Large-scale trustworthy distributed collaboration

Claudia-Lavinia Ignat

▶ **To cite this version:**

Claudia-Lavinia Ignat. Large-scale trustworthy distributed collaboration. Distributed, Parallel, and Cluster Computing [cs.DC]. Université de Lorraine, 2021. tel-03229173

HAL Id: tel-03229173
https://inria.hal.science/tel-03229173

Submitted on 18 May 2021

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

École doctorale IAEM Lorraine

Large-scale trustworthy distributed collaboration

THÈSE

présentée et soutenue publiquement le 23 avril 2021

pour l'obtention d'une

Habilitation de l'Université de Lorraine

(mention informatique)

par

Claudia-Lavinia Ignat

Composition du jury

Rapporteurs :	M. Prasun DEWAN	Professeur, University of North Carolina at Chapel Hill
	Mme Valérie ISSARNY	Directrice de Recherche, Inria
	M. François TAIANI	Professeur des Universités, Université de Rennes I
Examinateurs :	Mme Sihem AMER-YAHIA	Directrice de Recherche CNRS
	M. François CHAROY	Professeur des Universités, Université de Lorraine
	Mme Isabelle CHRISMENT	Professeur des Universités, Université de Lorraine
	M. Fabien GANDON	Directeur de Recherche Inria
	M. Pascal MOLLI	Professeur des Universités, Université de Nantes

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

Mis en page avec la classe thesul.

Acknowledgments

To begin with, I would like to thank jury members for having accepted my invitation in this difficult period of pandemics. Many thanks to Prasun Dewan, Valérie Issarny and François Taiani for their encouraging reports on my work and their thorough comments and questions during my habilitation defense. Also I would like to thank Sihem Amer-Yahia, François Charoy, Isabelle Chrisment, Fabien Gandon, Pascal Molli for their very interesting and constructive questions during the defense. Special thanks to Isabelle Chrisment for having accepted to be the jury president and for all the administrative work. I would like to express my gratitude to François Charoy as a scientific guarantor for my habilitation but also for being a great team leader and for his support, confidence and freedom he gave me with respect to my research work.

The work I report in this habilitation manuscript is the result of collaboration with several people. I would like to thank all my co-authors, specially Valerie Shalin (Wright State University) and Weihai Yu (UiT The Arctic University of Norway) for the fruitful ideas and discussions we had together. I also thank my PhD students Hien Thi Thu Truong, Quang Vinh Dang, Hoang Long Nguyen and Hoai Le Nguyen and all the other students that I supervised for having worked with me on the different parts of my research project.

I would like to thank all permanent members of team Coast (Khalid Benali, François Charoy, Claude Godart, Gérald Oster, Olivier Perrin and the others) and all non permanent members (Luc André, Victorien Elvinger, Quentin Laporte-Chabasse, Stéphane Martin, Matthieu Nicolas and all the others that belonged to Ecoo/Score/Coast) for all scientific and non scientific discussions that made a very pleasant working environment. Many thanks to Claude Godart and Pascal Molli that welcomed me in their team at my arrival in France.

I would like to thank Moira Norrie, my PhD supervisor, that played an important role in my professional life, for having inspired me with the beauties of a research career.

Many special thanks to my husband and colleague Gérald Oster for his valuable support and encouragements during all these years and for the scientific discussions that sometimes expand even in the free time. I thank him as well as to our girls Julie, Natalie and Alexandra for all the love and happiness they bring to my life.

Last but not least I would like to thank to my parents and my brother who gave me all their support and love even from far away. I specially dedicate this thesis to my father for his constant interest and concern on the evolution of my career and for his encouragements over the last years to write this manuscript.

to my father

Contents

Introduction

1 Motivation

Douglas Engelbart's pioneering work on augmenting human intellect in 1968 demonstrated for the first time multi-user systems with groupware capabilities. Remote users could share the same screen and one user could see what the other was writing. However, the first groupware systems defined as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." [253] appeared only early 1990s.

The CSCW Matrix [265] represented in Figure 1 categorizes groupware software according to the degree of physical proximity of the group members (members can be collocated or can be situated remotely) and the degree of synchronicity of writing activities if users write at the same time or at different times. Synchronous authoring tools, also referred to as real-time collaborative authoring systems, imply that users write at the same time and changes made by one user are immediately transmitted to other group members. Asynchronous authoring tools support groups of individuals who contribute at different times by taking turns of revising and editing the shared artifact. The meeting room is an example of face-to-face interaction, i.e. interaction that takes place at the same place and at the same time, while a bulletin board is an example of asynchronous interaction that takes place at the same place but at different times. An example of groupware belonging to the synchronous distributed interaction is a group editor or a video conference system that allows real-time collaboration. An email system belongs to asynchronous distributed interaction. IBM Lotus Notes, known today as HCL Notes, was one of the first commercial groupware allowing remote group collaboration.

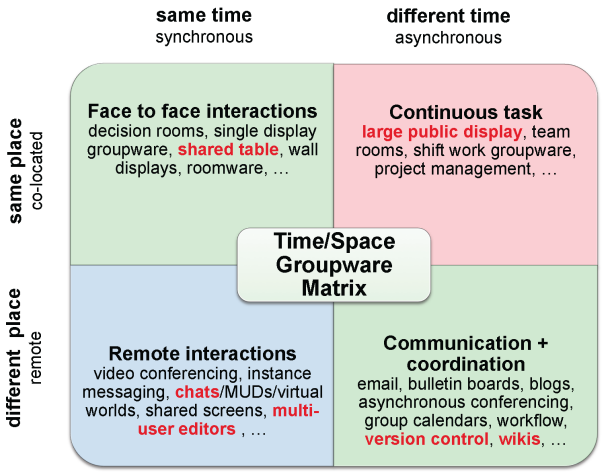


Figure 1: Groupware Time Space Matrix

Most of groupware systems developed at that time tried to avoid inconsistency that appears when users execute simultaneously conflicting operations on shared data. The simplest solution to avoid inconsistency is to prevent simultaneous action over shared data by defining a single, global stream of activity over the shared data space. In order to ensure this single, global stream of activity, groupware systems used turn-taking protocols or locking mechanisms. Turn-taking [251, 257, 263, 252, 255, 264, 262] allows only one active participant at a time, the one who "has the floor", the other users being blocked from editing. Locking [266, 203, 250, 242, 247] guarantees that users access objects in the shared workspace one at a time and concurrent editing is allowed only if users lock and edit different objects. These mechanisms restrict the collaboration on the shared documents and also yield unacceptable delays for obtaining the locks.

Later on, the multi-synchronous mode of collaboration [237] was proposed to abandon the attempt to construct a single stream of activity over the shared data space. Instead, it allows to maintain multiple, simultaneous streams corresponding to multi-user activities, and then looks to manage divergence between these streams. In this collaboration mode users can work independently with different streams of activity on the shared data. These streams can diverge and hence users have different views of the shared data. Divergence can arise due to delays in the propagation of operations on the shared data or execution of conflicting operations. These divergent streams synchronise at a later time and hence re-establish a common view of the shared data.

The first group editor that offered an unrestricted collaboration where participants were able to edit a document anywhere in the document and at any time by relying on a multi-synchronous mode of collaboration was GROVE [259]. The mechanism achieving the reconciliation of the divergent streams of collaboration is called operational transformation.

Starting with about 2005 we are facing an increasing interest in collaborative editors (i.e. ACE) which combined with the Web 2.0 movement caused an explosion of interest in web-based document editing tools. It received a lot of attention from both industry and academia and gained in popularity due to the wide availability of free services such as Google Drive introduced in 2012 as a cloud storage with office suite Docs, Sheets and Slides. As by July 2018, Google Drive had more than 1 billion users.

Early 1990s when first groupware systems appeared, a general skepticism existed regarding the use of these systems: "Isn't it chaotic to all edit in the same document, even the same paragraph, at the same time?" "Why would a group ever want to edit in the same line of text at the same time?" [253]. In the later years groupware systems started to be used in scenarios involving a small set of users such as for the writing of a research article. Nowadays we notice a change in the scale from several users to a community of users. For instance, we can cite open-source projects where a community of developers contribute to code of a common software project or Wikipedia where during an important event many users modify the wiki page related to that event or during a conference where a community of participants takes notes during a presentation.

Most commonly used collaborative systems are those provided by large service providers such as Facebook and Google. While these collaborative services offer very interesting functionalities, they feature certain limitations. Most of the platforms hosting these collaboration services rely on a central authority and place personal information in the hands of a single large corporation which is a perceived privacy threat. Users must provide and store their data to vendors of these services and have to trust that they will preserve privacy of their data, but they have little control over the usage of their data after sharing it with other users.

These systems do not scale well in terms of the number of users. Scenarios involving a large

number of concurrent editing users are emerging, such as students of a class or participants in a conference that collaboratively take notes. For instance, GoogleDocs limits the number of persons that can simultaneously modify a document to 50. If this number is exceeded, next users that join the document will have the rights limited to reading the document. An example of large scale collaborative editing that is not possible with the current technology is the MOOC (Massive Open Online Course) on online education opened in 2013 where the 40,000 participants were asked to access to parts of the Google Docs documents created for the course. Due to the high number of concurrent edits to the same documents, the system crashed and finally the lecture was cancelled.

Furthermore, user communities cannot deploy these kind of service applications since they generally rely on costly infrastructures rather than allowing sharing infrastructure and administration costs.

The vision underlying my research work is to move away from centralized authority-based collaboration towards a large scale peer-to-peer (P2P) collaboration where control over data is given to users who can decide with whom to share their data. The risk of privacy breaches is decreased in this P2P collaboration as only part of the protected data is exposed at any time. The main strengths of P2P systems [112] are their independence of a centralized control and of a dedicated infrastructure. Participating nodes are owned and operated by independent individuals and therefore administration costs of the system are shared. Distinctive characteristics of P2P systems are high scalability, resilience to faults and attacks and a low deployment barrier for new services.

However, these peer-to-peer collaborative systems feature some challenges. In order to provide efficient data availability, data is typically replicated and users are allowed to concurrently modify replicated data. One of the challenges is to develop optimistic replication algorithms that maintain consistency of the shared data in the face of concurrent modifications. These algorithms have to be reliable, i.e. after the reception of all modifications done by users the state of the data copies has to converge. They also have to be scalable, i.e. have a complexity that does not depend on the number of users. These algorithms have also to be explainable, i.e. their decision must be comprehensible by users and therefore user intentions must be preserved. Another challenge is how to maintain group awareness, i.e. an “understanding of the activities of other users which provides a context for your own activity" [245]. Group awareness helps users track the changes that other collaborators have made to shared data and avoid potential conflicts or redundant work.

In this large scale peer-to-peer collaboration a main question is how to choose your collaborators that you can trust in order to share them your data. A rational decision would be based on the evaluation of previous collaborative behaviour of your collaborators. In a large scale collaboration we cannot remember the interactions of all users with whom we collaborated and sometimes we have a false impression on the contribution of our collaborators. Some of them are modest and do not know to put themselves forward, while some others know to exaggerate their contribution. A trust value that is computed automatically based on previous collaboration behavior would be of great help to users. A main challenge in this task is how to compute this trust value according to past collaboration in order to be able to predict future user behavior.

2 Research Work

My research work is structured around two axes of research in the domain of peer-to-peer collaborative systems that tackle the challenges previously described: *collaborative data management*

and *trustworthy collaboration*.

2.1 Collaborative data management

This axis refers to the design and evaluation of various approaches related to management of distributed shared data including optimistic data replication and group awareness.

Two main optimistic replication approaches exist for maintaining consistency: operational transformation approaches (OT) and commutative replicated data types (CRDT).

Operational transformation [259] relies on two components: a set of transformation functions specific to a document type that specify how concurrent modifications are merged and a generic integration algorithm that detects concurrent changes and applies the appropriate transformation functions. This mechanism is currently integrated in GoogleDocs. Operational transformation has several limitations. The approach is not scalable with the number of users and requires either a central server for message dissemination or expensive distributed consensus protocols. Moreover, the algorithmic complexity is quadratic on the number of concurrent operations. Furthermore, the approach requires to detect concurrency between operations. It has been shown that the optimal data structure for concurrency detection is proportional to the number of collaborators. This structure is very costly in terms of storage and communication as it has to be attached to every disseminated modification. Finally, the design of correct transformation functions is complex and costly.

In order to address limitations of operational transformation, CRDTs (Commutative Replicated Data Types) [154, 100] were proposed. The main idea is to define data structures where parallel modifications are conflict free, i.e. these modifications are commutative. This approach has several advantages. It does not require a-posteriori synchronisation as concurrent modifications are executed without conflict and produce the same state on all copies independently of the execution order. The architecture can be therefore completely distributed. The algorithmic complexity is much lower than that of operational transformation mechanism.

My contributions relate to both families of algorithms. I proposed operational transformation mechanisms for complex data such as hierarchical text documents [16], XML [44, 41] (in collaboration with Gérald Oster) and wikis [13] (in collaboration with Gérald Oster and Luc André).

CRDTs have a limitation: the atomic elements of the data structure need to be uniquely identified. There is therefore a significant increase in the meta-data necessary for the update of the data structure. Together with Luc André, Stéphane Martin and Gérald Oster, I proposed a CRDT approach for strings that considerably limits the meta-data [32]. This CRDT approach is the underlying consistency maintenance mechanism in MUTE (https://coedit.re/), the peer-to-peer web-based collaborative editor that we developed in our team [71]. In collaboration with Weihai Yu (Professor, University of Tromso) and Luc André, I also proposed an undo mechanism for string-based CRDTs [29]. In collaboration with Weihai Yu and Victorien Elvinger, I designed a generic undo support for CRDTs [20]. In collaboration with Weihai Yu, I proposed a CRDT for relational databases [19].

I evaluated existing operational transformation and CRDT algorithms [46] (in collaboration with members of the ARC RECALL project), [36, 33] (in collaboration with members of SCORE/Coast team: Mehdi Ahmed-Nacer, Gérald Oster, Hyun-Gul Roh, Pascal Urso) by means of their theoretical complexity and simulations.

Understanding real-time constraints from users point of view provides a critical validation of research leaded by the collaborative editing community that continues to develop merging algorithms under the uniform assumption of high responsiveness requirements for real-time col-

laboration. However, no study related distributed systems performances with their perception by users. I studied real-time constraints from user point of view in terms of delays in real-time collaborative editing. Delays exist between a user executed modification and the visibility of this modification to other users. These delays can be due to the network and underlying architecture, but also due to the underlying consistency maintenance algorithms. In the scope of Quang Vinh Dang's thesis, by means of simulations, I measured delays in popular real-time collaborative editing systems such as GoogleDocs and Etherpad in terms of the number of users that edit a shared document and their typing frequency [58]. In the context of USCOAST Inria associated team that I led and in collaboration with Valerie Shalin, Olivia Fox and Meagan Newman from the Department of Psychology, Wright State University, François Charoy and Gérald Oster, by means of experimental study with users I studied the effect of delay on group performance [31, 30]. User groups had to perform three typical collaborative editing tasks under a constant but undeclared delay in the propagation of changes between group members. Results demonstrated the negative effect on the quality of task performance after a threshold delay. Delays measured in GoogleDocs are largely greater than the artificial delays experienced in our lab experiment. Results of our study support the assertion that delay associated with conventional consistency maintenance algorithms will impede group performance and motivate the need for optimized algorithms.

In the context of the PhD thesis of Hoai-Le Nguyen, I also analysed real collaboration traces such as of software projects that used Git where we studied management and resolution of conflicts [57, 12]. In [18] we provided a characterisation of conflicts in real-time collaborative editing in terms of their occurrence in time and position inside the document by analysing ShareLatex collaboration traces.

I proposed awareness mechanisms for localisation of concurrent modifications in the form of annotation and visualisation of changes in source code documents [42, 39] (in collaboration with Gérald Oster), textual documents [51, 43] and web sites [40] (in collaboration with Moira Norrie, Gérald Oster and Stavroula Papadopoulou). In the context of these awareness mechanisms I studied the balance between awareness and privacy and allowed users to filter information about their changes according to their preferences [43, 62] (in collaboration with Moira Norrie, Stavroula Papadopoulou, Gérald Oster, Pascal Molli and Hala Skaf-Molli). I also studied the balance between disturbance and awareness of concurrent updates and allowed users to define focus regions without being disturbed by work of other users [24] (in collaboration with Gérald Oster and Weihai Yu).

Existing tools for supporting parallel work feature some disadvantages that prevent them to be widely used. Very often they require a complex installation and creation of accounts for all group members. Users need to learn and deal with complex commands for efficiently using these collaborative tools. Some tools require users to abandon their favorite editors and impose them to use a certain co-authorship application. In [38], together with Gérald Oster and Pascal Molli, we proposed the DooSo6 collaboration tool that offers support for parallel work, requires no installation, no creation of accounts and that is easy to use, users being able to continue working with their favorite editors. User authentication is achieved by means of a capability-based mechanism. A capability is defined as a couple (object reference, access right). If a user possesses this capability he/she has the specified right to the referenced object. The system manages capabilities for publishing and updating shared projects. The prototype relies on the data synchronizer So6 (`http://www.libresource.org/`).

2.2 Trustworthy collaboration

In the peer-to-peer collaboration users might want to collaborate only with those users they trust. I am interested in assessing users trust according to their behaviour during collaboration in a large scale environment.

In the context of the PhD thesis of Hien Truong co-supervised with Pascal Molli, I proposed a contract-based collaboration model [15, 37, 61] where trust in users is established and adjusted based on their compliance to the contracts specified by the data owners when they share the data. Observation of adherence to or violation of contracts that is used to adjust trust levels is done by means of an auditing mechanism [35] and relies on logs that register all user activities on the shared data. Our contract-based collaboration model allows the specification of contracts, merging of data and contracts and resolution of conflicting contracts. A trust metric for computing user trust levels was proposed based on auditing user compliance to the given contracts. The model detects if the collaboration logs were tampered by relying on hash-chain based authenticators [34].

In the context of the PhD thesis of Quang-Vinh Dang and in collaboration with Valerie Shalin, I proposed an experimental design for testing the proposed trust-based collaboration model [11]. We employed trust game, a money exchange game that has been widely used in behavioural economics for studying trust and collaboration between humans. In this game, exchange of money is entirely attributable to the existence of trust between users. In the context of trust game we proposed a trust metric that reflects user behaviours during the collaboration [27].

In order to compute the trust score of users according to their contributions during a collaborative editing task, we need to evaluate the quality of the content of a document that has been written collaboratively. In the context of the PhD thesis of Quang-Vinh Dang, I investigated how to automatically assess the quality of Wikipedia articles in order to provide guidance for both authors and readers of Wikipedia. In this context we proposed three automatic assessment methods of the quality of Wikipedia articles. In the first approach we introduced readability features for a better prediction of quality [28]. The other two approaches are based on a deep-learning mechanism that automatically learns features from document contents rather than manually defining them [26, 14, 23].

3 My journey

I earned my PhD degree in 2006 at ETH Zurich in the Global Information Systems (GlobIS) research group under the supervision of Moira Norrie. In my PhD thesis I developed algorithms for maintaining consistency in the collaboration over hierarchical documents. I analysed the collaboration over textual, XML and graphical documents. I extended the operational transformation approach initially applied for linear structures to work for hierarchical models such as textual and XML documents. My approach for maintaining consistency allows any existing operational transformation algorithm for linear structures to be applied recursively over the hierarchical structure of the document [56, 68, 47, 48, 50]. For maintaining consistency over graphical documents I used a novel mechanism based on a distributed serialisation of operations [55, 49, 67]. In my approach for graphical documents users can define the types of conflicts between the operations and the policy for the resolution of conflicts. I analysed separately the real-time and asynchronous modes of collaborative interaction over a central repository [17, 53, 54, 65, 52, 66]. In real-time communication changes are transmitted immediately to other users, while in the asynchronous mode of collaboration users work in isolation and synchronise their

changes at a later time. I developed an editor for each mode of collaboration over textual, XML and graphical documents.

I did a postdoc between October 2006 and September 2007 in the Inria research team ECOO (currently team Coast) at Inria Nancy-Grand Est and LORIA. My postdoc was in the context of the project ARC RECALL 2006-2007 (Réplication optimiste pour l'Édition CoLLaborative massive sur réseau P2P) under the supervision of Pascal Molli, currently Professor at Nantes University and head of GDD team. The goal of the RECALL project was the development of optimistic replication algorithms for supporting massive collaborative editing involving thousands of users. In the context of this project I was the main leader for the comparison and evaluation of several optimistic approaches for peer-to-peer collaborative editing [46, 77]. I also worked together with Gérald Oster (Maître de conférence at l'Université de Lorraine, team ECOO) on the adaptation for peer-to-peer environments of the optimistic replication algorithm for hierarchical documents developed during my thesis by combining it with the TTF (Tombstone Transformational Functions) approach [64] developed by team ECOO.

I was recruited in 2007 as Researcher at Inria in the team ECOO. Since my arrival in the team ECOO I enlarged the spectrum of my work on collaborative data management with a focus on large scale peer-to-peer environments. My PhD thesis focused on optimistic replication in the context of groupware systems but where the large scale constraint was not taken into account. Operational transformation was the most appropriate optimistic replication mechanism existing at that time and my thesis focused on extending this mechanism for complex data. In the period just before my arrival ECOO team proposed a new optimistic replication mechanism more suitable for peer-to-peer environments [154], which is considered the first CRDT (Conflict-free replicated data type), concept that was later formally defined [100]. Together with Mehdi Ahmed-Nacer, Luc André, Stéphane Martin, Gérald Oster, Hyun-Gul Roh and Pascal Urso, I continued to enrich the operational transformation theory for collaborative editing by developing algorithms for complex data types in the Web 2.0 such as wikis [59, 13], but also to develop new CRDTs in order to achieve better performances for large scale peer-to-peer collaborative editing [32]. We were also the first ones that developed a comparison among the main operational transformation and CRDT algorithms in terms of their theoretical complexity and by means of simulations [36]. These works were done in the context of the Wiki3.0 project for which I was the leader for Inria partner and respectively of the ANR STREAMS and ConcoRDanT projects.

Later on, in collaboration with Weihai Yu (UiT The Arctic University of Norway) who spent his sabbatical years 2013/2014 and 2018/2019 in our team (SCORE/Coast) and Luc André and Victorien Elvinger I focused on undo issues in collaborative editing based on CRDT approaches [29, 20]. If up to now I worked only on CRDTs for collaborative editing, very recently, in collaboration with Weihai Yu, I started to focus also on other types of CRDTs such as for relational databases [19].

At the end of my PhD thesis at ETH Zurich I started to investigate together with Moira Norrie and Stavroula Papadopoulou (ETH Zurich) issues on awareness in collaborative editing such as providing users an overview of the quantity of changes done on the shared document [51, 63]. After my arrival in Nancy I continued to work on collaborative awareness and extended the collaboration with the members of ECOO team (Gérald Oster, Pascal Molli and Hala Skaf-Molli) on localisation of concurrent concurrent changes and on the trade-off between awareness and user privacy [62, 45, 43, 42, 39, 76]. Together with Gérald Oster and Stavroula Papadopoulou, I also studied awareness for the collaborative editing of web pages where we analysed changes done on a web page and all its linked pages [40]. Using edit profiles we provided the visualization of the quantity of changes of a certain type performed at different levels of the current web page (section, paragraph, sentence) and its linked pages. In collaboration with Gérald Oster and

Weihai Yu, I also studied the balance between disturbance and awareness of concurrent updates [24].

I opened up a collaboration with psychologists from the Department of Psychology, Wright State University namely with Prof. Valerie Shalin on studying distributed systems performances with their perception by users, namely real-time constraints from user point of view in terms of delays in real-time collaborative editing [58, 31, 30, 73]. This collaboration was done in the context of the Inria USCOAST associated team for which I was the main investigator where we organised several exchange visits between our team and Wright State University. Prof. Valerie Shalin spent her sabbatical year in 2013 in our team. Results of our studies show that delay associated with consistency maintenance algorithms used by popular collaborative editing systems such as GoogleDocs and Etherpad impede group performance in a large scale context in terms of the number of users and their typing frequency and motivate the need for optimized algorithms.

I co-supervised four defended PhD theses.

- Hien Thi Thu Truong, *A Contract-based and Trust-aware Collaboration Mode, (October 2009 - December 2012), defended on December 2012*, co-supervised together with Pascal Molli (member of SCORE team till 2010 and then head of the GDD team, University of Nantes). Hien Thi Thu Truong is currently Senior Researcher at NEC Laboratories Europe. In the context of the PhD thesis of Hien Thi Thu Truong I opened up a new direction of research in our team on security and trust in large scale collaborative systems. We investigated replacing the traditional "hard" security models that are not very suitable for large scale distributed collaborative systems with "soft" security models. Rather than adopting an a priori strict enforcement of security rules, access is given first to data without control but with restrictions that are verified a posteriori. We proposed a contract-based collaboration model [15, 37, 61, 35, 75] where trust in users is established and adjusted based on their compliance to the contracts specified by the data owners when they share the data. In this PhD thesis we applied in a novel way CRDTs for maintaining consistency over both data documents and contracts over sharing those documents. In this collaboration model security of logs in terms of their tampering is ensured by the use of hash-chain based authenticators [60, 34].

- Quang Vinh Dang, *Trust assessment in large scale collaborative systems, (October 2014-December 2017), defended on January 2018*, co-supervised with François Charoy. Quang Vinh Dang is currently assistant professor at the Industrial University of Ho Chi Minh. In the context of the PhD thesis of Quang-Vinh Dang and in collaboration with Valerie Shalin, we proposed an experimental design based on trust game for testing a trust-based collaboration model [11]. We designed a trust metric that reflects user behaviour during the trust game [27]. The experimental design described in [11] confirmed that the availability of this trust metric improves user cooperation and that it predicts participants future behavior. In order to compute the trust score of users according to their contributions during a collaborative editing task, we need to evaluate the quality of the content of a document that has been written collaboratively. Together with Quang-Vinh Dang, we investigated how to automatically assess the quality of Wikipedia articles in order to provide guidance for both authors and readers of Wikipedia. In this context we proposed three automatic assessment methods of the quality of Wikipedia articles. In the first approach we introduced readability features for a better prediction of quality [28]. The other two approaches are based on a deep-learning mechanism that automatically learns

features from document contents rather than manually defining them [26, 14, 23]. We also investigated how to predict trust relations between users that did not interact in the past. Given a network in which the links represent the trust/distrust relations between users, we proposed an algorithm to predict future user trust/distrust relations [22]. We also proposed a social recommendation approach based on the topology of an anonymized network that combines user trust relations with user rating scores for items [25].

- Hoang Long Nguyen, *A Trust Based Authorization Model and Framework for the Cloud, (November 2015 - March 2019), defended on December 2019*, co-supervised with Olivier Perrin. Hoang Long Nguyen is currently a co-founder of Akachain (akachain.io) in Hanoi, Vietnam, a startup on the development of transparent applications based on consortium blockchain. In the scope of Hoang Long Nguyen's PhD thesis, we proposed Trusternity, a key transparency approach using the Ethereum blockchain for End to End Encryption (E2EE), namely for key verification [70]. We also proposed a method to detect Eclipse attacks on the blockchain [21].

- Hoai Le Nguyen, *Study of Conflicts in Collaborative Editing, (September 2015- December 2020), defended on January 2021* co-supervised with François Charoy. In the scope of Hoai Le Nguyen's PhD thesis, we studied collaborative user behavior in version control systems such as Git where we analysed the collaboration process of these projects at specific periods revealing how change integration and conflict rates vary during the project development life-cycle. Our study suggests that developers should use more intensively awareness mechanisms close to release dates when changes integration rate is higher. We also studied the mechanism adopted by Git to consider concurrent changes made on two adjacent lines as conflicting. Based on the high rate of false positives of this mechanism, our study suggests that Git should reconsider signalling adjacent line conflicts inside the source code files [12]. We also studied user behavior in real-time collaborative editors such as ShareLatex/Overleaf and characterised conflicts in terms of their occurrence in time and position inside the shared document [18].

4 Implication in research projects

I provide here a list of projects in which I participated after my PhD thesis:

- **Principal Investigator of USCOAST (User Studies on Trustworthy Collaborative Systems) Inria Associate team (2013-2018)** `http://uscoast.loria.fr/` between COAST and Department of Psychology, Wright State University.

  This project focused on the human evaluation of methods and algorithms for trustworthy collaboration. This project helped me to acquire knowledge and some practice into the design of user studies, expertise that I was lacking before the start of USCoast associated team. I also acquired knowledge regarding the policies for the ethical treatment of participants and expertise into statistical analysis of the study results. This project was a successful experience in interdisciplinary research. In the context of this project we contributed to the first study towards understanding real-time constraints from users point of view showing that delay associated with popular collaborative editors impedes group performance and motivating the need for optimized consistency maintenance algorithms [30]. This project helped me set up an experimental design for testing the influence of trust score availability during collaboration and showing that it has the same effect as

identity for improving cooperation while having the advantage of scalability [11]. Coast funding: between 10,000 and 15,000 €/year

- **Local Inria leader of the Wiki 3.0 (2009-2012)** financed by the call for projects ń Web 2.0 ż launched by the Minister of Economy in France (appel Web innovant du volet numérique du plan de relance). The objective of this project was the extension of the XWiki system by the creation of a new wiki generation which offers real-time editing and the integration of social interaction tools such as chat and micro-blogging. Partners of this project were XWiki SAS, INRIA and Mandriva. I was responsible with the design and integration of real-time editing features into the XWiki system. The solution we proposed for real-time editing of wiki pages [13] was released as an extension of XWiki `https://labs.xwiki.com/xwiki/bin/view/Projects/Wiki30`. As a follow-up of this editor, XWiki developed CryptPad editor which is currently largely used by their clients. Total funding: 406,636 €. Coast funding: 143 375 €
- **Local Inria leader of Region Lorraine TV Paint (2016–2017)** in collaboration with TVPaint Development. The result of this project was the proposal of an architecture and prototype of a collaborative system dedicated to 2D animation movies, that allows to manipulate digital artifacts in a collaborative way. Total funding: 140,000 €. Coast funding: 50,000 €
- **Local Inria leader of Region Grand Est TV Paint (2017–2019)**. This is a follow-up project in collaboration with TVPaint Development. Based on the previously proposed architecture and prototype, this project focused on the design and implementation of a commercial product dedicated to animation movies that can manipulate a large amount of data in a safe and secure manner. Total funding: 295,628 €. Coast funding: 81,600 €
- **Local Inria leader of the Deeptech project financed by BPI and coordinated by Fair&Smart (2020-2023)**. The goal of this project is the development of a platform for the management of personal data according to General Data Protection Regulation (GDPR). Other partners of this project are CryptoExperts and team READ from LORIA. The computational personal trust model that we proposed for repeated trust game [27] and its validation methodology [11] will be adapted for the Fair&Smart personal data management platform for computing trust between the different users of this platform. Our decentralised mechanism for identity certification relying on a blockchain [70, 21] will be transferred to Fair&Smart for user identification for their personal data management platform. Total funding: 1,200,000 € Coast funding: 266,000 €
- **Member of OpenPaaS::NG (2015-2019)** `http://ng.open-paas.org/` project funded by BpiFrance involving French industrial leaders in open-source software development (Linagora, Nexedi, XWiki) and academic partners in collaborative work (COAST) and recommender systems (DaScim, LIX). The result of this project was the development of a french open-source new generation collaboration platform for enterprises that offers various secure collaboration modes (real-time, connected, disconnected, ad-hoc) over shared data, support for virtual meetings with automatic summary and recommendations. The main feature of the project that provided its originality was the peer-to-peer architecture underlying the platform. Indeed, existing collaboration solutions are offered by large service providers such as Google that place user data in the hands of a central authority which represents a privacy threat. This project allowed us to transfer our knowledge on data consistency maintenance and peer-to-peer architectures. In the context of this project we devel-

oped our web-based peer-to-peer collaborative editor MUTE (`(https://coedit.re/)`)[71]. Some of our ideas on consistency maintenance algorithms were transferred to the collaborative editor CryptPad developed by XWiki which is currently used by their clients. In this project I was responsible for the data security over the peer-to-peer architecture (the work package on secure and reliable peer-to-peer collaboration). Total funding: 10,710,208 €. Coast funding: 1,001,000 €

- **Member of ANR STREAMS (Solutions for Peer-to-peer Real-time Social Web) (2010-2014)**. STREAMS project designed peer-to-peer solutions that offer underlying services required by real-time social web applications and that eliminate the disadvantages of centralised architectures. These solutions are meant to replace a central authority-based collaboration with a distributed collaboration that offers support for decentralisation of services. In the context of this project, together with Luc André, Stéphane Martin and Gérald Oster I developed LogootSplit [32], a CRDT for strings that reduces metadata overhead and thus achieves better performances for large scale peer-to-peer collaborative editing. LogootSplit is the underlying consistency maintenance algorithm of MUTE collaborative editor. This project also allowed me to investigate a "soft" security models for peer-to-peer collaboration where rather than adopting an a priori strict enforcement of security rules, access is given first to data without control but with restrictions that are verified a posteriori [15, 37, 61, 35]. Total Cost : 1,730,017 €, Funding: 526,325 €, Coast funding : 157,466 €

- **Member of ANR ConcoRDanT (CRDTs for consistency without concurrency control in Cloud and Peer-to-Peer systems) (2010-2014)** The ConcoRDanT project studied the Commutative Replicated Data Type (CRDT), i.e. a data type where all concurrent operations commute, a simple approach for ensuring eventual consistency that scales indefinitely. In the context of both ANR ConcoRDant and ANR STREAMS we evaluated and compared different CRDT and OT algorithms [36]. Total Cost: 1,197,521 €, Funding: 316,740 €, Coast funding: 135,009 €

- **Member of RNTL XWiki Concerto (2006-2009)** I was involved in the XWiki project starting from 2008. This project developed XWiki Concerto (http://concerto.xwiki.org), a peer-to-peer extension of the XWiki system, an open-source enterprise wiki. This Wiki web application over a P2P network offers access to mobility, i.e. users can edit offline their wiki pages from a variety of devices such as smartphones, PDAs or desktops. XWiki Concerto uses an epidemic propagation algorithm to broadcast changes on the overlay network, combined with the WOOT algorithm [154] to merge concurrent changes.

- **Member of ARC RECALL 2006-2007 (Réplication optimiste pour l'Édition CoLLaborative massive sur réseau P2P)** The goal of the RECALL project was the development of optimistic replication algorithms for supporting massive collaborative editing involving thousands of users. Wikipedia or open-source projects are achievements of such type of collaboration. The proposed algorithms support the development of collaborative applications over a peer-to-peer network offering in this way a good scalability, support for fault tolerance and reduced deployment costs. I was the main leader for the comparison and evaluation of several optimistic approaches for peer-to-peer collaborative editing [46].

5 Organisation of the manuscript

The manuscript is structured into three parts:

- Part I describes the design and evaluation of optimistic replication and group awareness approaches that I proposed. In Chapter 1 I provide an overview of existing optimistic replication and group awareness approaches. In Chapter 2 I present an OT mechanism for complex data such as wikis and in Chapter 4 a CRDT solution that reduces metadata overhead. In Chapter 3 I present an evaluation of main OT and CRDT and an experimental user design for evaluating the effect of algorithm performance on users [30]. Chapter 5 presents my main contributions for conflict prevention and analysis.
- Part II describes my work around replacing the traditional "hard" security models for large scale distributed collaborative systems with "soft" security models based on trust and on assessing users trust according to their behaviour during collaboration. Chapter 1 presents a short literature review on security and trust management in multi-synchronous collaboration. In Chapter 2 I present a contract-based collaboration model where contracts are specified by the data owners when they share the data and user trust is assessed according to the observation of adherence to or violation of contracts. In Chapter 3 I present a hash chain based authenticators approach for ensuring integrity and authenticity of logs of operations in multi-synchronous collaboration. Chapter 4 presents a computational trust model for repeated trust game where user trust values are updated based on the satisfaction level for the exchanges during the game. Chapter 5 describes an experimental design for testing the influence of trust score on user behavior.
- Part III presents my project in the domain of secure and trustworthy collaborative data management focussing on novel synchronisation approaches for complex data requiring a composition replication mechanisms, security aspects of collaborative data management and evaluation of collaborators according to their past contributions.

Part I

Collaborative Data Management

Multi-synchronous collaborative systems are distributed systems that manage shared data. Replication of shared data is necessary in multi-synchronous collaboration to offer data availability and achieve high responsiveness in the case of real-time communication. Various replication protocols exist in order to ensure replicas consistency.

Pessimistic replication is a family of replication protocols which gives user the impression of existence of a single replica [266]. Operations of read can be done on any replica, while a writing operation has to be atomically applied on all replicas. Turn-taking protocols [251] used by shared view systems such as SHARE [257] and RTCAL [264] and locking mechanisms [203] used by SASSE [242], GroupDraw [247], RCS [254] are pessimistic approaches for consistency maintenance that allow only one participant at a time to edit a shared object, the others being blocked from editing. Pessimistic replication is not suitable for multi-synchronous collaboration as during an update local data cannot be edited as the initiator of the update has an exclusive access.

Pessimistic replication protocols ensure a strong consistency, i.e. after a user performs an update, any subsequent access performed by the other users will return the updated value. However, these protocols do not tolerate network partitions due to network failures. This can be explained by the CAP theorem [214, 216, 198] that states that in a shared-data system it is impossible to simultaneously ensure all three desirable properties, namely (C) data consistency meaning that all nodes see the same data at the same time, (A) availability of shared data for update and (P) tolerance to network partition, i.e. the system continues to operate despite arbitrary partitioning due to network failures.

In a large-scale distributed system, network partitions exist, so consistency and availability cannot be both achieved. Optimistic replication [169] is an approach suitable for multi-synchronous collaboration where shared data is always available but consistency is relaxed. Compared to pessimistic approaches, optimistic replication does not need an atomic update, as replicas are allowed to diverge temporarily, but they are expected to converge eventually. However, reconciliation of divergent copies is a complex process. In Chapter 1 I provide an overview of existing optimistic replication approaches. I highlight the advantages and disadvantages of the two most suitable approaches exist for maintaining consistency in multi-synchronous collaboration: operational transformation approaches (OT) and commutative replicated data types (CRDT).

I contributed to both families of algorithms. While most existing OT algorithms target simple data such as linear structures where a text document is seen as a sequence of characters, in Chapter 2 I present an operational transformation mechanism for complex data such as wikis [13]. The main disadvantage of CRDT approaches is meta-data overhead and in Chapter 4 I present a solution for reducing this overhead [32]. In Chapter 3 I present an evaluation of main OT and CRDT in terms of their complexities [36] and an experimental design with users in order evaluate the effect of algorithm performance on users [30].

Group awareness, defined as an "understanding of the activities of others which provides a context for your own activity" [245], has been identified by the CSCW community as one of the most important issues in collaborative document authoring, independently of the collaboration mode, be it synchronous, asynchronous or multi-synchronous. I focused on workspace awareness, particularly on providing awareness about the states of the shared documents in various contexts. I was interested in what information should be provided to users to prevent conflicting changes in user groups and to understand divergence when conflicts cannot be avoided. In Chapter 1 I present a short overview of awareness mechanisms for conflicts prevention and on conflicts management once they occurred. I present my main contributions for conflict prevention and analysis in Chapter 5. I present an awareness mechanism that localizes concurrent changes in

the context of collaborative editing over code source [39, 42] and textual documents [43] and a trade-off mechanism between awareness and user privacy [43] where users can filter details about their changes transmitted to other users according to their preferences. I also present a study of conflicts in asynchronous collaboration by using Git [12].

1

Literature Review

In this chapter I provide a literature review on optimistic replication approaches and awareness mechanisms in collaborative systems.

1.1 Optimistic replication

This section presents a short overview of optimistic replication approaches with a focus on operational transformation and commutative replicated data types which are the most suitable approaches for maintaining consistency in multi-synchronous collaboration. It also describes some existing solutions for maintaining consistency over complex data.

1.1.1 Last writer wins

One of the well-known optimistic replication approaches is the "last writer wins" strategy [273] used in Usenet. In the case that two replicas are concurrently modified, one of these modifications is lost which makes the resolution mechanism of "last writer wins" strategy not suitable for multi-synchronous collaboration.

1.1.2 Serialisation

Serialisation [266, 202] is an optimistic replication mechanism that allows operations to be executed immediately and, only in the case that an out of order operation arrives at the site, a certain repairing approach is adopted in order to guarantee the correct ordering. A solution to repairing is the Time Warp mechanism [269] where the system returns to the state just before the out-of-order operation was received. Intermediate side effects have to be cancelled by sending anti-messages of the operations locally generated out-of-order and then the messages received are executed again, the late message being received in the right order this time. Another solution to repairing is the undoing of the operations that follow the remote operation in the serial order, followed by the execution of the remote operation and the re-execution of the undone operations. This technique of repairing has been implemented in the GroupDesign [243]. By using serialization mechanism some user updates are lost. Moreover, undoing and redoing operations decreases the response time. Therefore, serialization is not a suitable approach for multi-synchronous collaboration with real-time constraints. Bayou [235] is also based on a serialization mechanism. A primary site ensures a global continuous order over a prefix of the history of modifications. The prefix is further distributed to other sites. The primary site can be a cause of a bottleneck and therefore the approach is not suitable for real-time collaboration.

