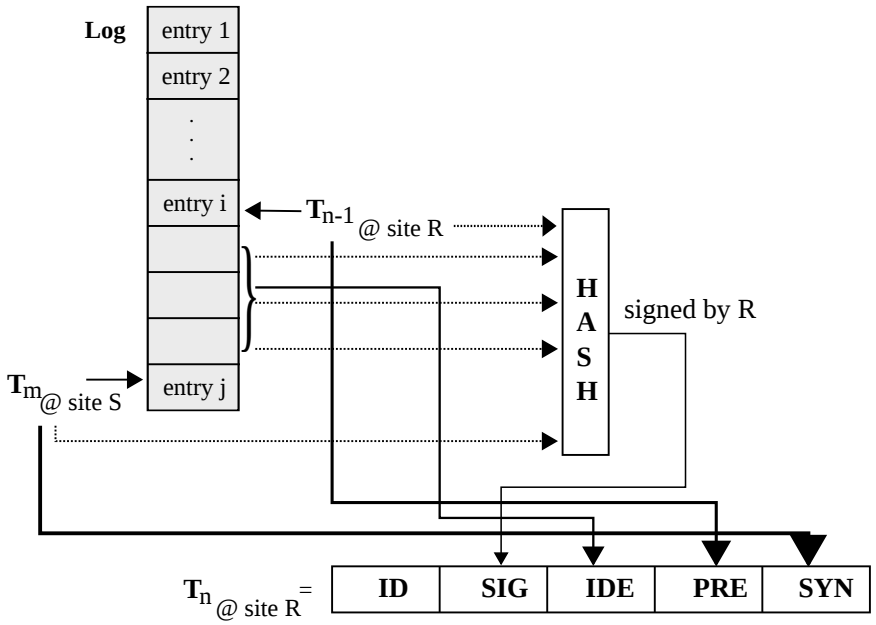


Figure 6.1. Structure of an authenticator.

When a user shares a document by sending the whole log, she creates an authenticator for log events computed based on the preceding authenticator and new updated events. The authenticator is signed by her private key and linked to the last operation(s) of the log. At the receiving site, the receiver performs reconciliation and creates a new authenticator at the reception.

The authenticators of a log of events are constructed whenever a site sends or receives a new change to/from another site. An authenticator is created in following cases:

- A site sends new changes to other sites. In this case, if a site sends a document without new changes, no new authenticator is needed.
- A site receives new changes from other sites. In this case, the receiving site will check the remote log, detect and resolve conflicts (if there are some conflicts among events). After these actions, if there are new changes that are added to the receiver's log, a new authenticator is created for this reception.

6.2.2 Example

We use the early example of history in this chapter where two sites collaborate on a shared document having initial version $V_0|\{e_0\}$ to illustrate the construction of authenticators. We assume that the initial version of the document V_0 consisting of operation e_0 was created by some site among collaborating sites. We further assume that all collaborators agreed on this initial version and that the corresponding log of this initial version does not need to be authenticated. Each of the two sites in our example performs parallel contributions based on the initial version of the document. In the example, *site 1* creates the new version $V_1|\{e_0, e_1\}$ and *site 2* creates $V_2|\{e_0, e_2\}$ concurrently. At a later time, *site 1* reconciles with updates from *site 2* and creates the up-to-date version V_3 | $\{e_0, e_1, e_2, e_3\}$. In Figure 6.2, the two sites, *site 1* and *site 2*, will create authenticators to authenticate their logs each time they do pushing or pulling. In what follows we describe in detail how authenticators are constructed.

- Firstly, when *site 2* pushes his log to *site 1*, it creates an authenticator $T_{1_{@site2}}$ where:

  $T_{1_{@site2}}.ID = < site2, e_2 >$ (linked to e_2),

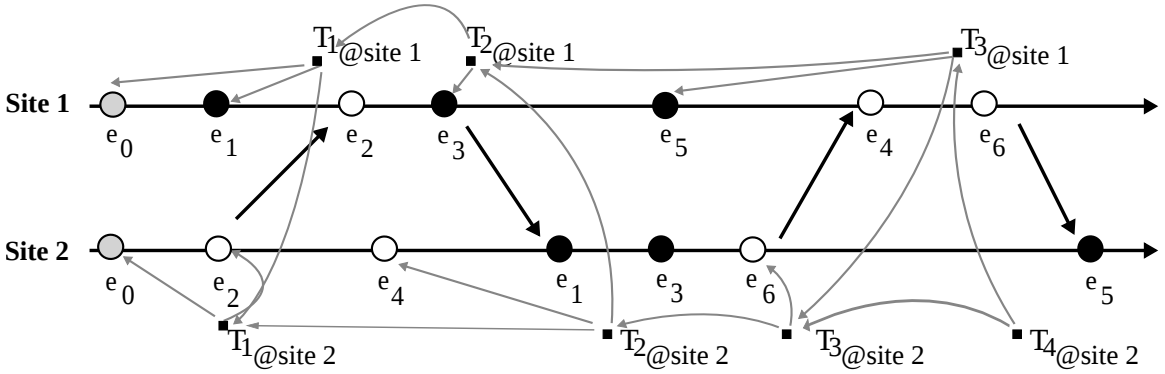


Figure 6.2. Example of constructing authenticators.

$T_{1_{@site2}}.SIG = \sigma_{site2}(hash(e_0 \parallel e_2))$

$T_{1_{@site2}}.IDE = [e_0, e_2]$

$T_{1_{@site2}}.PRE = \varnothing$ (no previous authenticator)

$T_{1_{@site2}}.SYN = \varnothing$ (no received remote authenticator)

- When *site 1* pulls changes from *site 2* and receives the log from *site 2*, it creates an authenticator $T_{1_{@site1}}$ where:

  $T_{1_{@site1}}.ID = < site1, \{e_1, e_2\} >$ (linked to e_1, e_2),

  $T_{1_{@site1}}.SIG = \sigma_{site1}(hash(e_0 \parallel e_1 \parallel T_{1_{@site2}}.SIG))$,

  $T_{1_{@site1}}.IDE = [e_0, e_1]$,

  $T_{1_{@site1}}.PRE = \varnothing$ (no previous authenticator),

  $T_{1_{@site1}}.SYN = T_{1_{@site2}}.ID$

- The two sites *site 1* and *site 2* then work concurrently and generate the new changes e_3 and e_4 respectively. When *site 1* pushes his changes and *site 2* pulls those changes, two authenticators $T_{2_{@site1}}$ and $T_{2_{@site2}}$ are constructed. The structure of $T_{2_{@site1}}$ is given below:

  $T_{2_{@site1}}.ID = < site1, e_3 >$ (linked to e_3),

  $T_{2_{@site1}}.SIG = \sigma_{site1}(hash(T_{1_{@site1}}.SIG \parallel e_3))$,

  $T_{2_{@site1}}.IDE = [e_3]$,

  $T_{2_{@site1}}.PRE = T_{1_{@site1}}.ID$,

$T_{2_{@site1}}.SYN = \varnothing$

- Similarly, $T_{2_{@site2}}$ is computed where:

  $T_{2_{@site2}}.ID = < site2, \{e_3, e_4\} >$ (linked to e_3, e_4),

  $T_{2_{@site2}}.SIG = \sigma_{site2}(hash(T_{1_{@site2}}.SIG \mid\mid e_4 \mid\mid T_{2_{@site1}}.SIG))$,

  $T_{2_{@site2}}.IDE = [e_4]$,

  $T_{2_{@site2}}.PRE = T_{1_{@site2}}.ID$,

  $T_{2_{@site2}}.SYN = T_{2_{@site1}}.ID$

- Again, *site 1* and *site 2* contribute independently to the document, e_5 is generated by *site 1* and e_6 is generated by *site 2*. The two sites then exchange their changes with each other by pushing and pulling other changes. New authenticators are computed at each site. *site 2* computes $T_{3_{@site2}}$ where:

  $T_{3_{@site2}}.ID = < site2, e_6 >$ (linked to e_6),

  $T_{3_{@site2}}.SIG = \sigma_{site2}(hash(T_{2_{@site2}}.SIG \mid\mid e_6))$,

  $T_{3_{@site2}}.IDE = [e_6]$,

  $T_{3_{@site2}}.PRE = T_{2_{@site2}}.ID$,

  $T_{3_{@site2}}.SYN = \varnothing$

- *site 1* computes $T_{3_{@site1}}$ where:

  $T_{3_{@site1}}.ID = < site1, \{e_5, e_6\} >$ (linked to e_5, e_6),

  $T_{3_{@site1}}.SIG = \sigma_{site1}(hash(T_{2_{@site1}}.SIG \mid\mid e_5 \mid\mid T_{3_{@site2}}.SIG))$,

  $T_{3_{@site1}}.IDE = [e_5]$,

  $T_{3_{@site1}}.PRE = T_{2_{@site1}}.ID$,

  $T_{3_{@site1}}.SYN = T_{3_{@site2}}.ID$

- In the last step shown in Figure 6.2, *site 1* pushes his changes to *site 2* and *site 2* pulls these changes from *site 1*. Because there are no new events since the authenticator $T_{3_{@site1}}$ was created, no new authenticator is created by *site 1*. However, the receiving site *site 2*

has to create a new authenticator since a new operation e_5 is added to the log. $T_{4_{@site2}}$ is therefore computed where:

$T_{4_{@site2}}.ID = < site2, \{e_5, e_6\} >$ (linked to e_5, e_6),

$T_{4_{@site2}}.SIG = \sigma_{site2}(hash(T_{3_{@site2}}.SIG \mid\mid T_{3_{@site1}}.SIG))$,

$T_{4_{@site2}}.IDE = []$,

$T_{4_{@site2}}.PRE = T_{3_{@site2}}.ID$,

$T_{4_{@site2}}.SYN = T_{3_{@site1}}.ID$

6.3 Algorithms

In this section, we present algorithms to construct authenticators and verify logs based on authenticators. We also prove that the proposed algorithms satisfy desired properties which we mentioned at the beginning of this chapter.