1.1.3 Multi-versioning

Multi-versioning is an optimistic replication approach that tries to achieve all operation effects and for each concurrent operation targeting a common object, a new object version is created. GRACE [189] and TIVOLI [236] are object-based collaborative graphical editors that use multi-versioning approach to maintain consistency over document copies. A multi-versioning mechanism ensures preserving user intentions, but it leads to multiple versions of graphical objects. No suitable interface is provided to allow users to navigate between object versions and no algorithm is proposed to merge object versions. The mechanism was applied for graphical objects but it would be not suitable for text documents as every concurrent change on the shared document would result in a new version of the document.

1.1.4 Operational transformation

CSCW community aimed to develop approaches that allow users to concurrently edit at any time any part of a shared document and provide solutions for merging concurrent changes that ensure convergence and preserve user intentions. Operational transformation (OT) was specifically designed to fulfill the requirements of multi-synchronous collaboration. It allows users to concurrently work on copies of the documents tolerating copies divergence and ensures that copies converge at a later time. The advantage of OT is its high responsiveness as it allows local operations to be executed immediately after their generation on the local copy of the document. These operations are broadcast synchronously or asynchronously to the other remote document copies. Incoming remote operations are transformed against concurrent operations that were executed on the document copy. Transformations are performed in such a manner that user intentions are preserved and, at the end, the copies of the documents converge. The technique has been implemented in many products including Google Docs, Google Wave, ACE, Gobby, SubEthaEdit. In the following I describe the main operational transformation approaches and their advantages and limits.

1.1.4.1 Basic notions

OT was mostly applied for textual documents that were modeled by a linear structure. Basic operations that can be performed on the document model are the following:

- $Ins(p, c)$ - inserts character c at position p
- $Del(p)$ - deletes the character at position p

OT model considers n sites, each site maintaining a copy of the shared document. When the document is modified on one site, the corresponding operation is executed immediately on that site and then it is sent for execution on the the other sites. Therefore, an operation is processed on four phases:

- generation on a site
- dissemination to the other sites
- reception by the other sites
- execution on the other sites

Between any two operations a causal relation can be established: one of the two operations precedes the other operation or the two operations are concurrent.

Causal ordering relation. Given two operations O_1 and O_2 generated at sites i and j, respectively, O_1 causally precedes O_2, $O_1 \rightarrow O_2$ iff: (1) $i = j$ and the generation of O_1 happened before the generation of O_2; or (2) $i \neq j$ and the execution of O_1 at site j happened before the generation of O_2; or (3) there exists an operation O_3 such that $O_1 \rightarrow O_3$ and $O_3 \rightarrow O_2$.

Concurrent operations. Two operations O_1 and O_2 are said to be concurrent, $O_1 \| O_2$ iff neither $O_1 \rightarrow O_2$, nor $O_2 \rightarrow O_1$.

For instance, in Figure 1.1, $O_1 \| O_2$, $O_1 \| O_3$ and $O_2 \rightarrow O_3$.

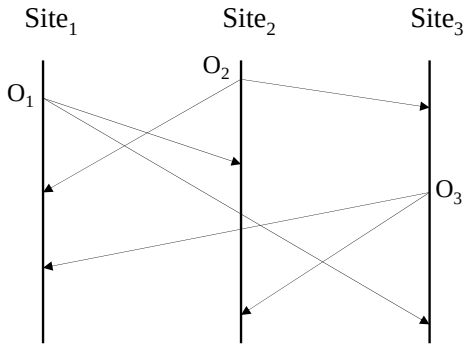
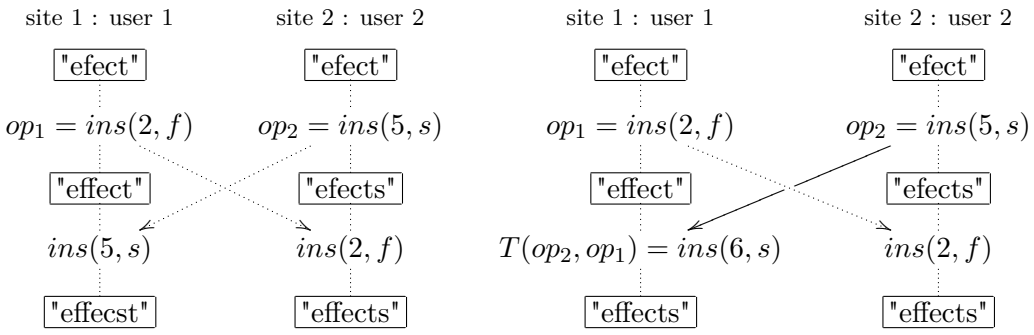


Figure 1.1: Concurrent and precedent operations: $O_1 \| O_2$, $O_1 \| O_3$, $O_2 \rightarrow O_3$

In what follows, I define the notion of operation context, as well as the notions of contextual equivalence and contextual precedence between operations, used by several operational transformation algorithms. The *state* of a document is associated with the list of operations that have to be executed to bring the document from its initial state to the current state. The *context* [201] of an operation O is the document state on which O's parameters are defined. Two operations O_a and O_b are *context equivalent* [201] denoted by $O_a \sqcup O_b$ if the contexts of these operations are equal. Given two operations O_a and O_b, O_a is *context preceding* [201] O_b denoted by $O_a \mapsto O_b$ if the state resulting after the execution of O_a is the context of O_b.

When an operation is received by remote sites the document state on which the operation is executed can be different than the document state on which it was generated (i.e. the context of the operation). In such a case the integration of the operation in original form on the remote sites might generate divergence of document copies at the different sites. In order to illustrate such a situation consider the following example. Consider that two sites $Site_1$ and $Site_2$ share the document with content *"efect"* and two users $user_1$ and $user_2$ generate two concurrent operations op_1=*Ins(2,f)* and op_2=*Ins(5,s)* at $site_1$ and $site_2$. As illustrated in Figure 1.2 (a), if operations are executed in their original forms at the remote sites the two document copies at $Site_1$ and $Site_2$ diverge.

In operational transformation approach, the received operations have to be transformed with respect to concurrent local operations before being executed. This transformation is done using transformation functions. This function denoted T takes as parameter two concurrent operations op_1 and op_2 defined on the same document state s and computes operation op'_1. This operation is equivalent to op_1 in terms of its effect, but it is defined on the state s' resulted after the execution of op_2 on state s. This operation is illustrated in Figure 1.2 (b). I first explain the execution of operations at $Site_1$. After operation op_1 is executed, when op_2 arrives at the site it needs to be transformed against operation op_1 to include the effect of this operation.



(a) Integration without transforming (b) Convergence achieved by transforming

Figure 1.2: Operational transformation.

As operation op_1 was generated at the same time as op_2 and it inserted a character before the character to be inserted by operation op_2, operation op_2 needs to adapt the position of insertion, i.e. increase its position by 1. In this way the transformed operation op_2 becomes op_2'=*T(*op_2*,*op_1*)= Ins(6,s)*. The result document becomes *"effects"*. At $Site_2$, in the same way, operation op_1 needs to be transformed against op_2 in order to include the effect of op_2. The position of insertion of op_1 does not need to be modified in this case as operation op_2 inserted a character to the right of the insertion position of op_1. Therefore, the transformed operation op_1 has the same form as the original operation op_1, i.e. *op'*$_1$*=T(*op_1*,*op_2*)= insert(2,f)*. We see that the result obtained at $Site_2$ respects the intentions of the two users and, moreover, the document replicas at the two sites converge.

The transformation function T is called by certain algorithms such as GOT [223] and GOTO [225] inclusion transformation (IT) and by other algorithms such as SOCT2 [229] forward transposition. The condition of performing the transformation is that the two operations are defined on the same document state.

But operational transformation scenarios can become more complex than the one illustrated in Figure 1.2. Consider again the scenario in in Figure 1.1. The relations between the operations are the following $O_1 \| O_2$, $O_2 \rightarrow O_3$ and $O_1 \| O_3$. Let us analyse how the operations are executed at $Site_1$. When operation O_2 arrives at $Site_1$, it is transformed against O_1 in order to include the effect of operation O_1. The function that transforms O_2 against O_1 is called in [223]inclusion transformation and is denoted by $IT(O_2, O_1)$. O_2 can be transformed against O_1 as both operations have the same context. But, when operation O_3 arrives at the site, it cannot be forward transposed against O_1 because O_1 and O_3 do not have the same context. The context of operation O_3 contains operation O_2, while operation O_1 does not. In order to respect the conditions for performing transformations, in the GOT and GOTO [223, 225] approaches, another function called exclusion transformation was defined while the SOCT2 [229] approach defined the backward transposition function. The *exclusion transformation* $ET(O_a, O_b)$ returns the form of the operation O_a that excludes the preceding operation O_b from its context. The condition of applying the exclusion transformation is that the state resulting after the execution of O_b is the context of O_a, i.e. $O_b \mapsto O_a$. In the previous example, O_3 would have to exclude the effect of O_2 before being transformed against O_1. The backward transposition function defined in SOCT2 changes the execution order of two operations and transforms them such that the same effect is obtained as if the operations were executed in their initial order and initial form.

In the previous example, a backward transposition between O_1 and the transformed form of O_2 against O_1, i.e. O_2', is performed. The backward transposition function computes the form of operation O_2' as if O_1 had not been executed, the result being O_2, and it computes the form of operation O_1, i.e. O_1', that includes the effect of O_2. In this way, O_3 can be transformed against O_1' as both have the same context, i.e. they both include operation O_2 in their contexts. The scenario illustrated in 1.2 is also known as partial concurrency scenario.

Operational transformation approach involves two main components: an integration algorithm and a transformation function. The integration algorithm is responsible with the reception, diffusion and execution of operations. It is independent of the manipulated data types. If necessary, it calls the transformation function. This transformation function is in charge with merging two concurrent operations defined on the same state. The transformation function is specific to a data type such as a sequence of characters as illustrated in the above example.

1.1.4.2 Transformation functions

Transformation functions have to satisfy two properties:

- Condition C_1

  Being given two concurrent operations op_1 and op_2, the relation $op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$ must hold. The notation $op_1 \circ op_2$ denotes the sequence of operations containing op_1 followed by op_2. This property expresses an equivalence between two sequences of operations, i.e. when the sequences of operations op_1 followed by $T(op_2, op_1)$ and op_2 followed by $T(op_1, op_2)$ are applied on the same initial document state, the same document state is produced.

- Condition C_2

  Being given three concurrent operations op_1 and op_2 and op_3, the relation $T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$ must hold. This condition expresses the equality between an operation transformed against two equivalent sequences of operations, i.e. op_3 transformed with respect to the sequence op_1 followed by $T(op_2, op_1)$ must produce the same operation as the transformation of op_3 with respect to the sequence op_2 followed by $T(op_1, op_2)$.

In [233] it was proved that conditions C_1 and C_2 are sufficient to ensure convergence of document copies independently of the order in which concurrent operations are transformed. Basically, these conditions ensure that transforming any operation with any two sequences of the same set of concurrent operations in different execution orders always yields the same result. However, as confirmed in [187] it is very difficult to design transformation functions that satisfy both conditions. The main difficulty comes from satisfying condition C_2.

1.1.4.3 Integration algorithms

Most existing OT integration algorithms such as dOPT [260], adOPTed [233], SOCT2 and GOTO are developed under two theoretical frameworks: [233] and [223]. The first framework [233] requires that OT algorithms preserve causality and achieve convergence. Transformation functions must satisfy the two properties C_1 and C_2. The second framework [223] further includes a new condition called *intention preservation*, in addition to causality preservation and convergence. However, the condition of intention preservation has not been formalized rigorously.

dOPT [260] algorithm pioneered OT mechanism. It identified but did not provide a solution for the case that transformations have to be performed between two operations that do not have the same context.

adOPTed [233] uses directed graphs to model interactions between users involved in the collaboration. The vertices of the graphs represent the application states, while the edges of the graph represent the original user requests or the result of operation transformations. In these approaches in order to compute the execution form of the remote operation, the computation of the intermediate states is required.

Approaches that followed such as SOCT2 [229, 221], GOT [223] and GOTO [225] avoided computing these intermediary states which is very costly. The principle of both SOCT2 and GOTO algorithms is that when a causally ready operation is integrated at a site, the whole log of operations is traversed and reordered such that all operations that are causally preceding the remote operation come before the operations that are concurrent to the remote operation in the history buffer. Afterwards, the remote operation has to be transformed according to the sequence of concurrent operations. SOCT2 and GOTO require transformation functions that satisfy conditions C_1 and C_2. In [187] it was shown that many proposed transformation functions fail to satisfy these conditions. The only existing transformation functions that satisfy conditions C_1 and C_2 are the ones proposed by the TTF (Tombstone Transformation Functions) approach [165]. In the TTF approach when a deletion of a character is performed, the character is not physically removed from the document, but just marked for deletion, i.e. deleted characters are replaced by tombstones. The space complexity is therefore high for the TTF approach.

As satisfying conditions C_1 and C_2 is very difficult, many algorithms tried to avoid satisfying both conditions C1 and C2 or only condition C2.

The GOT algorithm frees from satisfying both conditions C_1 and C_2 by defining a global serialisation order of execution of operations in order to ensure convergence. The global order between operations is defined by the sum of the state vector components and in case of equality by a predefined priority on the sites. As concurrent operations are delivered in an order that does not correspond to the serialisation order, an undo/do/redo scheme has been defined.

Undoing and redoing operations is very costly and therefore some algorithms aimed to eliminate condition C2 but without the need of undoing and redoing operations. SOCT3 and SOCT4 algorithms [215] operations are globally ordered according to timestamps generated by a sequencer. In both SOCT3 and SOCT4 an operation generated on a site is executed without delay. Afterwards, a timestamp is assigned to the operation and the operation is transmitted to the other sites.

In SOCT3 the reception procedure ensures a sequential execution of operations according to the ascending order of their timestamps by delaying the execution of an operation until the operations with lower timestamps have been executed.

In SOCT4 the broadcast of an operation at a site is deferred until all the operations which precede it according to the timestamp order have been received and executed on that site.

Jupiter [234] collaboration system developed at Xerox PARC is based on a client/server architecture in which clients communicate and synchronise with the server. Shared documents are replicated at all cooperating client sites, and also maintained at the central server. Consequently, only client-server communication is needed. In [234] a 2-way synchronisation approach has been proposed between a client and a server but where the server is considered that it can generate operations. A local operation done by a client is executed immediately and then propagated to the server. The server transforms the received operation against operations generated at its site. Similarly, when an operation sent by the server is received at the client site, it has to be transformed before it is executed on the local copy of the document. A state vector is used for

the synchronisation between a client and a server. It contains two elements: the first element represents the number of operations generated by the client and the second element represents the number of operations generated by the server.

An extension of the 2-way synchronisation approach where a client synchronises with a server has been extended to a multi-way synchronisation. This extension has been suggested in [234] and detailed in [206]. Each client-server pair synchronises their copies in the same way as was done by the 2-way communication in Jupiter. The generalisation of the 2-way synchronisation to the n-way synchronisation consists of the fact that when a message generated by a client is received by its corresponding server, it is transmitted to the other servers. In their turn, servers synchronise with the corresponding clients, as if that message had been generated locally. Forwarding a message from a server to other servers has to be atomically processed before another message is processed by any of the servers. Concurrent messages are processed in the order in which they arrive at the server. A client processes immediately the locally generated operations and then the remote operations are processed in the order in which they have been processed at the server side. Jupiter requires a total order broadcast of operations in order to ensure that all remote operations are integrated in the same order by all clients. Convergence is therefore achieved by enforcing the same transformation path at all sites. The final convergence state depends however on the order in which clients synchronise with the server.

ABT [106] approach introduces a constraint called admissibility preservation, which requires that the invocation of any remote operation does not violate the order of objects established by local operations invoked in their generation states. The approach maintains the history H as a sequence of insertion operations H_i followed by a sequence of deletion operations H_d, i.e. $H = H_i \cdot H_d$. Insertion operations and respectively deletion operations appear in the history in the order they have been executed. Before an operation o is propagated the effects of H_d, i.e. all deletions that happened before o are excluded from o. When o has to be integrated at a remote site the subsequence of the history from that site containing insertion operations H_i is transposed into $H_i = H_{ih} \cdot H_{ic}$, where H_{ih} contains insertion operations preceding o and H_{ic} contains insertion operations concurrent with o. When o is integrated it is transformed against $H_{ic} \cdot H_d$. The main idea of ABT is similar to the one of TTF as exclusion of all deletions that happened before the propagated operation and inclusion of all deleted operations before an operation is integrated is equivalent with having tombstones for deleted objects.

Some algorithms for data synchronisation in peer-to-peer environments were developed, but each one has some limitations. MOT2 [146] algorithm is based on operational transformation but does not use state vectors. It uses instead a pair-wise synchronization mechanism according to which it constructs a common history of operations for all sites. The disadvantage of this approach is that during a synchronization phase between two sites histories have to be sent to the other sites and their common prefix has to be determined which might affect performances. MOT2 approach requires TP1 and TP2 conditions for achieving convergence. The only transformation functions that ensure TP2 were proposed by the TTF approach that requires keeping tombstones.

1.1.4.4 OT Summary

Operational transformation has the advantage of being a generic and guided approach including a generic concurrency control algorithm for all data types and operation transformation functions specific to an application domain. However, while it is suitable for groupware with few number of users, it does not adapt well to large scale settings with a large number of users making frequent modifications. Several existing solutions require a central server for message dissemination [234]

or a processing order of messages [215] which limits the scalability. Solutions that do not impose these constraints use data structures that are a function of the number of users such as state vectors [260, 225] or/and of the number of operations such as history buffers [146]. These data structures are very costly in terms of storage and communication as they are attached to every operation. These solutions are not scaling well as their time complexity is proportional to the total number of users and to the total number of operations. Finally, the design of correct transformation functions is complex and costly.

Most existing OT algorithms were proposed for linear data structures with simple operations of insert and delete and in subsection 1.1.6 I discuss some few existing solutions for complex data.

1.1.5 CRDT algorithms

Starting with 2006, a new class of algorithms called CRDT (commutative replicated data types) that ensure consistency of highly dynamic content on peer-to-peer networks emerged. Unlike traditional optimistic replication algorithms, they do not require to detect concurrency between operations in order to ensure consistency. CRDT algorithms rely on natively commutative operations defined on abstract data types such as lists or ordered trees.

WOOT [154] is the first CRDT algorithm. Operations used by this algorithm are insertion and deletion of elements of a linear structure. Elements have associated an unique identifier. An insertion is defined by specifying the element content and the identifiers of the preceding and following elements. Concurrent operations determine partial orders between elements. Merging mechanism can be seen as linearising the partial order to obtain a total order. In order to obtain convergence, the total order has to be the same on all peers. As operations are integrated in any order at a site, merging has to be computed incrementally and independently of the order of arrival. WOOT incrementally computes the linearization order independently of the addition of partial order relationships. The advantage of this algorithm is that it is suitable for open user groups where users often join and leave the network, but it has the disadvantage that it is limited to linear structures. A disadvantage is that the algorithm uses tombstones, i.e. elements are not physically deleted but only marked for deletion. Since tombstones cannot be removed without compromising consistency, performance degrades in time. WOOTO [148] is an optimization of WOOT that uses element degree to compare unordered elements. Degree of an element is computed at the moment of its creation. When an element is inserted its degree is computed as the maximal degree of the neighbour elements +1. The order defined by element degrees ensure that elements with lower degrees are generated earlier than the ones with higher degrees.

To make concurrent assignments commute, in [163] a precedence order between operations is proposed. The approach is similar to last writer wins mechanism and it uses tombstones and vector clocks. The approach does not consider the case where concurrent updates should not be lost but merged. A follow-up work is the replicated growable array [103] CRDT. It supports not only insertion and deletion but also update operations which replace the content of an element without changing the size of the document. RGAs maintain a linked list of elements, via which local operations find their target elements with integer indexes. Meanwhile, a remote operation retrieves its target element via a hash table with a unique index of the target. As a unique index a quadruple vector type s4vector is introduced. A unique s4vector is issued with every operation, and the oldness or newness of multiple s4vectors can be determined transitively, respecting causality. Using the properties of uniqueness and transitivity, the s4vector associated with the insertion that creates an element is used as a unique index of the element, which is also used to resolve conflicts between concurrent insertions at the same position. An insertion

compares its s4vector with the s4vector identifiers of elements next to its target element, and adds its new element in front of the first encountered element that has an older s4vector. That is, a newer insertion inserts an element closely to its target position with higher precedence than relatively ordered concurrent insertions. Such precedence transitivity, realized with s4vectors, ensures consistency of concurrent insertions. As some other CRDTs, RGAs also uses tombstones for deleted elements. Tombstones should be preserved as long as they can be accessed by other remote operations.

TreeDoc [124] and Logoot [125] are based on the same main idea. They maintain a sequence of elements and support insertion and deletion operations on this sequence. An element in the sequence is identified by a unique identifier. Identifiers must be globally ordered identically to the sequence and it must always be possible to create a new identifier between two existing ones. The two approaches try to keep identifiers short.

TreeDoc approach uses a binary tree to represent the document. Element identifiers are represented by paths in the binary tree. The total order of identifiers is given by walking the tree in infix order. Deleted lines are kept as tombstones. The tree can become unbalanced and a balancing mechanism is needed in order to reduce the tree to a sequential array with zero overhead. A garbage collection mechanism that requires consensus was proposed that eliminates tombstones and balances the tree. The garbage collection mechanism aborts in the case that it is performed in parallel with insertion/deletion operations.

A Logoot position identifier is a sequence of 3-tuples: a digit in base BASE, a unique site identifier and a clock value. When an insertion is performed, Logoot allocates a free unique identifier ordered between the left and right position identifiers. There are cases where generation of such an identifier requires extension of the identifier of the left position with an additional layer. Different strategies can be adopted to produce the new identifier [108], all of them using randomness to prevent different replicas to produce concurrently close identifiers.

Existing CRDT algorithms suffer from meta-data overhead that could be excessive for real-time interactions, where the communication granularity can be as small such as a single character. In chapter 4 I describe a CRDT approach that I proposed for strings that considerably limits the meta-data [32].

1.1.6 Consistency maintenance for complex data

As we have seen, operational transformation was mainly developed for linear structured data such as a textual document seen as a sequence of characters where the main operations are insert and delete of characters. In this section I describe some few existing approaches that deal with rich text.

CoWord [171] is a plug-in that supports collaborative editing of Microsoft Word documents. It uses Transparent Adaptation (TA) approach for converting each high-level operation to a sequence of primitive operations. Operational Transformation (OT) technique is applied to this sequence of primitive operations for maintaining consistency among document copies. In CoWord, the set of primitive operations is *Insert*, *Delete* and *Update*. The editor supports creating any Word (rich text) document, but the transformation of high-level operations into primitive ones erases all information about the original high-level operation. For instance, the move of a character is transformed into one *Delete* operation followed by one *Insert*. Concurrently moving a sequence of ten characters while inserting a new character between the fourth and the fifth one will result in the ten characters moved (deleted then inserted), and the new single character inserted outside of its context. The expected result is to have the new character between the others, as the user intended to. Information about the move intention is discarded.

Google Docs and Etherpad[1] work in a similar way. All edits are transformed into three basic types of changes: inserting text, deleting a range of text and applying styles to a range of text. Transformations are provided for all pairs of these types of changes. However, these operations are not enough for capturing user intention. For instance, the previously provided scenario leads to the deletion of the moved sequence of characters together with the new inserted character and the re-insertion of moved sequence of characters at the new position. The new inserted character disappears.

Davis et al. [191] proposed an approach for real-time collaboration over SGML (Standard General Markup Language) documents. The operations that can be performed on the tree are the insertion of a subtree as a child of a specified node, the deletion of a subtree and the modification of the content of a node. As authors mention, this approach does not deal with high-level operations such as move. Indeed, dealing with special cases of concurrent moves is not trivial and the paper does not provide enough details on how this case can be managed.

Docx2Go proposed by Puttaswamy et al. [111] is a framework for collaborative editing over XML documents on mobile devices where not all devices have the complete document. It adapts the Logoot approach proposed by Weiss et al. [125] for XML documents. XML elements possess unique identifiers, the set of identifiers being ordered and dense. Docx2Go supports four types of operations at the element level: *Insert* adds a new element at a specific location in the relative order; *Delete* removes a specified element; *Update* changes the internal contents of an element; and *Move* changes the relative order of a specified element with respect to other elements. When one element is concurrently edited, the generated conflict can be resolved manually or automatically. In the case of a manual resolution the owner of the document is in charge of resolving the conflict. However, very often in collaborative editing multiple users edit the document and there is no explicit owner of the document. In the case of automatic resolution, when concurrent updates are performed on the same element, the element will be duplicated, each version of the element including the individual changes performed. For instance, if the element is a paragraph, the document will contain as many versions of the paragraph as the number of concurrent changes. The paper presents no awareness mechanism that would inform users about the different versions of the elements that were concurrently changed.

As we can see, some existing real-time collaborative editors were developed for rich text documents, but they do not take advantage of the underlying structure of the document and they model it as a linear sequence of elements. High level operations are translated into primitive insert, delete and update operations on basic elements. This approach leads to simple algorithms, but semantics of user operations is lost. As previously explained, a move of a block of text is obtained by deletion of the block character by character followed by re-insertion of each character of the block. If a user modifies a block of text that is concurrently moved, then his changes are either completely lost or they are placed outside the context of the moved block of text. In this case, user intention of modifying the paragraph is lost.

There is no suitable mechanism that offers an automatic resolution of conflicts that closely reflects user intentions. The previously highlighted issues are not simple bugs that can be corrected. My claim is that the only way that an approach preserves user intentions is that it deals with a rich set of operations for which intentions are precisely specified. In chapter 2 I describe our operational transformation based approach for rich text where we define operations with high-level semantic capturing user intentions.

[1] http://etherpad.org/

1.2 Awareness and conflicts management

This section presents a short overview of awareness mechanisms for conflicts prevention and on conflicts management once they occurred.

1.2.1 Conflicts prevention

Studies showed that in large projects the partition of software modules among developers is limited and developers can contribute to any part of the code [170]. Unfortunately, the awareness information provided by the currently available awareness–enabled multi-synchronous applications is limited. When users work with multi-synchronous collaborative applications, they remain isolated. They publish their changes to other users when they commit their changes to the repository and are informed about changes of other users only when they update their local copy. Some of the existing change awareness approaches [136, 161] track the changes that other collaborators have made to a group project by highlighting them since the last time the user saw that artifact. These approaches offer awareness for the changes committed in the repository and integrated in the local workspace of the user. However, when users are not aware of other users' activities while they are working, they may concurrently modify the same parts of the shared document. This could lead to conflicts or redundant work. A conflict would, for instance, be generated if a user proofreads a document section while another user concurrently deletes this section. Finally, redundant work would occur if two users concurrently perform identical tasks on their local copies of a document.

CVS watches [258] permit users to subscribe for changes performed on an artifact and to be notified by email when a user announces by means of a command his intent to modify that artifact. However, watches require the use of email as an external tool for coordination in software development. In [153] the email notification mechanism is replaced with a lightweight notification mechanism called Elvin together with a tickertape tool where CVS messages are displayed and where developers can also chat with one another. Elvin and the tickertape are integrated in the CVS. These approaches do not provide a presentation mechanism of the changes performed.

VC^2 [147] is an awareness tool that can be integrated with existing version control systems. The file system is monitored for changes performed on documents. A user working on a document receives a notification when another user starts editing the same document. The user can then ask the other user for committing his changes. The second user might accept or reject the request.

In [144] a real-time awareness is provided for collaborative software engineering. Warning messages are used to notify developers about concurrent activity. Developers can afterwards consult the list of conflicts. Moreover, based on a selected conflict, a user can set watches for concurrently edited elements such that he is informed when the collaborator finished editing the element.

State Treemap [209] informs users about states of shared documents such as *LocallyModified*, *PotentiallyConflict* – when two copies of the document are modified and none of the changes are published yet – or *WillConflict* – when a document copy is modified locally and some changes on that document have been committed.

Palantìr [185] is an awareness tool for distributed software development based on the same principle as State Treemap, the main difference being that severity information that computes the amount of changes performed among documents is provided.

In the divergence metrics approach [199], metrics are not based as in Palantìr and State

Treemap approaches on events triggered when the states of documents are changed, but they rather use information provided by operations that model concurrent changes. It is possible to compute the amount of concurrent changes performed on each document or an amount of conflicting/overlapping changes.

However, none of the previously mentioned approaches directly localises and presents the concurrent changes. Some of these approaches such as [258], [153] and [147] provide notifications about concurrent changes without any information on the changes performed. In [144] the notification mechanism allows users to consult conflicting changes that are listed in a separate document. The approaches in [209], [185] and [199] provide either a simple information that an artifact has been concurrently changed or a quantitative information about the concurrent changes performed on the same artifact. However, no information about the localisation of changes is provided.

There is a need of an approach where concurrent changes are localised and annotated on the local document without any burden for users to set watches and analyse their results. Moreover, in all of the previous mentioned approaches all uncommitted operations are sent to all members of the team and no privacy issues are taken into consideration. In chapter 5 I present an awareness mechanism that informs users about location of concurrent changes, users being able to consult the list of changes performed. I also present a mechanism that considers the trade-off between awareness and user privacy in multi-synchronous collaboration.

1.2.2 Conflicts management

In this section I present some literature review on conflicts management in version control systems which is the main topic of the study I present in section 5.2.

In Git [79], users can synchronize their changes with other users working in parallel with them. In this process, a merge is performed between local changes and remote changes. If concurrent changes refer to the same file, we say that these changes are conflicting. Conflicts on a file can be automatically resolved or they need user intervention for their resolution. We call the former category automatically resolved conflicts and the latter category unresolved conflicts. If conflicting changes refer to different non-adjacent lines of the file, the conflict is automatically resolved by the system. If, on the contrary, conflicting changes refer to the same or adjacent lines of the file, the conflict cannot be automatically resolved and the user has to manually resolve it. Unresolved conflicts also occur if a file is renamed and modified/deleted concurrently, if it is modified and deleted concurrently, if it is renamed concurrently by two users, if a user renames a file with the same name as another user gives to a concurrently created file and if two users concurrently add two files with the same name. Note that the name of a file is the whole path identifying that file.

The user study presented in [241] reports on conflict resolution experiences with the optimistic file system Ficus. Conflicts were classified into *update/update*, *remove/update* and *naming* conflicts. *Update/update* conflicts appear when two concurrent updates are performed on the same file. *Remove/update* conflicts appear when an update of a file and the removing of that file were performed concurrently. A *naming* conflict occurs when two files are independently created with the same name. The study found out that only about 0.0035% of all updates made to non-directory files resulted in conflicts and among them less than one third could not be resolved automatically. Authors mentioned that conflicts that cannot be resolved automatically are any *update/update* concurrent changes on source code or text files as they have arbitrary semantics and therefore require user intervention. Note that the definition of conflicts used in this study is slightly different than the one used in the domain of version control systems where

two updates done on the same file (source code or textual) lead to non-automatically resolved conflicts only if the updates refer to the same or adjacent lines in the file. All *update/remove* conflicts required human intervention and about 0.018% of all *naming* conflicts led to name conflicts which have to be resolved by humans.

The study in [207] analyses the software development history of the 5ESS switching system of Lucent Technologies over a period of 12 years. Each set of change requests representing all or part of a solution to a problem was recorded by the system. When a change from this set was made on a file, the system kept track of the lines added, edited or deleted. This set of changes composes a *delta*. It was found that 12.5% of all *deltas* were made to the same file by different developers within a day. 3% of all these *deltas* made within a day by different developers physically overlap. However, interference of these *deltas* is analyzed over a quite large period of time (1 day) and not all these *deltas* are performed concurrently.

In [141] authors investigated four large open-source projects (GCC, JBOSS, JEDIT and PYTHON) and found that in CVS the *integration rate* that measures the percentage of updates which were integrated with local user changes is very low (between 0.26% and 0.54%), and the *conflict rate* is between 23% and 47%. Low integration rates indicate that the parallel changes within single files are rare and have small impact to the development process. High conflict rates indicate that parallel changes affect the same location within a file or can not be integrated automatically by CVS.

Only few studies analysed parallel changes and conflicts for projects developed using distributed version control systems (DVCs). In [102], authors studied the *conflict rate* of textual conflicts in terms of change-sets (i.e. commits that refer to the whole project) for nine open-source projects using Git repositories. They found that the average conflict rate was 16%. However, no study analysed in detail textual conflicts in DVCSs and how people resolve these conflicts. In section 5.2 I present an analysis of conflicts in Git including textual conflicts and study how these conflicts are resolved.

2

Operational transformation for complex data

Operational transformation was mainly developed for linear structured data such as a textual document seen as a sequence of characters. During my PhD thesis, I developed a multi-level editing approach for maintaining consistency over hierarchical-based documents such as text and XML documents. The multi-level allows any existing operational transformation algorithm for linear structures to be applied recursively over the hierarchical structure of the document. In collaboration with Gérald Oster, I combined the multi-level editing approach [16] with the TTF approach [165]. The obtained algorithm constitutes the first approach for the reconciliation of XML documents adapted to peer-to-peer environments [41]. More precisely, I adapted the TTF approach for XML documents by developing transformation functions that satisfy the necessary properties C1 and C2 for consistency maintenance. However, the algorithm proposed in [41] deals only with simple operations such as insert, delete and update of the node attributes. As mentioned in chapter 1, this set of operations is not sufficient for preserving all user intentions. The solution for preserving user intentions is to model a rich set of operations for which intentions are precisely specified.

Together with Gérald Oster and Luc André, I designed a new operational transformation algorithm for maintaining consistency for rich text data such as wikis [13]. Our solution was transferred to XWiki and used to develop a real-time editor for XWiki wikis in the context of Wiki3.0 project. Existing wikis offer limited support for merging concurrent contributions on the same pages. Users have to manually merge concurrent changes and there is no support for automatic merging such as in the case of real-time collaborative editing. We proposed extending wiki systems with real-time collaboration. Our merging solution is based on an operational transformation approach for which we defined operations with high-level semantics capturing user intentions when editing wiki content such as move, merge and split. Our solution is the first one that deals with high level operations, existing approaches being limited to operations of insert, delete and update on textual documents. While it is impossible to exactly understand a user intention, by enhancing the set of operations we offer users more ways to express their actions and thus more chance to find the automatic conflict resolution appropriate. Our solution assumes the presence of a central server. This assumption matches with the XWiki architecture which is centralized.

In what follows I describe very briefly our wiki page document model and the set of operations that we considered. Then I present an overview of our underlying merging algorithm including the transformation functions among the considered operations. I will end the chapter with a

complexity analysis of our approach.

2.1 Document model

The source code of a wiki is called wikitext and is a combination of macros, meta information for a page and HTML. There is no one-to-one correlation between wikitext and HTML as certain features are present in one but not the other. Therefore, we cannot consider the DOM (Document Object Model) as the underlying representation of the document. There are also two other reasons why DOM cannot be considered as the basic document structure. Firstly, translation of user changes on the underlying DOM representation of the document is browser dependent and varies from one browser to the other. Secondly, users should be able to switch from editing the wiki page from a wiki syntax to WYSIWYG editor.

Representing the document model by using a linear structure is not feasible. For instance, styles and paragraphs contain text which in a linear structure would be represented by means of special characters for begin and end of the text. Correctly updating these characters when operations and their transformations are performed and guaranteeing that the document is well formed would be very complex.

The solution that we adopted was to design a specific hierarchical wiki structure for the underlying document model. This allows wikitext to be represented in a sufficiently abstract manner such that it can be modified and rendered back into wikitext without loss of information as well as into a variety of HTML formats or no-HTML formats such as plain text. Adopting a wiki specific model as the underlying structure was also the solution chosen by MediaWiki visual editor (`https://www.mediawiki.org/wiki/VisualEditor`).

The structure of the wikitext is described using elements, some of which contain other elements while others contain content, but never both. The elements composing the wikitext are listed below:

- **Paragraph** consisting of a series of lines of content.
- **Heading** consisting of a single line of content and a heading level.
- **List** consisting of a series of items, each containing a single line of content, depth and style information.
- **Table** consisting of a series of rows, each containing a series of cells composed of a series of elements.
- **Template** consisting of an application controlled content with any number of parameters composed of content/elements.

The meaning or appearance of text can be defined by annotating the previously described elements to obtain the following features:

- **Formatting**: Bold, italic, internal and external links, etc.
- **Rendering**: Images and templates.

The XWiki WYSIWYG editor has a hierarchical structure summarized by the following grammar:

```
document  =  element+
element   =  paragraph | heading | item | tableRow
paragraph =  p(content+)
heading   =  h[1-6](content+)
item      =  (ol|ul)@depth(content+)
tableRow  =  R(content+) cell+
cell      =  C(content+)
content   =  text | inline | link | image
inline    =  span @style (text)
link      =  a @href @style (text)
image     =  img @src (text)
```

The *document* is seen as a sequence of elements that can be paragraphs, headings, items of a list or rows of a table. A *paragraph* node, with node name `p` contains a sequence of contents whose structure is presented in the next paragraph of this section. *Heading* represents section titles of the document. Depending on the relative importance of each section, the names of those nodes are `h1`, `h2`, `h3`, `h4`, `h5` and `h6`. `h1` corresponds to the major section of the document, while `h6` corresponds to the least important section of the document. A heading has the same structure as a paragraph, i.e. it contains a sequence of contents. An *item* node of a list has also the same structure as a paragraph containing a sequence of contents. The node name can be either `ol` or `ul` with `depth` attribute. `ul` denotes an unordered list and `ol` denotes an ordered list. `depth` denotes the level of imbrication of the elements of the list. A table is composed of *tableRows*. Each row of the table is composed of a sequence of cells. The first cell in the row is denoted by `R`, while the other cells are denoted by `C`. A *cell* has the same structure as a paragraph, i.e. containing a sequence of contents. Note that rather than defining an element list or table, we define only item list elements with a level of imbrication and table rows. The reason is that we want to have as far as possible elements with the same structure such that we can define generic operations on these elements.

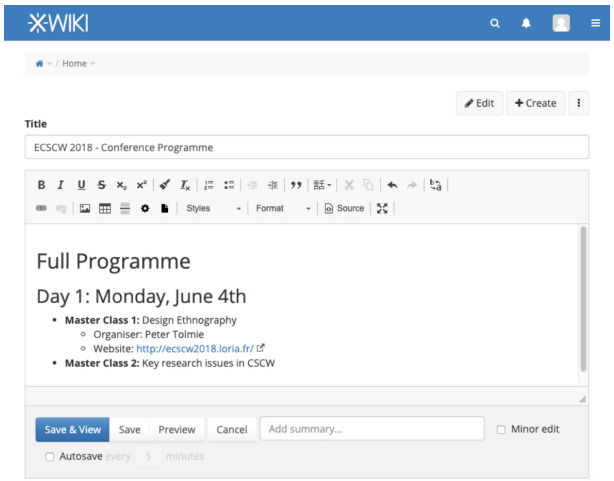
A *content* element can be:

- a *text* node containing a sequence of characters
- an *inline* node whose node name is `span` and that has a `style` attribute. An *inline* node is composed of a single text node.
- a *link* or anchor node whose node name is `a` and that has `href` and `style` attributes. The `href` attribute specifies the URL of the page the link refers to, i.e. the link's destination. The `style` attribute specifies the style of the link.
- an *image* node whose node name is `img` and that has a `src` attribute.

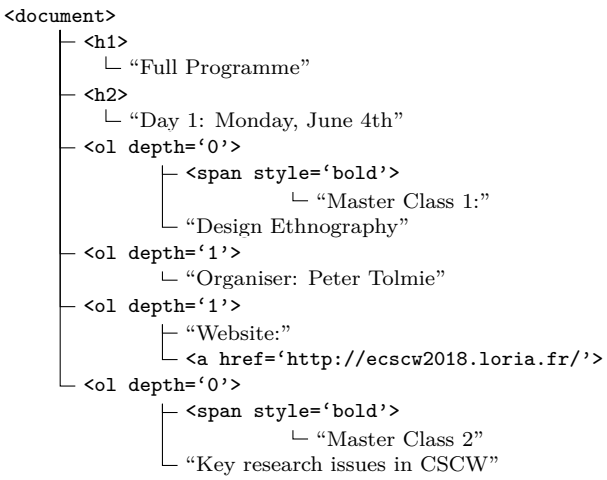
An *element* in the above presented document structure, i.e. in the first level of the document, directly under the root, is identified by an *index* in the list of elements. For instance, in Figure 2.1c, the element `<h2>` is identified by the index 1. Other interior nodes are identified by their *path* in the tree. For instance, in Figure 2.1c, the span node `<span style='bold'>` with the content `"February 2, 2016"` is identified by `[2,0]`. A character in a text node is identified by the *path* to the text node and a position *pos* inside the text node. For instance, the character `'e'` from the text node `"February 2, 2016"` is identified by the *path* `[2,0]` to the text node and the position 1.

```
= Full Programme =

== Day 1: Monday, June 4th ==

* **Master Class 1:** Design Ethnography
** Organiser: Peter Tolmie
** Website: http://ecscw2018.loria.fr/
* **Master Class 2:** Key research issues in CSCW
```

(a) Wikitext of the wiki page.



(b) Wiki page edited in XWiki.



(c) Document model.

Figure 2.1: The different representations of a wiki page.

2.2 Operations

Table 2.1 summarizes all operations that integrate user action semantics and that are applied on the document model presented in the previous section. For deleting an element the characters of the element are successively deleted and the empty element is kept in the document.

Table 2.2 classifies the operations according to their effect on the document structure: some of them modify the index of the targeted elements, while others modify the element type, the inner structure of the tree or the content of the leaf text nodes.

One of the main contributions of our approach is the design of split and merge operations working on the tree structure as well as the definition of the combined effects of these operations with all the other operations in our model.

`Split` and `MergeElement` were defined for our needs and behave in a specific way. Firstly, a `Split` operation splits an element at the deepest level in the tree, and duplicates every upper level elements to create a second branch from the root. For instance, Figure 2.2 illustrates that the split of `<p>ab<b>cd</b>ef</p>` after `c` leads to `<p>ab<b>c</b></p><p><b>d</b>ef</p>`.

Secondly, we only allow the merge of elements at a depth of one (direct children of the root). It is sufficient in our context, since the aim of our model is to focus on the modifications of the structure of the wiki document. The structure is defined by the first level of the tree model, the other levels being related to content. In our context, a `Split` occurs when one user presses enter with the caret inside an element : the element (paragraph, item...) is divided into two. Similarly, a `MergeElement` occurs when backspace is pressed while being between two elements.

When a `Split` occurs, we need to know whether it initially occurred at the beginning of the

InsertText	*pos*	insertion position into the leaf	Inserts one character in a leaf text node. *siteId* defines priority in case of two concurrent operations at the same position
	path	path from the root to the leaf	
	char	inserted character	
	siteId	unique site identifier	
DeleteText	*pos*	deletion position into the leaf	Deletes one character in a text. node.
	path	path from the root to the leaf	
NewElement	*index*	index of the new element	Creates a new element with an empty text node. *siteId* defines priority in case of two concurrent insertions at the same index
	content	type of the new element (h1,p...)	
	siteId	unique site identifier	
UpdateElement	*index*	index of the updated element	Updates an element. *siteId* defines priority in case of two concurrent updates at the same index
	content	type of the updated element (h1,p...)	
	siteId	unique site identifier	
MoveElement	*origin*	index of the element to be moved	Moves one element child of the root. *siteId* defines priority in case of concurrent moves of two different elements to the same location or of same element to two different locations
	dest	final index of the element	
	siteId	unique site identifier	
MergeElement	*index*	right element index	Merges two adjacent elements *content* and *leftChNb* are used during operation transformation
	content	type of the left element	
	leftChNb	number of children in the left element	
Split	*pos*	split position into the leaf	Splits one node in two. *splitLeaf* is used during operation transformation
	path	path from the root to the leaf	
	splitLeaf	true if a leaf needs to be split	
Style	*start*	start position of the range into the leaf	Adds/deletes a style to some text inside a leaf text node *siteId* defines priority for two concurrent modifications of the same attribute *splitStart* and *splitEnd* are used during operation transformation
	end	end position of the range into the leaf	
	path	path from the root to the leaf	
	param	name of the style	
	value	value of the style	
	siteId	unique site identifier	
	splitStart	true if start of the leaf not included in the range	
	splitEnd	true if end of the leaf not included in the range	

Table 2.1: Summary of operations and their parameters.

element. The reason is that, if the split is at the beginning of the element, this element is not split but moved. If the `Split` position is inside an element, the element is split and then the right part is moved.

When merging two elements, the position of the right element to be merged (the position in the children list of the root) is provided and the children of this element are appended to the children of the left element of the merge (the right one is discarded). We store in the operation's attributes the initial number of children in the left node, before the merge. This parameter is needed to transform any operation that updates the right node. The path of the transformed operation has its first level reduced by one and its second one increased by the number of left children. For instance, the transformation of `InsertText(2,[1,1],a,3)` against `MergeElement(1,p,2)` results into `InsertText(2,[0,2],a,3)`.

operation	structure modification			
	element index	element type	inner tree structure	leaf content
InsertText				×
DeleteText				×
NewElement	×			
UpdateElement		×		
MoveElement	×			
MergeElement	×	×	×	
Split	×		×	×
Style			×	×

Table 2.2: Classification of operations.

(a) initial document.

(b) after split execution.

Figure 2.2: Document model before and after a split after character ‘c’.

2.3 Merging algorithm and transformation functions

As the architecture is centralized, our merging algorithm can rely on a central server that will do the merging. We used operational transformation mechanism that is well known for its suitability for the real-time collaboration in a centralized architecture.

Any integration algorithm such as Jupiter proposed by Nichols et al. [234] or SOCT4 proposed by Vidot et al. [215] can be used. In our solution we used Jupiter algorithm, more precisely the generalisation of the 2-way synchronisation of Jupiter algorithm to a multi-way synchronisation that has been explained in [206]. In order to obtain convergence, Jupiter algorithm requires that the transformation functions satisfy the C1 condition.

We defined sixty four transformations corresponding to the eight considered operations. More details about these transformations can be found at [13]. Here I only provide a brief overview of our transformations. Interested reader can refer to the full set of transformation functions available at `https://gist.github.com/oster/04ca4fc1aaea7de58700`.

Table 2.3 summarizes the combined effect of the different classes of operations: the ones that modify elements index, elements type, the inner structure of the tree or the content of the leaf text nodes. Columns indicate the classes of operations to be transformed, while lines indicate the classes of operations against which the transformations are performed.

structure modification of the operation to be transformed according to	**structure modification performed by the operation to be transformed**			
	element index	element type	inner tree structure	leaf content
element index	update of the element index	.	.	.
element type	update of the element index	.	.	.
inner tree structure	update of the element index	.	update of the path to the leaf	.
leaf content	update of the element index	.	update of the path to the leaf	update of the position in the leaf

Table 2.3: Summary of the transformation functions.

2.4 Complexity discussion

Our solution reduces the number of operations needed to describe a task and therefore the size of the log is reduced and there is an improvement in the computation of transformations. Indeed, the proposed operations replace a set of operations on characters. For instance, `UpdateElement` operation replaces the set of operations corresponding to the deletion of each character composing the old content of the element and the insertion of each character of the new content of the element. A `MoveElement` operation replaces the set of operations corresponding to the deletion of each character composing the structure of the old element and the insertion of each of the deleted characters to the new position.

As we generate a much smaller number of operations than the number of operations generated by the character-based approaches, the communication traffic is reduced.

The complexity of Jupiter algorithm that we use is `O(n)` where `n` is the number of concurrent operations. As in our case the number of operations is reduced compared to the approaches based on characters, we achieve a better complexity and therefore better performances. If the complexity of existing algorithms working on simple operations on characters is `O(`n_1`)`, where n_1 is the number of concurrent operations generated by the k sites, the complexity of our approach is `O(`n_2`)`, where n_2 is the number of concurrent complex operations generated by the k sites with the property that $n_2 << n_1$. Moreover, in the case of simple operations on characters, these operations which are usually insert and delete, modify characters index and their transformations have to compute the updated index. In the case of our approach, operations are classified into ones that modify elements index, elements type, the inner structure of the tree or the content of the leaf text nodes. As operations modify different document structures, most of transformations among these operations need no computation as they return the original operation. For instance, operations that modify uniquely the elements index such as `NewElement` and `MoveElement` do not interfere with operations that modify leaf content such as `InsertText` and `DeleteText`, operations that modify the element type such as `UpdateElement` and operations that modify the inner tree structure such as `Style`.

3

Evaluation of optimistic replication approaches

I leaded the work done in the context of ARC RECALL project on the evaluation and comparison from a theoretical point of view optimistic replication algorithms suitable for decentralised environments [46], namely: MOT2 operational transformation approach [146], WOOTO CRDT approach [154, 148] and a serialisation and conflict resolution approach [176]. We evaluated these decentralised approaches in the context of the synchronisation of wiki pages and we compared them also with the existing centralised merging solution supported by MediaWiki. The evaluation was done according to the following criteria: communication complexity (the number of messages exchanged for all sites in order to achieve the final state for convergence), time complexity (the time necessary to obtain convergence), space complexity (the amount of required memory), first site convergence latency (the minimal number of rounds necessary for the first site to reach its final state) and convergence latency (the number of rounds necessary for all sites to converge to the final state).

In collaboration with Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh and Pascal Urso, I refined this evaluation methodology by including more criteria for a theoretical comparison between algorithms and by building a general framework that would allow comparison of algorithms performance against real traces of collaboration. Our framework includes the implementation of various algorithms and the support for running these algorithms in the same experimental settings and measuring their performance. We evaluated theoretically and practically using real traces of collaboration the main representative algorithms of OT (Operational Transformation) and CRDT (Conflict-free Replicated Data Types) working at the level of character. We chose WOOT with its optimisations WOOTO and WOOTH, Logoot and RGA as representative of CRDT family. We chose SOCT2 algorithm as the main representative OT algorithm that does not require a central server and that is suitable for peer-to-peer collaboration and that can use TTF (Tombstone Transformation Functions), the only transformation functions that satisfy C1 and C2 conditions necessary for correct transformations in peer-to-peer environments. This was the first work [36] that evaluated and compared different algorithms belonging to different classes and showed that the new family of CRDT algorithms outperforms OT algorithms in collaborative editing.

In what follows I describe the theoretical complexity of these algorithms and their evaluation with real traces of collaboration. I next describe how we evaluated the optimistic replication algorithms with with real traces of collaboration. I finally describe an experimental design done in collaboration with the Department of Psychology from Wright State University where we

studied the effect of algorithm complexity in terms of delays on users.

3.1 Theoretical complexity of main optimistic replication algorithms

The worst case complexity for WOOT, WOOTO and WOOTH, Logoot, RGA and SOCT2/TTF algorithms is presented in Table 3.1. We consider the time complexity of generation of a local user operation (single character insert or delete) and for the execution of a remote operation. We denote by R the number of replicas and by H the number of operations that had affected the document. We consider constant time for accessing an element in a hash table. In the worst case scenario for the approaches that use tombstones, the document size including tombstones equals H. For SOCT2 approach that uses state vectors we took into account the complexity of state vector creation, i.e. $O(R)$, associated with the operation at the moment of its generation.

Algorithm	local		remote	
	ins	del	ins	del
WOOT	$O(H^3)$	$O(H)$	$O(H^3)$	$O(H)$
WOOTO	$O(H^2)$	$O(H)$	$O(H^2)$	$O(H)$
WOOTH	$O(H^2)$	$O(H)$	$O(H^2)$	$O(log(H))$
Logoot	$O(H)$	$O(1)$	$O(H.log(H))$	$O(H.log(H))$
RGA	$O(H)$	$O(H)$	$O(H)$	$O(log(H))$
SOCT2/TTF	$O(H+R)$	$O(H+R)$	$O(H^2)$	$O(H^2)$

Table 3.1: Worst-case time-complexity analysis

The average complexity of each of the above described algorithms is presented in Table 3.2. We denote by:

- c the average number of operations concurrent to a given one,
- n the size of the document (non deleted characters),
- N the total number of inserted characters (including the ones that were deleted called tombstones),
- k the average size of Logoot identifier[2].
- $t = N - n$ the number of tombstones,
- $d = \lceil (t + c)/n \rceil$ the average number of elements (tombstones and concurrently inserted elements) found between to successive document elements

The space complexity of meta-data used by each replica is presented in Table 3.3. In average, algorithms using tombstones need to store N elements in their model. Logoot stores n identifiers with an average size of $O(k)$. SOCT2 additionally stores a log of operations, each one containing a version vector with size of $O(R)$.

More details about the above complexities can be found in [36].

[2]Theoretically, the size of a Logoot identifier is only bounded by H, but due to stochastic nature of Logoot identifier generation, it has only an infinitesimal chance to be proportional to H.

Algorithm	avg. local		avg. remote	
	ins	del	ins	del
WOOT	$O(N.d^2)$	$O(N)$	$O(N.d^2)$	$O(N)$
WOOTO	$O(N.d^2)$	$O(N)$	$O(N+d^2)$	$O(N)$
WOOTH	$O(N+d^2)$	$O(N)$	$O(d^2)$	$O(1)$
Logoot	$O(k)$	$O(1)$	$O(k.log(n))$	$O(k.log(n))$
RGA	$O(N)$	$O(N)$	$O(1+c/n)$	$O(1)$
SOCT2/TTF	$O(N+R)$	$O(N+R)$	$O(H.c)$	$O(H.c)$
with g.c.	$O(N+R)$	$O(N+R)$	$O(c^2)$	$O(c^2)$

Table 3.2: Average time-complexity analysis

Algorithm	space complexity	
	worst	average
WOOT-WOOTO-WOOTH	$O(H)$	$O(N)$
Logoot	$O(H^2)$	$O(k.n)$
RGA	$O(H)$	$O(N)$
SOCT2/TTF	$O(H.R)$	$O(H.R)$
SOCT2/TTF with g.c.	$O(H.R)$	$O(N+c.R)$

Table 3.3: Space complexity analysis of meta-data

3.2 Evaluation of optimistic replication algorithms with real traces of collaboration

Logs offered by commercial real-time collaboration systems such as Google Docs are not complete and freely available. For example, the revision log provided by Google server is a serialization of user operations transformed by the Jupiter algorithm [234]. Therefore, the revision logs open to the public do not include the information needed for replaying the real-time peer-to-peer collaboration, such as version vectors.

Due to unavailability of logs of the real-time peer-to-peer collaboration, we set up an experiment described in [36]. We asked students to perform several tasks by collaboratively writing documents by using TeamEdit [84] collaborative editor and we logged the operations generated during this experiment.

We logged the following user operations: insertions of a text block and deletions of a range of characters. Text blocks and ranges have a size of one character when a user types on the keyboard. They have a larger size when a user copy-pastes a text block or deletes a selected block. In order to apply the studied algorithms on the generated logs, user operations must be transformed into character operations.

We studied algorithm performance behaviour over time in order to analyse how algorithms may degrade over time due to tombstones or growing identifiers. The observed behaviour was approximately the same for the different groups and the different tasks. I show here the execution time for user operations (Figure 3.1) and character operations (Figure 3.2). We computed an average of the execution times for every hundred user operations and every three hundred character operations. The horizontal axis uses a linear scale representing the number of elapsed operations. The vertical axis uses a logarithmic scale and represents the average time, in microseconds, required to execute operations.

For performance behavior based on user operations as shown in Figure 3.1, we can notice peaks of low performance common to all algorithms due to operations inserting a block of hundreds, or thousands of characters. The performances of the algorithms using tombstones

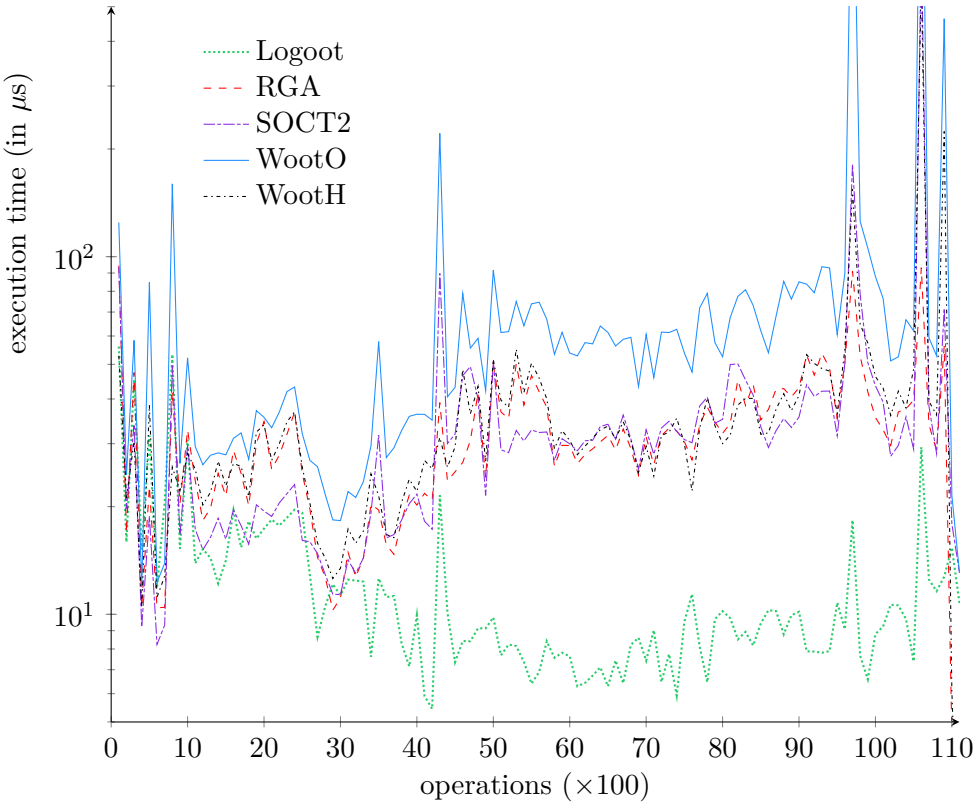


Figure 3.1: User operation execution times for a group of students that collaboratively edited a report

(WOOTs, RGA and SOCT2) eventually degrade over time. Indeed, all these algorithms have to count the number of tombstones and characters present before an inserted or suppressed string. The performance of Logoot remains good during the whole period. Experiments show that any algorithm counting tombstones (WOOTs, RGA and SOCT2) behaves worse than algorithms not based on tombstones such as Logoot.

For performance behavior based on character operations as shown in Figure 3.1, we can notice that the performance remains stable for Logoot. The performance of RGA and WOOTH are, in average, better than Logoot but have a more erratic behavior. The behavior of RGA and WOOTH is composed of a base line at 1 μs and some lower performance periods due to more frequent concurrent editing. RGA over-performs WOOTH in case of concurrent delete operations. The performances of WOOTO and SOCT2 degrade over time, SOCT2 having the most erratic behavior and the worst average performance.

We also studied the occupied memory by these algorithms [33]. As shown by the average theoretical space complexities, the worst algorithm is SOCT2 for which the occupied memory linearly grows with the number of operations. Among CRDT algorithms, the least performant is Logoot, whose memory size is growing due to identifiers size. The occupied memory of RGA, WOOTO and WOOTH is mainly due to tombstones and as few deletions occurred during the given task (less than 20% among the total number of operations), performance of these algorithms is good.

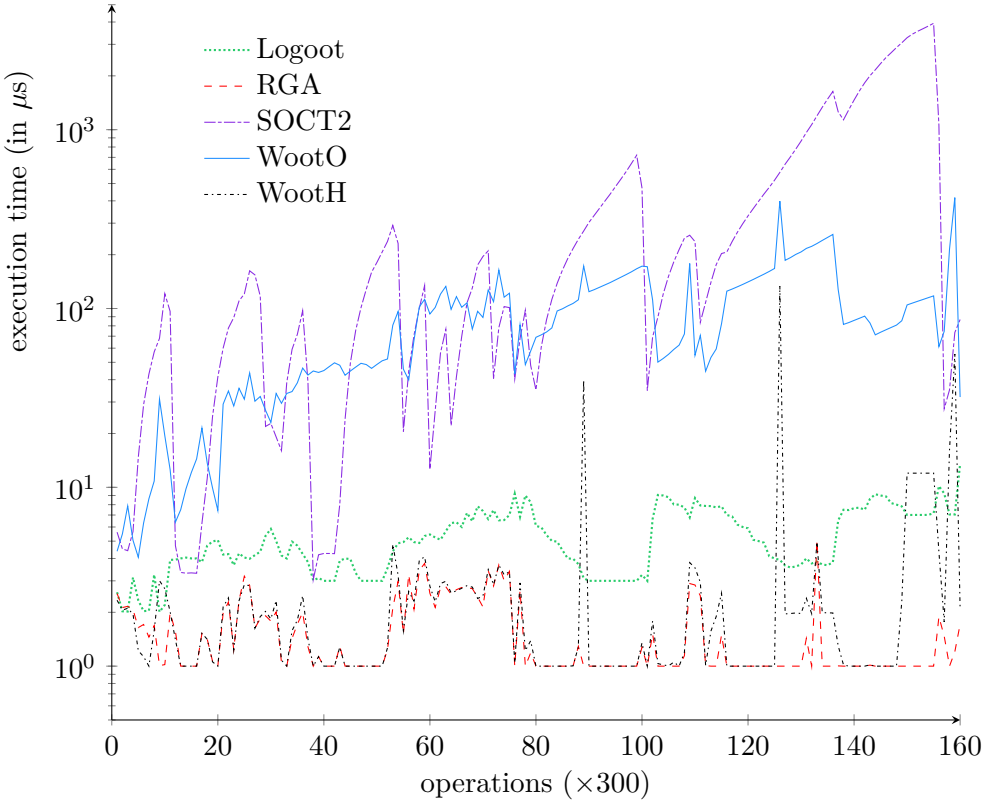


Figure 3.2: Character operation execution times for a group of students that collaboratively edited a report

3.3 Measurement of collaborative editors performance: a user perspective

In the context of the PhD thesis of Quang-Vinh Dang, I aimed at measuring the performance of real-time collaborative editing systems from users point of view. More precisely, I aimed studying the perceived delay by users in online real-time collaborative editing systems in their normal working environment, i.e. using web browsers. We studied delays experienced by users in GoogleDocs and Etherpad, two of the most popular real-time collaborative editors. Delays in these systems can be defined as the time that a modification done by a user is visible to the other users.

Delays can be caused by different reasons:

- network delay due to physical communication technology be it copper wire, optical fiber or radio transmission
- complexity of various algorithms for ensuring consistency: where most of them depend on the number of users and number of operations that users performed.
- on the type of architectures: For thin client architectures the computation for algorithms for maintaining consistency is done mainly on the server, which becomes a bottleneck in the case of a high number of users and operations. Hence, the delay for seeing operations of other users is high. For thick client architectures the computation is done mainly on the client side and delays are lower in this case.

Setting up an experiment with numerous real users that edit concurrently a shared document would not be possible with current tools. Existing tools restrict the number of users editing a document and most of them are not open-source in order to allow code instrumentation for delay measurement. We instead simulated user behavior by means of agents that use popular web-based real-time collaborative editing services currently available in the market: Google Docs and Etherpad. In order to simulate the real user behavior on web browsers, we selected Selenium [152], a widely used web-based testing tool [92].

Three types of simulated users have been defined:

1. Writer: writes a specific string to the shared document.
2. Reader: waits and reads the specific string from the writer.
3. DummyWriter: writes random strings to the shared document. Random strings are different from the specific string. DummyWriters are used to simulate concurrent users.

Each simulator (Writer, Reader, DummyWriter) performs its task on different Google Chrome browser windows. The delay is measured by the time period between the moment the specific string is written by the Writer and the moment when the specific string is read by the Reader. In order to avoid clock synchronisation issues, both Reader and Writer are executed on the same computer.

For each real-time collaborative editing system, i.e. Google Docs and Etherpad, we measured the performance (delay) in different settings by varying the number of users who modify the document at the same time, and their typing speed, i.e. the number of characters each user types to the document in one second. As the number of users that can concurrently modify a document in Google Docs was limited to 50 by the time we performed the study, we varied the number of users from one to 50. We simulated the normal speed for typing which is two to four characters per second [238], but we also tried with higher speeds such as six to eight characters/second in the case users copy and paste large blocks of text. We therefore varied the typing speed from one to eight characters per second.

We created five shared documents and then evaluated the delays in turn on each of these documents and for each combination of settings (number of users and typing speed). In order to further eliminate random effects on the performance achieved, for each of the shared document and for each combination of settings, we repeated the experiment four times.

We used five local computers to simulate users. Users were assigned to computers in a load balancing fashion with respect to CPU and memory of those computers [3].

The delays obtained in GoogleDocs for an increasing number of users with an average frequency of typing of two characters per second are illustrated in Figure 3.3. We obtained delays of over 15 seconds and after 35-37 users the system became unusable. For a typing speed of 4 characters per second we obtained even higher delays. What we can notice is that the performance of the system is good for up to eight or ten users and afterwards the behavior is unstable.

In Etherpad users are disconnected if the number of concurrent users is higher than 10.

We did the same measurements in MUTE, our peer-to-peer web based collaborative editor, as in GoogleDocs. The delays obtained in GoogleDocs for an increasing number of users with an average frequency of typing of 2 characters per second are illustrated in Figure 3.4. In order to compare the delays obtained in MUTE with the ones of GoogleDocs, the figure illustrates the variation with maximum 37 users. We notice that delays are up to 2 seconds.

[3]The implementation is available at `https://github.com/vinhqdang/collaborative_editing_measurement`

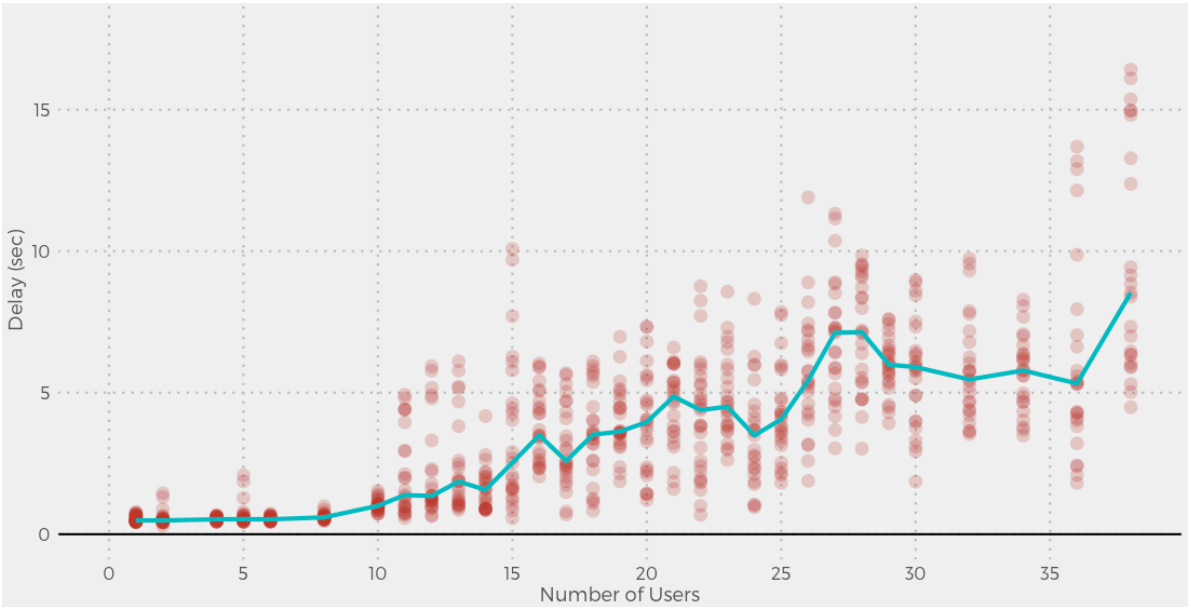


Figure 3.3: Performance of Google Docs with a typing speed of two characters / second

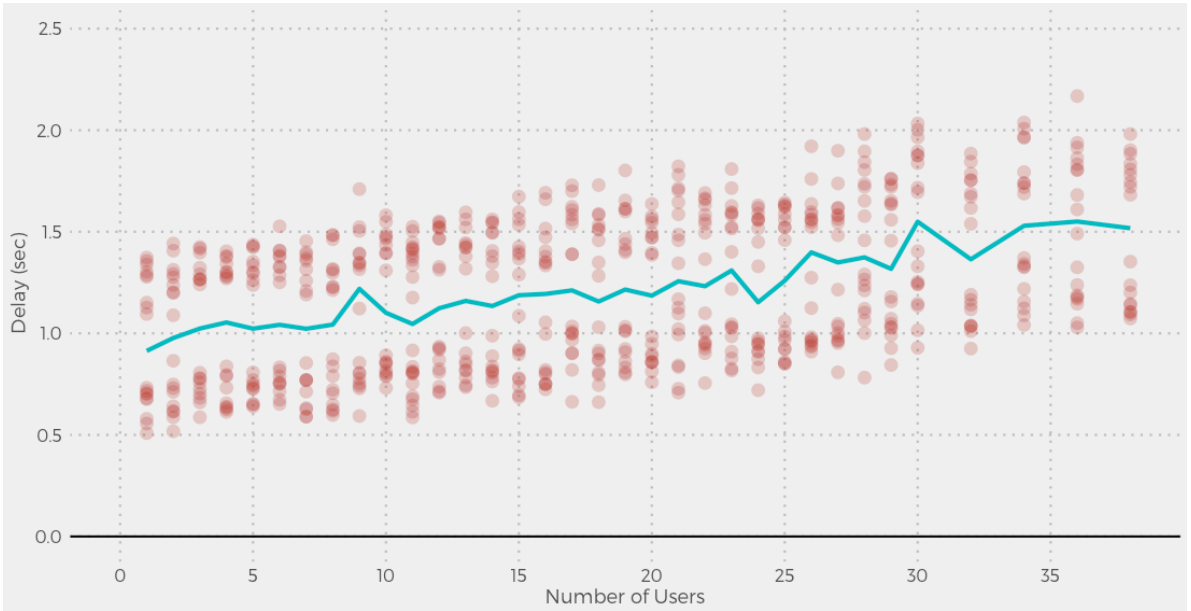


Figure 3.4: Performance of MUTE with a typing speed of two characters / second

We see that delays in MUTE are much lower than in GoogleDocs. This is also partly due to the underlying architecture. We deployed MUTE on our own server and experiments on MUTE were done locally, while GoogleDocs measurements rely on a remote server. However, what is interesting to notice is that while MUTE has a stable behavior with a very slight increase with the number of users, GoogleDocs becomes unstable and delays fastly increase if the number of users is higher than 10.

Based on the delays measured in GoogleDocs, in the next subsection I describe the effect of the delay on users.

3.4 Effect of delay on group performance: an experimental design

In the context of USCoast associated team (http://uscoast.loria.fr/) in collaboration with the Department of Psychology from Wright State University, I studied how delay influences group performance. It would have been difficult to examine the effect of delay during a real setting of collaborative note-taking during a conference. We could not use GoogleDocs for instance, because we cannot instrument it, it belongs to Google. It would have been necessary that everybody uses an editor that we can instrument. It would have been difficult to ask so many people to use that editor when they usually use GoogleDocs. So, what we did is that we mapped the real-world setting to a laboratory task that permits the systematic manipulation of delay. We simulate delay on artificial small tasks on small groups of users and we varied the delay in the range of 0 and 10 seconds.

We designed three typical collaborative editing tasks potentially sensitive to the effect of delay: proofreading, movie timeline sorting and collaborative note taking. The three tasks have different baseline time constants and hence different real time requirements. We organised user studies with 20 groups of four users that were asked to perform each collaborative task using the Etherpad collaborative editor under instructions that demanded interleaved work. Each group had to perform a sequence of three collaborative editing tasks:

- The first task was a proofreading task where we provided users with a text in French with several grammar and misspelling errors and we asked users to correct them.
- In the second task we provided users with a list of movies, we asked them to search for the release date of those movies and then sort them in an ascending order according to the release date.
- Last task was a note taking task where users were asked to listen to a short interview about cloud computing and take notes.

At the end of these tasks we asked users to fill in a survey mainly about their experience in collaborative editing and to estimate difficulty of each task. In order to perform the required tasks groups were asked to use the EtherpadLight collaborative editor and coordinate themselves by using a chat.Each group had to perform the three tasks under a constant but undeclared delay in the propagation of changes between group members. Software recorded each user's desktop activity, including task performance as well as chat for coordination. All participants received a 10 euro gift certificate at the end of the experiment.

For the analysis of the collected data we introduced outcome metrics for measuring the quality of the realised task but also process metrics for analysing user behaviour during achievement of

the tasks. Several statistical models were applied for studying the effect of delay on the various studied metrics.

We analysed the effect of delay for the sorting task in collaborative editing. In this task groups of users had to locate the release dates of an alphabetized list of movies, and sort them accordingly. We measured sorting accuracy based on the insertion sort algorithm, average time per entry, strategies (tightly coupled or loosely coupled task decomposition of the task), chat behavior and collisions between users. We found out that delay slows down participants which decrements the outcome metric of sorting accuracy. Tightly coupled task decomposition enhances outcome at minimal delay, but participants slow down with higher delays. A loosely coupled task decomposition at the beginning leaves a poorly coordinated tightly coupled sorting at the end, requiring more coordination as delay increases. More details about this study results can be found in [31].

We also analysed the effect of delay for the note-taking task where users listened 12 minutes to an interview about cloud computing and took notes during this time. After the end of the interview groups were given three additional minutes to revise their notes and generate a reconciled summary of their notes by continuing to use the collaborative editor. We divided the shared document into five sections corresponding to the five main parts of the audio interview. For each section of the document two participants were assigned the role of taking notes of the main content of the corresponding audio part. The other two participants were assigned the role of revising the notes taken by the first two participants. The roles were inverted for each section of the document. What we noticed that happened is that even by structuring the document and assigning roles, due to delay, notes about the same topic were taken two, three and even four times. What happened is that when two users want to take notes on the same topic, in the presence of delay, changes of one user are not immediately visible to the other user, so a user thinks that the other user is not taking notes, so he/she is taking the notes. In that way, finally, the notes are in double. If more than two users try to take notes simultaneously, finally the same idea will appear three or even four times.

We studied how does delay influence the quality of the final document in terms of the number of grammar errors present in the document, the amount of redundancy and the number of keywords present in the final document with respect to the transcript of the audio. We also wanted to answer to the question whether users try to adopt compensatory strategies to overcome delay by means of coordination over chat that we quantified according to the use of accord language and definite determiners. And finally we studied how do delay, experience and compensatory collaboration effort interact to affect task performance.

As dependent measures we analysed:

- *number of words* in the text base.
- *keywords* as a measure of document quality. It is computed as the number of main keywords present in the final version of the document provided by each group of users. We examined the number of keywords divided by the number of words.
- *redundancy* as another measure of document quality. It is computed as the sum of redundancies of each section in the document. Redundancy of a section was measured by analysing the recorded videos of the collaborative editing session and it represents the maximum number of occurrences in that section of any topic present in the audio.
- *error rate* as another measure of document quality. It is computed using Reverso tool that checks misspellings and grammar of a text in any language. We examined the number of errors divided by the number of words.

- *chat behavior* was studied for measuring coordination. We examined the number of words, accord language, and definite determiners.
- *survey responses*. For instance, we examined the experience of using collaborative editing of users. We divided the groups into high experienced and low experienced.

We found that the *error rate* is higher for groups that experienced a higher level of delay as shown in Figure 3.5. Redundancy is higher for groups in higher delay condition as shown in Figure 3.5 and results into an increase in the *number of words* with the delay condition as shown in Figure 3.6. We also found out that as the delay increases the *keywords* depicted by users decreases as shown in Figure 3.6. We separated the groups into high experienced and low experienced according to the data in the questionnaire. For high experienced groups *redundancy* increases with the delay, but for low experienced groups we could not see the same tendency. We also measured *chat behavior* by means of number of accord words and definite determiners which together provides a common ground knowledge which we considered as a measure of coordination. We have seen that low experienced groups used more coordination to manage *redundancy*. High experienced groups did not adjust their collaboration effort to manage *redundancy*. More details about an analysis of this task can be found in [30].

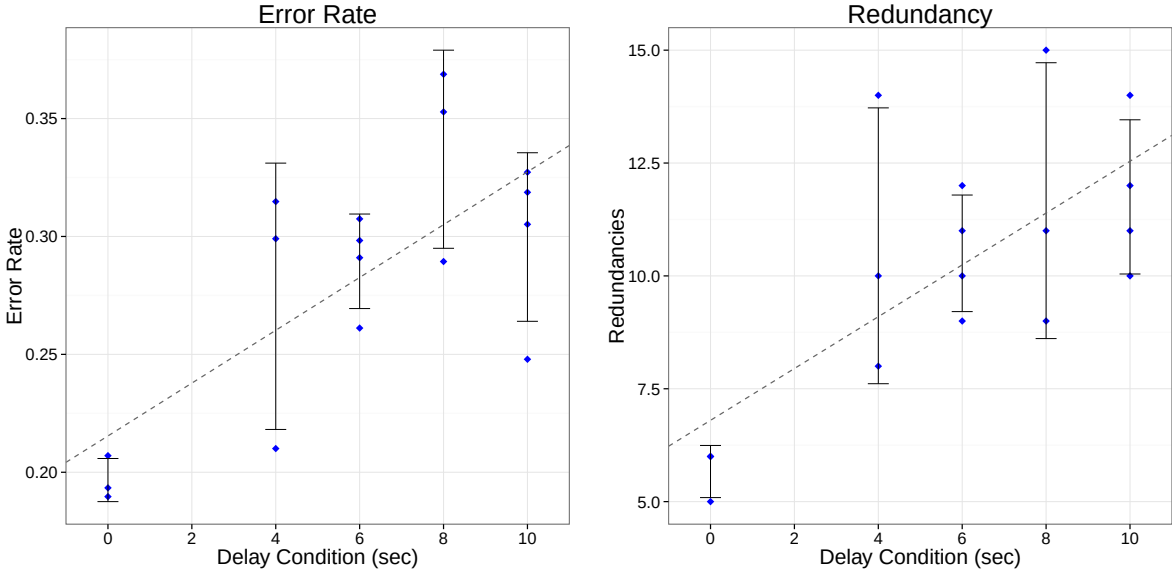


Figure 3.5: Error rate (left) and redundancy (right) as a function of delay condition

The studies we performed show that reducing delay influences the efficiency of the group and the quality of note taking. This finding is important because the choice of the underlying architecture of the collaborative editor as well as of underlying synchronisation mechanisms has an impact on the delay. If delays cannot be reduced by the choice of the architecture and synchronisation, an awareness approach should be provided to users such that they can compensate for the delay by coordination strategies.

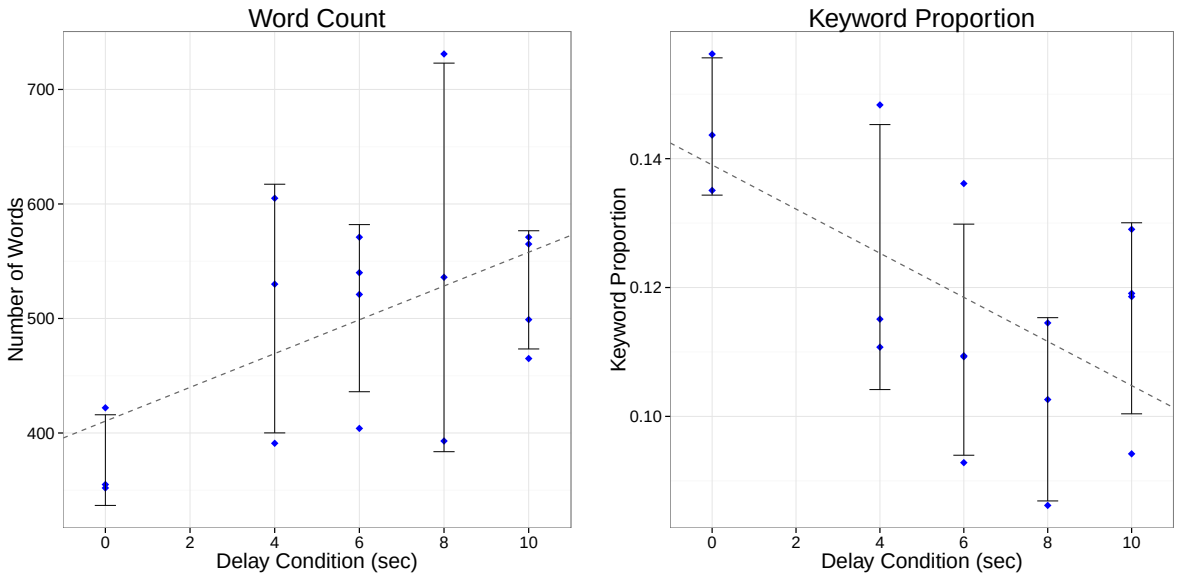


Figure 3.6: Number of words (left) and proportion of keywords (right) as a function of delay condition

4

CRDTs for ordered sequences

Existing replication mechanisms for collaborative editing consider that the shared content is made of a sequence of elements whose granularity is fixed. For instance a text content is generally seen as a sequence of characters or a sequence of lines. A coarse granularity of changes allows to keep low the overhead generated by additional structural metadata that maintains ordering between elements. But when the content of such an element is updated, this element has to be deleted and a new one is inserted. This can lead to duplicated content when two concurrent updates are done on the same element. In this case additional user actions are required to merge this duplication. On the other hand, a finer granularity of changes allows to compute finer merging of concurrent changes but implies a higher overhead. Additionally, the more elements the document is made of, the more computation is needed to apply and merge user changes. In collaboration with Gérald Oster, Stéphane Martin and Luc André, I designed a new replication mechanism for elements with variable granularity called LogootSplit. Our solution is based on a CRDT approach.