6.3.1 Authenticators Construction

Algorithm 6.1 presents the algorithm for the construction of an authenticator when a sender pushes his changes. An authenticator is computed from its preceding authenticator and the current generated events. The algorithm takes as argument the log of the sending site S and generates as output the authenticator computed by the sender site. The condition $E \neq \varnothing$ ensures that an authenticator is created only if the sender has generated new changes; otherwise the sender sends the log without computing a new authenticator.

Algorithm 6.2 presents the algorithm for the construction of the authenticator when a receiver site pulls changes from a sender site. An authenticator is computed from the preceding authenticator of local log, the current events generated by the receiver and the most recent authenticator of the remote log. The algorithm takes as arguments the two logs of sending site S and receiving site R and it generates as output the authenticator computed by the receiver site. If there are no new events sent by the sender that have to be added to the log of the receiver, then the receiver will not compute a new authenticator. Note that in synchronizing logs, authenticators that are linked to events must be also kept in the local workspace as they

authenticate previous events in the log.

Algorithm 6.1. Construct an authenticator for a sender (site S, log L_S)

$E \leftarrow$ list of new events S generates after $T_{m-1_{@S}}$;
if $E \neq \emptyset$ **then**
    $T_{m_{@S}}.ID \leftarrow <S$, identifier of most recently local operation at $S>$;
    $T_{m_{@S}}.SIG \leftarrow sign(hash(T_{m-1_{@S}}.SIG \mid\mid E))$;
    $T_{m_{@S}}.IDE \leftarrow E$;
    $T_{m_{@S}}.PRE \leftarrow T_{m-1_{@S}}.ID$;
    $T_{m_{@S}}.SYN \leftarrow \varnothing$;
else
    $T_{m_{@S}} \leftarrow \langle\rangle$;
return $T_{m_{@S}}$;

Algorithm 6.2. Construct an authenticator for a receiver (sites S and R, logs L_S and L_R)

$E \leftarrow$ list of new events R generates after $T_{n-1_{@R}}$;
$E_S \leftarrow$ list of new events from L_S added to L_R;
if $E_S \neq \varnothing$ **then**
    $T_{n_{@R}}.ID \leftarrow < R, T_{m_{@S}}.ID.opID \cup$ identifier of most recently local operation at $R>$;
    $T_{n_{@R}}.SIG \leftarrow sign(hash(T_{n-1_{@R}}.SIG \mid\mid E \mid\mid T_{m_{@S}}.SIG))$;
    $T_{n_{@R}}.IDE \leftarrow E$;
    $T_{n_{@R}}.PRE \leftarrow T_{n-1_{@R}}.ID$;
    $T_{n_{@R}}.SYN \leftarrow T_{m_{@S}}.ID$;
else
    $T_{n_{@R}} \leftarrow \langle\rangle$;
return $T_{n_{@R}}$;

We will consider *time* and *space* complexities of algorithms to construct and verify authenticators. Note that, for the space complexity for verification of authenticators, we exclude the space complexity for maintaining the log. The algorithm to create an authenticator in Algorithms 6.1 or 6.2 is $O(1)$ in time, and $O(|\Delta|)$ in storage, where Δ is the set of events whose identifiers are kept in $T_{@site}.IDE$. Since an authenticator is created each time a site sends or receives changes, the number of authenticators on a replicated object created by site S is the total number of interactions the site has done with other sites. Let Γ be the total number interactions of one site. Then each site needs $O(\Gamma \cdot |\Delta|_{max})$ space for all authenticators, where $|\Delta|_{max}$ is the maximum Δ of all authenticators. In synchronization, one log is updated to become the union of two

logs of sites S and R, and the new log shall need $O(\Gamma_S \cdot |\Delta_S|_{max} + \Gamma_R \cdot |\Delta_R|_{max})$ space for all authenticators. We can see that the storage complexity depends on the number of interactions and the number of events generated by two sites.

6.3.2 Authenticators-based Log Verification

The Algorithm 6.3 presents a mechanism to verify log entries based on authenticators. When a site receives a log of events accompanied by authenticators, it verifies the log based on these authenticators corresponding to entries in the log. The main idea of verification is to check the authenticity of events preserved by valid authenticators, including:

- If authenticators are valid (their signatures are correct). An authenticator is checked by verifying its digital signature by using the public key of signer.
- If the log entries are corresponding to these valid authenticators. When an authenticator passes signature checking, the content and the order of events are taken into account in the verification.

If all of these checkings pass, the log is authenticated. In contrast, a log with either events not authenticated or authenticated by invalid authenticators is unauthorized. With any detection of the corrupted data or falsified order of changes, authenticators will be invalid and the verification algorithm returns negative result. Authenticators help users being aware of attacks and once the log is unauthorized, the site which sent tampered log is made accountable for the misbehavior.

Let us revisit the example in the previous section. Let us assume one of two sites (i.e. *site 1* or *site 2*), for instance, *site 2* shares the document by sending its log to another site, say, *site 3*. Then *site 3* will verify the log it receives from *site 2*. To verify log, *site 3* has to verify the validity of authenticators and log entries. If it already received one part of log before, *site 3* can skip checking every authenticator linked to that part. We now describe the worst case when *site 3* receives the log from *site 2* for the first time and therefore every authenticator needs to be checked. *site 3* performs the following steps of the log verification procedure.

It starts by checking the most recent authenticator of *site 2*, $T_{4_{@site2}}$.

- Verify $T_{4_{@site2}}.SIG$ by using *site 2*'s public key.

Algorithm 6.3. Verify a log (site R, log L)

```
1  Q ← T_{n@R} ∈ L; // Q: queue of authenticators to verify
2  verified ← True;
3  while Q ≠ ∅ do
4      T ← Q.get();
5      check1 ← T.SIG is correct;
6      check2 ← order of events in L corresponds to T.IDE list;
7      check3 ← T.PRE precedes events in T.IDE;
8      check4 ← T.PRE and T.SYN precede T.ID;
9      if check1 & check2 & check3 & check4 then
10         mark events in T.IDE as checked;
11         put(Q, T.PRE);
12         put(Q, T.SYN);
13     else
14         verified ← False;
15         break;
16 if any operation in L is not checked then
17     verified ← False;
18 return verified;
```

- Verify $T_{4_{@site2}}.IDE$. As $T_{4_{@site2}}.IDE = []$ then check2 and check3 can be skipped.
- Verify the order of $T_{4_{@site2}}.ID$ (linked to e_5, e_6) and $T_{4_{@site2}}.PRE = T_{3_{@site2}}.ID$ (linked to e_6) by checking if the log is maintained correctly (if e_6 is logged before e_5). Similarly, the order of $T_{4_{@site2}}$.ID and $T_{4_{@site2}}.SYN = T_{3_{@site1}}.ID$ (linked to events e_5 and e_6) is checked.
- Since $T_{4_{@site2}}$ was constructed based on $T_{3_{@site2}}$ and $T_{3_{@site1}}$, these authenticators are put into a queue Q in order to be recursively verified.

If every above check passes then the authenticator $T_{4_{@site2}}$ is said valid. For other authenticators in queue Q, the verification is performed recursively and each verification follows steps in Algorithm 6.3. The verification finishes when queue Q is empty. The final checking result is only positive if all checks return positive result. Otherwise, the log will not be authenticated. Note that in this example, any deletion or re-ordering of events is detectable. For instance, if *site 2* tries to re-order events e_2 and e_3, this attack will be detected by authenticating the authenticator $T_{2_{@site1}}$ which is linked to e_3. We can see *site 2* cannot forge this order on behalf of *site 1* since *site 1* signed the authenticator linked to e_3. However, any re-ordering of concurrent events will

not change the verification result. The proof will be presented later.

Authenticators-based log verification has $O(1)$ complexity in space and $O(\Gamma)$ in time, where Γ is the total number of authenticators in the log. Since authenticators of a log are linked as a hash-chain in which an authenticator is linked to its preceding one, and due to the *forward-aggregated authenticity* property, it is enough to authenticate the log by checking only the most recent authenticator of a log. This verification process requires checking of all preceding authenticators. Therefore, the time complexity depends on the total number of all authenticators.

6.4 Proofs of Correctness

The algorithms, that have been presented previously for authenticators construction and logs verification, ensure the desirable properties for authenticating logs which are linearized from operation-based history.

Theorem 6.4.1. *A log is tamper-detectable by using authenticators. A misbehaving site cannot selectively insert, delete or change the happened-before order of other sites' events from the beginning or the middle of the log without being detected by next audit (Integrity).*

Proof. Let M be the misbehaving site who receives a log $L = [e_1, e_2, \ldots, e_i, e_{i+1}, e_{j-1}, e_j]$ from site R. Let us assume e_i and e_{i+1} were generated by R and e_{j-1} and e_j were received by R from S. Log L is accompanied with authenticators and the most recent authenticator is $T_{j@R}$ which is linked to events (e_{i+1}, e_j). Following Definition 6.2.1 and Definition 6.2.2, $T_{j@R}$ consists of:

$T_{j@R}.ID = <site\ R, \{e_{i+1}, e_j\}>$ (linked to events e_{i+1}, e_j),

$T_{j@R}.SIG = \sigma_{siteR}(hash(T_{i@R}.SIG \mid\mid e_i \mid\mid e_{i+1} \mid\mid T_{j@S}.SIG))$,

$T_{j@R}.IDE = [e_i, e_{i+1}]$,

$T_{j@R}.PRE = T_{i@R}.ID$,

$T_{j@R}.SYN = T_{j@S}.ID$

There are three cases that M can attack the log as follows.

- *Case 1 - misbehaving site M removes events at the beginning or in the middle of the log.*

  (i) If M selectively removes any operation in range from e_i to e_j from L, for example, e_i is removed by M but M still keeps e_{i+1}. The authenticator $T_{j@R}$ is then either invalid

(missing of e_i) or replaced by $T'_{j@R}$. However, $T'_{j@R}$ is invalid since it should be signed by R and M cannot forge R's signature to create $T'_{j@R}$ on behalf of R.