CRDT-based approaches require that atomic elements composing the data structure have to be uniquely identified. The main limitation of CRDT algorithms is that they suffer from meta-data overhead required for the update of the data structure. This metadata overhead can be excessive for real-time collaboration where the communication granularity can be as small such as a single character.

LogootSplit approach considerably reduces the size of the metadata. In addition to the standard operations of insertion and deletions of elements, the model underlying our solution consists of a split operation of elements into smaller elements that is atomically executed, i.e. without the need of executing a deletion of the original element followed by the insertion of the two smaller elements. It is therefore possible to initially create long size elements that can be fragmented in order to insert new updates.

LogootSplit is an extension of Logoot algorithm. Logoot associates each character with a unique identifier which remains constant during the lifetime of the document. Identifiers form a dense total order consistent with the element order. A partial order $<$ on a set X is said to be dense if, for all x and y in X for which $x < y$, there is a z in X such that $x < z < y$. To insert a new character at a precise position in the document the character is inserted with a new suitable identifier created on purpose. To remove an existing character the corresponding character identified has to be located and then the related character is removed with its associated identifier. In this algorithm, each identifier is a list of triples of integers of the form $< p, s, h >$. The first integer p is a priority number, the second one s references the user who generated the element – called the unique site identifier –, and the third one is equal to the

value of the user's logical clock h when this element was created – the clock is incremented each time an operation is generated –. The priority number is used to sort the characters, the site identifier to break ties if two users generated both an element with the same priority, and the clock value, together with the site identifier, ensure that triples are unique. If it is not possible to create a new triple between two others (for instance to insert a character between two existing characters), the new identifier will contain additional triples.

In Figure 4.1 we illustrate an example document with the content *concurency contrl* where each character has associated an identifier. We suppose that the user at site 1 having the logical clock 8 corrects the misspelling of the word *concurency* by adding the character r between the characters r identified by the identifier *<3,2,5>* and e identified by the identifier *<4,1,7>*. We suppose the new generated identifier assigned to r is equal to *<3,2,5><13,1,8>* which is between *<3,2,5>* and *<4,1,7>*. We suppose that the user at site 2 having the logical clock 10 corrects the misspelling of the word *contrl* by adding the character o between the characters r identified by *<12,3,1><7,8,2><13,3,6>* and l identified by *<12,3,1><7,8,2><14,3,7>*. We suppose that the new character o is identified by *<12,3,1><7,8,2><13,3,6><7,2,10>* which is between *<12,3,1><7,8,2><13,3,6>* and *<12,3,1><7,8,2><14,3,7>*.

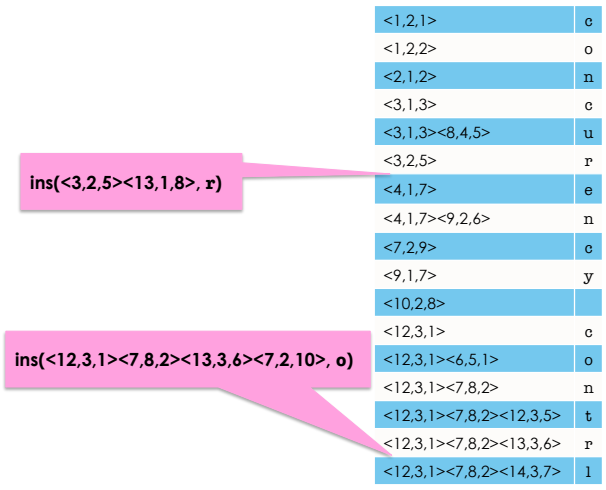


Figure 4.1: Logoot identifiers

The main issue of Logoot is that a huge amount of insertions in the same part of the document might lead to identifiers formed by very large lists of triples that are memory costly [36].

Coarse-grained data leads to the possibility of conflicting updates while fine-grained data requires more metadata. LogootSplit offers a solution for handling an adaptable granularity for shared data that overcomes the limitations of fixed-grained data approaches. LogootSplit deals with three types of operations: insertion which adds an element, deletion which deletes an element and split which splits an element. Elements consist of sequences of characters, each sequence of characters having associated unique identifiers that form a dense total order consistent with the order of the sequences of characters.

In what follows I shortly describe the LogootSplit approach including the description of identifiers for string elements and the execution of insertion and deletion operations on string elements. I also show the correctness of LogootSplit and describe its different implementations.

4.1 Identifiers

In LogootSplit, identifiers of an element are composed of two parts. The first part called *Base* is a sequence of integers representing the global identifier of the element. The last two integers of the *Base* correspond to the *site identifier* that generated the identifier and the *logical clock* of that site. The logical clock is incremented before the generation of an identifier. The second part of an identifier called *Interval* is composed of two integers representing the *begin* and *end* offsets of the element. Figure 4.2 illustrates the structure of a LogootSplit identifier. The integers composing LogootSplit identifiers have to belong to the interval (min, Max), where $min < 0 < Max$.

For comparing two identifiers we consider the sequence of integers composing the *Base* followed by the offset of the *Interval* of each identifier that we compare in lexicographical order. For instance $1, 1[0, 1] < 1, 2$ and $1, 1[0] < 1, 1, 0, 1, 3 < 1, 1[1]$.

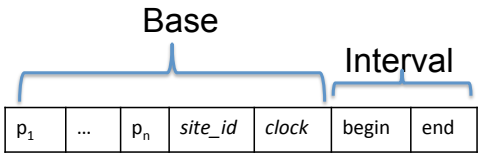


Figure 4.2: LogootSplit identifier

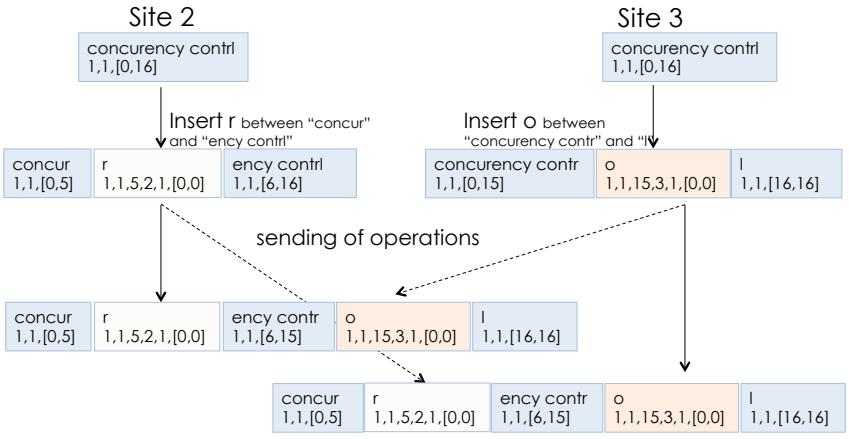


Figure 4.3: Concurrent insert operations in LogootSplit

4.2 Insertion operation

A user can generate an insertion from the view model. If the insertion position is between two elements, we generate an identifier greater than the *end* of the left element and lower than the *begin* of the right element. The *interval* corresponding to this identifier will be $[0, k-1]$ where k is the length of the string to be inserted. If $k - 1 > Max$ then we generate an element for the first Max characters of the string and then we execute recursively another insertion operation for the rest of the string.

If the insertion position is inside an element, we have to split the element. The two elements corresponding to the result of the split will have the same *base* as the original element. Supposing

that the *interval* of the original element is $[a, b]$ and that the insertion position is k inside the original element, the two elements corresponding to the result of the split will have the two intervals equal to $[a, a + k]$ and $[a + k + 1, b]$ respectively. An identifier for the new element will be generated to be greater than the *end* of the left element and lower than the *begin* of the right element.

The identifier of the new inserted element is then sent to remote sites which will execute the insertion.

Figure 4.3 shows the case when two sites concurrently execute insertions that split the same element. The two sites $Site_2$ and $Site_3$ share the same document with the content "concurency contrl" which was inserted by $Site_1$ which had the logical clock 1. The identifier of this 17-characters long string is $1, 1, [0, 16]$.

$Site_2$ with the logical clock 1 inserts "r" between "concur" and "ency contrl". The original element is split into two elements "concur" with the identifier $1, 1, [0, 5]$ and "ency contrl" with the identifier $1, 1, [6, 16]$. We see that the two split elements keep the same base as the original element and the original interval $[0, 16]$ is split into $[0, 5]$ and $[6, 16]$ according to the split position. The new generated element "r" will have the identifier $1, 1, 5, 2, 1, [0, 0]$. The base of this identifier is created by concatenating the base of the original element "1,1", the last element of the interval of the left split element "5", the site that generated the insert "2" and the clock "1". The interval of the newly created identifier is "[0,0]".

$Site_3$ with the logical clock 1 inserts "o" between "concurency contr" and "l". The original element is split into two elements "concurency contr" with the identifier $1, 1, [0, 15]$ and "l" with the identifier $1, 1, [16, 16]$. The new generated element "o" will have the identifier $1, 1, 15, 3, 1, [0, 0]$. The base of this identifier is created by concatenating the base of the original element "1,1", the last element of the interval of the left split element "15", the site that generated the insert "3" and the clock "1". The interval of the newly created identifier is "[0,0]".

When the insert of $Site_3$ of "o" with the identifier $1, 1, 15, 3, 1, [0, 0]$ is received by $Site_2$, the split between "ency contr" and "l" will be generated. Similarly, when the insert of "r" with the identifier $1, 1, 5, 2, 1, [0, 0]$ is received by $Site_3$, the split between "concur" and "ency contr" will be generated. The data structures at $Site_2$ and $Site_3$ will converge after the integration of all operations.

4.3 Deletion operation

When a deletion is issued we have to identify those elements or parts of the elements that are concerned between two positions in the document. A deletion is generated for each of these elements or parts of these elements. In the case that only a part of an element is deleted one or two splits might be generated. In the case that the part to be deleted is located at the begin or end of the element, two identifiers are generated, the one corresponding to the part that has to be deleted and the other one to the part that has to be preserved in the model. In the case that the part to be deleted is in the middle of the element, two splits are executed and three identifiers are generated, the identifier in the middle corresponding to the part that has to be deleted and the other two corresponding to the parts that have to be preserved in the model.

The identifiers of the parts that have to be deleted are sent to the remote sites that execute a deletion for each of the received identifiers. In the case that the original delete generated a split or the remote site executed some concurrent operations that generated a split, the remote site might not contain the identifier of the element to be deleted. Therefore, upon reception of a deletion, a remote site looks in their local model for the elements that have the same base as

the received identifiers, splits them if necessary and deletes them.

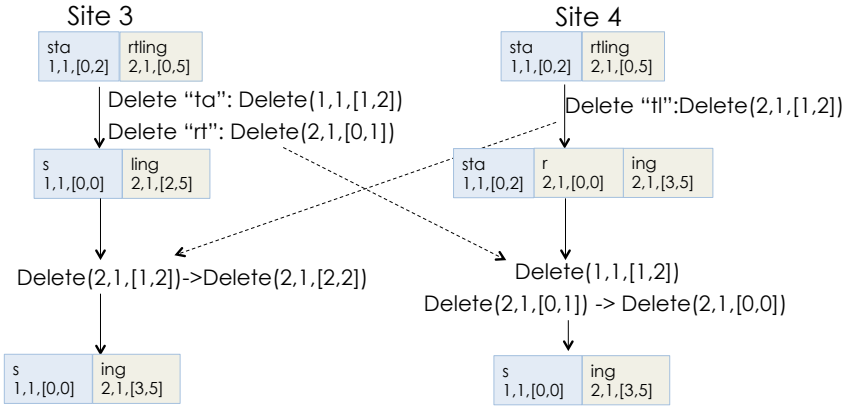


Figure 4.4: Concurrent delete operations in LogootSplit

Figure 4.4 shows an example of concurrent delete operations executed by $Site_3$ and $Site_4$. These two sites share the same document with the text "startling" consisting of two blocks: the first block with the identifier $1, 1, [0, 2]$ contains the text "sta" which was inserted by $Site_1$ which had the logical clock 1 and the second block with the identifier $2, 1, [0, 5]$ contains the text "rtling" which was inserted by $Site_2$ which had the logical clock 1.

$Site_3$ generates the deletion of "ta" from the first block corresponding to the operation $Delete(1, 1, [1, 2])$. The remaining content of the first block is "s" with the associated identifier $1, 1, [0, 0]$. Afterwards, $Site_3$ generates the deletion of "rt" from the second block corresponding to the operation $Delete(2, 1, [0, 1])$. The content of the second block becomes "ling" with the associated identifier $2, 1, [2, 5]$.

$Site_4$ generates the deletion of "tl" from the second block corresponding to the operation $Delete(2, 1, [1, 2])$. This deletion operation splits the second block into two blocks: "r" with the associated identifier $2, 1, [0, 0]$ and "ing" with the associated identifier $2, 1, [3, 5]$.

When operation $Delete(2, 1, [1, 2])$ is received by $Site_3$, the identifier of the delete operation does not exist anymore in the model as $Site_3$ already performed a deletion in an overlapping range of the block. The interval of the identifier of delete is adjusted in order to take into account the local concurrent delete. The locally executed operation is therefore $Delete(2, 1, [2, 2])$.

When operations $Delete(1, 1, [1, 2])$ and $Delete(2, 1, [0, 1])$ are received by $Site_4$, $Delete(1, 1, [1, 2])$ is executed as it is as no local concurrent operation was executed in the first block. The identifier of the second delete operation $2, 1, [0, 1]$ does not exist anymore at $Site_4$, as $Site_4$ already performed a deletion in an overlapping range of the block. The interval of the identifier of delete is adjusted in order to take into account the local concurrent delete, the operation becoming $Delete(2, 1, [0, 0])$.

The algorithms describing the generation of the identifiers, the execution of local and remote insertion and deletion operations are described in [32].

4.4 Correctness

LogootSplit algorithm respects the criteria of the CCI model: causality, consistency and intention preservation [222].

Regarding causality criterion, LogootSplit does not require causal diffusion of operations.

Due to the reception in any order of the operations, the deletion of an element can be received before the operation of creation of that element. LogootSplit has to distinguish between the cases of two concurrent deletions of an overlapping part of an element and the deletion of a part of an element whose insertion was not received. Both cases are characterised by the fact that when a deletion is executed, the part of the element that has to be deleted cannot be found. If we would consider that this corresponds to the first case (overlapping deletes), in the second case (deletion received before insertion) the insertion is executed afterwards, eliminating completely the effect of delete. In order to deal with these cases we keep a table containing the bases that were completely deleted from the model, i.e. there is no element in the model containing these bases. This table is browsed when a deletion does not lead to any modification in order to check whether the element was deleted or the insertion was not yet received. If it is the second case we can store the deletion in a table with pending deletions that can be checked for integration at regular times.

Regarding consistency and intention preservation criteria, LogootSplit ensures the following properties.

- **Convergence:** *When no new updates are generated and all messages (operations) have been delivered to all users' sites, user document copies have the same state.*

  Identifiers are unique, the *base* component of an identifier being composed of a unique site identifier and the number of operations generated at that site. Moreover, identifiers are totally ordered. Therefore, when all operations are received by all sites, the document copies are identical being composed by the same sequence of elements ordered by their corresponding identifiers.

- **Intention preservation of character insertions:** Each character inserted between two other characters in the document viewed by a user, needs to keep its relative position between its neighbors during the editing process.

  An identifier is uniquely constructed between two neighbor identifiers. Identifiers are sorted in a total order and are never modified during the editing process.

- **Intention preservation of string insertions:** Two concurrent string insertions at the same position lead to one string followed by another, the same order being respected on document copies.

  Two concurrent insertions at the same place have different bases as one of the component of the base is the site identifier. If the base of the first string is smaller than the one of the second, any character of the first string has a smaller identifier than any character of the second, hence the characters are not mixed up.

4.5 Implementation

We proposed three data structures that can be used to implement LogootSplit: a naive version, a string-based version and a tree-based version. Each of these data structures has its own benefits and drawbacks.

4.5.1 Naive representation (LogootSplitNaive)

In the naive implementation LogootSplitNaive, the data model is an ordered array of blocks, each block containing the element string and its identifiers, the blocks being ordered according

to the identifiers.

When a local operation is generated, the search function of an element based on the position in the view needs to sum up the lengths of all blocks from the beginning of the list until reaching the searched position. The cost of this function is $\mathcal{O}(l)$, l being the number of blocks in the list.

When remote operations are executed, in order to search elements in the model such as in the case of a remote delete and to search the position of insertion of a new element in the case of a remote insertion, a dichotomic search can be used based on the element identifier. The comparison between two identifiers requires a comparison of all integers composing the identifiers. The search function for the identifier of an element in the array has therefore a cost of $\mathcal{O}(i \times \log(l))$ where i is the identifier length and $\log(l)$ is the complexity of a dichotomic search with l blocks.

4.5.2 String-based representation (LogootSplitString)

The string-based data structure stores the characters of the final string in an array. Each character in this array is associated to the identifier base and to its position in the identifier interval. Character positions in the identifier interval are stored in an array while identifier bases are stored in a hashtable.

Figure 4.5 illustrates the string-based representation for the scenario described in Figure 4.3. The model represents the following five elements: "concur" with the identifier 1,1,[0,5], "r" with the identifier 1,1,5,2,1,[0,0], "ency contr" with the identifier 1,1,[6,15], "o" with the identifier 1,1,15,3,1,[0,0] and "l" with the identifier 1,1,[16,16]. Each character in the upper array is linked to its initial position in the identifier interval (the lower array) and to the base of its identifier.

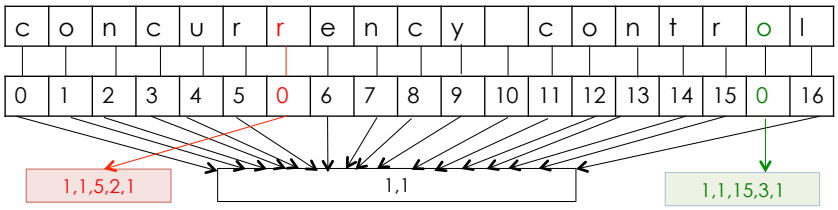


Figure 4.5: LogootSplitString representation

The search function based on the position is executed in constant time as it directly returns the element in the array at the required position. The search function based on identifier is similar to that of the naive representation and has a complexity of $\mathcal{O}(i \times \log(n))$ where i is the identifier length and $\log(n)$ is the complexity of a dichotomic search with n characters.

4.5.3 Tree-based representation (LogootSplitAVL)

The tree-based representation uses AVL [275], a self-balancing binary search tree. This data structure behaves similarly to the naive implementation but each block is now organized in an AVL tree rather than an array. Elements are stored in the nodes of the binary tree, the left subtree of a node contains the elements situated before the node (corresponding to smaller identifiers) while its right subtree contains the elements situated after the node (corresponding to higher identifiers).

The identifiers of the elements are represented as in the LogootSplitString implementation, i.e. the bases are stored in a hashtable and a node contains a string of characters, a link to an identifier base and the start position of the interval. Each node also contains the height of

the subtree used to balance the tree and the size of the string contained by this subtree. The size of the left subtree represents the position of the subsequence in the text. When the structure of the tree is modified rotations are executed in order to keep the tree balanced.

Figure 4.6 shows the evolution of the AVL tree for the scenario described in Figure 4.3. Figure 4.6 (a) shows the initial document structure. Figure 4.6 (b) shows the tree structure after the insertion of "r" between "concur" and "ency contrl" and after rotation of the tree in order to make it balanced. Figure 4.6 (c) shows the tree structure after the insertion of "o" between "ency contr" and "l" and after rotation.

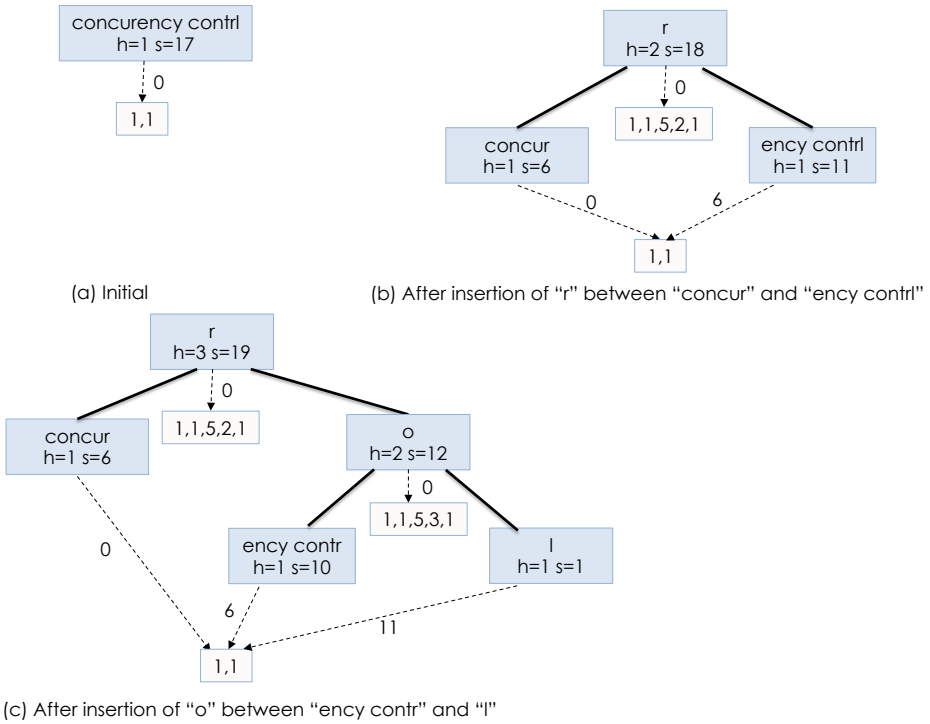


Figure 4.6: LogootSplitAVL representation

The search function based on position has a complexity of $\mathcal{O}(\log(l))$ where l is the number of blocks. The search relies on the size of a tree node which is contained in each node. Suppose we look for position p. We start from the root and we search the size of the left tree s_l. If $s_l > p$, we recursively continue the search in the left tree. Otherwise we update p with $p - s_l$ and we compare the updated value of p with the size of the string contained in the current node s_n. If $p < s_n$ then the search position is inside the node. Otherwise we update p with $p - s_n$ and we recursively continue the search on the right child. The complexity is logarithmic as the tree is balanced.

The search function based on identifier is performed in a similar manner. Starting from the root we compare the searched identifier with the node identifier. If the searched identifier is less than the identifier of the first character in the node, we continue the search with the left child. If the searched identifier is greater than the identifier of the last character in the node, we continue the search with the right child. Otherwise, the identifier is inside the node. The search is done in a logarithmic time, for which we have to consider the cost of identifier comparisons, leading to a complexity of $\mathcal{O}(i \times \log(l))$, where i is the size of an identifier.

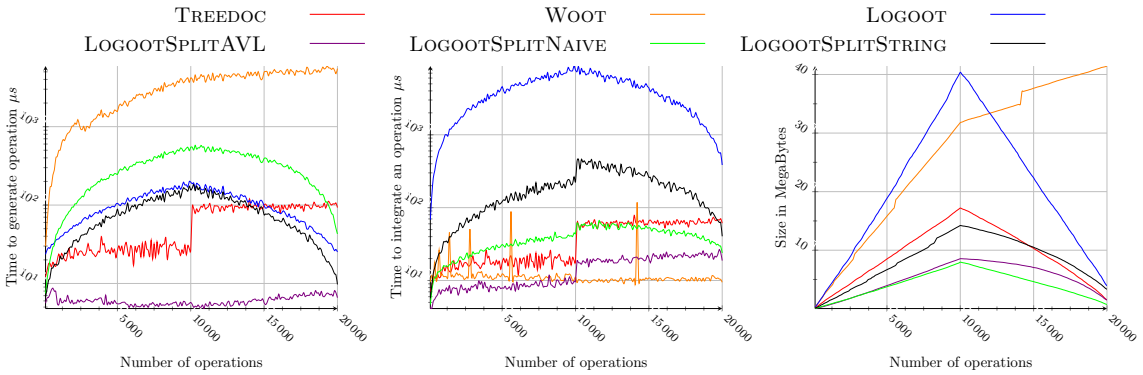
4.5.4 Evaluation

Table 4.1 presents the average-case time complexity of the different implementations of LogootSplit algorithms as well as three main CRDT algorithms working at character level Logoot, Treedoc and WOOT family of algorithms for which we selected WOOTH. We denoted by n the size of document, l the number of blocks, d the number of deleted elements, s the string added or deleted, f the number of times a block was split and i the size of an identifier. In the worst case the number of blocks l is equal to the size of document n. This case happens when users created a document by inserting the text content character by character.

Table 4.1: Summary of algorithms time complexity

Algorithm name	**Search**		**Insertion**		**Deletion**	
	of a Position	of an Identifier	Local	Remote	Local	Remote
LogootSplitNaive	l	$i \times \log(l)$	$l + i$	$i \times \log(l) \times f$	$l + f$	$i \times \log(l) \times f$
LogootSplitString	1	$i \times \log(n)$	$s + i$	$i \times \log(n) \times f + s$	f	$i \times \log(n) \times f$
LogootSplitAVL	$\log(l)$	$i \times \log(l)$	$\log(l) + i$	$i \times \log(l) \times f$	$f + \log(l)$	$i \times \log(l) \times f$
Logoot	1	$i \times \log(n)$	$s \times i$	$i \times \log(n) \times s$	s	$i \times \log(n) \times s$
Treedoc	i	i	$s \times i$	$s \times i$	$i \times s$	$i \times s$
Woot	$n + d$	1	$n + d + s$	s	$n + d + s$	s

l: # of blocks, n: size of the document, i: the size of an identifier, d: # of deleted elements, s: size of the string to inserted/deleted, f: # of times a block was split



(a) Generation time (b) Remote integration time (c) Memory occupation

Figure 4.7: Experimentation results for random trace

We evaluated performances of different implementations of LogootSplit as well of Logoot, Treedoc and WOOT using JBenchmarker [4], a Java environment for the implementation, evaluation and comparison of different collaborative editing algorithms.

We measured the time of operation generation (when a user modifies the document), the integration time (when another user receives a modification) and the memory occupation. These measurements were done by executing several randomly generated traces of collaboration containing insert and delete operations. Figure 4.7 shows the results we obtained with a collab-

[4] https://github.com/score-team/replication-benchmarker

oration trace containing insert and delete operations of an average of 50 characters generated randomly inside the document. The trace contains 20000 operations, the first 10000 operations being mostly insertions (80%) and the second 10000 operations being mostly deletions (80%). As it can be seen from figure 4.7, our algorithm LogootSplitAVL has the best overall performances.

More discussions on the complexity analysis and performance evaluation of our algorithms can be found in [32].

5

Conflicts prevention and resolution

Group awareness refers to information which group members obtain about the other group members, group processes, and mutually employed objects with the goal of carrying out a certain task efficiently. In my research work I mostly focused on workspace awareness, a specific category of group awareness. I studied how to provide awareness about the states of the shared documents in various contexts. I was interested in what information should be provided to users to prevent conflicting changes and to understand divergence when conflicts cannot be avoided.

In collaboration with Gérald Oster, Moira Norrie and Stavroula Papadopoulou, I proposed an awareness mechanism for improving collaborative work in the context of software engineering [39, 42] and collaborative writing of textual documents [43]. In version control systems such as Git and Subversion frequently used for code development or writing of textual documents, users can work simultaneously on their shared documents and publish their changes at a later time. Unfortunately, users are not informed about concurrent changes while they work in their local workspaces. Our proposed awareness mechanism [39, 42] offers users the possibility to work in isolation but being informed in real-time by means of annotations about changes performed by other users. Users can continue working without the need of integrating concurrent changes which might lead to code that does not compile. Concurrent changes are localized by taking into account the structure of these documents (classes, methods, lines of the source code). For localizing and providing the content of concurrent changes, we used the operational transformation mechanism. The same approach of providing real-time awareness while users work in isolation was applied for the collaborative editing of textual documents that conform to a hierarchical structure. In this context we studied the trade-off between awareness and user privacy [43]. Users can filter details about their changes transmitted to other users according to their preferences. In section 5.1 I present this work on conflicts prevention by localisation of concurrent changes and taking into account user privacy preferences.

In the context of the PhD thesis of Hoai-Le Nguyen I studied conflicts management and resolution in open-source projects developed using Git version control systems. We studied concurrency and conflicts of four Git projects: Rails, IkiWiki, Samba and Linux Kernel. We analysed the collaboration process of these projects at specific periods revealing how change integration and conflict rates vary during the project development life-cycle. Our study suggests that developers should use more intensively awareness mechanisms close to release dates where changes integration rate is higher. We also studied the mechanism adopted by Git to consider concurrent changes made on two adjacent lines as conflicting. Based on the high rate of false positives of this mechanism, our study suggests that Git should reconsider signalling adjacent line conflicts inside the source code files [12]. In section 5.2 I present an overview of our study

on conflicts management in Git-based open source software.

5.1 Conflicts prevention

In this section I present my main contributions on preventing conflicting changes in collaborative work by making users aware of the localisation of concurrent changes. I also present how our solution on the localisation of concurrent changes can be adapted to respect user privacy where details about changes transmitted to other users can be filtered out according to user preferences.

5.1.1 Localisation of concurrent changes

In this subsection I describe the awareness mechanism we proposed in the context of centralised version control systems such as Subversion and CVS where users are made aware of the localisation of concurrent changes.

The main assumption of our novel awareness mechanism is that users are connected most of the time, even when they work in isolation. Since nowadays network connectivity is provided almost everywhere and it will continuously expand in the near future, our assumption seems feasible. However, we can support disconnected work, but without providing any awareness mechanism.

First I illustrate by means of an example our proposed annotation mechanism. I consider a very simple example involving two software engineers that collaborate on the source code of the same project stored on a central repository. The modifications they perform will overlap on the code of some common classes. Suppose that the first developer decides to remove the method isReal() from the class Integer illustrated in Figure 5.1.

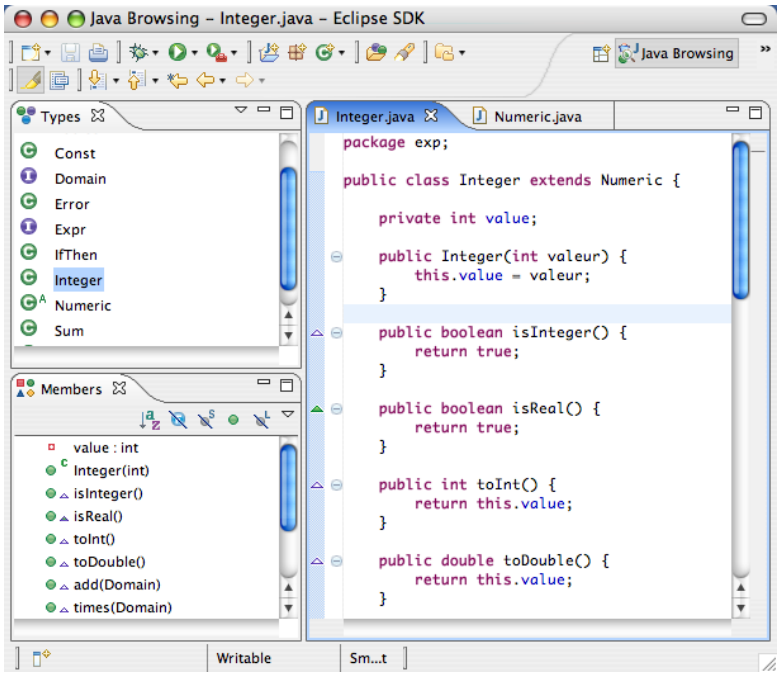


Figure 5.1: Initial document state

Concurrently, the second developer that uses the functionality of class Integer realises that the method isReal() should be corrected - it should return false as an integer should not be

considered to be a real.

I show next what awareness information is provided to the two developers. The first user who deletes the method isReal() will receive the changes of the second user and will be provided with the awareness information presented in Figure 5.2. By means of a marker the user will be informed that the class Integer is concurrently modified as shown on the top left hand side window of the interface. In the right hand side window of the interface an annotation marker will indicate that a line was concurrently modified by another user. The user can consult details about the concurrent changes and he will be provided with a window where the difference between his changes and the concurrent changes performed on the document is presented. The user can see that the method locally deleted was concurrently modified by another user. In this manner, the user can decide to contact the other user in order to discuss about the resolution of this conflict.

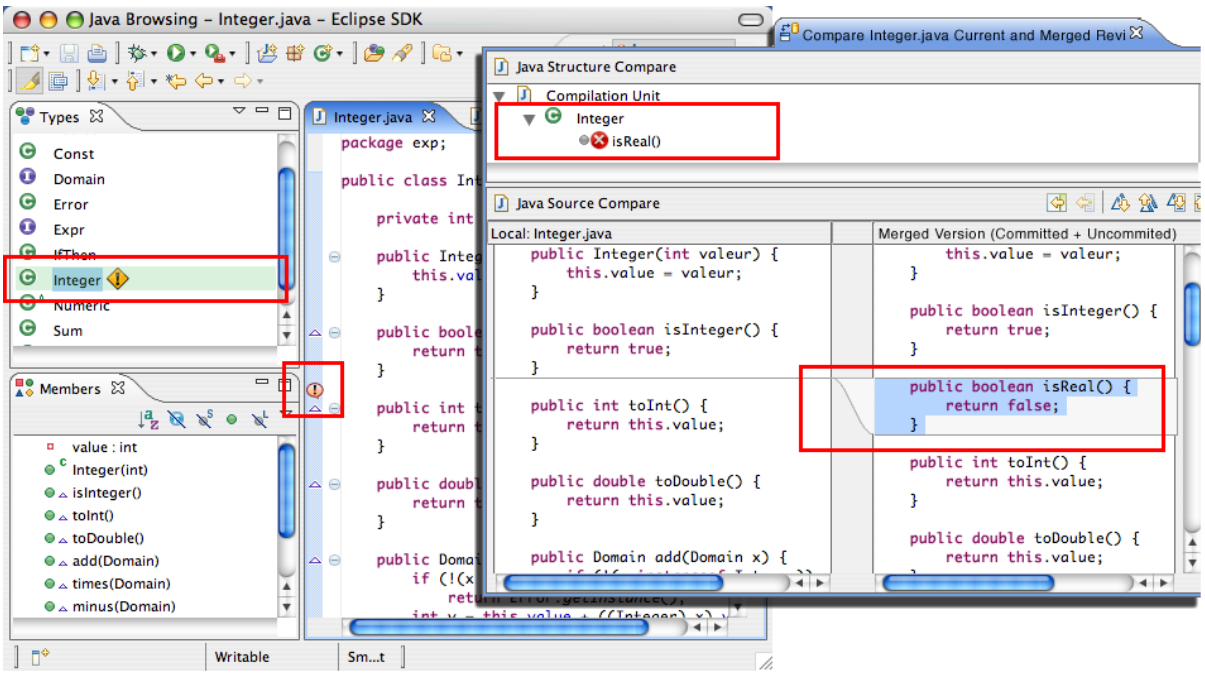


Figure 5.2: Interface at the first site after reception of concurrent operations

I show next what happens at the site of the second user that modifies the method isReal(). This user will receive the changes performed by the first user and will be provided with the awareness information presented in Figure 5.3. In the right hand side window the user will be notified that the method isReal() is deleted by annotating the lines composing this method. The left hand side windows displaying the class hierarchy and the methods belonging to class Integer will highlight the fact that class Integer was concurrently modified and method isReal() was deleted.

We integrated our awareness mechanism into an operation-based version control system that we built ourselves. The repository represents a document version V_i by storing the set of operations representing the difference between V_{i-1} and V_i. The initial version V_0 is represented by the initial document state.

We designed the three basic methods supported by a version control system, i.e. checkout, commit and update and the annotation mechanism. In the checkout phase, a request is sent to the repository to specify the version of the document that is intended to be checked out.

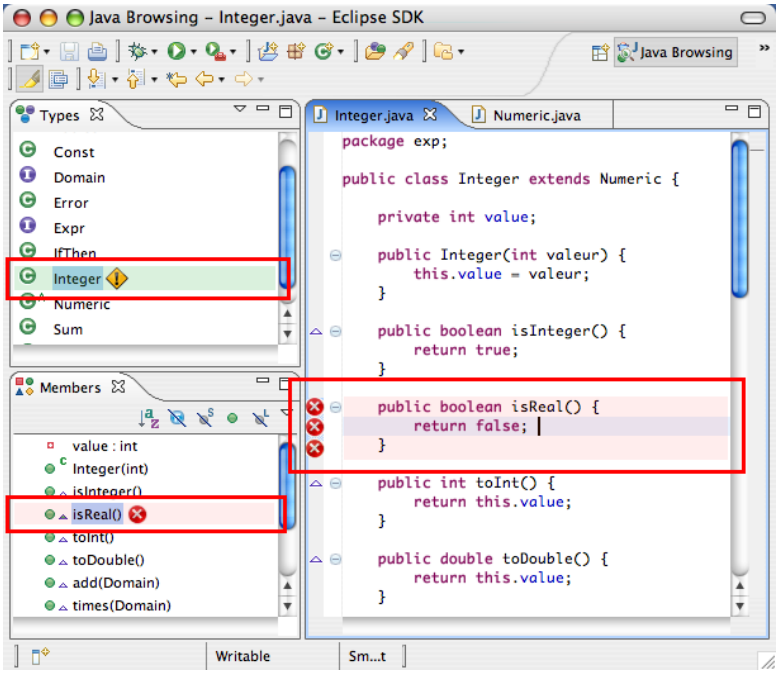


Figure 5.3: Interface at the second site after reception of concurrent operations

The repository sends to the client the initial version of the document and the set of operations representing the difference between the requested document version and the initial document version. The client then executes the received list of operations on the received initial document state. It also sets the base document version to the requested version.

In the commit phase, a check is first performed as to whether the user can commit the changes to the repository. If the base version of the document in the local workspace is equal to the last version in the repository, a commit can be performed. Otherwise, an update is necessary before committing the changes. In the case that a commit is allowed, the repository simply stores the operations that were performed in the local workspace.

In the update phase, the repository sends to the local workspace a list of operations representing the delta between the base version in the local workspace and the latest version in the repository. Upon receiving the list of operations from the repository, the local workspace performs the SOCT4 [215] merging algorithm to update the local version of the document. Details about the algorithms implementing checkout, commit and update phases can be found in [39, 42].

In what follows I describe the basic principles underlying our annotation mechanism.

In a version control system users can simultaneously work on different versions of the shared project. For instance, Figure 5.4 shows that $User_1$ performs changes on version V_2, $User_2$ on version V_{n-2} and $User_m$ on version V_n.

Each time a user commits changes to the repository, the repository will inform the other users about the committed changes and user documents will be annotated with these changes. Uncommitted changes locally performed by users will be periodically sent directly to the other users and their documents accordingly annotated. Different types of markers will be used for annotation of committed and uncommitted changes.

Users can continue working without integrating committed and uncommitted changes, being informed by means of annotations about the concurrent changes performed. An annotation

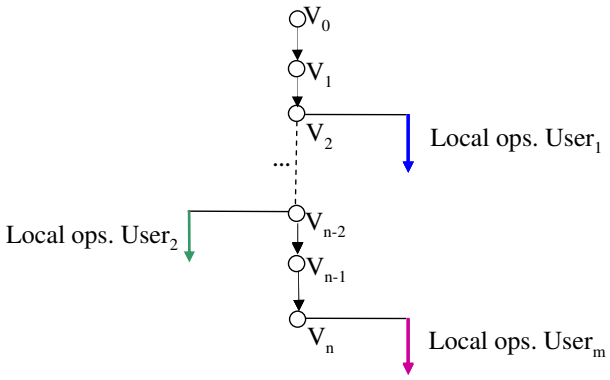


Figure 5.4: Concurrent editing

operation is a remote committed or uncommitted operation not yet integrated in the local document, but that is used to annotate the document. Since execution effect of an annotation operation does not change the document state, local operations performed in the workspace do not have to take into account its execution. However, for computing the proper localisation of annotation operations in the current document, they have to take into account concurrent executions of local operations.

In order to manage annotations, each user workspace maintains the following data structure (*LL*, *RCL*, *RUL*, *bv*):

- *LL* is the local log of operations containing the operations executed in the local workspace.
- *RCL* is the remote log containing the committed and non-updated operations from the repository that are used in the annotation process of the local document.
- *RUL*[*i*] is the remote log containing the non-committed operations performed by $User_i$ that are used in the annotation process on the local document.
- *bv* is the identifier of the base version of the local workspace, i.e. the latest version in the repository that was integrated in the local workspace.

Operations in *RCL* and in *RUL* have to be transformed against the operations in *LL* in order to annotate the local version of the document. Transformations can be performed only if operations are defined on the same state.

We defined an operation as *O*(*type*, *position*, *content*, *bv*, *uid*, *sid*, *committed*) where

- *type* is the type of operation
- *position* is the position in the document where the operation is applied
- *content* is the content of the operation
- *bv* is the base version of the local document
- *uid* is the identifier of the user that generated the operation
- *sid* is the sequence identifier of the operation, i.e. the number of operations including the operation itself that were generated by user *uid* starting from version *bv*

- *committed* is a boolean indicating if the operation is committed or not.