(ii) If M removes any operation before e_i, for example, e_1 is removed, then the authenticator $T_{i_{@R}}$ is invalid and this makes $T_{j@R}$ invalid consequently.

Therefore, if a misbehaving site removes any operation in the middle of the log, the log will not be authenticated by valid authenticators.

- *Case 2 - misbehaving site M changes the happened-before order of events on behalf of others.*

  If M changes the happened-before order of any events from e_i to e_j then the events list $T_{j@R}.IDE$ will be invalid. When e_{i+1} is generated by site R, e_j is generated by site S, and R receives e_j from S after generating e_i, e_{i+1}, we say e_{i+1} and e_j are concurrent and other users can change the order of e_{i+1} and e_j. In the case of changing order of concurrent events, the authenticator $T_{j@R}$ is still valid (it passes the check of Algorithm 6.3 - line 7). However, if R continues to work on the document and adds new operation e_k after e_j, then commits an authenticator $T_{k@R}$, other sites cannot change the order of e_i, e_j and e_k since this misbehavior will make the authenticator $T_{k@R}$ invalid by the checking procedure in Algorithm 6.3 - line 8.

  Therefore, if a misbehaving site changes any happened-before order, the log will not be authenticated.

- *Case 3 - misbehaving site M inserts an operation at the beginning or into the middle of log.*

  We assume misbehaving site M inserts an operation e_m in the middle of existing log between e_i and e_j. If M claims operation e_m was generated by site R then the log will be not authenticated since none of existing authenticators of site R authenticates e_m and it therefore cannot pass the verification process in Algorithm 6.3 - line 17, 18.

  If M claims e_m was generated by himself, then it needs to commit an authenticator to authenticate e_m. In such case, e_m is considered concurrent with other events, so it can be inserted into any position in the log L and L is still authenticated.

  Therefore, a misbehaving site only can insert its own events into its local log. The site cannot claim its insertion as events on behalf of others because it cannot authenticate such

events.

In summary, it is impossible to forge the integrity of a log without being detected by using authenticators. □

Theorem 6.4.2. *Authenticators preserve concurrency-collision-freeness property.*

Proof. In the proof of theorem 1, we use a log L of site R, $L_R = [e_1, e_2, \ldots, e_i, e_{i+1}, e_{j-1}, e_j]$. We assume events e_i, e_{i+1} are concurrent with e_{j-1}, e_j. Thus the order between them can be interchangeable in any linearization of history. Let us consider that site S maintains a different log of same history $L_S = [e_1, e_2, \ldots, e_{j-1}, e_j, e_i, e_{i+1}]$. We will prove that the log verification will return the same result on checking L_S and L_R.

When sites S and R share logs with each other, we suppose that $T_{i_{@R}}$ and $T_{j_{@R}}$ are committed by site R before and after receiving the log from site S; $T_{j_{@S}}$ and $T_{i_{@S}}$ are committed by site S before and after receiving log from site R. The log verification by checking $T_{i_{@S}}$ and $T_{j_{@R}}$ yields the same result regardless the order of concurrent events. Indeed, $T_{i_{@S}}$ and $T_{j_{@R}}$ are valid only if they pass four checks (Algorithm 6.3, line 6 - 9). Consider check1 and check2 were passed, therefore they must pass check3 and check4 to be completely verified. The check3 only deals with the order of preceding authenticator against events list IDE ($T_{i_{@S}}$ with e_i, e_{i+1}, $T_{j_{@R}}$ with e_{j-1}, e_j) and these orders are preserved as proved in Theorem 1. The check4 deals with the orders of preceding and synchronized authenticators with respect to the committed authenticator. Since $T_{j_{@R}}$ is committed after $T_{i_{@R}}$ and $T_{j_{@S}}$ (linked to events e_{i+1}, e_j), the check4 for $T_{j_{@R}}$ passes. Similarly, the check4 for $T_{i_{@S}}$ passes. Therefore, regardless the logging order of concurrent events, the verification yields same result of checking two logs L_S and L_R. □

Theorem 6.4.3. *Authenticators are forward-aggregated.*

Proof. This property is achieved by using hash-chain based authenticator, so that $T_{i_{@site}}$.SIG includes $T_{i-1_{@site}}$.SIG in its construction. □

Theorem 6.4.4. *Every site which is in possession of history can verify authenticators by using the public key of the site which committed them. A site which created an authenticator cannot deny having constructed it (public verifiability).*

Proof. Non-repudiation is an important feature of digital signatures. By this property, a site that has signed authenticators cannot at a later time deny having signed them. Suppose that site S has signed an authenticator for events $e_1, e_2, ..., e_i$ and shared them with another site. At later time, site S wants to change the history by removing e_i (e.g *insert line X*). In that case, site S should add a new operation e_j (e.g *delete line X*) instead of removing operation e_i since this will make authenticator $T_{i_{@S}}$ invalid. Once a log has been shared with other sites, site S cannot remove its events due to the using of non-repudiation signature for committed authenticators. Authenticators are linked to events and replicated together with logs, therefore anyone can authenticate them. □

6.5 Experiment

As the time complexity for the creation of authenticators is not significant, we evaluated the time complexity of the algorithm for log verification based on authenticators. Verification is done when a site clones or pulls remote log and it needs to check if the remote log is shared correctly without any tampering.

We carried out experiments on real logs from projects that used Mercurial as a distributed tool for source code management. We chose randomly two projects: Hgview project [108] and one branch of OpenJDK project [131]. The project *Hgview* includes almost 700 committed patches stored in repository gathering contributions from 20 developers with 115 interactions between them. One branch of *OpenJDK* stored about 350 committed patches in repository which were created by 31 developers with 253 interactions between them. A committed patch is a sequence of events that a user commits. It is also called a log entry. We implemented our experiments by using Python programming language.

In the histories of projects developed with Mercurial or any other distributed version control system, we are unable to know when a user pulls changes. We can only have information about push events. We therefore considered the worst case scenario where a pull is performed at each new entry in the repository by an arbitrary user Y that had never interacted before with any other user X that contributed to the project. If previous interactions were taking place between users Y and X, an optimization could be applied.

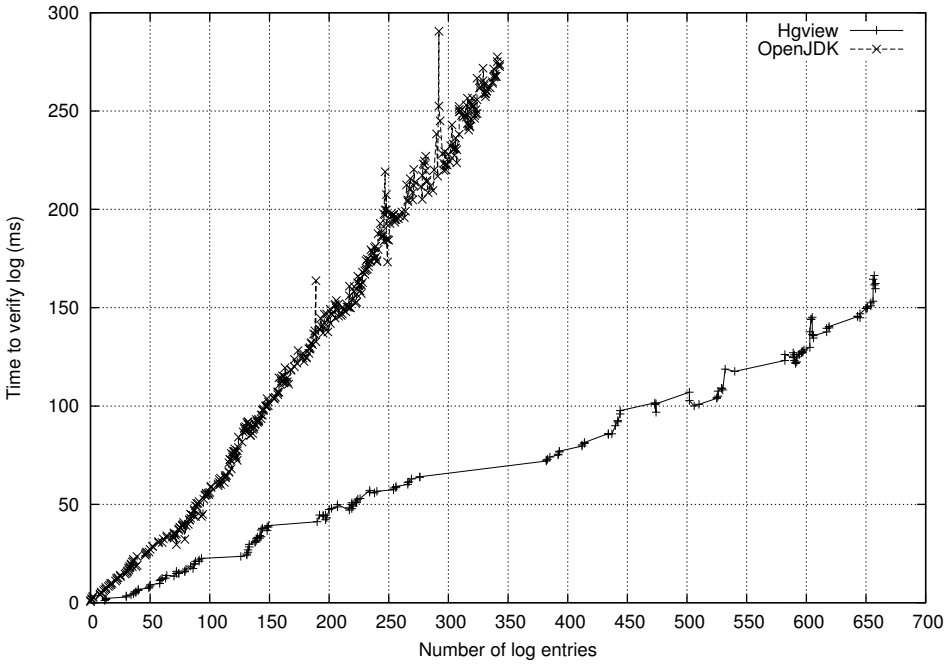


Figure 6.3. Time overhead to check authenticators created for Hgview and OpenJDK repositories.

In the experiment, we have first traversed the repository to extract entries and user names who contributed to the project. Then we generated for each user one RSA key pair which is later used to sign authenticators. Authenticators are created based on linearized logs of repositories. Finally, verification time is measured for the worst case scenario where a newcomer clones the repository and she has to check all authenticators created by previous contributors. The results are computed by the average values of five run times. Figure 6.3 presents the experimental results for the worst case behavior. In two experiments with the input data from Hgview project and OpenJDK project, the checking time grows linearly with the increasing number of authenticators. It is observed that it takes less than 50 milliseconds (ms) to verify a log if its size is less than 100 entries (this size is common for the size of all files observed in these projects). However, to check the whole repository, the verifying time depends mostly on the number of interactions between users (this means also the number of authenticators). In Figure 6.3, we notice that the runtime to verify log of the *Hgview* project is less than that of *OpenJDK* project even though its log size is bigger.

The main conclusion to be drawn from the results is that adding authenticators to secure logs does not create a significant time overhead for collaborative systems even for the worst case.

Sites can reduce the time to verify log by skipping authenticators which are already checked when previous pulls were performed.

6.6 Related Work

In this section, we give a review of existing securing log schemes and highlight their non suitability for securing logs in multi-synchronous collaboration.

Several works introduce a trusted server to be used for verification. This approach makes the system open to a single point of failure. Peterson et al. [141] presents an approach to secure version history in a versioning file system. The approach proposed a design of a system based on generating message authentication codes MACs for versions and archiving them with a third party. A file system commits to a version history when it presents a MAC to the third party and at a later time, an auditor can verify this history. Thus it requires that the system trusts the third party that maintains MACs correctly. In the general model of multi-synchronous collaboration, users have no need to rely on a trusted third party and therefore any user should be able to verify logs.

In a similar direction, Haeberlen et al. designed PeerReview [79] to provide accountability and fault detection for distributed systems. It guarantees the eventual detection of all Byzantine faults. PeerReview framework contains tamper-evident logs and commitment, consistency and audit protocols. However, each node should have an identical log with others. It does not support that each node can keep different orders of events as in operation-based multi-synchronous collaboration where users maintain different streams of activity on the shared data. The framework offers three applications of overlay multicast, network file system and peer-to-peer email, however, all of these applications do not deal with parallel modifications of data that is the case in multi-synchronous collaboration.

The integrity of audit logs has traditionally been protected through the use of one-way hash functions. There is a line of work that addresses the forward-secure stream integrity for audit logs. Ma et al. proposed a set of secure audit logging schemes and aggregate signatures [111, 110, 112]. Forward security ensures the integrity of log entries in the log stream and no selective deletion or re-ordering to stream is possible. Recently, Yavuz et al. proposed their work to secure audit

log such as BAF [196] that was developed to achieve at the same time the computationally efficient log signing and the resistance of log truncation attack. This work could be applied only for a particular case of the multi-synchronous collaboration where all users maintain the same linearization of the collaboration history. However, it cannot be applied for the general case of the multi-synchronous collaboration where users work on different streams of activity on the shared data corresponding to different linearizations of collaboration history.

There is another line of work that relies on authenticated data structures to secure logs in distributed systems [76, 116, 115, 136]. While these approaches are computationally efficient, they do not deal with history for collaboration. Maniatis et al. introduced Timeweave [115] that uses a time entanglement mechanism to preserve the history state of distributed systems in a tamper-evident manner. However, Timeweave does not handle the information flows synchronization which is required for multi-synchronous streams where concurrent events appear in different orders in replicas.

Apart from above approaches, there are works that address securing logs for replication systems. Spreitzer et al. [176] uses hash chain to protect modification orders of a weakly consistent, replicated data system. Kang et al. [92, 91] proposed SHH for optimistic replication using hash values as in Merkle tree [121] for revision identifiers to protect causality of version history. It supports the purpose of securing version history construction when the log was pruned in limited storage environments such as mobile computing; and for checking distributed replicas' convergence. By ensuring decentralized ordering correctness, SHH can guarantee that all updates are not vulnerable to a decentralized ordering attack. However, SHH cannot ensure the integrity of data in the sense that it is original or forged. Using SHH the sender signature cannot be included in summary hashes since that makes them different even if the merged versions are identical, thus it makes replicas diverge. Without digital signature in summary hashes, SHH cannot protect history from attacks of unauthorized actions and it cannot provide authenticity and accountability. Similar approaches to SHH in which hashes are used as identifiers are implemented in distributed version control systems such as Git history [107] and Mercurial history [132].

Concerning securing document history, Hasan et al. [80] proposed a mechanism of preventing history forgery for a document history where a document refers to a file or database. In [80],

the term "provenance" is used for the history of the ownership of items and actions performed on them. The authors present a provenance-aware system prototype that captures history of document writes at the application layer. To prevent all potential attacks on provenance chain, it requires trusted pervasive hardware infrastructure at the level where tracking is performed. However, contributions to the shared document/database are done sequentially and the approach does not deal with merging of parallel contributions to the shared document.

A different work, Mella et al. [120] proposed a framework to the document control flow in a highly distributed environment. The proposal is aimed at cooperative updates on a document flow with delegation and security policies. However, it considers one stream of update process rather than a multi-way flow of updating with reconciliation as in multi-synchronous model. Moreover, security access control policies are defined at document's attributes level that means each document atomic element is marked with a label containing a set of access control policies that apply to it. The approach described in [120] secures different XML elements, while we aim to secure patches of events.

In the domain of database security, Mahajan et al. [113] proposed Depot to secure replicated database in the cloud. Among all issues addressed in Depot, we focus on the issues of consistency, integrity and authorization. Depot addresses these issues in the context of database where data is stored in the form of key/value and update is the main operation performed over database. We consider collaborative systems with more events beyond update, i.e. insert, delete content to/from the shared document. Depot ensures consistency by using version vectors and version history hashes. Each update is signed by authorized node to enforce consistency and integrity. This would be too costly in a collaborative working environment where users produce a huge number of events on the shared document. Our approach secures logs without requiring that each user signs each operation. Authenticators are created for a patch of events each time the log is pushed/pulled to/from one user.

The state of the art of secure audit logging research was also surveyed by Accorsi [6]. Though secure audit logging was intensively investigated, we are not aware of any work that ensures secure audit logs for a collaboration history with partial order where users maintain different total ordered logs of the collaboration history corresponding to their activity streams.

6.7 Summary and Discussion

In this chapter, we introduced a solution using authenticators for a security challenge in operation-based multi-synchronous collaboration. Authenticators are used to ensure integrity and authenticity of logs of events corresponding to different streams of activity during a collaborative process. While tamper-resistance is impossible to be ensured in multi-synchronous collaboration without a central provider, tamper-detection should be guaranteed. We presented an approach for securing logs that made misbehaving users accountable in collaborative systems without the need of a central authority. We provided proofs of correctness of our approach and analyze the complexities of our algorithms. We also conducted a set of experiments testing our proposed approach on real histories of collaboration extracted from real projects using Mercurial. The results show the feasibility of our approach that can be used to provide security, trustworthiness and accountability to distributed collaborative systems.

Chapter 7

Conclusion

In this thesis we have introduced the C-PPC model, a contract-based and trust-aware multi-synchronous collaboration model. The core idea underlying this model is that the contract, which is built from deontic concepts of the permitted, the obligatory and the forbidden, is integrated inside the collaborative model. People who join a collaboration are expected to comply to given contracts. The contract compliance is used as a measurement for trust that one person puts into another one as her personal assessment. The contract-based C-PPC model indirectly provides a trust awareness mechanism at two views.

(1) A user is aware that when she performs actions she should respect given contracts to gain good trust levels others will put on her. The system can notify her the contracts that she is holding so that she should respect those contracts in doing actions and the possible conflicting between contracts when she synchronizes data. To gain good trust levels she should take into account all concerning notifications.

(2) The user is aware of behavior of other users that is reflected in their assigned levels of trust.

Trust and contract are interrelated. Trust is evaluated based on contract compliance. Contracts are given based on trust levels. We have applied this idea in designing the C-PPC model. Users start up the collaboration based on their social trust. Throughout the life time of working process, people reassess trust so that they can decide how to continue to collaborate with others. All operations of users on the shared data are recorded in one log that is a linear extension of

multiple working streams. To check contract compliance, log auditing is performed locally by users. By applying a trust metric, a single user recomputes trust value that she assigns to others after each log audit.

One of the key challenges we faced throughout this thesis is that in multi-synchronous model no central authority exists to maintain logs as well as to perform log auditing. Even though users work collaboratively on the same shared data, they maintain individual work streams which are synchronized when they share their works with each other. Since users work independently and concurrently, managing an enforcement mechanism for contracts is more difficult than the same issue in a model with single sequential workflows only. For instance, when individual logs are synchronized, not only operations but also contracts will be merged. Existing CRDT solutions solve only the problem that concurrent operations can be merged in any order. However, dealing with contracts that must be maintained in correct order raises new issues. We have addressed and solved this issue in the thesis.

7.1 Outcomes

The key conclusion we have reached throughout this thesis is that adding contracts to multi-synchronous model is practical and benefic for users. In the introduction to this thesis (in chapter 1), we posed a set of questions. We revise each question here and discuss its answer through major results presented in this thesis.

- *Which rights and obligations (contracts) should be respected in using shared data? How existing approaches deal with this problem? Furthermore how to express contracts and integrate them in the multi-synchronous model?*

We used deontic model introduced by the philosopher Von Wright which states that every kind of actions can be expressed as permitted, obligatory or forbidden (chapter 2). By surveying existing approaches (chapter 3) we highlighted the gap that traditional access control is too strict and solve partly the problem of controlling how data is used after it is given away. Moreover it deals mostly with *a priori* check and *preventive* enforcement. More advanced and closer to our work, usage control deals with *a posteriori* check after shared data has been used and *preventive*

enforcement. In order to provide more flexible approach for collaborative environments, we aim at a model that deals with *a posteriori* check and *detective* enforcement. We described how contracts are embodied with shared data. Contracts are kept in log as events with special attributes: event type is contract and modality is either P, F, or O. We distinguished write events, communication events and contract events by their event types (chapter 4). This distinction is used in log auditing by comparing write or communication events in accordance with contract events. Integrating contracts into multi-synchronous model implies their integration in the replication mechanism. This requires a merging solution for action events and contract events. Contributions for the merging algorithm, the solutions for contract conflict and the method for contract ordering were proposed (chapter 4). To conclude, by answering this question we get the first outcome of this thesis with a general model for contract-based multi-synchronous collaboration that is called the C-PPC model.

- *Which enforcement mechanism is used in contract-based model? When is log auditing performed and how are auditing results used as a measurement to assess user trust?*

As mentioned before, our approach adopts *detective* enforcement rather than *preventive* enforcement. Since trust is introduced into the model as a precondition to start up the work between users, users are free to bring their contributions without being enforced to respect given contracts for each user. User actions are audited only afterward by individuals. The audit results are used to compute trust levels assigned to audited persons by an auditor. We used a trust metric which allows to assess trust value based one past actions that are good, bad or unknown. Results range in an interval [0..1]. The obtained trust values could be used as a main part of a trust model. The third outcome of our work is a novel trust metric and log auditing algorithm presented in chapter 5. Note that though our work offers an alternative for managing data *a posteriori*, it does not replace other strong enforcement mechanisms for secured data management or other usage control approaches. Dealing with completely open and decentralized collaborative environments raises some difficulties that make strong enforcement mechanism not straightforward to be applied.