These operation parameters allow to establish the relations of causality and concurrency among operations, as well as causally readiness of an operation or a set of operations at their execution at a certain site.

In order to capture changes at a low granularity level such as the character, we modeled the document as a sequence of characters and we represent changes performed on the document by means of the following two types of operations:

- *insert(p,c)* - inserts character c at position p
- *delete(p)* - deletes the character at position p

The content of an *insert* operation is the character that was inserted. In the case of a *delete* operation the content is not specified, it can be determined from the specified position.

The linear structure of the document can be mapped to the structure of a source code document composed of packages, classes, methods and lines of code. Therefore, if a change was performed at a certain character of the document, we can annotate that a change was performed at the corresponding line of code, method, class or package.

A local operation has an associated base version at the moment of its generation. When the local version of the document is updated with new changes from the repository, the local operations will have associated the new base version of the document.

The algorithms underlying the annotation of concurrent committed and uncommitted operations are described in [39, 42].

5.1.2 Trade-off between awareness and privacy

Under the assumption that a user is continuously connected, it is possible that she receives in real-time non-committed parallel modifications in order to annotate his local copy of the document. This presumes that users agree to send in real-time their local non-committed operations. Unfortunately, this assumption might violate user privacy as users may not agree to send draft changes of their work.

There is a trade-off between privacy and the usefulness of awareness: if users agree to have less privacy, other group members are provided with rich awareness.

We provided a filtering mechanism that deals with this trade-off. We filter non-committed local operations before sending them to other users by masking some operation parameters. We call these operations *ghost operations*.

A ghost operation is the result operation obtained by filtering an original operation according to user privacy preferences. Given that an original operation is defined by its type and a list of parameters as *operation*=<*type*,(*parameter*)*>, the ghost operation g(*operation*)=<*filter*(*type*), (*filter*(*parameter*))*> is obtained by filtering the type and the parameters of the original operation according to user privacy preferences.

The ghost operation might not contain a parameter belonging to the real operation, it might contain it in the original form or it might filter it. A filtered parameter is formed by the filtered name, the filtered type and the filtered value of the original parameter.

Let me consider the coding example described in the previous subsection and illustrated in Figure 5.1. First user removes the method isReal() by generating the operation op_1=*delete*(*User*$_1$, *Integer.java*, 15–18).

This operation removes the lines 15 to 18 describing the definition of the method isReal() from the file *Integer.java*. The second user modifies method isReal() by updating the content of the line 16 with the content return false;. Therefore, operation op_2=*update*(*User*$_2$, *Integer.java*, 16, "return false;") is generated.

Suppose that the two users decide to send ghost operations describing their activity while working in isolation. The first user decides to apply the privacy policy allowing to send the full content of his modifications as ghost operations. Therefore, he sends the following ghost operation: $g(op_1)$=*delete*(*User*$_1$, *Integer.java*, 15–18). In order to make other users aware about his modification, the second user decides to apply the privacy policy that hides the content of his changes but shares their location. Therefore, the form of the generated ghost operation is $g(op_2)$=*update*(*User*$_2$, *Integer.java*, 16) signifying that line 16 is under modification.

I describe first what happens at the site of the first user who deletes the method isReal(). After the reception of the ghost operation sent by the second user $g(op_2)$, awareness information concerning activity of the second user can be presented as depicted in the Figure 5.5.

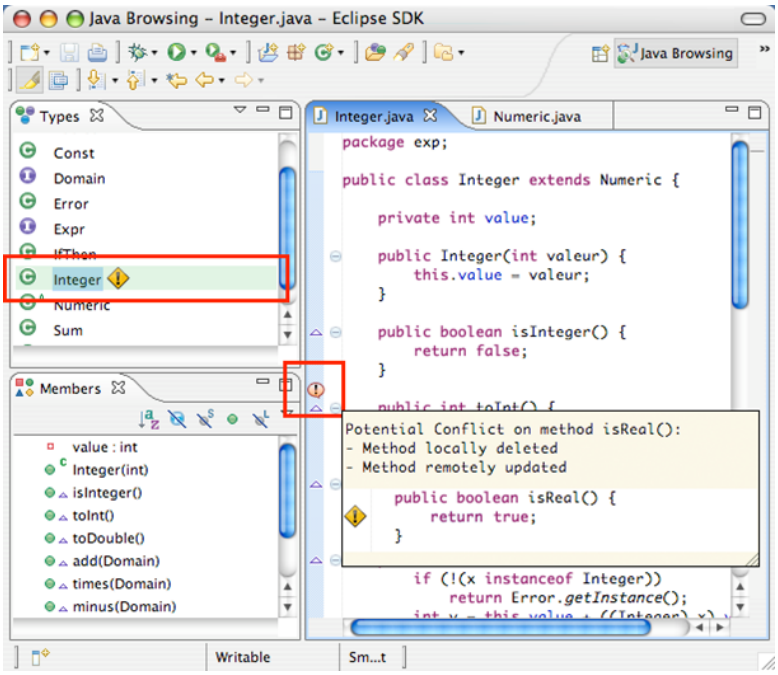


Figure 5.5: Interface at the first site after reception of ghost operations

Since the ghost operation $g(op_2)$ contains information about the target file, it is possible to indicate by means of a marker that the class Integer is concurrently modified as shown on the top left hand side window of the interface. In the right hand side window an annotation marker will indicate that a line was concurrently modified by another user. The position of the modified line is computed by using the line number indicated by the ghost operation. The associated annotation in Figure 5.5 informs the user that the method isReal() locally deleted was modified by another user. In this manner, the user can decide to contact the other user in order to resolve the conflict.

The site of the second user will be notified that the method isReal() is deleted by annotating the lines composing this method as shown in Figure 5.3 and by displaying in the class hierarchy that class Integer was concurrently modified and method isReal() was deleted.

The proposed awareness mechanism that takes into account user privacy was proposed in

the context of multi-synchronous collaboration of text documents [43]. Users can specify the detail of information made available to their collaborators and the computation of awareness is based on metrics that measure the effect of changes for the different types of changes, on the different syntactic document levels and document parts. For the visualisation of awareness, we employed the concept of edit profiles.

5.2 Conflicts management and their resolution (a case study)

In this section I present an overview of a study that I conducted on conflicts management in Git-based open source software. In contrast to centralized version control systems, Git does not support the centralized logging feature of all user activities. The best overview of user activities is provided by the commit history (including merges) from the primary repository. To identify concurrences and conflicts in each project, we replayed the merges done by users and computed the conflicts. We created a *shadow* repository and recursively re-integrated developer's changes into this repository. We analysed concurrency and conflicts on Git repositories of four projects: Rails [82], IkiWiki [80], Samba [83] and Linux Kernel [81].

5.2.1 Integration rate and conflict rate

We computed the number of concurrent updates to a same file and the number of these updates that resulted in unresolved conflicts. Similar to [141] we computed the *integration rate* and *conflict rate* as provided in Table 5.1. *File updates* represents the total number of updates to files throughout the development cycle. *Integration rate* represents the proportion of concurrent updates to a same file over all updates to files. *Conflict rate* is calculated by the proportion of updates to a same file that resulted in unresolved conflicts over concurrent updates to files. The file updates were collected from all commits of the project. And by re-integrating all developer's changes, we computed the concurrent updates to a same file and the concurrent updates to a same file that resulted in unresolved conflicts.

Project name	*File updates*	*Integration rate*	*Conflict rate*
Rails	117,960	4.04%	16.26%
IkiWiki	37,327	1.08%	50.50%
Samba	306,182	0.68%	87.84%
Kernel	1,278,247	10.99%	4.86%

Table 5.1: Concurrency and conflicts on files

We can notice that Kernel and Rails projects have larger integration rate than IkiWiki and Samba, where Kernel has by far the highest integration rate. This can be explained by the large size of Kernel project in terms of the number of files. In contrast with the *integration rate*, Rails and Kernel have smaller *conflict rates* than IkiWiki and Samba. This is mainly due to the working policy of developers for each project.

The lack of a central server that holds a reference copy of the project introduces more parallelism between user versions allowing them to diverge more in distributed version control systems than in centralised version control systems. For instance, Kernel, Rails, IkiWiki and Samba projects developed in Git have significantly (99% confidence level) higher *integration rate*

(22, 8, 2 and 1.5 times respectively) than projects in CVS analysed in [141]. However, the higher *integration rate* does not result into higher *conflict rate*. For instance, Kernel and Rails have 5 and 1.5 times respectively lower *conflict rate* than projects in CVS whereas Samba and IkiWiki have almost 2 times higher *conflict rate*. Integration rates are not equally distributed over the life time of projects but are higher close to release dates as we showed in [12]. In order to prevent conflicts, close to release dates developers should use awareness mechanisms about the location of their changes.

5.2.2 Conflict types

We also measured the proportion of the different conflict types: content conflicts referring to conflicts inside a file, remove/update conflicts referring to concurrent removal and update of a file and naming conflicts referring to concurrent renaming of the same file or of two files with the same name. Table 5.2 presents the proportion of conflict types of the four projects that we studied. We found that content conflicts are far the most frequent conflicts with a proportion of 46% - 90% from all conflicts.

Project name	*Content conflicts*	*Remove/Update conflicts*	*Naming conflicts*
Rails	89.68%	2.97%	7.35%
IkiWiki	46.31%	1.48%	52.22%
Samba	64.47%	34.44%	1.09%
Kernel	90.96%	8.63%	0.41%

Table 5.2: Proportion of conflict types

5.2.3 Conflict resolution

Several files can be in conflict during a merge. Table 5.3 reports on the total number of merges performed during the lifetime of each project and the number of merges that led to unresolved conflicts. When a merge is not resolved automatically by Git, users need to resolve it manually. A user can decide to rollback to a previous version. Table 5.3 also provides the rollback rate, i.e the number of times users manually resolve conflicts by reverting to a previous version.

Project name	*No. of merges*	*Unresolved conflict merge rate*	*Rollback rate*
Rails	9,728	4.34%	1.66%
IkiWiki	1,037	7.52%	5.13%
Samba	1,281	10.07%	6.98%
Kernel	38,961	9.11%	0.70%

Table 5.3: Frequencies of conflicting merges

5.2.4 Adjacent-line conflicts

We analysed Git mechanism of considering concurrent modifications of two continuous lines as being in conflict called adjacent-line conflicts. Figure 5.6 illustrates an example of adjacent line

conflict with both expected and real merge result. $User_1$ makes changes on line 2 and $User_2$

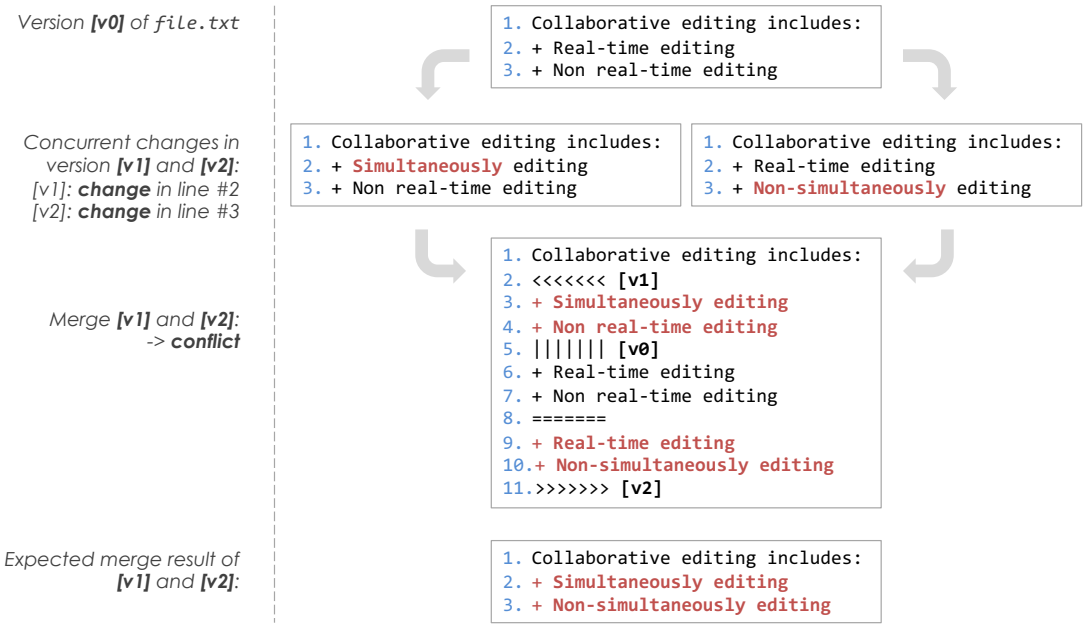


Figure 5.6: Adjacent-line conflicts: expected and real merge results

makes changes on line 3. They then merge their changes and Git generates a *'CONFLICT (content)'*. Our hypothesis is that this is not a content conflict because these changes are made on two different lines. Git should merge them successfully by applying changes from both users. To test our hypothesis, we analysed all content conflicts in the four projects to detect all adjacent-line conflicts. We then analysed the adjacent-line conflict resolutions that were manually fixed by the authors to check if both changes done on the adjacent lines were applied. Table 5.4 shows the number of adjacent-line conflicts and their possible resolutions: applying both changes, applying a change from one user only (either $User_1$ or $User_2$) and applying no changes.

Project name	*Adjacent-line conflicts*	*Applying both changes*	*Applying one change*	*Other cases*
Rails	80	46	28	6
IkiWiki	4	3	1	0
Samba	41	10	23	8
Kernel	367	312	51	4

Table 5.4: Adjacent line conflicts and resolutions

The proportions of applying both user changes are 24.39% in Samba, 57.5% in Rails, 75% in IkiWiki and respectively 85.01% in Linux Kernel. We did the same analysis on how users resolve normal content conflicts and Table 5.5 presents a comparison of the results obtained for adjacent-line conflicts and normal-content conflicts.

We compared the frequency of applying both changes for adjacent-line conflicts to that of applying both changes for normal content conflicts. Table 5.6 presents for each project the standardized mean-difference effect size (SdMD) between proportions of applying both changes

	Adjacent-line conflicts		Normal content conflicts	
Project name	*No. of conflicts*	*Applying both changes*	*No. of conflicts*	*Applying both changes*
Rails	80	57.50%	317	5.67%
IkiWiki	4	75.00%	22	9.09%
Samba	41	24.39%	1149	14.19%
Kernel	367	85.01%	1326	13.38%

Table 5.5: Adjacent-line and normal content conflicts

for adjacent-line and normal content conflicts [78]. Only Samba has a low SdMD and its lower bound of confidence interval (CI) is less than zero. Excepting Samba, we obtained significant confidence at level of 95% that adjacent-line conflicts are resolved more often by applying both changes than normal content conflicts. Applying both changes in the case of a conflict means that the concurrent changes were not in conflict, so the conflict was not necessary to be detected.

Project name	*SdMD*	*Lower 95% CI*	*Upper 95% CI*
Rails	1.8872	0.4319	3.6909
IkiWiki	2.0614	1.4932	2.2812
Samba	0.405	-0.0384	0.8483
Kernel	2.1837	1.9854	2.382

Table 5.6: Standardized mean difference(SdMD) effect size between adjacent-line conflicts and normal content conflicts

We found that across the four studied repositories around 75% of the adjacent-line conflicts were false positives. Adjacent-line conflicts detection was designed to warn the developers that there are two changes on adjacent lines that might be related and that developers should check whether the changes are conflicting. For around 25% of cases adjacent-line conflict warnings indeed helped developers to discover the conflicts reducing the costs in later development phase. However, for the other 75% developers did some un-needed extra work for resolving the conflicts. Moreover, if concurrent changes occur on two adjacent lines Git signals a conflict, but not in the case of two lines separated by two or more lines. Our results suggest that Git should reconsider signalling conflicts on adjacent lines inside the source code file which requires developers to do in most of the cases some extra work for removing the conflict. A solution would be that Git sends a warning message to the users in the case of concurrent changes on adjacent lines, but does not represent this conflict inside the source code file. Note that in Subversion and Darcs version control systems, concurrent modifications on adjacent lines are not considered as conflict. Our suggestion would be useful for tool builders to help developers in avoiding wasting time on trivial merge conflicts.

Part II

Trustworthy Collaboration

Multi-synchronous collaboration [237] is the most general collaboration mode allowing private working in cycles of divergence and convergence. Users work simultaneously in isolation in their workspace and their changes are synchronized at a later time. Shared data diverges when users work in isolation and converges later after synchronization of shared data.

Large scale multi-synchronous collaboration supporting a large number of users (communities rather than groups) to collaborate over large data (billions of documents) requires revisiting traditional security models that check user granted rights before allowing access to data. Such security models are too strict [224] and they do not scale well. I investigated replacing the traditional "hard" security models for large scale distributed collaborative systems with "soft" security models. Rather than adopting an a priori strict enforcement of security rules, access is given first to data without control but with restrictions that are verified a posteriori.

In this collaboration, it is very difficult to control what users will do with the data after it has been released to them and that they will not misbehave and violate usage policy. In the scope of Hien Thi Thu Truong's thesis, I modeled usage policy expressed in terms of obligation, permission and prohibition by means of contracts. The main issue we addressed was how contracts can be expressed and checked within multi-synchronous collaboration model and what actions can be taken in response to users who misbehaved [15, 37, 61]. For checking to what extent users respected or not their contracts, the collaboration logs containing both data and contracts are audited [35]. According to auditing results, users adjust their trust levels assigned to their collaborators. In Chapter 2 I present this contract-based collaboration model and its experimental evaluation.

Users can also misbehave by tampering the logs for their convenience by modifying data or contracts. This attack raises the threat that honest users might get forged content of shared data. Replicas with corrupted updates might never converge with other replicas with valid updates which is critical in multi-synchronous collaboration. Log auditing also requires that logs are not tampered as otherwise log auditing results are not correct. In the scope of Hien Thi Thu Truong's thesis, I proposed a hash chain based authenticators approach in order to ensure integrity and authenticity of logs of operations corresponding to different streams of activity during collaborative process [34]. As tamper resistance cannot be ensured in multi-synchronous collaboration without a central provider, we ensure tamper detection by making misbehaving users accountable for their misbehavior. The authenticators-based approach that we proposed is general and can be applied for logs of operations of any type. In Chapter 3 I present this authenticators-based approach and its experimental evaluation.

In this large scale multi-synchronous collaboration without a central authority a main question is how users choose their collaborators with whom they want to share their data. A rational decision would be based on the evaluation of previous collaborative behaviour of their collaborators. Participant names usually serve as a retrieval cue for recalling a participant's expertise and personally observed, previous behavior. However, it is difficult for a typical user to remember and recall each user in a collaboration that contains thousands of members that do not know each other personally. Studies confirmed that it is not possible for an average user to analyze every on-line interaction [217] and that maintenance of partner-specific trust values requires a significant overhead [167]. Therefore, users need assistance in assessing the trustworthiness of their partners. A rational decision would be to compute the partner trustworthiness based on the evaluation of their previous collaborative behaviour. A trust value that is computed automatically based on previous collaboration behavior would be of great help to users. A main challenge in this task is how to compute this trust value according to past collaboration in order to be able to predict future user behavior.

A solution would be to compute this trust value based on the observation of the respect or

violation of contracts in the contract-based collaboration model that we proposed. Another solution for computing trust score of users would be according to the quality of their contributions during a collaborative editing task.

In the scope of Quang Vinh Dang's thesis, I addressed the question whether providing users with a trust value of their partner would help them in their decisions of choosing their partners. We developed an experimental design for testing user acceptance of the proposed trust-based collaboration model for large communities of participants. We employed game theory, and trust game in particular, as a standard because it provides a solid foundation of research methods and findings, giving us a basis for expected performance. Firstly we proposed a computational personal trust model for repeated trust game that deals with fluctuating user behavior in the collaboration [27]. In this trust model user trust values are updated based on the satisfaction level for the exchanges during the game. In Chapter 4 I describe this trust metric and its evaluation. Secondly, in collaboration with Valerie Shalin we designed an experiment that tested variations of the trust game: with and without explicit computation of user trust values based on the computational trust model we proposed [11]. We organised a user study with 30 participants that confirmed that the availability of this trust metric improves user cooperation and that it predicts participants future behavior. In Chapter 5 I describe this experimental design for testing the influence of trust score on user behavior.

Before presenting in detail the above mentioned contributions in the domain of trustworthy collaboration, I present in Chapter 1 a short literature review related to each main aspect of these contributions.

1

Literature Review

My contributions in the domain of trustworthy collaboration are related to several topics in the area of privacy and data management in multi-synchronous working environment such as contract-based models, usage control models, access control models, trust management and log auditing for log-based collaboration. In this chapter I provide a short overview of the state of the art on these topics.

1.1 Contract-based models

The contractual approach 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. Contracts exist in many systems. It is either implicit in communication protocols, software licenses, downloading and sharing policies in P2P file-sharing systems or explicit in paper-based contracts of using network services. The push-pull-clone model for collaborative editing source code was adopted in distributed version control systems but users are uniformly trusted and there are no contracts specified during collaboration. 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 other people. In contract-based models rules have to be explicitly expressed and checked by the system.

Some existing works [270, 240, 231, 192] focus on contract frameworks for negotiating and controlling resource usage in a distributed system. Contracts express the terms under which nodes in network promise to offer and to get payment with regard of exchanging resources. However, these works do not consider contracts for end-users. Some other existing works [116, 95, 149, 137, 99, 107, 101] focus on contracts for end-users but they require an independent framework to express contracts. Some works need a central authority to enforce contracts [204, 190].

In the multi-synchronous collaboration we require that contracts are objects that are part of the replication mechanism: contracts are associated with mutable data and they are mutable themselves. The synchronisation mechanism has to deal both with data and contracts. The existing contract-based models cannot be applied to multi-synchronous collaboration systems where contracts are replicated and synchronized. We do not need a central authority to check contract compliance. We adopt the same view as the contract model in [174] on using a distributed trust model to verify *a posteriori* whether contracts were respected or not and therefore

to make parties accountable.

1.2 Access control

Providing an authorization mechanism suitable for distributed collaborative systems is a difficult task. Indeed, collaborative systems aim at facilitating the collaboration between users by allowing them to access all information and resources that they need. On the other side, access control seeks limiting the access and use of data and resources only to users with proper authorization.

[168] presents a survey of security mechanisms for collaborative systems. Discretionary access control (DAC) [274] and its variants such as ACL (Access Control Lists) and Capability Lists specify the rights that each subject possesses for each object. Collaborative systems such as Grove [253], RTCAL [268], BSCW [226] and Google Cloud Storage use ACLs. The main disadvantage of this access control model is that it lacks the ability to support dynamic changes of access rights.

In Role-based access control (RBAC) [248, 232], rights are associated to a role or a set of roles organized in a hierarchy and not to individual users as in DAC. SUITE [244] and PREP [256] are examples of collaborative systems using RBAC as a control access mechanism. RBAC approaches are largely used and they offer administrative advantages once the roles were implemented [93]. However, they require a substantial role engineering effort. Moreover, RBAC cannot accommodate dynamic distributed systems parameters such as time of the day and user location. Furthermore, it requires that in the context of multiple users, all users agree on the same set of designed roles, which is very challenging.

Attribute-based Access Control (ABAC) [220], [200] mechanisms were proposed to overcome disadvantages of RBAC and offer a flexible access control adaptable for distributed systems. In these mechanisms access requests are evaluated by means of rules involving attributes of entities (users or resources). An example of user attribute is the user identifier. ABAC entails a substantial attribute engineering effort. Furthermore, attributes have no meaning until they are associated with a user, object, or relation, and it is not practical to audit user permissions.

In the CSCW domain, Dewan et al. [244, 219, 224] highlight the shortcomings of standard access control. As opposed to standard access control models that are considered as pessimistic, optimistic access control [224, 212] can be seen as a new 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.

The above mentioned access control mechanisms feature two main difficulties in the context of collaborative systems. The first one relates to the design of security policies that are satisfactory for all partners involved in a collaboration. A challenging issue is how to manage partner joining and leaving to the group. The second difficulty is the necessity of sending at each user action an access request and waiting for its answer from a trusted central authority which maintains the security policies. This delay is critical for the real-time collaboration where the number of updates is high. Moreover, in the case of a federation of organisations agreeing on such an authority is almost impossible. Furthermore, in order to maintain its autonomy with respect to its autorisation management, it should be possible for a partner to revoke previous granted rights without contacting an external authority. Indeed, it is possible that once several users collaborated, they wish that some of them do not have anymore access to new modifications on

shared documents. However, it is normal that users that had access to a document version and that contributed to the document keep this document version even though they do not have anymore access to next versions of the document. Moreover, access control mechanisms deal only with access to data. They do not deal with the usage of data after users have been granted the access to the data.

1.3 Usage Control

Usage control can be considered as an evolution of access control. The term *usage control* was firstly introduced by Park and Sandhu with the notion UCON [194] for controlling access to and usage of data. At the beginning the term *usage* means usages of rights on digital objects. The UCON model [194] and the family of ABC models - authorizations (A), obligations (B) and conditions (C)- [183, 172, 151, 135, 133] derived from the UCON model consist of three core components (subjects, objects and rights) and several additional components (authorization rules, conditions and obligations). The core components are associated with several attributes that are involved in the authorization process. Conditions represent factors such as a specific time period when an access is allowed. Obligations have to be fulfilled by a subject to get access. Access is granted when a subject holds certain rights on specific objects. UCON model is more general than attribute-based access control as it deals with the fulfillment of obligations.

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 in [162] 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 dealing with the protection of digital information when it is shared, copied and distributed in an open environment.

Usage control is a suitable approach for open and distributed environments (e.g peer-to-peer, grid, web services, ad-hoc networks, cloud, etc) that addresses the inadequacies of access controls [114]. It is an active area with many proposed frameworks and models in the literature [162, 139, 131, 120]. A survey of usage control models [110] describes usage control in four independent layers starting from the high level specification to the low level of enforcement mechanisms and implementation. [151, 133, 119] presents applications of usage control for collaborative systems.

Usage control policies can be enforced in a detective or preventive manner [138]. The *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 the systems that participants use are heterogeneous, it is very difficult to put unanticipated circumstances into preventive policies and to ensure the polices can be enforced across such different systems as well.

Detective enforcement is a flexible 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. 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 has been given. Using logs to verify users actions are common in many works [208, 143, 145, 134, 122].

Usage control with detective enforcement is a very flexible mechanism suitable for collaborative environments. However, most usage control models do not consider replicated data as in multi-synchronous collaboration models. In these collaboration models, data is replicated and modified by different users at different moments of time. Data usage control must be part of the replication mechanism and conflicts between usage policies (contracts) have to be managed in those collaboration models.

The only existing usage control approach for ensuring security and privacy in a weakly consistent replication system where users are not uniformly trusted was presented in [109]. Access control policy claims are treated as data items. The guards added to replication protocol enforce specified policies at 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. However, this approach only expresses rights but not obligations that each replica should follow. Moreover, only the author of an item can define the policy associated to it and hence there is no requirement to resolve conflicts between policies. However, in the general multi-synchronous collaboration we need to deal with policy conflicts as multiple contributors can specify different contracts on the shared document. Moreover, the system described in [109] uses a state-based replication where each site applies updates to its replica without maintaining a change log rather than an operation-based replication as in the most general case of multi-synchronous replication.

1.4 Log auditing

Log auditing technique is a general principle in systems supporting observation. Keeping and managing event logs is frequently used for ensuring security and privacy. This approach has been studied in many works. In [143], a log auditing approach is used for detecting misbehavior in collaborative work environments, where a small group of users shares a large number of documents and policies. In [134, 208], authors present a logical policy-centric framework for behavior-based decision making. The framework consists of a formal model of user past behaviors which is based on event structures. However, these models [143, 134, 208] require a central authority that has the ability to observe all actions of all users. This assumption is not valid for a purely distributed multi-synchronous collaboration. The complexity of the log auditing mechanism compared to centralized solutions comes from the fact that each user has only a partial overview of the global collaboration and can audit only users with whom he collaborates. Therefore, a user can take decisions only from the information he possesses from the users with whom he collaborates.

1.5 Trust measurement and trust game

In a distributed collaboration model where access is given first to data without control but with restrictions that are verified a posteriori, trust management is an important aspect. The concept of trust in different communities varies according to how it is computed and used. My contributions rely on the concept of trust which is based on past user behaviors [197]. Trust is not immutable and it changes over time. Thus trust should be managed by using a

trust model. A trust model includes three basic components[159] that are gathering behavioral information, scoring and ranking peers and rewarding or punishing peers. Most of existing P2P trust models (e.g. EigenTrust model[186]) propose mechanisms to update trust values based on direct interactions between peers. In a collaboration model where restrictions are verified a posteriori, log auditing helps one user evaluate others either through direct or indirect interactions. No existing trust model considers log auditing result for trust assessment.

I did a literature review of game theory in the fields of cognitive science, psychology and economics and investigated whether there is a game theory model that could reflect collaborative document sharing and that deals with user reputation and trust. The most appropriate game theory model is the trust game, also known as the investment game developed by Berg in 1995 [239], a money exchange game that is widely used in economics to study trust between users [104, 90]. In the traditional trust game, an investor (also called “sender" or “trustor") can invest a fraction of his money, and the broker (also called “receiver" or “trustee") can return only part of his gains. If both players follow their economics-based best interest, the investor should never invest and the broker should never re-pay anything. The observed money exchange is entirely attributable to the existence of trust. I illustrate next an example of the exchanges between the “sender" and the “receiver". Initially the sender sends an integer amount between 0 and 10 units to the receiver. The receiver gains three times the amount sent. For instance, if the sender sent 7 money units, the receiver will gain $3 * 7 = 21$ units. Subsequently, the receiver can select an amount between 0 and the gained amount (in this case, 21) to return to the sender. However, the returned amount is not further multiplied. Suppose the receiver returned 11. The final payoff to the sender is 11 units, and the payoff to the receiver is $21 - 11 = 10$ units.

The trust game can be one-shot, i.e. the game ends after one round of money exchange, or repeated, i.e. it lasts several rounds [188, 180, 157]. The pairs of users could be fixed [175] or re-assigned before each round [96]. These games provide different kinds of partner information to players, such as their gender, age and income [130], or their past interaction history [166, 96].

Our aim was to design a trust metric that computes partner trust as a basis for the prediction of partner behavior in the repeated trust game. Game theory predicts that, in trust game, sender will send 0 and receiver will send back 0 [179]. However, in experimental game theory we usually do not observe this user behavior. In fact, the sending behavior of users in large-scale settings follows the normal distribution [104].

User trust in trust game was measured as an average value of previous sending amount [211, 96, 104, 129]. However, this average trust metric can not deal with malicious user fluctuating behavior where users may strategically oscillate between collaboration periods when they aim gaining the trust of the other users and sudden betrayal.

1.6 Log Authentication

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 a secure logging machine such as a write-only medium (e.g CD/DVD), a tamper-resistant-hardware or a trusted hardware to prevent adversary from modifying logs [181]. However, in real-world applications deployed over large scale distributed environments, it is impractical to assume the presence of such devices. Existing solutions belonging to the later approach [251, 246, 117, 173, 121, 158, 113, 193, 218, 115] are adapted only for collaboration based on a single global stream of activity over shared data allowing a single user at a time to access objects in the shared workspace.

PeerReview framework [145] ensures accountability and eventual detection of all Byzantine

faults in a distributed system. PeerReview maintains a secure log of the messages sent and received by each node. A node is automatically detected when its behavior deviates from that of a given reference implementation. However, each node should maintain an identical log with other nodes. PeerReview does not support that each node keeps different orders of operations as it is the case in multi-synchronous collaboration where users maintain different streams of activity on the shared data.

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. A set of secure audit logging schemes and aggregate signatures were proposed in [140, 132, 113]. Forward security ensures the integrity of log entries in the log stream and no selective deletion or re-ordering to stream is possible. BAF [115] was proposed to secure audit logs by achieving at the same time the computationally efficient log signing and the truncation-attack-resistant logging. This work can be applied only if 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 [205, 178, 193, 128]. While these approaches are computationally efficient, they do not deal with history for collaboration. Timeweave [193] 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 synchronized which is required in optimistic replication where concurrent operations appear in different orders in replicas.

Apart from above approaches, there are works that address securing logs for replication systems. In [227] hash chains protect modification orders of a weakly consistent, replicated data system. SHH approach [182, 177] for optimistic replication uses hash values as in Merkle tree [271] for revision identifiers to protect causality of version history. It serves mainly for 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 and distinguish whether data are forged or not. If the sender signature would be included in summary hashes, the merge executed at different sites of the same two versions would diverge even though the content of the merged result is the same. 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 [160] and Mercurial history [118].

A mechanism of preventing history forgery for a document/database history was proposed in [121]. The authors present a provenance-aware system prototype that captures the 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.

The document control flow framework proposed in [164] addressed 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 secures different XML elements, while we aim to secure patches of operations.

In the domain of database security, Depot [105] secures replicated database in the cloud. Depot addresses the issues of consistency, integrity and authorization in the context of database where data is stored in the form of key/value and update is the main operation performed over the database. Collaborative systems involve more operations 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 an authorized node to enforce consistency and integrity. This would be too costly in a collaborative working environment where users produce a huge number of operations on the shared document.

The state of the art of secure audit logging research was also surveyed in [127]. Though secure audit logging was intensively investigated, no existing work 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.

2

Trust-based collaboration

Push-Pull-Clone (PPC) is one of the paradigms supporting multi-synchronous collaboration. In 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 workspace this data as well as changes done on it. Users can then push their changes to many channels at any time they want, and other users who have granted rights may pull these changes from these channels. By using pull primitives replicas are synchronized.

In the context of the PhD thesis of Hien Thi Thu Truong, I designed a contract-based PPC model, where contracts are specified by data owners when they share the data and the adherence to or violation of contracts can be checked after users gained access to data [15, 37, 61, 35]. Audit of user compliance to the given contracts allows the computation of trust scores associated to users. This trust scores would help users collaborate with other users they trust.

As an example of use of our proposed contract-based PPC model we can consider implicit contracts in distributed version control systems (DVCS). DVCS systems widely used for source code development rely on the PPC paradigm. In open source projects, usage restrictions are expressed in the license of the code, while in closed source code projects, these restrictions are expressed in the contracts developers sign when accepting their job. In both cases, usage restrictions are checked *a posteriori* outside the collaborative environment with social control or plagiarism detection. As a result of observations concerning usage violation, trustworthiness on the users who misbehaved is implicitly decreased and collaboration with those users risks to be ceased. We aimed at building a contract-based model that can express usage restrictions which are checked within the collaborative environment.

The main issue in designing a contract-based multi-synchronous collaboration is that contracts are objects that are part of the replication mechanism. In our contract-based model each user maintains a local workspace that contains shared data and contracts for the usage restriction of that data as well as changes on data. Changes done locally on the data together with specified contracts are shared with other users. Algorithms for merging and for conflict resolution have to deal not only with data changes but also with contracts. For checking if users respect contracts, a log auditing mechanism is used. According to auditing results, users adjust their trust levels assigned to their collaborators.

In this chapter I describe:

- A PPC model extended with contracts for multi-synchronous working environment that we call the C-PPC model. The proposed model ensures consistency of the shared document.
- A log auditing mechanism to detect user misbehavior in C-PPC model.

- A set of experiments to evaluate the performance of the C-PPC model and the log auditing mechanism by using a peer-to-peer simulator.

2.1 The C-PPC model

Our model deals with modifications that users do on the different parts of shared documents and contracts specified by users while they exchange different versions of the document with other users. All these modifications and contracts are kept in logs of operations maintaining information about who did the operation and when the operation was performed.

A document is seen as a log of operations that have been done during the collaboration. 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 allowing 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.

For the sake of simplicity we consider that all users collaborate on a single shared document.

2.1.1 Events and logs

Events and log structures are defined as follows:

Definition 2.1.1 (Event)**.** *Let* $\mathbb{P}$ *be a set of operations* {insert, delete, update, 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.*

Definition 2.1.2 (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 his 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.

A special event attribute is *GSN* (*generate sequence number*) that is assigned to the event at the site where 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 the receiving site.

2.1.2 Contracts

A contract expresses usage policies which one user expects others to respect when they receive and use shared data. Contracts are built on the top of basic deontic logic[277] with the normative concepts of obligation, permission and prohibition representing what one ought to, may, or must not do. We handle contracts in a distributed manner, as part of a replication system.

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 as 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 $\sim P_{\sim A}$ or simpler by O_A. The proposition that the act named by A is indifferent, is symbolized by $(P_A)\&(P_{\sim A})$. In this symbolism, P, O, 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.

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, $A \to B$ as a name of their implication. We should note that "$\sim$" is stronger than "&", "&" is stronger than "$\vee$", and "$\vee$" is stronger than "$\to$". Several deontic logic laws were defined on the deontic operators [15].

2.1.2.1 Contract formalisation

Based on deontic concepts, we formalized contracts as follows.

Definition 2.1.3 (Contract primitive)**.** *A contract primitive is composed of a deontic operator followed by a write or a communication operation in $\mathbb{P} = \{op_1, op_2, \ldots, op_n\}$. A contract primitive is an event in the log where deontic operators P (the permitted), O (the obligatory), F (the forbidden) are modality attributes. 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 primitive c can refer to any operation.*

Definition 2.1.4 (Contract)**.** *A contract C is a set of contract primitives which are built on these 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.*

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) $C = \{c_1\}$ & $(c_1 \to c_2) \longrightarrow C = \{c_1, c_2\}$ (*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 relationship between two contract primitives or two operations).

The deducibility axiom A1 shows that 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 primitives given at a certain sharing time since those contracts that can be inferred from other contract primitives do not need to be specified. For instance, if we suppose that $O_{edit} \to P_{insert}$, then $C = \{O_{edit}\}$ & $(O_{edit} \to P_{insert}) \longrightarrow C = \{O_{edit}, P_{insert}\}$. This means that if a user receives an obligation to edit, she will have the permission to insert automatically since the setting of inference rule $O_{edit} \to P_{insert}$ holds. Another example, if system allows $P_{edit} \to P_{insert}$ then $C = \{P_{edit}\}$ & $(P_{edit} \to P_{insert}) \longrightarrow C = \{P_{edit}, P_{insert}\}$.

The axiom (A2) rules the merging process of different contract primitives that refer to a 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

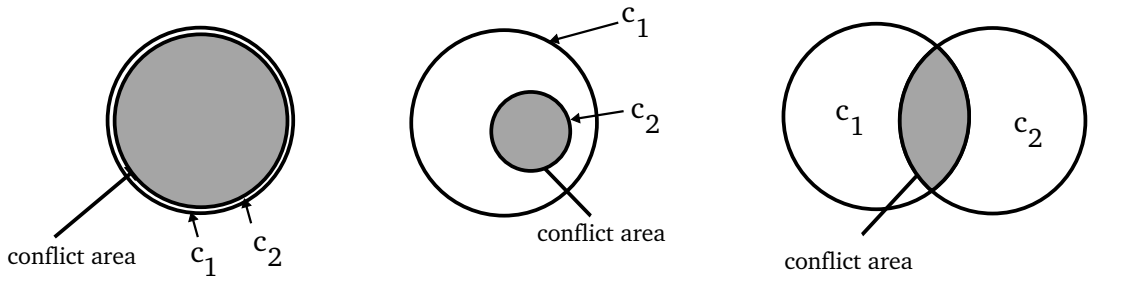
deducible as $C = \{c^1_{op}\}$. For instance, if $C = \{F_{op}, P_{op}\}$ and $F_{op} \succ P_{op}$ then it is deducible to have a contract $C = \{F_{op}\}$.

2.1.2.2 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 terms *deontic conflict* and *deontic inconsistency* have been used interchangeably in the literature. According to [276] inconsistency can be of three types: *total-total*, *total-partial* and *partial-partial*.

Total-total inconsistency means that 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 (Fig. 2.1a). 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 2.1: Three types of inconsistencies.

Total-partial inconsistency means that 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 (Fig. 2.1b). As an example, the total-partial inconsistency arises when an action is simultaneously permitted and forbidden.

Partial-partial inconsistency means that 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 (Fig. 2.1c). We can see this inconsistency in an example where 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 or both of the two norms without causing violation to the other. In this case, she can enter the theatre, and by attending the concert, she fulfills the obligation requiring her to attend it. In this case no conflict arises. However, if she does not have a ticket, then she cannot follow one norm without violating the other. Without having a ticket, while 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 violates the prohibition not allowing her to enter without ticket.

Through the example we see that the two norms are incompatible in one case but compatible in another case.

Definition 2.1.5 (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.*

Definition 2.1.6 (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}.*

2.1.2.3 Conflict resolution

In this subsection I present our solution to deal with inconsistencies of contracts.

Users should be informed about their contractual situation when a synchronization is performed. This would help them choosing between different contracts the ones that give them better benefit. In what follows I describe a method for ordering contracts based on the priority order of operations associated with them.

Depending on the application type, priorities between contract primitives can be specified. One way to select a contract in the case of conflict is to assign orders to various contract primitives, and then compare contracts composed of them. The hierarchical ordering of contracts enables users to give preferences to some contracts over others. In this case, the set of contract primitives is a partially ordered set and the ordering relations are intrinsic to the contracts in the system.