- *How to secure logs in an open, decentralized environment so that users cannot modify them to hide their misuses?*

Maintaining logs correctly is very critical to ensure that log auditing will return right results. We proposed a solution to authenticate logs which maintain all users actions and their given contracts. We used hash chain based to construct authenticators for authenticating logs. Based one these authenticators, any tampering to those logs is discovered before it is synchronized with the local log of a certain user. Authenticators deter users misusing shared data. However, they cannot prevent them from tampering the log. Log tampering will be detected after the fact when a user received a log from a remote user. To show the feasibility of the proposed solution, we carried out some experiments based on data from real projects using Mercurial software. We concluded that the overhead of using authenticators mainly depends on the frequency of exchange between users. The detail of this outcome can be found in chapter 6.

Beyond major outcomes, there are also several limitations in our work (see discussion in chapter 4). Namely, the growing infinitely of log size, the possibility to apply to multiple documents with file operations, and the time overhead caused by adding contracts to collaboration is still linear. We highlight such limitations so that any further consideration for applying the model must take them into account.

7.2 Outlook

Here we propose several ideas for future work that extend our current model.

- **Conducting user studies over practical applications**.

The work of this thesis was not yet validated in practical applications. Simulation environment that we used has several limitations compared to real world collaborative applications since it lacks human factors for making decision. Our work has immediate applications such as distributed version control systems or photo sharing over friend-to-friend networks. We took an example of building collaboratively a photo album to illustrate how the C-PPC model works. For future work, we plan to implement this application as a plug-in over social networks with real users. To fit our decentralized model, it is ideal to deploy applications over a distributed P2P social networks such as Friendica [65], Diaspora [55]. There are several goals and issues for conducting user studies.

(1) One of the most important issues we intend to evaluate the work of this thesis over practical applications is its usefulness for end-users. Today people are usually busy with overwhelming tasks. In practical applications we need to consider if users are pleasant with specifying contracts for usage control each time they share their data.

(2) Another important issue is the system responsive time to end-users. Adding contracts to multi-synchronous model certainly increases time overhead. It is necessary to check if the time overhead upsets users. If it is the case we need to proceed with suitable solutions to achieve good trade-off between a model with richer feature of using contracts and the traditional model with higher responsiveness.

(3) Another issue concerning to our future work for practical applications is about privacy. OECD (Organization for Economic Cooperation and Development) defined basic privacy principles including: collection limitation, data quality, purpose specification, use limitation, security safe, openness, individual participation, accountability. Within our current work where data usage control is provided as a new feature for multi-synchronous model, we consider data privacy from the point of *use limitation* that users will specify how their data will be used after given away. User activities are logged and each time a user shares the log, all past activities of all users are disclosed to receiver to serve for log audit and trust assessment. By doing this we introduce somewhat a privacy breach in the sense that activities of a particular user are given to some others whom she may not want to share. Ensuring this *anonymous* privacy while still provide log-audit-ability is an interesting challenge to address further especially in some applications where user privacy is a critical characteristic as well as the ability for cooperation.

- **Applying commutative hash to build authenticators**.

Ensuring logs are maintained correctly is critical in log-based collaboration. We have proposed to use authenticators to verify the validity of logs created and shared between participants. Our solution is based on a hash chain that is aggregated over time to time when new log entries are added, new interactions are performed. At each time of hash value computing, all data for hashing have to be used and the computation starts from the scratch. Since time overhead to

verify log with those authenticators is linear (shown in an experiment in chapter 5), we should find an alternative to get a better performance in constructing authenticators. Moreover, in our authenticators $T@site$ we had to make a rule for the order of the preceding authenticator and remote authenticator even they can be put in any order in hash computing. When $T@site$ is recomputed to verify log, the order is relaxed. In a future work, we would like to propose a solution for hashing without considering order between commutative elements involved in output hash.

As far as we know, one-way accumulators with *quasi-commutative* property proposed by Benaloh and Mare [23] is a worth direction to be investigated as a hashing alternative. The one-way accumulator offers a property that $h(h(x, y_1), y_2) = h(h(x, y_2), y_1)$ with function $h : X \times Y \rightarrow X$ for all $x \in X$ and for all $y_1, y_2 \in Y$. Using *quasi-commutative* function to time stamp log to make it impossible to be forged might provide space-efficiency and eliminate the need for a trusted central authority. This fits the logs for multiple streams collaboration within our context.

- **Extending wider ranges of contracts expressed in C-PPC model**.

In C-PPC model, contracts and actions are put in a relative temporal dependence in the sense that when actions and contracts refer to the same operation and the actions are done after the contracts have been issued, the actions should comply to these contracts. Otherwise, the actions are considered as violations. In real world, people usually express contracts with additional properties, such as *"it is permitted to edit this article before submission deadline only"*, *"it is obligatory to send a feedback an edited version after 3 days"*. Deontic logic is inadequate to express these additional information. Combining deontic logic and temporal logic therefore is an alternative to extend C-PPC model with more expressive contracts. With this extension, evaluating contract compliance requires additional procedures adapted to temporal logic. Moreover, auditing result can be extended to have more than three discrete values: bad, unknown or good. Consequently, the trust metric might need to be changed to cover all possible auditing results used for trust computation.

7.3 Closing Words

Above all, our work presents a complete collaboration model based on contracts that support trust-awareness. We have taken a first step toward building a trust-aware collaboration based on multiple streams. Our hope is that this thesis will inspire other researchers to further explore how trustworthy collaboration can be deployed over open, decentralized environments to support collaborative works of human beings.

List of Publications

Journal articles

(1) Hien Thi Thu Truong, Claudia-Lavinia Ignat, Pascal Molli, "A Contract-extended Push-Pull-Clone Model for Multi-Synchronous Collaboration", *International Journal of Cooperative Information Systems*, Vol.21, No.3(2012), pp. 221-262, World Scientific.

Conference papers

(2) Hien Thi Thu Truong, Claudia-Lavinia Ignat, Pascal Molli, "Authenticating Operation-based History in Collaborative Systems", *In Proceedings of the 17th ACM International Conference on Supporting Group Work, Group 2012*, pp. 131-140, Sanibel Island, Florida, USA, October 2012.

(3) Hien Thi Thu Truong, Claudia-Lavinia Ignat, Pascal Molli, "Securing Logs in Operation-based Collaborative Editing", *The Twelfth International Workshop on Collaborative Editing Systems, IWCES12*, in conjunction with ACM CSCW 2012, Seattle, Washington, USA, February 2012.

(4) Hien Thi Thu Truong, Claudia-Lavinia Ignat, Mohamed-Rafik Bouguelia, Pascal Molli, "A contract-extended push-pull-clone model", *In Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2011*, pp. 211-220, Orlando, Florida, USA, October 2011 (Best Paper Award).

(5) Hien Thi Thu Truong, Mohamed-Rafik Bouguelia, Claudia-Lavinia Ignat, Pascal Molli, "Collaborative Editing with Contract over Friend-to-Friend Networks", *Atelier Protection*

de la Vie Privée / Géolocalisation et Vie Privée (APVP2011), Toulouse, France, 2011.

(6) Hien Thi Thu Truong, Claudia-Lavinia Ignat, "Log Auditing for Trust Assessment in Peer-to-Peer Collaboration", *In Proceedings of the 10th International Symposium on Parallel and Distributed Computing - ISPDC 2011*, pp. 207-214, Cluj-Napoca, Romania, 2011.

Publications before PhD

(7) Dang Thu Hien, Trinh Nhat Tien, Truong Thi Thu Hien, "An Efficient Identity-Based Broadcast Signcryption Scheme", *In Proceedings of the Second International Conference on Knowledge and Systems Engineering (KSE 2010)*, pp. 209-216, Hanoi, Vietnam, 2010.

(8) Truong Thi Thu Hien, Shin-ichiro Eitoku, Tomohiro Yamada, Shin-yo Muto, Masanobu Abe, "An Ontological Approach to Lifelog Representation for Disclosure Control", *In Proceedings of the 13th IEEE International Symposium on Consumer Electronics - ISCE 2009*, pp.934-938, Kyoto, Japan, 2009.

(9) Truong Thi Thu Hien, Shin-ichiro Eitoku, Tomohiro Yamada, Shin-yo Muto, Masanobu Abe, "Ontology Model of Personal Life-log for Fine-grained Disclosure Control", EICE Technical Report., IEICE-OIS2008-90, OIS-462, pp.89-94, Okinawa, Japan, 2009.

Bibliography

[1] Break-glass: An approach to granting emergency access to healthcare systems. White paper. Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC), 2004.

[2] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '00, pages 116–129, London, UK, UK, 2000. Springer-Verlag.

[3] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6 - Volume 6*, HICSS '00, pages 6007–, Washington, DC, USA, 2000. IEEE Computer Society.

[4] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, 2001.

[5] Alan Abrahams, David Eyers, and Jean Bacon. An asynchronous rule-based approach for business process automation using obligations. In *Proceedings of the 2002 ACM SIGPLAN workshop on Rule-based programming*, RULE '02, pages 93–103, New York, NY, USA, 2002. ACM.

[6] R. Accorsi. Safe-keeping digital evidence with secure logging protocols: State of the art and challenges. In *IT Security Incident Management and IT Forensics, 2009. IMF '09. Fifth International Conference on*, pages 94 –110, sept. 2009.

[7] Rafael Accorsi and Thomas Stocker. Automated privacy audits based on pruning of log

data. In *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops*, EDOCW '08, pages 175–182, Washington, DC, USA, 2008. IEEE Computer Society.

[8] William J. Adams and Nathaniel J. Davis. Toward a decentralized trust-based access control system for dynamic collaboration. In *IEEE Workshop on Information Assurance and Security*, page 324, United States Military Academy, West Point, NY, USA, 2005.

[9] B. Thomas Adler and Luca de Alfaro. A content-driven reputation system for the wikipedia. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 261–270, New York, NY, USA, 2007. ACM.

[10] Montresor Alberto and Jelasity Márk. PeerSim: A scalable P2P simulator. In *Proceedings of the 9th International Conference on Peer-to-Peer, P2P'09*, pages 99–100, Seattle, WA, September 2009.