Depending on the operation types, the order of contract primitives is given as follows. Operations are categorized to 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 within group and the order of the group.

We formalized the method to order single operations as well as sets of operations. Our method is inspired from the work of Cholvya and Hunterb[184]. I present below the ordering of operations within single category and across multiple categories.

- Ordering categories of operations: Each category includes a set of operations referring to a specific kind of action. Given two categories $\Lambda_1 = [\alpha_1, ..., \alpha_m]$ and $\Lambda_2 = [\beta_1, ..., \beta_n]$, the ordering of Λ_1 and Λ_2 is 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 orders 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 $\mathbb{P}$, $C = \{c_1, c_2, \ldots, c_m\}$, $c_i = P_{op_j}|O_{op_j}|F_{op_j}$, $c_i \in C$, $op_j \in \mathbb{P}$, $1 \leq i \leq m$, $1 \leq j \leq n$, where contract primitives are ordered following operation order.

To order contracts, we set norms in some kind of hierarchy. Without losing generality, let us assume that deontic operators are ordered as $P \succ O \succ F$ and operations in $\mathbb{P}$ are ordered as $op_1 \succ op_2 \succ \ldots \succ op_n$. 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 $s[j]_3 = 2$, where $1 \leq i \leq m$;
  (2) if $c_i = O_{op_j}$ then $s[j]_3 = 1$, where $1 \leq i \leq m$;
  (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 specified on operation op_j. We consider that the absence of obligations and prohibitions implies the permission. Therefore, $s[j]$ is set the value 2 similarly to the case that op_j is permitted.

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

For instance, given a set $\mathbb{P}$ of two operations (n=2) in the order $op_1 \succ op_2$ 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 decision in case of inconsistencies to choose the contracts with more benefits. It is important to make users aware that what is added to the system might introduce inconsistency. Furthermore, in a peer-to-peer network with no central authority that maintains the consistency of contracts, once conflicts are detected, they should be resolved or adapted by users.

2.1.2.4 Repealing contracts

In addition to adding contracts to data, our approach supports removal of given contracts. We consider the overriding rule to repeal contracts issued in the past.

Overriding rule allows that an old contract is 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 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 received after c_1).

I present next an example of contract overriding. Suppose that Alice realizes that the operation op under the contract primitive $c_{op} = F_{op}$ she gave to Bob some time ago should not be forbidden any longer because conditions that made the prohibition of performing op have changed. She wants to permit Bob to do op. Since previous changes associated with given contracts were logged and shared with other users, the only solution for removing the prohibition is by compensation. Alice can override the prohibition by giving a new contract to Bob. Once the new contract is accepted by Bob, the prohibition is removed for him.

2.1.3 Log management and consistency maintenance during collaboration

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

2.1.3.1 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 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.
- 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 (kept in log) by u.
  2. e is a *contract* or *communication* event given to u by another site v. Recall that sending site does not keep *contract* or *communication* events in its local log.

An important feature of 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.

2.1.3.2 Properties of C-PPC Model

The C-PPC model is based on the CCI consistency model[222] which requires preserving *causality*, ensuring document *convergence* and preserving user *intention*.

Regarding causality preservation, the C-PPC model preserves the following two types of causal relationships (denoted as $\xrightarrow{c}$): *causal* relation (based on *happened-before* defined by Lamport[272]) and *semantic causal* relation. These causal relations between events are used in the auditing mechanism for detection of users that did not respect the given contracts.

- *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 example in Figure 2.2(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 example in Figure 2.2(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 (see example in Figure 2.2(c)). The contract event given by site u to other sites should depend on the current contracts of u: ($e_1.evt = e_2.evt = contract$) and ($e_1.attr.to = e_2.attr.by$) and ($e_1.attr.RSN < e_2.attr.GSN$) $\Longrightarrow$ ($e_1 \xrightarrow{c} e_2$).

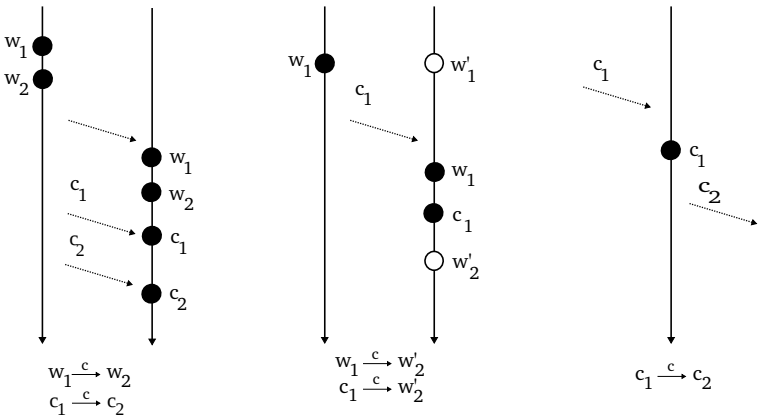


Figure 2.2: Causal relations between events (w_i denotes *write* events and c_i denotes *contract* or *communication* events).

In C-PPC model, logs are propagated by using anti-entropy[267, 228], 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 replicas. 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.

Anti-entropy ensures the *happened-before* relation between events as defined by Lamport[272] without using state vectors[261] or causal barriers[230]. We say that event e_1 *happened-before* e_2, denoted as $e_1 \xrightarrow{hb} e_2$, if e_2 was generated on 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 partial order set H to the linear extensions on the same ground set E.

These linear extensions are individual logs observed by different sites. Figure 2.3 shows an example of a history and its congruent linear extensions.

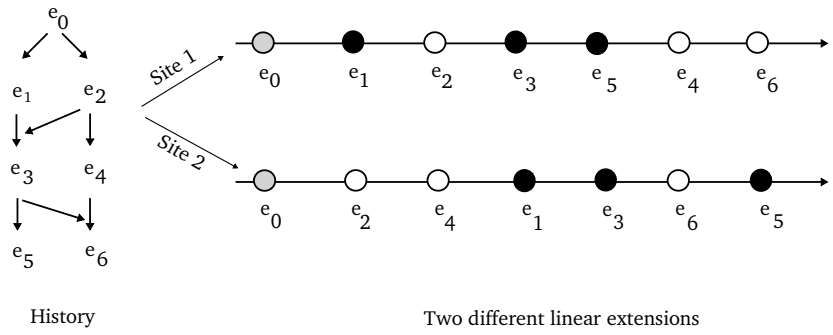


Figure 2.3: An example of history and logs

In collaborative systems, where multiple sites collaborate on the same shared data object, 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 site keeps therefore an individual log as a linearization of history built on a subset of a ground set of operations. There are remaining events of global history built on entire ground set of events that are not visible for the site.

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. Document state can be achieved by re-executing operations in the log. This collaboration model requires that the log is not tampered. Our solution for the construction and verification of authenticators to secure logs are presented in next chapter. Authenticators prevent re-ordering of log events and therefore causality is preserved. If the log was tampered, the receiving site might discard it and the trust level of the site that misbehaved would be decremented.

The C-PPC model uses CRDTs for document convergence. It ensures that the copies of the shared document are identical in the presence of different contracts received by different sites when the same set of *write* operations was executed at those sites. However, the shared document might be in different states on two sites since the shared document is not uniformly distributed due to the use of contracts and the trust levels of users. Users can use log-auditing mechanism to detect contract conflicts and logs are synchronized if and only if all conflicts are resolvable. Conflicts can be resolved by the rejection of the owner or by the overriding of user with high role in system (the order of users can be voted between users in system).

And finally, intention preservation in C-PPC model is ensured by causality preservation and CRDT algorithm.

2.1.3.3 Pushing Logs containing Contracts

In C-PPC model contracts are given to restrict usage on the shared document when a user shares/pushes a document to another user. Added contracts respect the following properties:

- A contract is attached at the end of the log of events representing the document modifications and the accepted contracts.
- When 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' where $C' \leq C$ (contracts are compared as presented in section 2.1.2.3).
- A user cannot specify a new contract which conflicts with her current contracts. For instance, if user u has contract $C = \{O_{op}\}$, then she cannot add F_{op} to C.
- The contracts a user specifies to different users might be different. These users do not know the contract of the other users 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.

2.1.3.4 Pulling and Merging Pairwise Logs

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 and therefore keeping the local version in its current state or to abandon completely the local document version and accept the new remote version. If conflicts can be resolved the local log L is merged with the remote log L' as shown in the *merge* algorithm 1, *clock* being the logical clock of the local site.

I describe next how our *merge* algorithm ensures causality not only between *write* events but also between *communication* events and *contract* events. The *commit sequence number CSN* is used to track the last event committed by one site. As previously mentioned, the attributes of event e, $e.attr.GSN$ and $e.attr.RSN$ record the values of the clock at its generation and reception, respectively. Every event has the *GSN* attribute assigned before the log is propagated, but the *RSN* attribute is assigned to *communication* and *contract* events at the receiving site during the synchronization. The value of the commit sequence number (*CSN*) of an event e committed by site u is computed as follows:

- If e is a *write* event generated by u, *CSN* is assigned the value of attribute $e.attr.GSN$. 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 a to a site v and committed by site v, *CSN* is assigned the value of attribute $e.attr.RSN$. The site who commits e is extracted from attribute $e.attr.to$.

The fact that the *merge* algorithm ensures that new events are added only to the end of log enables the property that if the log of a site u contains an event e committed by v with a commit

Algorithm 1: merge(L, L', clock)

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

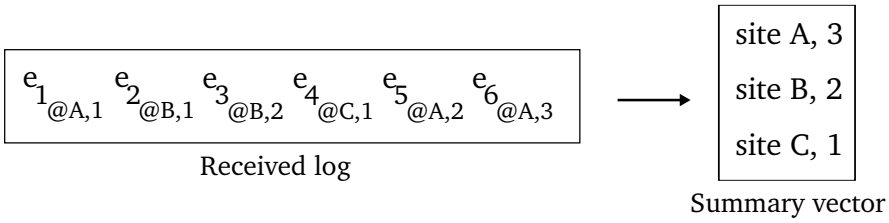


Figure 2.4: An example of Summary Vector

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 Fig.2.4). A summary vector is a set of time-stamps CSN, each from a different site. 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 that needs to be exchanged together with the corresponding operation and whose size is the number of sites. Instead, summary vector is maintained locally at sites and its size is the number of other sites whose events are known to the site.

It is possible to replay *write* events from log to get document state. We can use any existing CRDT approach[124, 108, 103] in which concurrent operations can be replayed in any causal order as they are designed to commute in order to ensure document consistency. The complexity of *merge* Algorithm 1 is $O(n)$ where n is the size of the remote log L'.

2.1.4 Log Auditing and Trust Assessment

Through a mechanism of log auditing we check whether user actions comply with contracts. Log auditing is an approach that adopts *a posteriori* enforcement of controlling compliance of users

after the fact. The main principles of our auditing mechanism for the C-PPC model are:

1. Users can audit the log 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 incompleteness evidence. However, this assumption is suitable to human society where a person is assessed only based on some of her noticed behavior.

2. Logs that reflect actions done by users and that are input to the auditing mechanism must be maintained correctly. I present in chapter 3 a solution based on authenticators for detecting log tampering. Log tampering is detected at time of synchronization before the log is accepted by receivers.

3. When users discover other users that misbehaved, they update their trust levels. Users use trust models to manage their friend reputation. The trust levels obtained from auditing result are used as input data for a trust model.

We identified three types of attacks that might lead to contract violation.

- Malicious users tamper logs to eliminate or modify contracts or other events in the log. We consider that a user u is *malicious* if she re-orders, inserts or deletes events in the log. For instance, u removes some obligations that she does not want to fulfill. This kind of attack is detected by the log authentication mechanism described in chapter 3.

- Malicious users perform actions that are forbidden by the specified contracts. These action events are labelled as *bad*.

- Users neglect obligations that need to be fulfilled. For instance, a user receives an obligation *"insert is obligatory"* but she never fulfills this obligation. If at a given moment a log auditing mechanism is performed and no event that fulfills the obligation is found, we cannot claim that the user misbehaved. The user might fulfill the obligation at a later time. The given obligation is labelled 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, his misbahaviour should be detected by other users. The auditing mechanism returns a trust value that is computed from the number of events labelled 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 prevents users in case a contract is violated by reminding them the obligations they hold.

Our auditing mechanism aims at finding contract violations and making users accountable for their irresponsible actions by adjusting their trust levels following a trust metric. The general idea of the auditing algorithm (Algorithm 2) executed locally by each user is to browse their log and check each event appearing in the log whether it conforms to the given contracts. For each violation of a particular user found, 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*

Algorithm 2: audit$(L, lastCheckedPos)$

```
1  for i = lastCheckedPos + 1 to length(L) do
2      e ← i^th event in L;
3      if (e.evt =' contract') then
4          v ← e.to;
5          if (v ∈ contracts) then
6              contracts[v] ← contracts[v] ∪ {e};
7          else
8              contracts[v] ← {e};
9          if e overrides c in contracts[v] then
10             contracts[v] ← contracts[v] \ {c};
11         if (e.attr.modal =' O') then
12             if (v ∈ obligations) then
13                 obligations[v] ← obligations[v] ∪ {e};
14             else
15                 obligations[v] ← {e};
16             if (v ∈ numberOfUnknownEvents) then
17                 numberOfUnknownEvents[v] ← numberOfUnknownEvents[v] + 1;
18             else
19                 numberOfUnknownEvents[v] ← 1;
20     else
21         v ← e.by;
22         if e violates contracts[v] then
23             if (v ∈ numberOfBadEvents) then
24                 numberOfBadEvents[v] ← numberOfBadEvents[v] + 1;
25             else
26                 numberOfBadEvents[v] ← 1;
27         if e fulfills c in obligations[v] then
28             obligations[v] ← obligations[v] \ {c};
29             numberOfUnknownEvents[v] − −;
30     V ← V ∪ {v};
31     if (v ∈ numberOfAuditedEvents) then
32         numberOfAuditedEvents[v] ← numberOfAuditedEvents[v] + 1;
33     else
34         numberOfAuditedEvents[v] ← 1;
35 foreach v in V do
36     current_trust[v] ← exp(−λ ∗ (numberOfUnknownEvents[v] + k ∗ numberOfBadEvents[v])/((1 +
         k) ∗ numberOfAuditedEvents[v]);
37     if (v ∈ trust) then
38         trust[v] ← α ∗ current_trust[v] + (1 − α) ∗ trust[v];
39     else
40         trust[v] ← current_trust[v];
```

events. The number of contract violations of user v over all the total audited events done by v is used to compute the trust level of v.

Algorithm 2 *audit* takes as input the local log L of user u and the position in the log *lastCheckedPos* identifying the last event checked by the auditing mechanism. L is browsed to check whether log events respect or not the given contracts. This corresponds to the case where the user u audits actions of all other users v who performed events in the log. Trust values of all audited users v in V are recomputed based on auditing results.

contracts[v] and *obligations*[v] are used to keep a set of contracts and a set of obligations which user v holds, respectively (*obligations*[v] $\in$ *contracts*[v]).

For each event e in the log L, we check its event type, contract or write event. If e is a contract given to user v then it is added to *contracts*[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 *contracts*[v]. 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*). *auditedEvents*[v] is used to count the total number of audited events. All users v audited by u are inserted in set V.

Before the auditing mechanism is applied, several variables are initialised as shown in the procedure initialization. The set of audited users V is initialised to the empty set. *contracts* and *obligations* are represented as a *map* between users and a set containing log events (contracts and obligations respectively). *numberOfUnknownEvents*, *numberOfBadEvents*, *numberOfAuditedEvents*, *current_trust*, *trust* are represented as a *map* between users and integers.

Procedure initialization

```
1 V ← new set();
2 contracts ← new map();
3 obligations ← new map();
4 numberOfUnknownEvents ← new map();
5 numberOfBadEvents ← new map();
6 numberOfAuditedEvents ← new map();
7 current_trust ← new map();
8 trust ← new map();
```

A user u can perform log auditing at any time at local site. As a result of log auditing, user u updates the trust values of users in the system. Log analysis has a time complexity $O(n)$ where n is the number of events that are audited. In case auditing creates significant overhead, users might skip auditing some parts of log with events performed by highly trusted users. However, in case these users behave badly, they are discovered only in a next auditing phase.

In order to manage trust levels, we need a decentralized trust model. The trust level of a user assessed by one another could be aggregated from log-based trust, reputation trust and recommendation trust. In [94] we proposed an example of a trust metric, where the trust value is computed directly by users based on auditing of their local log and not from an indirect source such as a recommendation.

The current trust value for a user is computed based on auditing results consisting of the number of writing events that violate contracts defined as *bad*, the number of contract events that are not fulfilled defined as *unknown* since it is unknown whether they will be fulfilled in the future and the number of remaining events that are neither bad nor unknown, defined as *good*.

We consider trust values range in the interval [0..1]. If all audited events are good, then the trust value equals 1. Otherwise, the *unknown* and *bad* events decrease the trust value. We denote by x_1, x_2 and x_3 the number of *good*, *unknown* and *bad* events, respectively and by $y = x_1 + x_2 + x_3$ the total number of audited events. We assume the number of *bad* events, x_3, is k times stronger than x_2 in decreasing trust. Applying weighted mean of x_2 and x_3 to compute

the decrement they cause on trust values over y audited events, we have:

$$malicious\_rate = \frac{\frac{x_2 + k * x_3}{1+k}}{y} = \frac{x_2 + k * x_3}{y + k * y} \tag{2.1}$$

We wanted to define a trust function that varies according to the number of malicious events. In a system where violations are assumed happening rarely trust should be decreased quickly if violations are found, while in a system where violations are assumed easily happening trust should not be decreased strongly. In the C-PPC model we assume that users perform actions mostly according to their given contracts, so violations will occur rarely. We considered that when bad events occur, trust value should decrease quickly. This assumption is realistic as in social life where one bad action might strongly decrease the reputation of a trusted person. Derived from this hypothesis, we designed the following trust function that decreases exponentially with the amount of bad events or unknown events found, λ being the decreasing coefficient, u being the auditor user and v the auditee user:

$$current\_trust(u, v) = exp(-\lambda * \frac{x_2 + k * x_3}{y + k * y}) \tag{2.2}$$

Figure 2.5 depicts an example of trust values computed from the following parameters. The total number of events $y = 100$. The *bad* events are three times more dangerous than unknown event, $k = 3$. The decreasing rate for trust in case of malicious events is $\lambda = 5$. The graph plots function $exp(-5 * \frac{x+3*y}{400})$. We can see that trust has the highest value 1 when no malicious events (bad and unknown) are found. Otherwise, trust decreases exponentially, however, it decreases more slowly on the dimension of *unknown* than the dimension of *bad*.

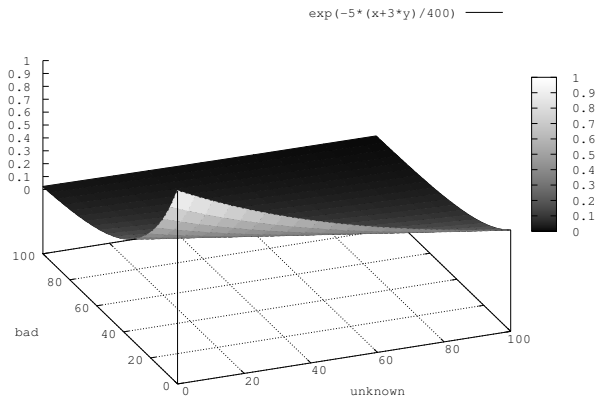


Figure 2.5: Current trust computed from the variations of number of bad and unknown events that are found after auditing, $0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$.

Trust value of user v assessed by user u is updated at each step n when an auditing is performed by u. Given the last trust value of user v assessed by user u, $trust_{n-1}(u, v)$, the new trust value for user v is computed by u as an aggregation between the current trust $current\_trust(u, v)$ and the last trust value $trust_{n-1}(u, v)$ as in the equation 2.3 with $n \geq 2$. α stands for the importance of the current trust against the old trust value computed from the last auditing phase, where $0 < \alpha < 1$. $trust_1(u, v) = current\_trust(u, v)$ is the trust between u and v computed after the first auditing phase.

$$trust_n(u,v) = \alpha * current\_trust(u,v) + (1-\alpha) * trust_{n-1}(u,v). \quad (2.3)$$

2.2 Experimental evaluation of the C-PPC model and auditing mechanism

In this section, I present the evaluation of our proposed model by performing some experiments using a peer-to-peer simulator and give some discussions around the proposed model.

Due to the unavailability of real data traces of collaboration including contracts, we evaluated the feasibility of C-PPC model through simulation using PeerSim [126]. We analysed the ability of detecting misbehaving users and we estimated the overhead generated by using contracts.

We simulated a network of 200 users as either honest or misbehaving. The ratio of honest/misbehaving users in different experiments varies depending on the purpose of evaluation.

We generated randomly the collaboration data flow including operations, contracts and users with whom to share. The network topology with which users share logs to their neighbors is 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 no more than three other users at each step. Similarly, the number of operations and contracts generated by one user each time is at most ten operations and three contracts (if we consider only three types of actions in our system: insertion, deletion and sharing).

Each node in the network represents one user. Between two interactions, nodes generate local operations randomly but must follow their current contracts. Nodes keep their assigned contracts temporarily in order to be able to generate correct operations. While honest nodes generate allowed operations, misbehaving nodes generate operations that violate their contracts.

Since contracts are generated randomly with the only condition that they should be no bigger than the node's current contract according to the order described in subsection 2.1.2.3, conflict certainly arises during the simulation between contracts of different nodes. However, logs are always maintained under consistent contracts. We omit negotiation protocols for contracts. Once a node detects conflicts before a synchronisation, the remote log is rejected and the node waits for next cycle or for other nodes which send logs without conflicts.

2.2.1 Experiment 1 - Misbehavior detection

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 users that can detect him. Figure 2.6 shows the results recorded after each cycle. We can see that at the beginning the misbehaving user is detected by a few users at then the number of users that detect his misbehavior increases along with the number of interactions.

Second, we compute the percentage of misbehaving users that can be detected. We select randomly one honest user to observe the percentage of misbehaving users that she can detect. Figure 2.7 shows the result according to the number of synchronizations done by the selected user with others. We can see 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

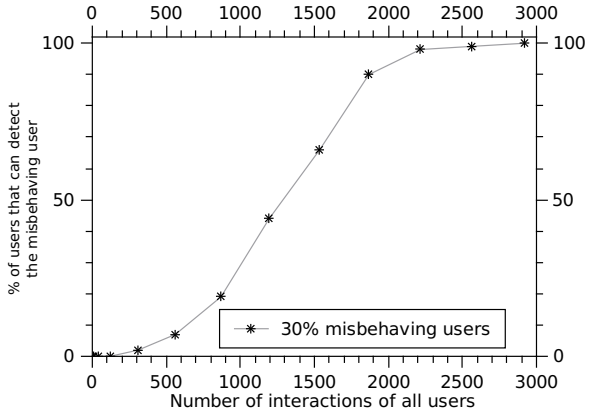


Figure 2.6: Detection of one selected misbehaving user with respect to the total number of interactions.

than 80% of misbehaving users are detected. The drastic change between the fourth and the fifth synchronization is due to a synchronization of the log of selected user with a remote log that contains misbehavior of most 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. We can see that the ability to detect misbehaving users depends also on the topology of collaborative network.

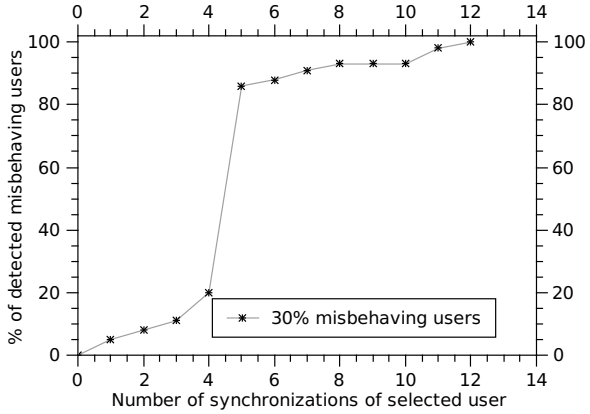


Figure 2.7: Percentage of detected misbehaving users with respect to the number of synchronizations done by selected honest user.

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 2.8 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 (respectively 5%, 30%, 80% of misbehaving users). The results show that the system functions well both in the case of a high or low population of

misbehaving users.

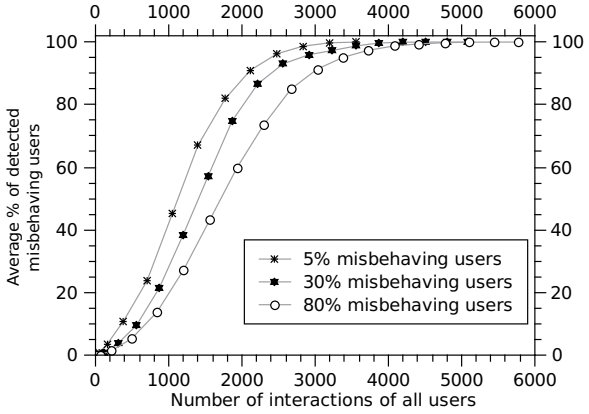


Figure 2.8: Average percentage of detected misbehaving users with respect to the total number of interactions in the collaborative network.

2.2.2 Experiment 2 - Overhead estimation

We evaluated the time overhead generated by using contracts for the synchronization and auditing mechanism. Following the same data flow, we compared two models: with and without contracts. In the model without contracts, synchronization requires merging logs of write events only. In the model with contracts synchronization requires merging logs of write and contract events. Additionally, an auditing mechanism for user misbehavior detection has to be applied.

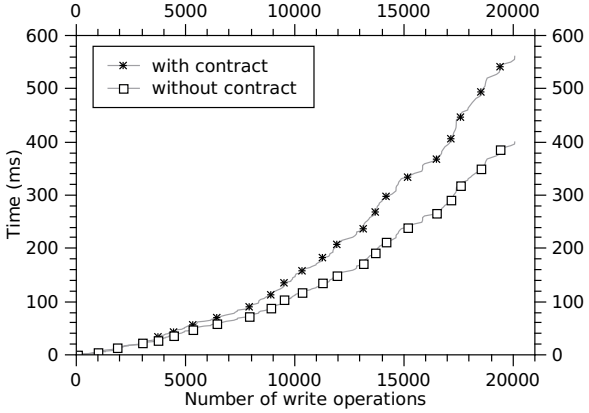


Figure 2.9: Synchronization time with growing number of *write* events

We computed for each model the total time (T) of all the synchronizations performed by a given user to build the same state of the document, $T = \sum t_i$, where t_i is the time required for the i^{th} synchronization. Figure 2.9 shows the result according to the number of write operations in the local log. We can see that the time overhead generated by using contracts is reasonable since the difference of time overhead computed for the two models increases slowly with the number of events.

3

Authentication of Logs in Multi-synchronous Collaboration

In the context of the PhD thesis of Hien Thi Thu Truong I studied how to secure collaboration logs in the Push-Pull-Clone (PPC) model [60, 34]. In this collaboration model users can misbehave by tampering history for their convenience. For instance, they can remove some content of the history or change the order of some operations from the history. This 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. This is also critical for 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, by modifying the history, a contributor may introduce security holes in the system under the name of another contributor. Therefore, there is a need to ensure integrity of the log, and in case the log was tampered, the misbehaving user should be detected.

In this chapter I describe first very shortly the collaboration model underlying PPC. Then I describe the threat model and the security properties to be ensured by our authentication mechanism. I describe next the authenticators structure and construction followed by the verification of authenticators-based logs. I also analyse the space and time complexity for authenticator construction and log verification as well as present an experimental evaluation of the proposed authentication mechanism.

3.1 Description of the Collaboration model

In chapter 2 I described the C-PPC model where the PPC model was enhanced with contracts when data content was committed and merge had to deal with both data content and contracts. Log auditing was applied to check to what extent users respected or not the contracts they accepted. Log auditing requires that logs are not tampered. Our solution for log authentication can be applied for the C-PPC model where logs contain events of types write, communication and contracts. However, we present the log authentication mechanism for the more general case of PPC model where logs contain operations of a general type.

Each site keeps a log of operations that have been performed during a collaborative process, $L = [op_1, op_2, ..., op_n]$, each operation being parameterized depending on an application domain. Operations are stored in the log in an order that is consistent with the order they were generated.

The order of addition of operations in the log is compatible with the "happened-before" relation between operations [272]. We say that op_a *happened-before* op_b, denoted as $op_a \xrightarrow{hb} op_b$, if op_b was generated on some site after op_a was either generated or received by that site. The "happened-before" relation is transitive, irreflexive and antisymmetric. Two operations op_a and op_b are said concurrent if neither $op_a \xrightarrow{hb} op_b$ nor $op_b \xrightarrow{hb} op_a$.

Logs of operations are propagated in a weakly consistent manner from one site to another one. Users decide by means of primitives push and pull when, with whom and what data to be sent and synchronized. When a pull is performed by *User_1* from the channel where *User_2* pushed his changes, in order to minimize traffic overhead, an anti-entropy mechanism is used [267]. Only the part of the log of *User_2*, that is new to *User_1* since the last time that two users synchronized, is sent to *User_1*. The remote log from *User_2* is synchronized with the local log of *User_1*. The synchronization mechanism requires to detect the concurrency and happened-before order between operations of different sites. Also the conflicts between concurrent operations must be resolved. Replicas are consistent if their states are identical when they have applied the same set of operations. For our approach, we use CRDTs [124, 108] which design operations to be commutative from the start. When reconciliation is performed, operations from the remote log, that have not been previously integrated into the local log, are simply appended to the end of the local log. The log propagation mechanism uses anti-entropy which preserves happened-before order between operations. Therefore, the reconciliation mechanism ensures happened-before order between operations as well as it allows concurrent operations to appear in logs in different orders.

As described in section 2.1.3.2, the global stream of activity of all sites is defined by a partially ordered set of operations, where operations respect the happened-before relation. Each site, however, can see only operations in his workspace that it generated locally or received from other sites. The site keeps therefore an individual log as a linearization of history built on a subset of a ground set of operations, where operations preserve the partial order. There are remaining operations of global history built on entire ground set of operations that are not visible for the site. Sites can maintain logs that are different linearizations of the same set of operations.

3.2 Security properties to be ensured

This section specifies the threat model addressed by our authentication mechanism and the security properties ensured by this mechanism.

There are two types of malicious users: insiders and outsiders. We consider that an inside adversary 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.

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.

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 operations, 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.

The following properties are addressed to authenticate operation-based history in collaborative systems.

Integrity. Adversaries are infeasible to forge a log, such as modify its entries or put new forged operations into log, without being detected. Integrity is the most important property required for securing logs in operation-based replication. Ensuring the integrity of a non-replicated document can be done easily by using cryptographic signatures or checksums. However, ensuring the integrity of a replicated document represented by a log of operations is more difficult as operations cross multi-contributors and some of them might be adversaries.

Concurrency-collision-freeness. In a history H, some operations 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.

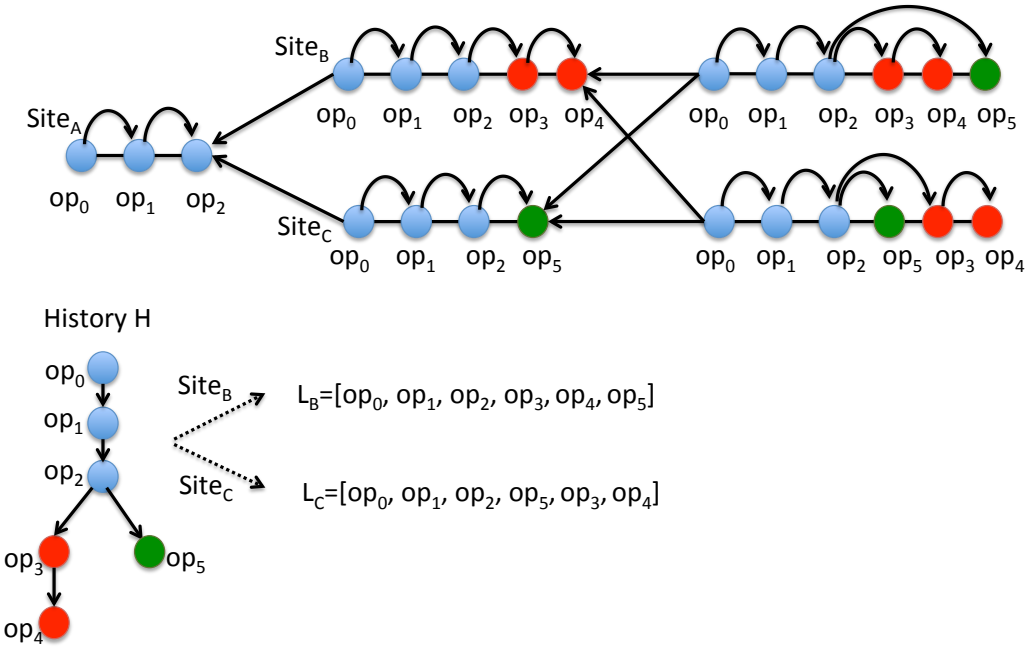


Figure 3.1: An example of different log linearizations in the PPC Model. $Site_B$ and $Site_C$ perform some concurrent operations and afterwards synchronize their changes. The arcs between operations represent the happened-before relations. $Site_B$ and $Site_C$ have different linearizations after the synchronization

In Figure 3.1, $Site_A$ performed operations op_0, op_1 and op_2 in this order. $Site_B$ and $Site_C$ pulled these changes from $Site_A$ and they performed some operations in parallel: $Site_B$ performed op_3 and op_4, while $Site_C$ performed op_5. Afterwards $Site_B$ merges its changes with $Site_C$, operation op_5 is added at the end of its log, and therefore the log of operations at its site becomes $L_B = [op_0, op_1, op_2, op_3, op_4, op_5]$. $Site_C$ merges its changes with $Site_B$, operations op_3 and op_4 are added at the end of its log, and therefore the log of operations at its site becomes $L_C = [op_0, op_1, op_2, op_5, op_3, op_4]$.

This is an example of a history H built on the ground set of operations $P = \{op_0, op_1, op_2,$

op_3, op_4, $op_5\}$ which is linearized into two logs by two sites $L_1 = [op_0, op_1, op_2, op_3, op_4, op_5]$ and $L_2 = [op_0, op_1, op_2, op_5, op_3, op_4]$ with relation "$\xrightarrow{hb}$" recorded in logs. Both logs preserve orders of all operations 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 [210, 140, 115].

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 done 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 any trusted central authority.

3.3 Authenticators structure and construction

In this section, I present our approach to construct authenticators to deter users from log tampering while preserving the above mentioned properties.

When a sending site sends its log to a receiving site, it creates an authenticator for its log that is attached to the log. The receiving site creates a new authenticator when it receives the log. We make a practical reasonable assumption [156, 164] that 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. The private key of the key pair is used to sign entries of log that prevent malicious sites modifying operations on behalf of other sites. We also use a cryptographic hash function that is 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-resistance property can be used to establish the uniqueness of logs at a certain moment when an authenticator is created.

An authenticator is a log tamper-evidence 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 $[op_1,\ op_2]$ where $op_1 \xrightarrow{hb} op_2$. When user U pushes his changes, he creates an authenticator for the sequence of operations in the log such that it should not be tampered by any other user. For instance, a receiver of this log should not be able to re-order op_1 and op_2 to change the happened-before order of op_1 and op_2.

Definition 3.3.1. *An authenticator, denoted as Υ_{site}, is defined as a tuple $\langle ID,\ SIG,\ IDE,\ PRE,\ SYN\rangle$ where:*

ID is the authenticator identifier which is a tuple $(siteID, opIDs)$ where $siteID$ is the identifier of the site which creates the authenticator and $opIDs$ is the set of operation identifiers that the authenticator is linked to;

SIG is the signature value (the private key of the site is used for the signature);

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

PRE and SYN are the identifiers of preceding authenticator and sender authenticator with which the synchronisation was performed respectively.

Definition 3.3.2. *The SIG of an authenticator Υ_{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 $\Upsilon_S^m.SIG = \sigma_S(hash(\Upsilon_S^{m-1}.SIG \mid\mid E))$ with condition that $E \neq \varnothing$; and the receiver computes $\Upsilon_R^n.SIG = \sigma_R(hash(\Upsilon_R^{n-1}.SIG \mid\mid E \mid\mid \Upsilon_S^m.SIG))$ with the condition that new update(s) from S were appended to log of R, where:*

Υ_S^m is the most recent authenticator committed by sender S;

Υ_R^n is the most recent authenticator committed by receiver R;

Υ_S^{m-1} is the preceding authenticator of Υ_S^m;

Υ_R^{n-1} is the preceding authenticator of Υ_R^n;

$E = [op_{i_1}, op_{i_2}, ..., op_{i_r}]$ is the list of subsequent changes generated after preceding authenticator;

$\sigma_{site}(\cdot)$ denotes the signature of the 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.*

When a user shares a document by sending the whole log, she creates an authenticator for log operations computed based on the preceding authenticator and new updated operations. The authenticator is signed by her private key and linked to the last operation 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 operations are constructed whenever a site sends or receives a new change to/from another site. An authenticator is created in the 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 operations). After these actions, if there are new changes that are added to the receiver's log, a new authenticator is created for this reception.

The structure of an authenticator computed by a receiver is illustrated in Figure 3.2. The events in the log $[entry_1, ..., entry_p]$ are authenticated by the last authenticator computed by the receiver site Υ_R^{n-1}. Υ_R^{n-1} is attached to the event $entry_p$. The events $[entry_{p+1}, ..., entry_q]$ are the locally executed events since the last computed authenticator. The events $[entry_{q+1}, ..., entry_r]$ are the events received from the sender that were added to the log. The events received from the sender were authenticated by Υ_S^m which was attached to the most recent event received from the sender $entry_r$. The ID of the authenticator will be the pair $< R, \Upsilon_S^m.ID.opIDs \cup entry_q.ID >$. R is the receiver's site identifier. The list of operation identifiers to which the receiver authenticator will be linked to is a union between the list of operation identifiers to which the sender authenticator is linked to and the identifier of the most recent local event. The signature SIG of the authenticator Υ_R^n is computed by signing with R's key the hash of the concatenation of the preceding receiver's authenticator Υ_R^{n-1}, the local events $[entry_{p+1}, ..., entry_q]$ and sender's authenticator Υ_S^m. The list of operation identifiers IDE used to compute the signature is the list of identifiers of the local events $[entry_{p+1}, ..., entry_q]$. The preceding authenticator PRE is Υ_R^{n-1} and the sender authenticator with which the synchronisation was performed SYN is Υ_S^m.

The algorithms supporting the creation of authenticators as well as the proof that the proposed algorithms satisfy the desired properties are presented in [34].

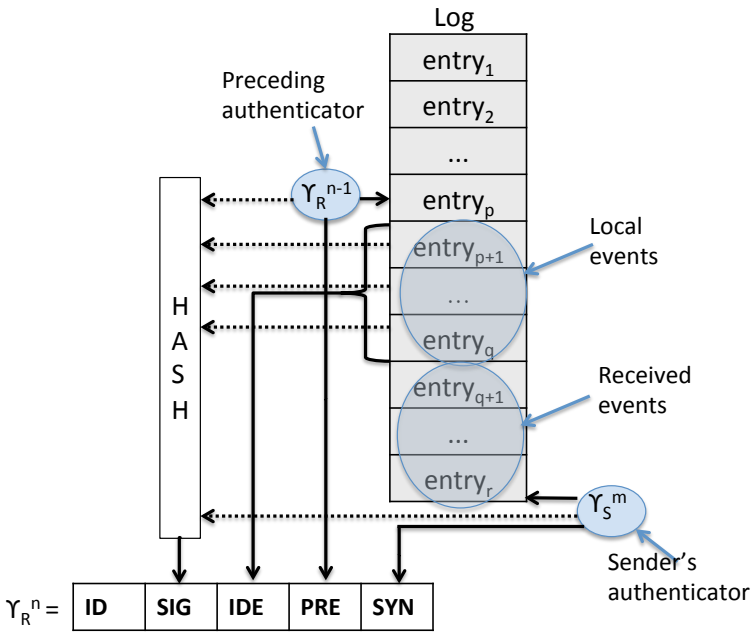


Figure 3.2: Structure of a receiver's authenticator.

3.4 Example of authenticators construction

For the example in Figure 3.1, we illustrated in Figure 3.3 the construction of authenticators:

Firstly, when $Site_A$ pushes his log to $Site_B$ and $Site_C$, it creates an authenticator Υ^1_A where:
$\Upsilon^1_A.ID = (Site_A, op_2)$ (linked to op_2),
$\Upsilon^1_A.SIG = \sigma_{Site_A}(hash(\varnothing \mid\mid op_0 \mid\mid op_1 \mid\mid op_2))$ (no previous authenticator)
$\Upsilon^1_A.IDE = \{op_0, op_1, op_2\}$
$\Upsilon^1_A.PRE = \varnothing$ (no previous authenticator)
$\Upsilon^1_A.SYN = \varnothing$ (no received remote authenticator)

When $Site_B$ pulls changes from $Site_A$ and receives the log from $Site_A$, it creates an authenticator Υ^1_B where:
$\Upsilon^1_B.ID = (Site_B, op_2)$ (linked to op_2)
$\Upsilon^1_B.SIG = \sigma_{Site_B}(hash(\varnothing \mid\mid \varnothing \mid\mid \Upsilon^1_A.SIG))$ (no previous authenticator, no local operations)
$\Upsilon^1_B.IDE = \varnothing$ (no new local operations)
$\Upsilon^1_B.PRE = \varnothing$ (no previous authenticator),
$\Upsilon^1_B.SYN = \Upsilon^1_A$

Similarly, when $Site_C$ pulls changes from $Site_A$ and receives the log from $Site_A$, it creates an authenticator Υ^1_C that has a similar structure to Υ^1_B.