[11] Carlos E. Alchourron and Eugenio Bulygin. The expressive conception of norms. In Risto Hilpinen, editor, *New Studies in Deontic Logic*, pages 95 – 124. D. Reidel Publishing Company, 1981.

[12] Alvaro Arenas and Michael Wilson. Contracts as trust substitutes in collaborative business. *Computer*, 41(7):80–83, July 2008. IEEE Computer Society Press.

[13] Donovan Artz and Yolanda Gil. A survey of trust in computer science and the semantic web. *Web Semant.*, 5(2):58–71, June 2007.

[14] Robert Axelrod. *The evolution of cooperation.* Basic Books, New York, 1984.

[15] Franz Baader, Andreas Bauer, and Marcel Lippmann. Runtime verification using a temporal description logic. In *Proceedings of the 7th international conference on Frontiers of combining systems (FroCoS'09)*, pages 149–164, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] Jean Bacon, Ken Moody, and Walt Yao. A model of oasis role-based access control and its support for active security. *ACM Trans. Inf. Syst. Secur.*, 5:492–540, November 2002.

[17] Ronald M. Baecker and Others. *Readings in Human Computer Interaction: Toward the Year 2000*, chapter 11: Groupware and Computer Supported Cooperative Work, pages 741–782. Morgan Kaufmann, second edition, 1995.

[18] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (SP'06)*, pages 184–198, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Adam Barth, John Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 279–294, Washington, DC, USA, 2007. IEEE Computer Society.

[20] David Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zălinescu. Enforceable security policies revisited. In *Proceedings of the First international conference on Principles of Security and Trust (POST'12)*, pages 309–328, Berlin, Heidelberg, 2012. Springer-Verlag.

[21] David A. Basin, Matús Harvan, Felix Klaedtke, and Eugen Zalinescu. Monitoring usage-control policies in distributed systems. In *Eighteenth International Symposium on Temporal Representation and Reasoning (TIME'11)*, pages 88–95, Lübeck , Germany, September 2011. IEEE.

[22] Bazaar. Bazaar - Version Control System. `http://http://bazaar.canonical.com`, 2007.

[23] Josh Benaloh and Michael de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, EUROCRYPT '93, pages 274–285, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[24] Jeremy Bentham and Herbert Lionel Adolphus Hart. *Of Laws in General*. University of London, Athlone Press, 1945.

[25] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. Knowl. Data Eng*, pages 827–842, 2004.

[26] Andrew D. Birrell, Roy Levin, Michael D. Schroeder, and Roger M. Needham. Grapevine: an exercise in distributed computing. *Commun. ACM*, 25:260–274, April 1982.

[27] BitKeeper. BitKeeper - The Scalable Distributed Software Configuration Management System. `http://www.bitkeeper.com/`, 1997.

[28] Bob Blakley. The emperor's old armor. In *Proceedings of the 1996 workshop on New security paradigms*, NSPW '96, pages 2–16, New York, NY, USA, 1996. ACM.

[29] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust-management system version 2, 1999.

[30] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–, Washington, DC, USA, 1996. IEEE Computer Society.

[31] Achim D. Brucker and Helmut Petritsch. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 197–206, New York, NY, USA, 2009. ACM.

[32] Sonja Buchegger, Doris Schiöberg, Le Hung Vu, and Anwitaman Datta. Peerson: P2p social networking - early experiences and insights". In *Proceedings of the Second ACM Workshop on Social Network Systems 2009, co-located with Eurosys 2009*, Nürnberg, Germany, March 31 2009.

[33] Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17:603–619, July 2008.

[34] Michelle Cart and Jean Ferrie. Asynchronous reconciliation based on operational transformation for P2P collaborative environments. In *COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 127–138, Washington, DC, USA, 2007. IEEE Computer Society.

[35] Jan G. Cederquist, Ricardo Corin, M. A. C. Dekker, Sandro Etalle, J. I. den Hartog, and Gabriele Lenzini. Audit-based Compliance Control. *International Journal of Information Security*, 6(2):133–151, March 2007.

[36] Sudip Chakraborty and Indrajit Ray. Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, SACMAT '06, pages 49–58, New York, NY, USA, 2006. ACM.

[37] Sun Chengzheng, Jia Xiaohua, Zhang Yanchun, Yang Yun, and Chen David. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, March 1998.

[38] Dickson K. W. Chiu, S. C. Cheung, and Sven Till. A three-layer architecture for e-contract enforcement in an e-service environment. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 3 - Volume 3*, HICSS '03, pages 74.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[39] Laurence Cholvya and Anthony Hunterb. Merging requirements from a set of ranked agents. *Knowledge-Based Systems*, 16(2):113 – 126, March 2003.

[40] Cheun N Chong and Zhonghong Peng. Secure audit logging with tamper-resistant hardware. In *18th IFIP International Information Security Conference (IFIPSEC), volume 250 of IFIP Conference Proceedings*, pages 73–84. Kluwer Academic Publishers, 2003.

[41] David Clark. Face-to-Face with Peer-to-Peer Networking. *Computer*, 34(1):18–21, January 2001.

[42] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory 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.

[43] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, pages 317–334, 2009.

[44] Frédéric Cuppens and Nora Cuppens-Boulahia. Modeling contextual security policies. *International Journal of Information Security*, 7(4):285–305, July 2008.

[45] Darcs. Darcs - Version Control System. `http://http://bazaar.canonical.com`, 2003.

[46] Anupam Das and Mohammad Mahfuzul Islam. SecuredTrust: A Dynamic Trust Computation Model for Secured Communication in Multiagent Systems. *IEEE Transactions on Dependable and Secure Computing*, 9(2):261–274, March 2012.

[47] Aspassia Daskalopulu, Theo Dimitrakos, and Tom Maibaum. E-contract fulfilment and agents' attitudes. In *In Proceedings of the ERCIM WG E-Commerce Workshop on The Role of Trust in e-Business*, 2001.

[48] Darren Davis, Fabian Monrose, and Michael K. Reiter. Time-Scoped Searching of Encrypted Audit Logs. In *ICICS*, pages 532–545, 2004.

[49] Debian. Debian Social Contract. `http://www.debian.org/social_contract#guidelines`, april 2004.

[50] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

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

[52] Gerardine Desanctis and R. Brent Gallupe. A foundation for the study of group decision support systems. *Management Science*, 33(5):589–609, May 1987.

[53] Prasun Dewan and Honghai Shen. Controlling Access in Multiuser Interfaces. *ACM Transactions on Computer-Human Interaction*, 5(1):37–62, March 1998.

[54] Prasun Dewan and HongHai Shen. Flexible meta access-control for collaborative applications. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 247–256, New York, NY, USA, 1998. ACM.

[55] Diaspora. Diaspora. `http://joindiaspora.com`, 2012.

[56] Alan Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003. chapter 19, page 466-467.

[57] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.

[58] Paul Dourish. The parting of the ways: divergence, data management and collaborative work. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'95, pages 215–230, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[59] W. Keith 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.

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

[61] Sandro Etalle and William H. Winsborough. A posteriori compliance control. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 11–20, New York, NY, USA, 2007. ACM.

[62] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proceedings of the 15th National Computer Security Conference*, pages 554 – 563, Baltimore MD, USA, 1992.

[63] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4:224–274, August 2001.

[64] Mattern Friedemann. Virtual Time and Global States of Distributed Systems. In Michel Cosnard et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Chateau de Bonas, France, October 1989. Elsevier Science Publishers B. V.

[65] Friendica. Friendica. `http://friendica.com/`, 2012.

[66] Keith Frikken, Mikhail Atallah, and Jiangtao Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Trans. Comput.*, 55(10):1259–1270, October 2006.

[67] Gallery. `http://gallery.menalto.com/`, 2000.

[68] Gallery. Gallery - User Survey 2009. `http://codex.gallery2.org/Gallery3:Season_of_Usability_2009_Survey_Results`, 2009.

[69] Pedro Gama and Paulo Ferreira. Obligation policies: an enforcement platform. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 203–212, Stockholm, Sweden, june 2005.

[70] Diego Gambetta. *Can We Trust Trust?*, chapter Trust: Making and Breaking Cooperative Relations, pages 213–237. Department of Sociology, University of Oxford, 1988. This was originally published in hardcopy in 1988, this is electronic copy circa 2000.

[71] Deepak Garg, Limin Jia, and Anupam Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS'11)*, pages 151–162, New York, NY, USA, 2011. ACM.

[72] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan 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.

[73] Yolanda Gil and Varun Ratnakar. Trusting information sources one citizen at a time. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 162–176, London, UK, UK, 2002. Springer-Verlag.

[74] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent available partition-tolerant web services. In *In ACM SIGACT News*, page 2002, 2002.

[75] Cheh Goh and Adrian Baldwin. Towards a more complete model of role. In *Proceedings of the third ACM workshop on Role-based access control*, RBAC '98, pages 55–62, New York, NY, USA, 1998. ACM.

[76] Michael T. Goodrich, Roberto Tamassia, and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. *DARPA Information Survivability Conference and Exposition,*, 2:1068, 2001.

[77] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(4):659–685, 2006.

[78] Andreas Gutscher. A Trust Model for an Open, Decentralized Reputation System. In *Proceedings of the Joint iTrust and PST Conferences on Privacy Trust Management and Security (IFIPTM 2007)*, July 2007.

[79] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. *SIGOPS Oper. Syst. Rev.*, 41:175–188, October 2007.

[80] Ragib Hasan, Radu Sion, and Marianne Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *FAST*, pages 1–14, 2009.

[81] Manuel Hilty, Alexander Pretschner, Christian Schaefer, and Thomas Walter. Duke - distributed usage control enforcement. In *POLICY*, page 275, 2007.

[82] Jason E. Holt. Logcrypt: forward security and public verification for secure audit logs. In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research - Volume 54*, ACSW Frontiers '06, pages 203–211, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[83] Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 177–189, New York, NY, USA, 2004. ACM.

[84] Tom Hvitved, Felix Klaedtke, and Eugen Zălinescu. A trace-based model for multiparty contracts. *Journal of Logic and Algebraic Programming*, 81(2):72 – 98, 2012. Formal Languages and Analysis of Contract-Oriented Software (FLACOS'10).