$Site_B$ executes the new changes op_3 and op_4 and then pushes its changes, and therefore authenticator Υ^2_B is created. The structure of this authenticator is given below:
$\Upsilon^2_B.ID = (Site_B, op_5)$ (linked to op_4)
$\Upsilon^2_B.SIG = \sigma_{Site_B}(hash(\Upsilon^1_B.SIG \mid\mid op_3 \mid\mid op_4))$,
$\Upsilon^2_B.IDE = \{op_3, op_4\}$,
$\Upsilon^2_B.PRE = \Upsilon^1_B$,
$\Upsilon^2_B.SYN = \varnothing$ (no received remote authenticator)

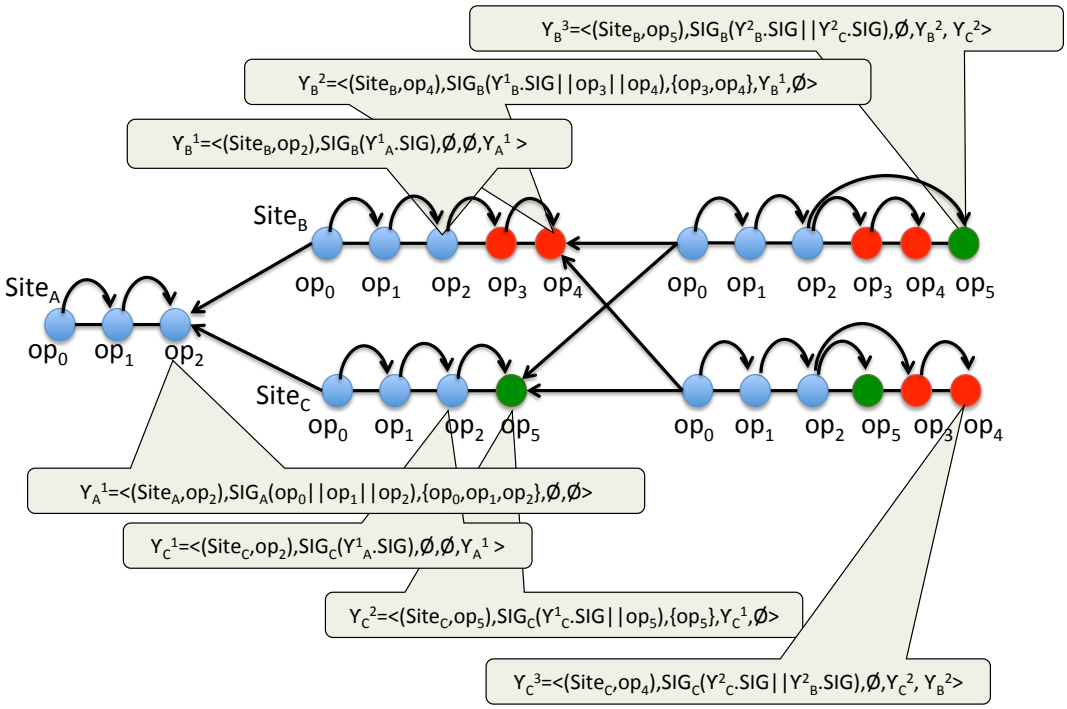


Figure 3.3: Example of constructing authenticators.

Similarly, $Site_C$ executes the new change op_5 (concurrently with op_3 and op_4) and then pushes its changes. The authenticator Υ^2_C that is created has a similar structure to Υ^2_B.

When $Site_B$ pulls the changes from $Site_C$, the log at $Site_B$ $[op_0, op_1, op_2, op_3, op_4]$ is merged with the log at $Site_C$ $[op_0, op_1, op_2, op_5]$. The changes concurrently done by $Site_C$, i.e. op_5, are appended to the log at $Site_B$. As while pull was executed new changes were added to the log, a new authenticator Υ^3_B is created. The structure of this authenticator is given below:

$\Upsilon^3_B.ID = (Site_B, op_4)$ (linked to op_5)
$\Upsilon^3_B.SIG = \sigma_{Site_B}(hash(\Upsilon^2_B.SIG \,||\, \varnothing \,||\Upsilon^2_C.SIG))$ (no local changes)
$\Upsilon^3_B.IDE = \varnothing$
$\Upsilon^3_B.PRE = \Upsilon^2_B$
$\Upsilon^3_B.SYN = \Upsilon^2_C$

Similarly, when $Site_C$ pulls the changes from $Site_B$, the authenticator Υ^3_C is created that has the structure similar with Υ^3_B.

3.5 Authenticators-based log verification

When a site receives a log of operations 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 operations preserved by valid authenticators, including checking:

- if authenticators are valid, i.e. their signatures are correct. An authenticator is checked by verifying its digital signature based on the public key.
- if the log entries are corresponding to these valid authenticators. When an authenticator passes signature checking, the content and the order of operations are taken into account

in the verification.

If all of these checkings pass, the log is authenticated. In contrast, a log with either operations not authenticated or authenticated by invalid authenticators is unauthorized. With any detection of the corrupted data or falsified order of changes, authenticators will be not valid 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. The algorithm supporting the authenticators-based log verification is presented in [34].

3.6 Space and time complexity for authenticator construction and log verification

Creation of an authenticator is $O(1)$ in time, and $O(|\Delta|)$ in storage, where Δ is the set of operations whose identifiers are kept in $\Upsilon_{Site}.IDE$. 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 had 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 operations generated by two sites.

Authenticators-based log verification has $O(1)$ complexity in space and $O(N)$ in time, where N 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.

3.7 Experimental evaluation

We evaluated experimentally our proposed solution for log authentication. 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 logs and it needs to check if remote logs are shared correctly without any tampering.

We carried out experiments on real logs of two projects Hgview [85] and one branch of OpenJDK [86] that used Mercurial as a distributed tool for source code management. 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 operations that a user commits. It is also called a log entry. We implemented our experiments in Python.

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 operations. 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.

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 checks all authenticators created by previous contributors. The results are computed by the average values of five run times. Figure 3.4 presents the experimental results for the worst case behavior. In both 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. 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 3.4, 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.

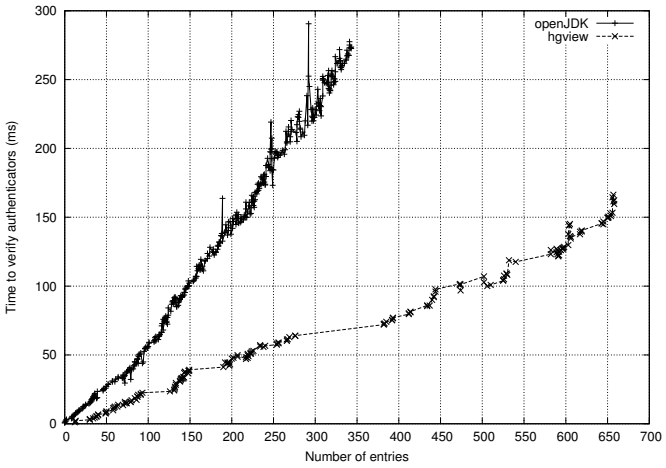


Figure 3.4: Time overhead to check authenticators created for Hgview and OpenJDK repositories.

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 logs by skipping authenticators which are already checked when previous pulls were performed.

4

Measuring trust score among users in trust game

In the context of the PhD thesis of Quang-Vinh Dang, I proposed a computational trust metric, which serves as a model for user behavior, to reflect and predict user's behavior in repeated trust games [28]. Based on users past behavior in the repeated trust game, we can model and predict their next behavior. We showed that our model is robust to several types of attacks. We validated our model by using an empirical approach against data sets collected from several trust game experiments. We proved that our trust metric is consistent with users' opinion about partners trustworthiness.

In this chapter I present our computational trust metric for measuring trust among users in the trust game. I first present the requirement for the trust metric. I then first present how to compute trust score among two users based on a single interaction among these users and then the aggregated trust score, which is the cumulative trust score over multiple interactions among users. I also describe how the trust metric deals with fluctuating behavior of users during the game. Finally, I present an evaluation of our trust metric.

4.1 Requirements

The trust score function needs to satisfy the following requirements:

1. The trust value is higher if the sending amount is higher.
2. The trust value can distinguish between different types of users: honest and malicious ones.
3. The trust value considers user behavior over time.
4. The trust value encourages a stable behavior rather than a fluctuating one.
5. The trust value is robust against attacks.

4.2 Current trust

Separate trust scores are calculated for each player for each round, i.e. for each interaction between two players. The round number is denoted as t.

In each round, two users interact by sending a non-negative amount. For senders, the maximum amount they can send is set to 10, and for receivers, the maximum amount they can send is the amount they received from the sender (i.e. three times of what the sender sent). For both roles, we normalize the sending amount of both roles to $send\_proportion_t$ as the sending proportion of a user at round t, with $t \geq 1$:

$$send\_proportion_t = \frac{sending\_amount_t}{maximum\_sending\_amount_t} \tag{4.1}$$

It is obvious that $\forall t, 0 \leq send\_proportion_t \leq 1$.

We defined the trust metric for a single interaction between users as *current_trust*. $current\_trust_t$ is a function of $send\_proportion_t$, meaning that the trustworthiness of a user in a single interaction depends on how much she sends to her partner in round t. $current\_trust_t$ should have a value between 0 and 1 inclusive. This function should satisfy the following properties (for convenience, I use the notation $f(x), f : [0,1] \rightarrow [0,1]$ for the function of $current\_trust_t$, with x being $send\_proportion_t$):

- $f(x)$ is continuous in $[0,1]$.
- $f(0) = 0$, meaning that *current_trust* is 0 if the user sends nothing.
- $f(1) = 1$, meaning that *current_trust* is 1 if the user sends the maximum possible amount.
- $f'(x) > 0$ with $x \in [0,1]$, meaning that *current_trust* is strictly increasing when *send_proportion* increases from 0 to 1. $f'(x)$ denotes the derivative of function $f(x)$.
- $f''(x) \leq 0$ with $x \in [0,1]$ meaning that the function is concave, i.e. the closer to 1 the value of *current_trust* is, the harder is to increment it.
- $f'(x^-) = f'(x^+), \forall x \in [0,1]$, meaning that the function is smooth, i.e. there is no reason that at some point the current trust increases roughly less than previously.

We proposed the following function that satisfies the above mentioned conditions:

$$current\_trust_t = \log(send\_proportion_t \times (e-1) + 1) \tag{4.2}$$

where $current\_trust_t$ is the *current_trust* function at round t and $send\_proportion_t$ is the value of *send_proportion* at round t.

4.3 Aggregated trust

We needed to calculate the aggregate trust score, which is the cumulative trust score over multiple rounds. The requirement for our aggregate function was that it has to encourage a stable behaviour rather than a fluctuating one. Our aggregate function was inspired by the trust model SecuredTrust [97] for computing trust and reputation of interacting agents in a multi-agent system. SecuredTrust computes trust of agents based on the satisfaction level of the transactions between agents in the presence of highly oscillating malicious behavior.

Similar to SecuredTrust, we defined the aggregate trust score function as an exponential averaging update function over the previous interactions between the two users that gives more weighting to the current computed trust than the simple average does.

$$aggregate\_trust_t = \alpha_t \times current\_trust_t + (1 - \alpha_t) \times aggregate\_trust_{t-1} \tag{4.3}$$

The weight α_t changes based on the accumulated deviation β_t.

$$\alpha_t = threshold + \frac{c \times \delta_t}{1 + \beta_t} \tag{4.4}$$

$$\delta_t = |current\_trust_t - current\_trust_{t-1}| \tag{4.5}$$

$$\beta_t = c \times \delta_t + (1 - c) \times \beta_{t-1} \tag{4.6}$$

The δ_t is the change of current trust value by two sequential interactions t and $t-1$ between two users, where $current\_trust_0 = 0$. We calculated δ_t to see how much a person changes her behavior since her last interaction.

$\beta_0 = 0$. c is a constant that controls to what extent we react to the recent change of the current trust. A higher value of c gives more significance to the recent change of the current trust (δ_t) than to the accumulated deviation β_{t-1}. We set $c = 0.9$.

It is easy to prove that α_t is bigger if δ_t is bigger, and vice versa. It means that, if the trust computed from the current interaction is much different from accumulated trust of all previous interactions, the current interaction will play a more important role in the final trust value.

The *threshold* is used to prevent α_t from saturating to a fixed value. *threshold* was set to 0.25.

4.4 Expected trust

$$expect\_trust_t = trend\_factor_t \times current\_trust_t + (1 - trend\_factor_t) \times aggregate\_trust_t$$

Expected trust in a user represents expected behaviour of that user and is computed based on $current\_trust_t$, $aggregate\_trust_t$ and a $trend\_factor_t$ that is tuned according to the difference between the $current\_trust_t$ and $aggregate\_trust_t$.

$$trend\_factor_t = \begin{cases} trend\_factor_{t-1} + \phi \\ \quad \text{if } current\_trust_t - aggregate\_trust_t > \epsilon \\ trend\_factor_{t-1} - \phi \\ \quad \text{if } aggregate\_trust_t - current\_trust_t > \epsilon \\ trend\_factor_{t-1} \quad \text{otherwise} \end{cases} \tag{4.7}$$

The $trend\_factor_t$ at round t represents the recent trend of user behavior. We encourage user benevolent behavior in recent interactions by increasing $trend\_factor_t$ by ϕ when $current\_trust_t$ exceeds $aggregate\_trust_t$ by the threshold ϵ. ϕ was empirically tuned to 0.1 as in SecuredTrust. As a result, the contribution of $current\_trust_t$ to the $expect\_trust_t$ increases, while the contribution of $aggregate\_trust_t$ to the $expect\_trust_t$ decreases. We have to pay attention to the

chosen value of ϵ. A small value of ϵ encourages users benevolent behavior by helping them to quickly recover from past interactions that resulted in a low $aggregate\_trust_t$. However, a very small value of ϵ can be abused by malicious users to easily recover from their malicious past actions. ϵ was empirically tuned to 0.3 as in SecuredTrust.

If $current\_trust_t$ decreases compared to $aggregate\_trust_t$ by at least the threshold ϵ, $trend\_factor_t$ is decreased by ϕ, which results into a lower contribution of $current\_trust_t$ and a higher contribution of $aggregate\_trust_t$ respectively to the $expect\_trust_t$.

If the absolute difference between $current\_trust_t$ and $aggregate\_trust_t$ is less than ϵ, $trend\_factor_t$ remains unmodified.

4.5 Dealing with fluctuating behaviour

Some malicious users may strategically oscillate between collaboration periods when they aim gaining the trust of the other users and sudden betrayal. We added a $change\_rate_t$ variable into our model to punish this kind of behaviour.

$$adj\_atf_t = \begin{cases} \frac{atf_t}{2} & \text{if } atf_t > \text{MAX\_ATF} \\ atf_t & \text{otherwise} \end{cases} \tag{4.8}$$

$$atf_t = \begin{cases} adj\_atf_{t-1} + \frac{(current\_trust_t - aggregate\_trust_t)}{2} \\ \quad \text{if } current\_trust_t - aggregate\_trust_t > \phi \\ adj\_atf_{t-1} + (aggregate\_trust_t - current\_trust_t) \\ \quad \text{if } aggregate\_trust_t - current\_trust_t > \phi \\ adj\_atf_{t-1} \quad \text{otherwise} \end{cases} \tag{4.9}$$

$$change\_rate_t = \begin{cases} 0 \quad \text{if } atf_t > \text{MAX\_ATF} \\ \cos\left(\frac{\pi}{2} \times \frac{atf_t}{\text{MAX\_ATF}}\right) \quad \text{otherwise} \end{cases} \tag{4.10}$$

In order to measure how much deviation we are willing to tolerate for trust fluctuation, we defined the accumulated trust fluctuation (atf) similar to SecuredTrust. The accumulated trust fluctuation is a non-decreasing function until it reaches the threshold value MAX_ATF. The increase depends on the change over time of user's behavior.

Both kinds of *fluctuate behaviors* are punished: the latest sending amount is suddenly higher or lower than usual behavior. However, it is obviously that the latter case (when $current\_trust_t$ decreases compared to $aggregate\_trust_t$ by at least the threshold ϕ) is more critical than the former one (when $current\_trust_t$ increases compared to $aggregate\_trust_t$ by at least the threshold ϕ). Therefore, the punishment in the latter case should be stronger, i.e. atf_t is increased with the decrease of $current\_trust_t$ with respect to $aggregate\_trust_t$, while in the former case atf_t is increased with half of the increase of $current\_trust_t$ with respect to $aggregate\_trust_t$. If the behavior is stable or it changes within the allowed range (defined by the constant ϕ), atf_t will not change.

When atf_t reaches the threshold value MAX_ATF, it means that accumulated change in user behavior over time reaches the level of betrayal and therefore $change\_rate_t$ drops to 0. Otherwise, $change\_rate_t$ decreases if atf_t increases. In order to encourage benevolent behaviour even after a betrayal, in the next round after the betrayal the value of atf_t is divided by 2.

The cosine function is used in the computation of $change\_rate_t$ as the cos function has a low degradation rate in the initial stage, and a high degradation rate in the case of repeated fluctuating behavior. It means that, if a user starts adopting a fluctuating behavior the punishment is low, but it increases fast while fluctuating behavior persists.

Finally, the trust value after round t is calculated by:

$$trust\_value_t = expect\_trust_t \times change\_rate_t \tag{4.11}$$

4.6 Evaluation of the trust metric

In this section I describe an evaluation of our trust metric by means of simulations and of real data obtained by different experimental studies of the repeated trust game.

4.6.1 Simulations

Our trust metric punishes fluctuating user behaviors and detects user behavior patterns, i.e. it distinguishes between different types of user profiles: low, medium and high. This subsection shows by means of simulations that our trust metric satisfies these criteria.

We defined three types of user profiles according to the values of *send_proportion*: low, medium and high. Similar to [91], we defined that a user with a low profile sends in average 20% of the maximum possible amount, while for a medium profile user the *send_proportion* is 50% and for a high profile user it is 80%. We also defined a *fluctuate profile* user corresponding to a user that first tries to behave well and then deviates.

By means of simulations for the mentioned user profiles, we compared the behavior of our trust metric with the average trust metric where the trust score is calculated by an average of previous sending amounts [188, 123, 175, 96, 157, 211, 104]. The behavior of our trust metric in the first 10 rounds is displayed in Fig. 4.1 and the behavior of the simple average trust metric is displayed in Figure 4.2.

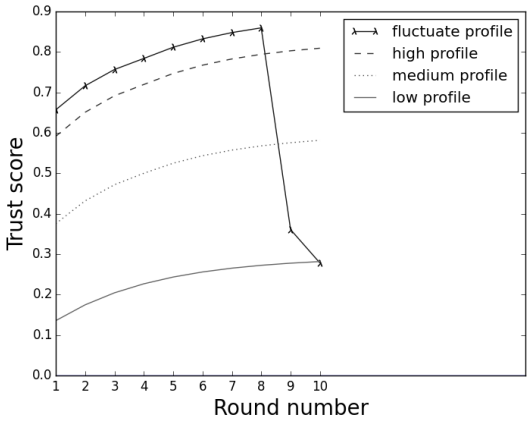


Figure 4.1: Our trust metric on different user types in the first 10 interactions.

We can easily see that, our trust metric can cope and punish the *fluctuating* behavior very well, as it reduces the trust score of *fluctuating* user to the same as of a *low profile* user. On the other side, the simple average metric cannot distinguish between *fluctuating* and *high profile* users.

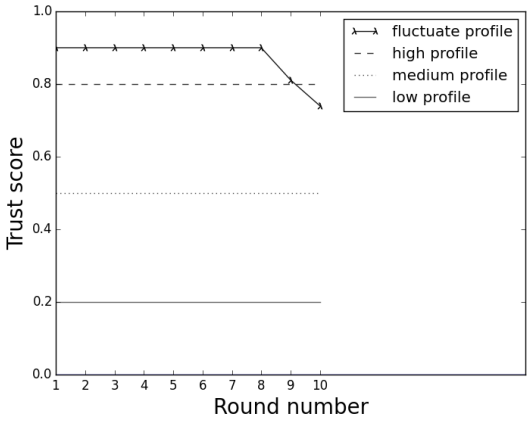


Figure 4.2: The average trust metric on different user types.

However, users do not have a constant sending behavior as we assumed above. But rather they vary their behavior around some average values and after a large number of rounds, their trust scores follow a distribution. In particular, we defined the behavior of *low profile*, *medium profile* and *high profile* as normal distributions with means of 0.2, 0.5 and 0.8 respectively, with standard deviation of 0.15 (this standard deviation value has been approximated from [104]). In what follows I show that our trust metric can distinguish between these different user types. In order to distinguish between different profiles, these distributions must satisfy the following properties:

- The trust values assigned to *fluctuating* users should be similar with the trust values assigned to *low profile* users, and should not overlap with the trust values assigned to *medium profile* users.
- The difference between two mean values should be at least the sum of two standard deviations. If we denote by $mean_{low}$, $mean_{medium}$ and $mean_{high}$ the mean values of trust scores of *bad profile*, *medium profile* and *high profile* respectively, and by std_{low}, std_{medium} and std_{high} the corresponding standard deviations, then:

$$mean_{low} + std_{low} \leq mean_{medium} - std_{medium} \tag{4.12}$$

$$mean_{medium} + std_{medium} \leq mean_{high} - std_{high} \tag{4.13}$$

- The ratio of any two variances of these distributions should not be larger than 3, as suggested by Keppel [249].

Our *current_trust* function defined by Equation 4.2 satisfies these requirements.

It is not easy to find a *current_trust* function that satisfies these above requirements. For instance, if we replace our *current_trust* formula by a new formula such as *current_trust* = *send_proportion*, this trust metric will not be able to distinguish between *medium profile* and *fluctuating* users. As shown in Fig. 4.3, after ten rounds, the new trust metric will assign overlapping trust scores to *medium profile* and *fluctuating* users, but our metric still can distinguish between these two user profiles as displayed in Figure 4.4.

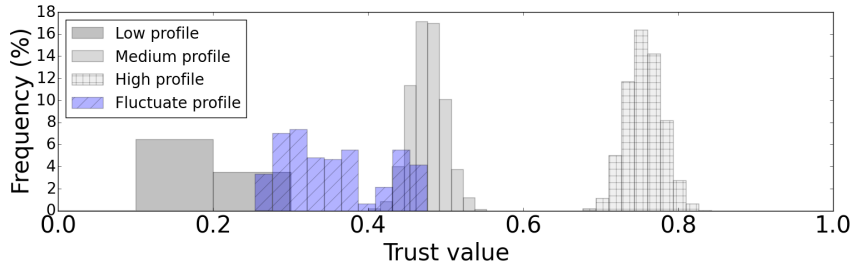


Figure 4.3: Distribution of the trust metric *current_trust* = *send_proportion* after ten rounds. The trust scores assigned to *fluctuating* users overlap with trust scores assigned to *medium profile* users.

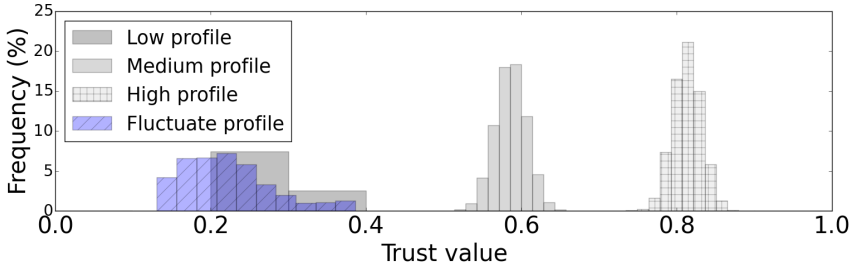


Figure 4.4: Distribution of our trust metric after ten rounds. The trust scores assigned to *fluctuating* users do not overlap with trust scores assigned to *medium profile* users.

4.6.2 Evaluation with real data

In this subsection I present an evaluation of our trust metric according to real data obtained by different experimental studies of the repeated trust game.

4.6.2.1 Power of trust metric to evaluate partner's past behaviour

We first evaluated our trust metric according to user ratings obtained during the repeated trust game experiment [195] where users could rate in each round their partners sending behavior. The three levels proposed were: negative, neutral or positive. Based on the data published in this study, we created three virtual users called *positive user*, *neutral user* and *negative user* respectively corresponding to the levels of possible ratings. These virtual users follow the average behavior of real users who have the corresponding rating.

We evaluated our trust metric applied for the behavior of these virtual users. Since we are using a continuous rating score and [195] used a discrete rating score, the two rating scores do not match completely. However, we should expect that our trust metric does not conflict with the results in [195], i.e. for any two behaviors A and B, if A was rated higher than B (for instance, positive versus neutral or positive versus negative), our trust metric should assign a higher trust score to A than B. As shown in Figure 4.5, our trust metric assigns in all cases higher trust values to *positive user* than *neutral user*, and higher trust values to *neutral user* than *negative user*. Hence, our trust metric matches users opinion about trustworthiness of partner behavior in repeated trust games.

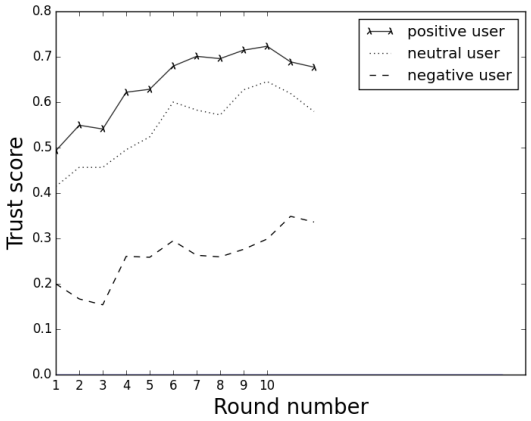


Figure 4.5: Validating trust metric with real users ratings.

4.6.2.2 Power of trust metric to evaluate partner future behaviour

In this subsection I show that our trust metric can predict the future behavior of users.

We analyzed the performance of our trust metric on three data sets comprising the behavior of a total of 174 participants in repeated trust games. The first data set was collected by a repeated trust game experiment implemented by using zTree [142] that we conducted in our laboratory (see more details in chapter 5). Our experiment involved 30 participants grouped into five sessions, each with six people. In each round, the six participants were paired randomly, and for each pair the roles of sender or receiver were assigned randomly. We ensured that during the session each user interacted exactly five times with each other user. The two other data sets were obtained by the experiments described in [96] involving 36 participants and in [98] involving 108 participants.

The datasets show that changes in user behavior in repeated trust games are very usual. Figure 4.6 illustrates the average and standard deviation of sending amount proportions of each user in the three datasets previously mentioned. The standard deviations of user sending proportions are large compared with their average sending proportions, meaning that users often change their sending behavior during the experiments. This confirms that fluctuating behavior is a fact in all three data sets and that it is important to design a trust metric that copes with this behavior.

Based on the behavior log we applied our trust metric on users behavior at a certain round, then used the output trust score as the independent variable to predict users behavior in the next round. For all rounds starting with round five, we found a high correlation between the output trust scores and user behavior in the next round. I present next the results of our analysis for rounds five and ten.

In our analysis the independent variables are the trust values for each user after fourth and ninth interaction and the dependent variables are the sending proportions of users in the fifth and tenth round. For the data in [96], we tested the relationship between our trust metric and the user behavior at round five and ten. However, because of the design of the experiment in [98], we could only test the relationship between our trust metric and user behavior at round five. Figure 4.7 displays the prediction of user sending behavior at round ten by using the data set from our experiment. Figure 4.8 displays the prediction of user sending behavior at round five by using the Bravo dataset.

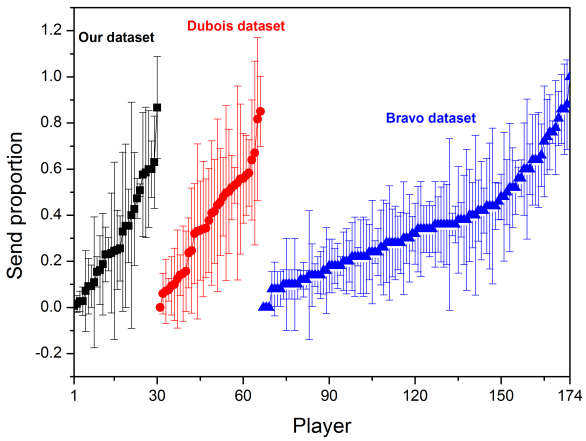


Figure 4.6: Average and standard deviation of sending proportions in datasets.

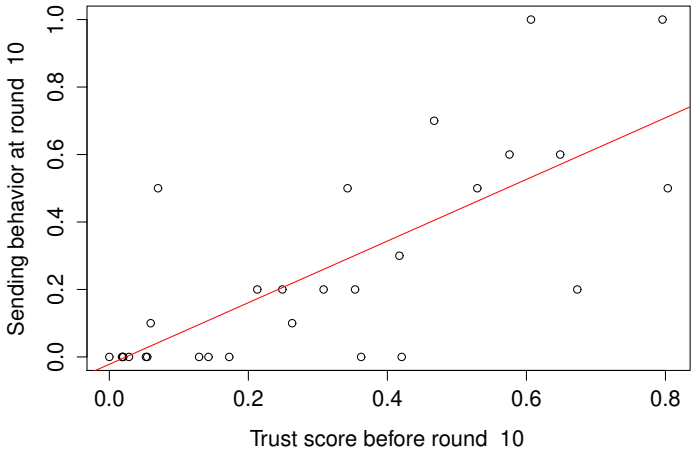


Figure 4.7: Relationship between trust metric and user behavior at round ten in our dataset.

The summary of all linear regressions previously mentioned is displayed in Table 4.1, where the independent variable (x-axis in Fig. 4.7 and Fig. 4.8) is the trust value our metric assigned to each user before a particular round, and the dependent variable is the behavior of this user in this round (y-axis in Fig. 4.7 and Fig. 4.8). We can notice that the slopes of all regressions are significant, meaning that our trust metric predicts well users behavior. Similar results were obtained for the same analysis in other rounds (i.e. a significant slope value and a positive r-value).

As there is no prior work in predicting users behavior in repeated trust game, we compared our model with two other baseline models: average model and null model. Average model predicts that, the next sending amount of a user is equal to the average of her previous sending amounts. On the other hand, the null model predicts that, the next sending amount of a user is equal to her previous sending amount. In order to compare the performance of these

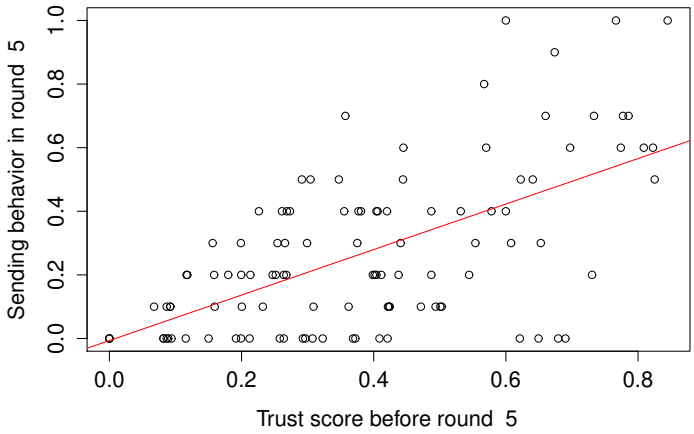


Figure 4.8: Relationship between trust metric and user behavior at round five in the Bravo dataset.

Table 4.1: Regression between trust metric and future users' behavior.

	Intercept	**Slope**	**Adj.R^2**
Our dataset (round 5)	0.071	0.701***	0.319
Our dataset (round 10)	-0.022	0.913***	0.542
Bravo dataset (round 5)	-0.006	0.715***	0.362
Dubois dataset (round 5)	0.072	0.848***	0.356
Dubois dataset (round 10)	0.027	0.855***	0.357

We denote '***' as significant level of 99.9%.

three models, we calculated the predicting values of each of these models. We computed the adjusted R^2 value for each model from round five to round ten and then calculated the average of adjusted R^2. The higher average R^2 a model achieves, the better this model is in predicting users behavior.

The comparison of performance of different models is displayed in Table 4.2. For our data and data of Dubois, we calculated an average adjusted R^2 values in predicting users behavior from round five to ten. As Bravo's dataset contains only five rounds, we computed the average adjusted R^2 values in predicting users behavior at round five. We can see that, our model outperforms the other two baseline models in predicting users behavior in repeated trust games.

Table 4.2: Comparison of R^2 values of different predicting models.

	Average model	**Null model**	**Our model**
Our data	0.42	0.43	0.55
Dubois's data	0.28	0.34	0.40
Bravo's data	0.3	0.32	0.36

We note that, a low R^2 value is usual in predicting human behavior, but in many cases, it does not mean that the prediction is useless [196]. For instance, Ashraf et al. [155] used a list of ten factors to predict users' behavior in one-shot trust game, and achieved the average R^2 of 0.25.

5

Influence of trust score on user behavior

In the context of USCoast Inria associated team, in collaboration with Valerie Shalin (Department of Psychology, Wright State University) and in the context of the PhD thesis of Quang-Vinh Dang, I developed an experimental design for testing user acceptance of the proposed trust-based collaboration model for large communities of participants described in chapter 2. We employed game theory, and trust game in particular, as a standard because it provides a solid foundation of research methods and findings, giving us a basis for expected performance. Nevertheless, the adaptation of these economics-inspired tasks is non-trivial, requiring new cover stories that reflect realistic choices and payoffs and extended models of user trust.

The question we targeted precisely was whether providing users with a trust value of their partner would help them better decide the amount of money exchanged during the trust game. For this, we adapted the trust game to a repetitive setting and we designed a trust metric that I described in chapter 4 that takes into account user behaviour in the repetitive trust game. Trust values are updated based on the satisfaction level for the exchanges during the game.

We designed an experiment that tested variations of the trust game: with and without explicit computation of user trust values according to behavior during previous collaborations [11]. This experiment was implemented using zTree that is commonly used in economy for implementing and testing such games. We followed standard practice for informed consent, user instruction, and data collection and analysis. We particularly investigated whether the availability of either trust scores or ID improves user cooperation.

In this chapter I describe the precise research questions that we addressed, the experiment that we designed and the results that we obtained.

5.1 Research questions

We studied how the availability of partner trust score and identity (ID) impacts participant behavior and the appropriateness of the used trust metric for computing trust scores in the repeated trust game. We grouped our research questions as follows:

RQ1 *Does showing partner trust score or ID change user cooperative behavior?* If so, is there a significant difference in cooperative user behavior with only trust scores relative to ID only? Is there a significant difference in user cooperative behavior resulting from the availability of both trust score and ID compared to the availability of only one of these two features? Does cooperative behavior change over time?

RQ2 *Does the trust calculation predict participant's future behavior? Do participants follow the guidance of the trust calculation?*

As senders and receivers have two different roles and may behave differently, we analyzed these research questions separately from both the senders and receivers points of view.

5.2 Experimental design

5.2.1 Participants and task

We recruited 30 participants organised into five independent groups of six participants.

In each game a participant played at least 25 rounds with the other five partners in the group in a random order, namely five rounds with each of these partners where she served as sender and receiver equally often. At the beginning of the first game each participant received 10 money units. In each round, the sender moved first. She knew how much money she had, and had to decide the amount she wanted to send to the receiver. After that, the receiver received a message indicating how much she had at the beginning of this round, how much she received from the sender, and how much she will have after having received. Then, the receiver decided how much she wanted to return.

We payed the participants an amount based on how much they virtually earned during the experiment. To assure continuing incentive throughout the session, each person who participated received a coupon of 10 euros, but the person who earned most, i.e. who had the highest payoff among other people in the group, received an additional coupon of 10 euros.

5.2.2 Independent Variables

We crossed the availability of ID and partner trust scores to create four different games as shown in Table 5.1. IDs, such as "Mr. Black" or "Mrs. Green", were assigned to participants, fixed during a game and varied between games. Trust scores were calculated as described in the previous chapter. Trust scores were always calculated for each participant in a pair, but only displayed according to experimental condition and only partner scores were available. The theoretical trust score value ranges from 0.0 to 1.0 inclusive, presented when available with two significant digits. Participants started with the neutral value of 0.5 [89].

		ID presented	
		False	**True**
Trust presented	**False**	**Simple Game**: The trust game when participants are given no information about partners	**Identity Game**: The trust game when participants are given only partner
	True	**Score Game**: The trust game when participants are given only trust scores of partners	**Combined Game**: The trust game when participants are given both trust the scores and ID of their partners

Table 5.1: Game descriptions

We calculated user reputation score as distinct from trust score by averaging all previous sending proportion amounts of that user in both roles sender or receiver.

5.2.3 Design

The experimental conditions were organized as a split-plot factorial with group as a between subjects factor and Show-ID and Show-Trust as within subjects, such that each group of six participants participated in the set of four randomly ordered games. In each round, participants were paired randomly within their group and assigned randomly the sender or receiver role. We ensured that within each game, a participant was paired with a particular other participant at least five times.

5.2.4 Dependent Measures

The four dependent measures used in our study are: **sending proportion by senders**, **sending proportion by receivers**, **average sending proportion by senders** and **average sending proportion by receivers**.

Sending proportion by senders is the net amount the sender sends to the receiver over 10, which is the maximum amount the sender could send.

Sending proportion by receivers is the net amount the receiver sends back over the amount she received after being tripled.

Other studies [180, 96] also used sending proportion measures in order to normalize the sending behavior of receivers for comparison. For example, sender A sent 6 to receiver B, and B sent back 9 to A. In this round, the net sending amount of A and B are 6 and 9 respectively, the sending proportion of A is $6/10 = 0.6$ and the sending proportion of B is $9/18 = 0.5$.

Consistent with [180, 96] for all analyses of receiver behavior, we eliminated the zero transaction between the sender and the receivers, (i.e. the sender sends 0 and the receiver is obliged to send 0), for two reasons. First, receiver behavior is completely determined by the sender, so that the receiver's behavior is not informative. Moreover, in this case, the sending proportion for the receiver (0 divided by 0) is not calculable. However, the zero-sending amount is retained in the analysis of sender behavior.

For the sender, there are exactly 375 sending proportion data points in each game (25/2 senders $\times$6 players in a group $\times$5 groups). For receiver, the number of sending proportion data points varies between 250 and 340 due to the elimination of the zero transaction.

Average sending proportion by senders is the average of sending proportions by each sender over all trials in the game. Taking an average distributes the effect of the zero transaction and also eliminates trial as a repeated factor in analysis.

Average sending proportion by receivers is the average sending proportion the receiver sends back to the sender over all trials in the game, without the zero transaction case.

There are 30 average sending proportion data points corresponding to 30 participants, for both sender and receiver. For the receiver, the zero-transaction data is removed before calculating the means.

5.2.5 Procedure

All groups participated independently using z-Tree [142] hosted on our laboratory computers. At the beginning of each session, all participants read the instructions presenting the purpose of the experiment, a short description of the four games, the payment procedure and some example screenshots illustrating the interaction of users with the z-Tree tool. Instructions informed participants that they would play the games in an arbitrary order. For each of the games participants were told what partner information would be displayed during each interaction: for the Simple Game no information, for the Identity Game the partner identity in the form of an

ID, for the Score Game a partner trust score computed according to her behaviour in previous interactions (without any details about the metric) and for the Combined Game, the partner identity and trust score. Participants did not know the number of rounds they would play in each game. After confirming that they had read and understood the instructions, participants reviewed and signed an informed consent form prior to commencing the experiment. Participants sat in different rooms to avoid any communication during the experiment. Each participant used a computer running our z-Tree application. All senders in the group finished their decision making process before proceeding to the next trial. They waited for every receiver to respond before starting a new round. This eliminated response time cues as an indication of player identity. No other means of communication or identification were available. Participants were informed of their cumulative earnings at each round. It was possible to play with a negative balance but this never occurred.

The repeated measures design resulted in 100 rounds across the four games. A session usually lasted two hours. At the end of the experiment participants filled out a questionnaire regarding general information such as university major and game preference.

5.3 Results

5.3.1 Showing trust score and ID equivalently improves the sending proportion

Showing either trust score or ID improves sending proportion for senders but showing both partner information sources does not change the sent proportion relative to one source as shown in Figure 5.1.