[85] Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli, Michelle Cart, Jean Ferrie, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. A Comparison of Optimistic Approaches to Collaborative Editing of Wiki Pages. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2007*, page 10, White Plains, New York, USA, November 2007. IEEE Computer Society.

[86] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. Towards a theory of accountability and audit. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 152–167, Berlin, Heidelberg, 2009. Springer-Verlag.

[87] Robert Johansen. *GroupWare: Computer Support for Business Teams*. The Free Press, New York, NY, USA, 1988.

[88] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, March 2007.

[89] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 120–, Washington, DC, USA, 2003. IEEE Computer Society.

[90] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International*

Conference on World Wide Web, WWW 2003, pages 640–651, Budapest, Hungary, May 2003. ACM Press.

[91] Brent ByungHoon Kang. *S2D2: A Framework for Scalable and Secure Optimistic Replication*. PhD thesis, EECS Department, University of California, Berkeley, Oct 2004.

[92] ByungHoon Kang, Robert Wilensky, and John Kubiatowicz. The hash history approach for reconciling mutual inconsistency. In *ICDCS*, pages 670–677, 2003.

[93] Ray Kaplan. *Chapter 3 - A Matter of Trust in Information Security Management Handbook, Edition by Tipton, Harold F. and Krause, Micki*. Fifth Edition. CRC Press LLC, 2004.

[94] Krukow Karl, Nielsen Mogens, and Sassone Vladimiro. A Logical Framework for History-based Access Control and Reputation Systems. *Journal of Computer Security*, 16(1):63–101, January 2008.

[95] Anne-Marie Kermarrec, Antony I. T. Rowstron, Marc Shapiro, and Peter Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC'01*, pages 210–218, 2001.

[96] Michael Koch. Design issues and model for a distributed multi-user editor. *Computer Supported Cooperative Work*, 3:359–378, 1996.

[97] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *In SIGARCH*, pages 190–201, 2000.

[98] Arun Kumar, Neeran Karnik, and Girish Chafle. Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36:53–66, July 2002.

[99] Leslie Lamport. Times, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[100] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010.

[101] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 108–119. VLDB Endowment, 2004.

[102] Benedikt Ley, Volkmar Pipek, Christian Reuter, and Torben Wiedenhoefer. Supporting improvisation work in inter-organizational crisis management. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 1529–1538, New York, NY, USA, 2012. ACM.

[103] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, PADL '03, pages 58–73, London, UK, UK, 2003. Springer-Verlag.

[104] Mui Lik, Mohtashemi Mojdeh, and Halberstadt Ari. A Computational Model of Trust and Reputation. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, HICSS 2002*, pages 2431–2439, Waikoloa, Big Island, Hawaii, January 2002. IEEE Computer Society.

[105] Ernst Lippe and Norbert Van Oosterom. Operation-based merging. *SIGSOFT Softw. Eng. Notes*, 17:78–87, November 1992.

[106] Xin Liu, Anwitaman Datta, Krzysztof Rzadca, and Ee-Peng Lim. Stereotrust: a group based personalized trust model. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 7–16, New York, NY, USA, 2009. ACM.

[107] Jon Loeliger. Collaborating with Git. *Linux Magazine*, June 2006.

[108] Logilab.org. hgview. `http://www.logilab.org/project/hgview`, 2012.

[109] Xin Luo, Yixian Yang, and Zhengming Hu. Controllable delegation model based on usage and trustworthiness. In *Knowledge Acquisition and Modeling, 2008. KAM'08. International Symposium on*, pages 745 –749, dec 2008.

[110] Di Ma. Practical forward secure sequential aggregate signatures. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 341–352, New York, NY, USA, 2008. ACM.

[111] Di Ma and Gene Tsudik. Extended abstract: Forward-secure sequential aggregate authentication. In *IEEE Symposium on Security and Privacy*, pages 86–91, 2007.

[112] Di Ma and Gene Tsudik. A new approach to secure logging. *TOS*, 5(1), 2009.

[113] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst.*, 29(4):12:1–12:38, December 2011.

[114] Dahlia Malkhi and Douglas B. Terry. Concise version vectors in winfs. *Distributed Computing*, 20(3):209–219, 2007.

[115] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *Proceedings of the 11th USENIX Security Symposium*, pages 297–312, Berkeley, CA, USA, 2002. USENIX Association.

[116] Petros Maniatis and Mary Baker. Authenticated append-only skip lists. *Acta Mathematica*, 137:151–169, 2003.

[117] Srdjan Marinovic, Robert Craven, Jiefei Ma, and Naranker Dulay. Rumpole: a flexible break-glass access control model. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, SACMAT '11, pages 73–82, New York, NY, USA, 2011. ACM.

[118] Stephen Paul Marsh. *Formalising Trust as a Computational Concept.* PhD thesis, Department of Mathematics and Computer Science, University of Stirling, 1994.

[119] Michelle L. Mazurek, J. P. Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, Richard Shay, Kami Vaniea, Lujo Bauer, Lorrie Faith Cranor, Gregory R. Ganger, and Michael K. Reiter. Access control for home data sharing: Attitudes, needs and practices. In *Proceedings of the 28th*

international conference on Human factors in computing systems, CHI '10, pages 645–654, New York, NY, USA, 2010. ACM.

[120] Giovanni Mella, Elena Ferrari, Elisa Bertino, and Yunhua Koglin. Controlled and cooperative updates of xml documents in byzantine and failure-prone distributed systems. *ACM Trans. Inf. Syst. Secur.*, 9:421–460, November 2006.

[121] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

[122] Daniel Le Métayer. Formal methods as a link between software code and legal rules. In *Proceedings 9th International Conference on Software Engineering and Formal Methods, SEFM 2011*, pages 3–18, 2011.

[123] Daniel Le Métayer, Manuel Maarek, Valérie Viet Triem Tong, Eduardo Mazza, Marie-Laure Potet, Nicolas Craipeau, Stéphane Frénot, and Ronan Hardouin. Liability in software engineering: overview of the lise approach and illustration on a case study. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010*, pages 135–144, 2010.

[124] Zoran Milosevic. *Enterprise Aspects of Open Distributed Systems.* PhD thesis, oct 1995.

[125] Zoran Milosevic, Audun J"sang, Theo Dimitrakos, and Mary Anne Patton. Discretionary enforcement of electronic contracts. In *Proceedings of the Sixth International ENTERPRISE DISTRIBUTED OBJECT COMPUTING Conference (EDOC'02)*, EDOC '02, pages 39–, Washington, DC, USA, 2002. IEEE Computer Society.

[126] Mircrosoft. Sharepoint - Collaboration Software for the Enterprise. `http://sharepoint.microsoft.com`, 2010.

[127] C. Molina-Jimenez, S. Shrivastava, and M. Strano. A model for checking contractual compliance of business interactions. *Services Computing, IEEE Transactions on*, 5(2):276 –289, april-june 2012.

[128] Pascal Molli, Hala Skaf-Molli, Gérald Oster, and Sébastien Jourdain. Sams: Synchronous, asynchronous, multi-synchronous environments. In *Proceedings of the Seventh International Conference on CSCW in Design, CSCWD'02*, pages 80–84, 2002.

[129] Alessio Moretti. Why the logical hexagon? *Logica Universalis*, 6:69–107, 2012.

[130] Roger Muriel and Goubault-Larrecq Jean. Log Auditing through Model-Checking. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations, CSFW 2001*, pages 220–234, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[131] OpenJDK. OpenJDK. `http://openjdk.java.net`, 2012.

[132] Bryan O'Sullivan. *Mercurial: The Definitive Guide*. O'Reilly Media, 2009.

[133] Gordon J. Pace and Gerardo Schneider. Challenges in the specification of full contracts. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 292–306, Berlin, Heidelberg, 2009. Springer-Verlag.

[134] Keshnee Padayachee and Jan H. P. Eloff. Enhancing optimistic access controls with usage control. In *TrustBus'07*, pages 75–82, 2007.

[135] Keshnee Padayachee and Jan H. P. Eloff. Adapting usage control as a deterrent to address the inadequacies of access controls. *Computers & Security*, 28(7):536–544, 2009.

[136] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 437–448, New York, NY, USA, 2008. ACM.

[137] J. Park and R. Sandhu. The ucon-abc usage control model. In *ACM Transactions on Information and System Security*, pages 7(1):128–174, 2004.

[138] Jaehong Park and Ravi Sandhu. Towards usage control models: beyond traditional access control. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 57–64, New York, NY, USA, 2002. ACM.

[139] Olivier Perrin and Claude Godart. A model to support collaborative work in virtual enterprises. *Data Knowl. Eng.*, 50(1):63–86, July 2004.

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

[141] Zachary N. J. Peterson, Randal Burns, Giuseppe Ateniese, and Stephen Bono. Design and implementation of verifiable audit trails for a versioning file system. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 20–20, Berkeley, CA, USA, 2007. USENIX Association.

[142] G.J. Popek, R.G. Guy, Jr. Page, T.W., and J.S. Heidemann. Replication in ficus distributed file systems. In *Proceedings of the workshop on management of replicated data, IEEE*, 1990.

[143] Dean Povey. Optimistic security: a new access control paradigm. In *Proceedings of the 1999 Workshop on New Security Paradigms*, NSPW '99, pages 40–45, New York, NY, USA, 2000. ACM.

[144] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[145] Nuno M. Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *ICDCS*, pages 395–403, 2009.

[146] Alexander Pretschner, Manuel Hilty, and David Basin. Distributed Usage Control. *Commun. ACM*, 49:39–44, September 2006.

[147] Alexander Pretschner, Manuel Hilty, David A. Basin, Christian Schaefer, and Thomas Walter. Mechanisms for usage control. In *ASIACCS*, pages 240–244, 2008.

[148] Alexander Pretschner, Florian Schütz, Christian Schaefer, and Thomas Walter. Policy evolution in distributed usage control. *Electr. Notes Theor. Comput. Sci.*, 244:109–123, 2009.

[149] Cristian Prisacariu and Gerardo Schneider. A formal language for electronic contracts. In *Proceedings of the 9th IFIP WG 6.1 international conference on Formal methods for open object-based distributed systems*, FMOODS'07, pages 174–189, Berlin, Heidelberg, 2007. Springer-Verlag.

[150] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stéphane Weiss. Multi-synchronous collaborative semantic wikis. In *Proceedings of the 10th International Conference on Web Information Systems Engineering*, WISE '09, pages 115–129, Berlin, Heidelberg, 2009. Springer-Verlag.

[151] Agrawal Rakesh, Kiernan Jerry, Srikant Ramakrishnan, and Xu Yirong. Hippocratic Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002*, pages 143–154, Hong Kong, China, August 2002. VLDB Endowment.

[152] Prakash Ravi, Raynal Michel, and Singhal Mukesh. An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, 41(2):190–204, March 1997.

[153] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, December 2000.

[154] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297, New York, NY, USA, 1996. ACM.

[155] Reinhard Riedl. Rethinking trust and confidence in european e-government linking the public sector with post-modern society. In *I3E*, pages 89–108, 2004.

[156] Sherwood Rob, Lee Seungjoon, and Bhattacharjee Bobby. Cooperative Peer Groups in NICE. *Computer Networks*, 50(4):523–544, March 2006.

[157] Rodrigo Rodrigues and Peter Druschel. Peer-to-peer systems. *Commun. ACM*, 53(10):72–82, October 2010.

[158] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel Distrib. Comput.*, 71(3):354–368, March 2011.

[159] Martin Roscheisen and Terry Winograd. A Communication Agreement Framework for Access/Action Control. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 154–, Washington, DC, USA, 1996. IEEE Computer Society.

[160] Alf Ross. *On Law and Justice*. Berkeley:University of California Press, 1959.

[161] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA, 2001. ACM.

[162] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37:42–81, March 2005.

[163] Ravi Sandhu and Xinwen 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.

[164] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29:38–47, February 1996.

[165] Ravi S. Sandhu and Jaehong Park. Usage control: A vision for next generation access control. In *MMM-ACNS*, pages 17–31, 2003.

[166] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput.*, 39(4):447–459, April 1990.

[167] Greenberg Saul. Personalizable groupware: accommodating individual roles and group differences. In *Proceedings of the second conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'91, pages 17–31, Norwell, MA, USA, 1991. Kluwer Academic Publishers.

[168] Greenberg Saul, Roseman Mark, Webster Dave, and Bohnet Ralph. Human and technical factors of distributed group drawing tools. *Interacting with Computers*, 4(3):364–392, 1992.

[169] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22:299–319, 1990.

[170] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*, pages 4–4, Berkeley, CA, USA, 1998. USENIX Association.

[171] Brian Shand and Jean Bacon. Policies in accountable contracts. In *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 5-7 June 2002, Monterey, CA, USA*, pages 80–91. IEEE Computer Society, 2002.

[172] Brian Ninham Shand. Trust for resource control: Self-enforcing automatic rational contracts between computers. Technical Report UCAM-CL-TR-600, University of Cambridge, 2004.

[173] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th international conference on Stabilization, safety, and security of distributed systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.

[174] HongHai Shen and Prasun Dewan. Access control for collaborative environments. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, CSCW '92, pages 51–58, New York, NY, USA, 1992. ACM.

[175] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[176] Mike J. Spreitzer, Marvin M. Theimer, Karin Petersen, Alan J. Demers, and Douglas B. Terry. Dealing with server corruption in weakly consistent, replicated data systems. In *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '97, pages 234–240, New York, NY, USA, 1997. ACM.

[177] Federico Stagni, Alvaro Arenas, Benjamin Aziz, and Fabio Martinelli. On usage control in data grids. In *IFIPTM*, pages 99–116, 2009.

[178] Gunnar Stevens and Volker Wulf. A new dimension in access control: studying maintenance engineering across organizational boundaries. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, CSCW '02, pages 196–205, New York, NY, USA, 2002. ACM.

[179] Gunnar Stevens and Volker Wulf. Computer-supported access control. *ACM Transactions on Computer-Human Interaction*, 16(3):12:1–12:26, September 2009.

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

[181] Roshan 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.

[182] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.

[183] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, January 2009.

[184] Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi. *Peer-to-Peer Computing*. Springer, Singapore, 2009.

[185] W3C. World Wide Web Consortium (W3C) - A P3P Preference Exchange Language 1.0 (APPEL1.0). `http://www.w3.org/TR/P3P-preferences/`, 2000.

[186] W3C. World Wide Web Consortium (W3C) - P3P: The Platform for Privacy Preferences. `http://www.w3.org/P3P/`, 2000.

[187] Kevin Walsh and Emin Gün Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing (Awarded Best Paper). In *NSDI*, 2006.

[188] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-Undo: Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162–1174, August 2010.

[189] Bernard Williams. *Formal Structures and Social Reality*, chapter Chapter 1 in Gambetta,Diego (ed.) Trust: Making and Breaking Cooperative Relations, electronic edition, pages 3–13. Department of Sociology, University of Oxford, 2000.

[190] Marianne Winslett, Neil Ching, Vicki Jones, and Igor Slepchin. Assuring security and privacy for digital library transactions on the web: client and server security policies. In *Proceedings of the IEEE international forum on Research and technology advances in digital libraries*, IEEE ADL '97, pages 140–151, Washington, DC, USA, 1997. IEEE Computer Society.

[191] Ted Wobber, Thomas L. Rodeheffer, and Douglas 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.

[192] Rosalinde Klein Woolthuis, Bas Hillebrand, and Bart Nooteboom. Trust, contract and relationship development. *Organization Studies*, 26(6):813–840, June 2005.

[193] G. H. Von. Wright. *Norm and action : a logical enquiry.* Routledge and Kegan Paul - Humanities Press, London, New York, 1963.

[194] Georg Henrik Von Wright. Deontic logic. *Mind*, 60(237):1–15, January 1951. Oxford University Press.

[195] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 16:843–857, 2004.

[196] Attila Altay Yavuz and Peng Ning. Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *Proceedings of the 2009 Annual Computer Security*

Applications Conference, ACSAC '09, pages 219–228, Washington, DC, USA, 2009. IEEE Computer Society.

[197] Alexander Yip, Benjie Chen, and Robert Morris. Pastwatch: A distributed version control system. In *IN NSDI*, 2006.

[198] Xinwen Zhang. *Formal model and analysis of usage control.* PhD thesis, Fairfax, VA, USA, 2006. AAI3221391.

[199] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. A usage-based authorization framework for collaborative computing systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, SACMAT '06, pages 180–189, New York, NY, USA, 2006. ACM.

[200] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi S. Sandhu. Toward a usage-based security framework for collaborative computing systems. *ACM Trans. Inf. Syst. Secur.*, 11(1), 2008.

Résumé

De nos jours, les technologies de l'information offrent aux utilisateurs la possibilité de travailler avec n'importe qui, à n'importe quel moment, de n'importe où et avec plusieurs dispositifs hétérogènes. Cette évolution favorise un nouveau modèle distribué de collaboration de confiance où les utilisateurs peuvent travailler sur des documents partagés avec qui ils ont confiance. La collaboration multi-synchrone est largement utilisée pour soutenir le travail collaboratif en maintenant des flux simultanés de l'activité des utilisateurs qui divergent et convergent continuellement. Cependant, ce modèle n'offre pas de support concernant l'expression et la vérification de restriction d'usage des données. Cette thèse présente C-PPC, un modèle de collaboration basé sur les contrats et sur la confiance. Dans ce modèle, des contrats sont utilisés comme des règles d'utilisation des données et les utilisateurs collaborent en fonction des niveaux de confiance qu'ils attribuent aux autres en accord avec le respect des contrats. Nous formalisons les contrats en utilisant des concepts déontiques: permission, obligation et prohibition. Les contrats sont inclus dans les historiques d'opérations des données partagées. Le modèle C-PPC fournit un mécanisme pour la fusion des modifications des données et des contrats. N'importe quel utilisateur peut auditer les historiques à n'importe quel moment et les résultats de l'audit sont utilisés pour actualiser les niveaux de confiance en se basant sur une métrique de confiance. Nous proposons une solution reposant sur des authentificateurs basés sur les chaînes de hachage pour garantir l'intégrité des historiques et la responsabilité des utilisateurs. Nous fournissons des algorithmes pour construire les authentificateurs et vérifier les historiques puis nous prouvons leur correction. Des résultats expérimentaux montrent la faisabilité du modèle C-PPC.

Mots-clés: traitement réparti, informatique distribuée, travail collaboratif, logique déontique, contrat, confiance, collaboration multi-synchrone, contrôle d'usage, modèle de confiance, audit, authentificateur.

Abstract

Nowadays, information technologies provide users ability to work with anyone, at any time, from everywhere and with several heterogeneous devices. This evolution fosters a new distributed trustworthy collaboration model where users can work on shared documents with whom they trust. Multi-synchronous collaboration is widely used for supporting collaborative work by maintaining simultaneous streams of user activities which continually diverge and converge. However, this model lacks support on how usage restrictions on data can be expressed and checked within the model. This thesis proposes "C-PPC", a multi-synchronous contract-based and trust-aware collaboration model. In this model, contracts are used as usage rules and users collaborate according to trust levels they have on others computed according to contract compliance. We formalize contracts by using deontic concepts: permission, obligation and prohibition. Contracts are enclosed in logs of operations over shared data. The C-PPC model provides a mechanism for merging data changes and contracts. Any user can audit logs at any time and auditing results are used to update user trust levels based on a trust metric. We propose a solution relying on hash-chain based authenticators that ensures integrity of logs and user accountability. We provide algorithms for constructing authenticators and verifying logs and prove their correctness. A set of experimental results shows the feasibility of the C-PPC model.

Keywords: multi-synchronous collaboration, usage control, trust model, contract, log auditing, authenticator.