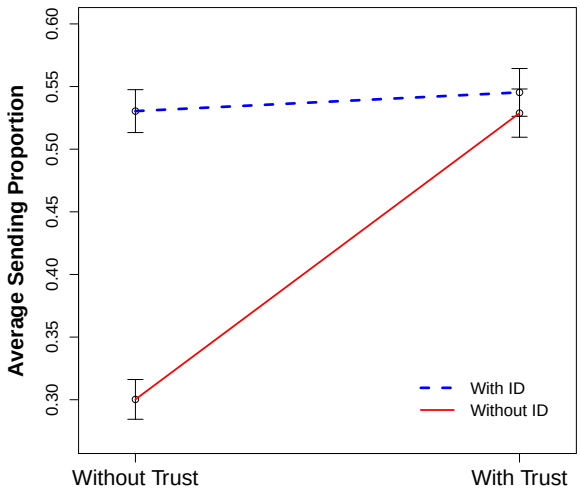


Figure 5.1: Interaction between trust score and ID availability for sender.

The same holds for receiver return proportions, i.e. showing either trust score or ID improves receiver return proportions, but showing both partner information sources does not change the sent amount relative to one source as shown in Figure 5.2.

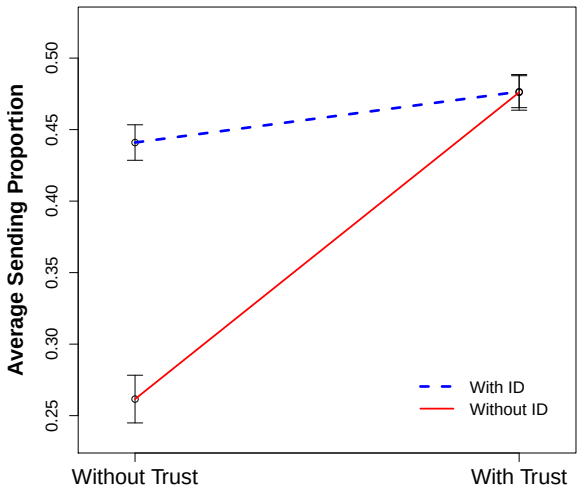


Figure 5.2: Interaction between trust score and ID availability for receiver.

This observation was verified by paired comparisons between games. [104] claimed that in large-scale the send proportion of senders as well as receiver return proportion in trust game follows the normal distribution. We computed paired t-test based confidence intervals (yoking by sender ID and receiver ID respectively) in Table 5.2 and Table 5.3 to examine differences between the Simple Game and any other tested game. We demonstrated that either trust score or ID increases with no additive effect the average sending proportions of senders and receivers respectively (results are significant with $p < 0.001$). The negative signs indicate that the sending amount of participants in Simple Game is less than the sending amount of these participants in other games.

Game in Comparison with Simple Game	**95% confidence interval**	**p-value**	**Df**
Identity Game	(-0.32, -0.14)	9e-06	29
Score Game	(-0.35, -0.13)	9e-05	29
Combined Game	(-0.35, -0.14)	7e-05	29

Table 5.2: Paired t-based confidence intervals for senders average sending proportion in the Simple Game compared to other games.

Game in Comparison with Simple Game	**95% confidence interval**	**p-value**	**Df**
Identity Game	(-0.23, -0.10)	2e-05	29
Score Game	(-0.25, -0.08)	3e-04	29
Combine Game	(-0.26, -0.11)	4e-05	29

Table 5.3: Paired t-test confidence intervals for receivers' average sending proportion in Simple Game compared to other games.

The differences between the other three games (Identity, Score and Combined Games) are

not significant, i.e. $p > 0.10$, for both senders and receivers. We conclude that $IdentityGame \approx ScoreGame \approx CombinedGame > SimpleGame$ for average sending proportion for senders and receivers.

5.3.2 Showing trust score and ID controls cooperative behavior

We addressed the claim that providing identification or trust score controls cooperative behavior. We considered the cases of non cooperation where senders/receivers send 0 and the dependence of sending behavior on trust score values.

5.3.2.1 Non cooperation by means of sending 0

The percentage of times that a sender sends 0 in Simple Game, Identity Game, Score Game and Combined Game are 33.3%, 9.3%, 13.6% and 12.7% respectively. Senders are more likely to send 0 in the Simple Game. In order to show this, we performed a logistic regression on the frequency of 0 transactions (i.e. we modeled a binary dependent variable related to the existence of 0 transaction or not) for all rounds with sending participant, Show-Trust and Show-ID as predictors. The results are statistically significant with $p < 0.001$ indicating also an interaction between Show-Trust and Show-ID, $z = 5.607$.

The percentage of times that a receiver sends 0 in Simple Game, Identity Game, Score Game and Combine Game are 36.8%, 8.5%, 8.3% and 4.5% respectively. Receivers are also more likely to return 0 in the Simple Game. The logistic regression on the frequency of 0 transactions for all trials with sending participant, Show-Trust and Show-ID as predictors indicates an interaction between Show-Trust and Show-ID $z = 3.68$, $p < 0.01$.

5.3.2.2 Dependence of sending behavior on trust score

We examined correlations between average sending behavior and trust scores for both senders and receivers. We also examined for both senders and receivers correlations between sending behavior and trust scores separately for rounds 4 and 5 when trust scores have sufficient data to stabilize.

Table 5.4 presents regression analyses for senders between average sending behavior as the criterion with sender trust values and participant trust values as predictors. Sender behavior is positively correlated with his own trust value for all games. The trust function predicts sender behavior well. Moreover, when partner trust is available, it controls sending behavior. Notably, this is the only analysis suggesting any difference between the availability of partner identity and the trust score, as partner trust score does not predict sending behavior in games without a trust score. Hence, partner trust score availability controls cooperation.

Table 5.5 presents regression analyses for receivers between average sending behavior as the criterion with sender trust values, participant trust values and amount received from the sender as predictors. Receiver behavior is positively correlated with his own trust value for all games. This confirms our ability to predict receiver cooperation (i.e., receiver trustworthiness) from past trust values. However, receiver behavior is only related to partner trust in the Combined Game. Moreover, model fits are not as good for receivers as they are for senders. We have explored models that include interactions between amount received and trust values. These often improve the relatively smaller adjusted R^2 we obtain for receiver behavior. Such models suggest the need for different trust functions for sender and receiver, to accommodate the asymmetry in their relationship.

	without Trust		with Trust	
	without ID (Simple)	**with ID** (Identity)	**without ID** (Score)	**with ID** (Combine)
Own trust	12.80***	9.31***	7.36***	8.33***
Partner trust	1.65	1.73	5.69***	4.69***
Adjusted R^2	0.85	0.75	0.88	0.89
F(2,27)	86.03	43.57	106.9	117.1

Table 5.4: Trust regression analysis for average sending behavior of senders. The table reports on t(27) values. '*' $p < 0.05$, '**' $p < 0.01$, '***' $p < 0.001$.

	without Trust		with Trust	
	no ID (Simple)	**with ID** (Identity)	**no ID** (Score)	**with ID** (Combined)
Own trust	6.003***	8.936***	4.617***	3.927***
Partner trust	0.687	0.978	0.237	-2.158*
Partner sending amount	-2.214*	-1.849	-1.469	0.587
Adjusted R^2	0.565	0.746	0.415	0.494
F(3,26)	13.53	29.36	7.854	10.44

Table 5.5: Trust regression analysis for average sending behavior of receivers. The table reports on t(26) values.'*' $p < 0.05$, '**' $p < 0.01$, '***' $p < 0.001$.

We performed separate multiple regression analyses for each game, for rounds 4 and 5 when trust scores have accrued sufficient data. The criterion variable is the sending proportion of the participants to their partners. Table 5.6 provides the results of a regression of the senders sending proportion on a model with her trust value and the trust value of her partner for both rounds. In all cases, the sender trust value predicts sending behavior. Moreover, the partner trust value also predicts sending behavior in the presence of ID or trust score information, confirming sender attention to these sources. Adjusted R^2 values range from 0.26 to 0.70, with lower values resulting from the game with no information.

	without Trust		with Trust	
	no ID (Simple)	**with ID** (Identity)	**no ID** (Score)	**with ID** (Combine)
Round 4	df = 72	df = 72	df = 72	df = 72
Own trust value	6.46***	5.80***	3.89***	7.28***
Partner's trust value	0.67	3.24**	6.98***	4.41***
Adj. R^2	0.36***	0.40***	0.66***	0.70***
Round 5	df = 72	df = 72	df = 72	df = 72
Own trust value	4.87***	7.13***	3.19**	7.11***
Partner's trust value	1.16	4.54***	7.38***	3.52***
Adj. R^2	0.26***	0.55***	0.67***	0.70***

Table 5.6: Trust regression analysis on senders sending proportion with t-values for individual slope tests. '*' $p < 0.05$, '**' $p < 0.01$, '***' $p < 0.001$.

Table 5.7 provides comparable information for receiver behavior, answering the question of how well we can predict whether a participant is trustworthy. These regression models included own trust value, partner trust value and the amount just received (i.e., three times the amount

sent). While receivers were never aware of their own trust values, our trust function is a good predictor of receiver behavior *when trust score is not provided.* This does support our claim that the trust function is a good predictor of trustworthiness. However, the mere presence of trust scores in the trust score conditions dampens its predictive capability. Partner trust value is rarely predictive. Receivers did not rely on this systematically. Adjusted R^2values range from 0.08 to 0.45 with higher values in the conditions where trust score is not provided.

	without Trust		**with Trust**	
	no ID (Simple)	**with ID** (Identity)	**no ID** (Score)	**with ID** (Combine)
Round 4	df = 42	df = 62	df = 60	df = 60
Own trust value	3.41**	7.21***	1.98	1.76
Partner's trust value	0.02	1.40	1.63	0.50
Amount received	-0.53	-1.62	-2.37*	0.33
Adj. R^2	0.18*	0.45***	0.08	0.10*
Round 5	df = 39	df = 61	df = 61	df = 60
Own trust value	4.21***	3.56***	3.06**	1.09
Partner's trust value	0.14	2.10*	0.74	1.53
Amount received	-2.19*	0.06	-1.75	-0.16
Adj. R^2	0.30***	0.29***	0.13*	0.09*

Table 5.7: Trust regression analysis on receivers sending proportion with t-values for individual slope tests. '*' $p < 0.05$, '**' $p < 0.01$, '***' $p < 0.001$.

5.4 Discussion

We analyzed our research questions distinguishing between sender and receiver points of view.

RQ1 *Does showing partner trust score or ID change user cooperative behavior?*

Overall increases in the proportion returned and reductions in the frequency of 0 unit returns for both senders and receivers show evidence of the influence of trust score or ID on cooperation. We also demonstrated that the availability of a trust score has a similar impact on boosting cooperation as the availability of identities. We also showed that the availability of both features has no additional benefit to cooperation as the availability of only one of these features.

RQ2 *Does the trust calculation predict participants future behavior ?*

With respect to senders, we provide excellent predictive models for average behavior. These average models always depend positively on own trust values, and on partner trust values when trust values are available. Sender behavior is also well modeled at the round level, always depending upon own trust values and on partner trust values for all games except the Simple Game. Senders are attending to the specific values shown for partners. We note that the effect is not to encourage blind cooperation, but rather cooperation in response to the available information. Low partner trust scores elicit low sending amounts.

With respect to receivers, models of average return proportions behavior do depend on own-trust. This supports a claim for some ability to predict trustworthiness. Models at the round level are best when the trust score is not available. This unexpected result is possibly due to strategic differences in receiver behavior. While receiver models did include an additional factor (partner sending amount), our general impression is that the models of receiver behavior are more complex than models of sender behavior and not yet accommodated by the trust function used. Moreover, unlike the sender, deceitful receiver behavior is not punished until the

subsequent round. These considerations suggest that the trust function should differ for sender and receiver.

We have not identified the source of leverage on the success of the trust function for senders. We noted three different influences: the specification of partners, the management of change over time and the treatment of variability, particularly punishment in response to non-cooperative behavior. Limitations in the receiver model highlight this claim, where the role of amount received may interact with the partner trust values in ways that we have not yet captured.

The trust function used considers only the sending proportion as a parameter, but not for instance the amount sent by the partner. This trust model fits well for a sender that initiates the interaction by sending an initial amount. But the trustworthiness value associated with a receiver should depend not only on the return proportion but also on the amount received. We might consider associating a higher trustworthiness with a receiver that received 6 and returned 1 than to someone that received 30, but returned the same proportion. The receiver that received 30 obtained the maximum possible amount but did not reciprocate the granted trust. These suggestions further reinforce the need to consider the measurement of trust from a psychometric perspective, capturing the relationship between physical quantities and behavioral response.

We have demonstrated that the presence of partner information benefits cooperative behavior. As our experiment suggested, the trust score has a similar effect on cooperative behavior relative to ID. Therefore, trust scores may complement current systems that employ ID to identify users, helping users define the trustworthiness of their connections. While it is possible for participants to change their ID in on line systems, they cannot change the trust level other participants assigned to them. If a trust score is available, participants do not need to remember individuals by name, nor do they need to assess previous experience with imprecise mental calculations. Instead, they can make decisions based on their partner current trust score. Such a system greatly facilitates engagement with large scale collaborative networks. Our proposed solution for computing partner trust scores scales well with the number of partners.

Part III

Perspectives

Secure and trustworthy collaborative data management

In the last years my research work concerned trustworthy large scale peer-to-peer collaborative data management characterised by the absence of a central authority where users maintain their data and decide with which other users they want to share their data. In this context of large scale peer-to-peer collaboration, my future research project will focus on novel synchronisation approaches for complex data requiring a composition replication mechanisms, on security aspects of collaborative data management and on evaluation of collaborators according to their past contributions.

1 Composition of replication mechanisms

The behaviour of various optimistic algorithms is poorly understood under intensive use in different architectural settings, with different scenarios and different kind of data structures. Each family of algorithms has advantages and disadvantages. Operational transformation has the advantage of being a generic and guided approach including a generic concurrency control algorithm for all data types and operation transformation functions specific to an application domain. Operational transformation works well on groupware settings with few number of users, but does not adapt well to large scale settings with a large number of users making frequent modifications. Furthermore, several existing solutions impose constraints either on the system architecture such as client/server-based architectures [234] or on the processing order of generated modifications [215] both of which limit extension to large scale settings. Solutions that do not impose these constraints use data structures that are a function of the number of users such as state vectors [260, 222] or/and of the number of operations such as history buffers [260, 222, 146]. Hence, the time complexity of existing algorithms is proportional to the total number of users and to the total number of operations which threatens scalability to large settings. Alternatively, CRDTs [154, 125, 124] provide an optimistic replication solution with weaker requirements, whose underlying data structures are independent of the total number of users and operations. However, CRDTs are not generic as they implement different solutions for concurrency handling for different data types and they suffer from meta-data overhead.

There are two families of CRDT approaches, namely state-based and operation-based [100]. State-based CRDTs directly ship their modified state to other replicas which merge received states with their own state. As the merge function is associative, commutative, and idempotent, state convergence is achieved independently of the order of merging states and of repeated merge execution. The possible states of a state-based CRDT must form a join-semilattice. Each modification inflates the state and thus each state supersedes previously merged state. We can represent the set of states as a partially ordered set where each pair of two states has a

corresponding join (least upper bound) produced by the merge function. State-based CRDTs are suitable for unreliable communication as convergence is achieved in the presence of message duplications and message losses. As each state supersedes previously merged states, state-based CRDTs natively offer causal consistency.

In operation-based CRDTs, replicas propagate updates to every other replica that applies them to their local replica state. The execution of an operation is done in two phases *prepare* and *effect*. *prepare* is performed only on the local replica and looks at the operation and current state to produce a message that aims to represent the operation, which is then shipped to all replicas. Once received, the representation of the operation is applied remotely using *effect*. The *effect* operation must be reliably executed in all replicas, where it updates the replica state. It has been shown that if *effect* operations are delivered in causal order, replicas will converge to the same state if concurrent *effect* operations commute [100]. If *effect* operations may be delivered without respecting causal order, then all *effect* operations must commute. If *effect* operation may be delivered more than once, then all *effect* operations must be idempotent. Most operation-based CRDT designs require exactly-once and causal delivery.

Operation-based and state-based synchronisation have both advantages and disadvantages. Operation-based synchronisation is more efficient since you only ship small updates, but requires exactly once causally-ordered broadcast. On the other side, state-based synchronisation requires only reliable broadcast, but features communication overhead by shipping the whole state. Rather than applying one approach or the other, a system should be able to implement both approaches and depending on the size of the updates and the size of the data state could decide which approach to apply at a certain moment. If only several updates were performed on a shared state, propagating the complete state to remote replicas is inefficient and an operation-based approach is more suitable. If a large number of updates are performed on a same small state, propagating all operations is inefficient and a state-based operation that propagates the final state would be more suitable. Similarly, in a real-time collaborative editing application where operation-based synchronisation is necessary in order to ship updates in real-time, synchronisation with a peer that has been offline for a long period of time could be done by means of a state-based synchronisation where complete states are shipped. This combination of state-based and operation-based has not been studied.

I would also like to study composition of state-based CRDTs with operation-based CRDTs and with operation transformation. In [88] general techniques were provided for composing state-based CRDTs using lattice merge functions. However, composition of various operation-based CRDTs and of operation-based CRDTs with operational transformation has not been previously studied. Operation-based CRDTs require that effect operations commute which is a difficult task from the design point of view. Adding new operations to a well-defined operation-based CRDT requires that the new operations commute with the existing operations. Two solutions can be studied. The first possible solution consists in creating a new CRDT which could be composed with the initial CRDT. If these new operations are commutative among them, a new CRDT can be designed. However, it has to be studied what are the necessary conditions in order that the composition of the two new CRDTs is a CRDT. The second possible solution that can be studied is to apply operational transformation between the new operations and the operations belonging to the initial CRDT. Indeed, operational transformation allows transforming an operation against other conflicting operations by making them commute after the transformation. However, conditions of combining operation-based CRDTs with operational transformation have to be studied.

Several state-based and operation-based CRDTs were formalised in the literature. However, these CRDTs were not studied from the point of view of time and space complexity. I plan to

analyze the complexity of these algorithms according to a theoretical point of view, but also from a practical point of view with the help of real collaboration traces on implementations of these CRDTs.

I am also interested in the design of specific CRDTs that require global invariants to be maintained for correctness. This is the case of relational databases that have to maintain different integrity constraints such as uniqueness, reference integrity and numeric constraints. Recently, together with Weihai Yu (University of Tromso), we designed Conflict-free Replicated Relations (CRRs) based on a specific CRDT for relational databases [19]. This work would allow us to study maintenance of integrity constraints over CRRs.

In collaborative editors a selective undo allows a user to undo an earlier operation, regardless of when, where and by which user the operation was generated. There is currently no generally applicable undo support as stated in the manifesto on CRDTs [87]. In collaboration with Weihai Yu and Luc André, we proposed in [29] a layered commutative replicated data type (CRDT) that supports selective undo of string-wise operations. In [20], in collaboration with Weihai Yu and Victorien Elvinger, we designed a generic support of selective undo for delta-based CRDTs, an optimised class of state-based CRDTs. Our solution uses lattice theory and proposes an abstraction that captures the semantics of concurrent undo and redo operations through equivalence classes. However, a generic undo solution has to be proposed for operation-based CRDTs.

2 Secure collaborative data management

I want to propose a security mechanism adapted for distributed collaborative systems without a central authority. The security mechanism has to deal with user access rights to the shared documents as well as with end-to-end encryption of data with key management suitable for user dynamic groups. The mechanism has to be easy to use and will be tested with users.

Existing access control mechanisms feature two main difficulties in the context of collaborative systems. The first one relates to the design of security policies that are convenient for all partners involved in a collaboration. A challenging issue is how to manage partner joining and leaving to the group. The second difficulty is the necessity of sending at each user action an access request and waiting for its answer from a trusted central authority which maintains the security policies. This delay is critical for the real-time collaboration where the number of updates is high. Moreover, in the case of a federation of organisations agreeing on such an authority is almost impossible. Furthermore, in order to maintain its autonomy with respect to its autorisation management, it should be possible for a partner to revoke previous granted rights without contacting an external authority. Indeed, it is possible that once several users collaborated, they wish that some of them do not have anymore access to new modifications on shared documents. However, it is normal that users that had access to a document version and that contributed to the document keep this document version even though they do not have anymore access to next versions of the document.

Besides access control, end-to-end encryption is very important for ensuring security of mutable data in the collaboration. Large collaborative service providers such as Dropbox, iCloud and GoogleDrive adopted encryption solutions in order to store only encrypted version of shared documents. However, for facilitating the usage of their services, encryption keys are stored by the service providers which gives them the possibility of accessing the non encrypted data and being subject of different attacks. My goal is to adopt suitable end-to-end encryption for collaboration over mutable data where messages sent between participants are end-to-end encrypted

and servers do not need accessing a non encrypted data.

Another aspect related to security of collaborative data is group key management. Suppose that a group of users collaborate on a document and exchange data encrypted with a symmetric key. If the group decides to eliminate a user from the group, this user should not have anymore access to the document. A new key has to be generated and shared among the remaining group members which requires several rounds of communication among users. This process has performance issues and leads to interruptions of participants work. In a large scale collaboration where several users join and leave very often the group, the generation and sharing of group keys is critical. My goal is to propose an efficient group key management suitable for large dynamic groups.

In order to avoid the use of a central server that stores all data, access rights as well as data are replicated. I propose to develop CRDT algorithms for the synchronisation of access rights that will be composed with CRDT algorithms for data synchronisation.

In a classical access control mechanism, when a user requests access to a resource on a remote machine, the remote machine checks whether the user has been granted an access to the resource and if it is the case the user can access it. In a decentralised collaboration data are already replicated and if no additional measures are considered, a user can have access to data independently of the access rights. A data encryption mechanisms has to be conceived in order to deal with access to replicated data. In this case the user could check the list of authorised users and share with them the data encryption keys.

The main challenge is to compose CRDTs for data with CRDTs for access control and preserve causality between these two types of data. Indeed it is important to determine whether a user has the right to execute an operation on a document, i.e. whether an access right was granted or revoked before he executed an operation. Issues appear when users execute operations on the document while their rights of executing these operations are concurrently revoked: decisions have to be taken whether these operations can be kept or they have to be cancelled.

This security mechanism could be implemented into our MUTE peer-to-peer collaborative editor and tested by means of user studies.

3 Users trust evaluation based on the quality of their past contributions

In a large scale peer-to-peer collaboration a critical question is how to choose your collaborators that you can trust in order to share them your data. I propose to automatically compute a trust value for a user based on his previous collaborative editing behavior. A main challenge in this task is how to compute this trust value according to past collaboration in order to be able to predict future user behavior.

In previous work I focused on predicting the quality of the content of a document collaboratively written in Wikipedia [28, 26, 14, 23]. In future work I would like to focus on predicting the quality of a user contribution based on the quality of his past contributions. For this I propose using the trust metric proposed in [27] for trust game that predicts participants future behavior based on their past behavior while dealing with fluctuations of their behavior. This trust metric can be applied for user contributions in Wikipedia by considering that user interactions in the trust game are user contributions throughout an article revisions. A critical challenge is how to define the quality of a user contribution. For this I would like to investigate different quality metrics based on the edit length (i.e. the length of contribution in terms of characters) but also

on the edit longevity (i.e. how long an edit of a user persists).

In previous work I investigated whether providing users with a trust value of their partner would help them better decide the amount of money exchanged during the trust game [11]. However, the experiment was done at a small scale, with groups of six users. I plan additional adaptations to the experiment design including simulated participants to permit repeated samples of large scale collaboration. The simulated users behavior can be inferred from real users data, or predefined with some patterns. I would like also to experimentally investigate whether providing users with the trust value of their partner based on their direct interaction is more useful to users than computing the reputation of a specific user [213].

This metric as well as our proposed validation methodology received a lot of interest from the Fair&Smart company for their personal data management platform that respects General Data Protection Regulation (GDPR) for computing trust between the different users of this platform and will be transferred in the context of a DeepTech project financed by BPI and coordinated by Fair&Smart (2020-2023). In the context of this project I will lead the activity of computing trust scores among users and enterprises based their behavior during their past interactions. This trust score will predict future behavior of a user or enterprise. For instance, it is possible to compute a trust score based on the respect or violations of consents established before the collaboration. Consents will be logged in the platform and an auditing mechanism would be able to check whether they were respected or not. In order to test the feasibility of this trust mechanism between users before its implementation on the Fair&Smart platform I propose an experimental design based on game theory. I propose to investigate a contract-based game such as [150] which would model by means of contracts the interactions between users and enterprises. An implementation of this kind of game would allow us to organise experimental studies with users in order to ensure that the proposed trust score metric is well reflecting user perception.

Bibliography

Personal publications

Theses

[1] **Claudia-Lavinia Ignat**. July 2006. "Maintaining Consistency in Collaboration over Hierarchical Documents". PhD thesis. ETH Zurich. DOI: 10.3929/ethz-a-005361349.

[2] **Claudia-Lavinia Ignat**. June 2000. "Implementation of a Framework supporting Web-based Information Modelling". Diploma thesis. ETH Zurich.

Books

[3] Iosif Ignat and **Claudia-Lavinia Ignat**. 2007d. *Data Structures and Algorithms (rédigé en Roumain)*. Editura Albastra. ISBN: 978-973-650-213-2.

[4] Iosif Ignat and **Claudia-Lavinia Ignat**. 2002b. *Computer programming. Algorithms Description and the Fundaments of C/C++ (rédigé en Roumain)*. Cluj-Napoca, Romania: Editura Albastra. ISBN: 973-650-093-4.

[5] Iosif Ignat and **Claudia-Lavinia Ignat**. 2001a. *Data Structures and Algorithms - guide to lab sessions (rédigé en Roumain)*. Cluj-Napoca, Romania: U.T. Pres. ISBN: 973-8335-09-4.

International editorial activities

[6] **Claudia-Lavinia Ignat**, Pernille Bjørn, and Prasun Dewan. 2018a. "Special Issue: ECSCW 2018: The 16th European Conference on Computer-Supported Cooperative Work, The International Venue on Practice-centred Computing and the Design of Cooperation Technologies". In: *ECSCW 2018 - 16th European Conference on Computer-Supported Cooperative Work*. Vol. 27. 3–6. Nancy, France: Springer Verlag, pp. 291–1020. DOI: 10.1007/s10606-018-9334-0. HAL: hal-01917238.

[7] Michael S. MacFadden, Agustina Ng, **Claudia-Lavinia Ignat**, Ning Gu, and Chengzheng Sun. Feb. 2017. "The Fifteenth International Workshop on Collaborative Editing Systems". In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW 2017, Portland, OR, USA, February 25 - March 1, 2017, Companion Volume*, pp. 351–354. DOI: 10.1145/3022198.3022653. HAL: hal-01652759.

[8] Michael MacFadden, Agustina, Ning Gu, **Claudia-Lavinia Ignat**, Haifeng Shen, David Sun, and Chengzheng Sun. Feb. 2014. "The fourteenth international workshop on collaborative editing systems". In: *Computer Supported Cooperative Work, CSCW '14, Baltimore, MD, USA, February 15-19, 2014, Companion Volume*, pp. 331–334. DOI: 10.1145/2556420.2558861. HAL: hal-01088762.

[9] Agustina, Ning Gu, **Claudia-Lavinia Ignat**, Michael MacFadden, Haifeng Shen, David Sun, and Chengzheng Sun. Feb. 2013. "The thirteenth international workshop on collaborative editing systems". In: *Computer Supported Cooperative Work, CSCW 2013, San Antonio, TX, USA, February 23-27, 2013, Companion Volume*, pp. 299–300. DOI: 10.1145/2441955.2442028. HAL: hal-00924922.

[10] Agustina, Ning Gu, **Claudia-Lavinia Ignat**, Pascal Molli, Haifeng Shen, David Sun, and Chengzheng Sun. Feb. 2012. "The twelfth international workshop on collaborative editing systems". In: *CSCW '12 Computer Supported Cooperative Work, Seattle, WA, USA, February 11-15, 2012 - Companion Volume*, pp. 25–26. DOI: 10.1145/2141512.2141526. HAL: hal-01652759.

Articles in peer-reviewed international journals

[11] **Claudia-Lavinia Ignat**, Quang-Vinh Dang, and Valerie L. Shalin. Sept. 2019. "The Influence of Trust Score on Cooperative Behavior". In: *ACM Trans. Internet Technol.* 19.4, 46:1–46:22. DOI: 10.1145/3329250. HAL: hal-02307981.

[12] Hoai Le Nguyen and **Claudia-Lavinia Ignat**. Dec. 2018. "An Analysis of Merge Conflicts and Resolutions in Git-Based Open Source Projects". In: *Computer Supported Cooperative Work* 27.3-6, pp. 741–765. DOI: 10.1007/s10606-018-9323-3. HAL: hal-01917249.

[13] **Claudia-Lavinia Ignat**, Luc André, and Gérald Oster. Mar. 2017. "Enhancing rich content wikis with real-time collaboration". In: *Concurrency and Computation: Practice and Experience*. DOI: 10.1002/cpe.4110. HAL: hal-01404024.

[14] Quang Dang and **Claudia-Lavinia Ignat**. Nov. 2016. "Quality assessment of wikipedia articles: a deep learning approach". In: *SIGWEB Newsletter* 2016.Autumn, 5:1–5:6. DOI: 10.1145/2996442.2996447. HAL: hal-01393227.

[15] Hien Thi Thu Truong, **Claudia-Lavinia Ignat**, and Pascal Molli. Sept. 2012. "A contract-extended push-pull-clone model for multi-synchronous collaboration". In: *Journal of Cooperative Information Systems* 21.03, pp. 221–262. DOI: 10.1142/S0218843012410031. HAL: hal-00761038.

[16] **Claudia-Lavinia Ignat** and Moira C. Norrie. Dec. 2008. "Multi-level editing of hierarchical documents". In: *Journal of Computer Supported Cooperative Work* 17.5-6, pp. 423–468. DOI: 10.1007/s10606-007-9071-2. HAL: inria-00337480.

[17] **Claudia-Lavinia Ignat** and Moira C. Norrie. Dec. 2006. "Customisable collaborative editing supporting the work processes of organisations". In: *Computers in Industry* 57.8, pp. 758–767. DOI: 10.1016/j.compind.2006.04.005. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCI06.pdf.

Articles in proceedings of peer-reviewed international conferences

[18] Hoai Le Nguyen and **Claudia-Lavinia Ignat**. Sept. 2020. "Time-position characterization of conflicts: a case study of collaborative editing". In: *The 26th International Conference on Collaboration Technologies and Social Computing (CollabTech 2020)*. Tartu, Estonia. DOI: 10.1007/978-3-030-58157-2_5. HAL: hal-02983533v1.

[19] Weihai Yu and **Claudia-Lavinia Ignat**. Oct. 2020. "Conflict-Free Replicated Relations for Multi-Synchronous Database Management at Edge". In: *The IEEE International Conference on Smart Data Services (SMDS 2020)*. Beijing, China, pp. 113–121. DOI: 10.1109/SMDS49396.2020.00021. HAL: hal-02983557.

[20] Weihai Yu, Victorien Elvinger, and **Claudia-Lavinia Ignat**. Dec. 2019. "A Generic Undo Support for State-Based CRDTs". In: *The International Conference On Principles of Distributed Systems (OPODIS 2019)*. Neuchâtel, Switzerland, 14:1–14:17. DOI: 10.4230/LIPIcs.OPODIS.2019.14. HAL: hal-02370231.

[21] Hoang-Long Nguyen, Jean-Philippe Eisenbarth, **Claudia-Lavinia Ignat**, and Olivier Perrin. July 2018. "Blockchain-Based Auditing of Transparent Log Servers". In: *DBSec 2018 - 32nd Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy*. Proceeding of Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference. Bergamo, Italy, pp. 21–37. DOI: 10.1007/978-3-319-95729-6_2. HAL: hal-01917636.

[22] Quang-Vinh Dang and **Claudia-Lavinia Ignat**. Oct. 2018. "Link-Sign Prediction in Dynamic Signed Directed Networks". In: *CIC 2018 - 4th IEEE International Conference on Collaboration and Internet Computing*. Philadelphia, United States, pp. 36–45. DOI: 10.1109/CIC.2018.00-42. HAL: hal-01881035.

[23] Quang-Vinh Dang and **Claudia-Lavinia Ignat**. 2017b. "An end-to-end learning solution for assessing the quality of Wikipedia articles". In: *Proceedings of the 13th International Symposium on Open Collaboration, OpenSym 2017, Galway, Ireland, August 23-25, 2017*, 4:1–4:10. DOI: 10.1145/3125433.3125448. HAL: hal-01559693.

[24] Weihai Yu, Gérald Oster, and **Claudia-Lavinia Ignat**. Sept. 2017. "Handling Disturbance and Awareness of Concurrent Updates in a Collaborative Editor". In: *CDVE 2017 - 14th International Conference on Cooperative Design, Visualization, and Engineering*. Ed. by Yuhua Luo. Vol. 10451. LNCS - Lecture Notes in Computer Science. Mallorca, Spain: Springer. DOI: 10.1007/978-3-319-66805-5_5. HAL: hal-01652656.

[25] Quang-Vinh Dang and **Claudia-Lavinia Ignat**. Oct. 2017. "dTrust: a simple deep learning approach for social recommendation". In: *The 3rd IEEE International Conference on Collaboration and Internet Computing (CIC-17)*. San Jose, United States, pp. 209–218. DOI: 10.1109/CIC.2017.00036. HAL: hal-01578316.

[26] Quang Vinh Dang and **Claudia-Lavinia Ignat**. June 2016. "Quality Assessment of Wikipedia Articles without Feature Engineering". In: *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries, JCDL 2016*. Newark, NJ, USA, pp. 27–30. DOI: 10.1145/2910896.2910917. HAL: hal-01351226.

[27] Quang Vinh Dang and **Claudia-Lavinia Ignat**. Aug. 2016. "Computational Trust Model for Repeated TrustGames". In: *Proceedings of the 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications - TrustCom 2016*. Tianjin, China, pp. 34–41. DOI: 10.1109/TrustCom.2016.0043. HAL: hal-01351250.

[28] Quang Vinh Dang and **Claudia-Lavinia Ignat**. Nov. 2016. "Measuring Quality of Collaboratively Edited Documents: the case of Wikipedia". In: *Proceedings of the 2nd IEEE International Conference on Collaboration and Internet Computing - CIC 2016*. Pittsburgh, USA, pp. 266–275. DOI: 10.1109/CIC.2016.044. HAL: hal-01388614.

[29] Weihai Yu, Luc André, and **Claudia-Lavinia Ignat**. June 2015. "A CRDT Supporting Selective Undo for Collaborative Text Editing". In: *Proceedings of the 10th International Federated Conference on Distributed Computing Techniques - DisCoTec 2015, Distributed Applications and Interoperable Systems - DAIS*. Grenoble, France, pp. 193–206. DOI: 10.1007/978-3-319-19129-4_16. HAL: hal-01246212.

[30] **Claudia-Lavinia Ignat**, Gérald Oster, Olivia Fox, Valerie L. Shalin, and François Charoy. Sept. 2015. "How Do User Groups Cope with Delay in Real-Time Collaborative Note Taking". In: *Proceedings of the European Conference on Computer Supported*

Cooperative Work - ECSCW 2015. Oslo, Norway, pp. 223–242. DOI: 10.1007/978-3-319-20499-4_12. HAL: hal-01238831.

[31] **Claudia-Lavinia Ignat**, Gérald Oster, Meagan Newman, Valerie Shalin, and François Charoy. Sept. 2014. "Studying the effect of delay on group performance in collaborative editing". In: *Proceedings of International Conference on Cooperative Design, Visualization and Engineering (CDVE 2014)*. Seattle, Washington, USA, pp. 191–198. DOI: 10.1007/978-3-319-10831-5_29. HAL: hal-01088815.

[32] Luc André, Stéphane Martin, Gérald Oster, and **Claudia-Lavinia Ignat**. Oct. 2013. "Supporting Adaptable Granularity of Changes for Massive-scale Collaborative Editing". In: *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2013)*. Austin, Texas, USA, pp. 50–59. DOI: 10.4108/icst.collaboratecom.2013.254123. HAL: hal-00903813.

[33] Mehdi Ahmed-Nacer, Pascal Urso, **Claudia-Lavinia Ignat**, and Gérald Oster. June 2012. "Evaluation de l'occupation mémoire des CRDTs pour l'édition collaborative temps-réel mobile". In: *Proceedings of 8èmes journées francophones Mobilité et Ubiquité (Ubimob 2012)*. Anglet,France, pp. 1–12. HAL: hal-00710258.

[34] Hien Thi Thu Truong, **Claudia-Lavinia Ignat**, and Pascal Molli. Oct. 2012. "Authenticating Operation-based History in Collaborative Systems". In: *Proceedings of the ACM International Conference on Supporting Group Work (Group 2012)*. Sanibel Island, Florida, USA: ACM, pp. 131–140. DOI: 10.1145/2389176.2389197. HAL: hal-00761045.

[35] Hien Thi Thu Truong and **Claudia-Lavinia Ignat**. July 2011. "Log Auditing for Trust Assessment in Peer-to-Peer Collaboration". In: *Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC 2011)*. Cluj-Napoca, Romania, pp. 207–214. DOI: 10.1109/ISPDC.2011.38. HAL: inria-00636177.

[36] Mehdi Ahmed-Nacer, **Claudia-Lavinia Ignat**, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. Sept. 2011. "Evaluating CRDTs for real-time document editing". In: *Proceedings of the 2011 ACM Symposium on Document Engineering (DocEng 2011)*. Nominated for best paper award. Mountain View, CA, USA, pp. 103–112. DOI: 10.1145/2034691.2034717. HAL: inria-00629503.

[37] Hien Thi Thu Truong, **Claudia-Lavinia Ignat**, Mohamed-Rafik Bouguelia, and Pascal Molli. Oct. 2011. "A contract-extended push-pull-clone model". In: *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2011)*. Best paper award. Orlando, Florida, USA, pp. 211–220. DOI: 10.1142/S0218843012410031. HAL: hal-00761038.

[38] **Claudia-Lavinia Ignat**, Gérald Oster, and Pascal Molli. Sept. 2009. "DooSo6: Easy collaboration over shared projects". In: *Proceedings of the 6th International Conference on Cooperative Design, Visualization and Engineering (CDVE 2009)*. Luxembourg, Luxembourg, pp. 56–63. DOI: 10.1007/978-3-642-04265-2_8. HAL: inria-00431680.

[39] **Claudia-Lavinia Ignat**. Aug. 2008. "Annotation of Concurrent Changes in Collaborative Software Development". In: *Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2008)*. Cluj-Napoca, Romania, pp. 137–144. DOI: 10.1109/ICCP.2008.4648365. HAL: inria-00337401.

[40] Stavroula Papadopoulou, **Claudia-Lavinia Ignat**, Gérald Oster, and Moira C. Norrie. Sept. 2008. "Intra/Inter-document Change Awareness for Co-authoring of Web Sites". In: *Proceedings of the International Conference on Web Information Systems Engineering (WISE 2008)*. Auckland, New Zealand: Springer, pp. 90–105. DOI: 10.1007/978-3-540-85481-4_9. HAL: inria-00343814.

[41] **Claudia-Lavinia Ignat** and Gérald Oster. Sept. 2008. "Peer-to-peer Collaboration over XML Documents". In: *Proceedings of the 5th International Conference on Cooperative Design, Visualization and Engineering (CDVE 2008)*. Mallorca, Spain: Springer, pp. 66–73. DOI: 10.1007/978-3-540-88011-0. HAL: inria-00343815.

[42] **Claudia-Lavinia Ignat** and Gérald Oster. Nov. 2008. "Awareness of Concurrent Changes in Distributed Software Development". In: *Proceedings of the International Conference on Cooperative Information Systems (CoopIS 2008)*. Monterrey, Mexico, pp. 456–464. DOI: 10.1007/978-3-540-88871-0. HAL: inria-00343812.

[43] **Claudia-Lavinia Ignat**, Stavroula Papadopoulou, Gérald Oster, and Moira C. Norrie. Nov. 2008. "Providing Awareness in Multi-synchronous Collaboration Without Compromising Privacy". In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW 2008)*. San Diego, California, USA: ACM Press, pp. 659–668. DOI: 10.1145/1460563.1460665. HAL: inria-00343817.

[44] **Claudia-Lavinia Ignat** and Gérald Oster. June 2007. "Flexible Reconciliation of XML Documents in Asynchronous Editing". In: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS 2007)*. Funchal, Madeira, Portugal, pp. 359–366. DOI: 10.5220/0002407103590366. HAL: inria-00139708.

[45] Hala Skaf-Molli, **Claudia-Lavinia Ignat**, Charbel Rahhal, and Pascal Molli. July 2007. "New Work Modes For Collaborative Writing". In: *International Conference on Enterprise Information Systems and Web Technologies (EISWT 2007)*. Orlando, Florida, USA, pp. 176–182. HAL: inria-00129222.

[46] **Claudia-Lavinia Ignat**, Gérald Oster, Pascal Molli, Michelle Cart, Jean Ferrié, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. Nov. 2007. "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)*. White Plains, New York, USA: IEEE Computer Society, pp. 474–483. DOI: 10.1109/COLCOM.2007.4553878. HAL: inria-00188329.

[47] **Claudia-Lavinia Ignat** and Moira C. Norrie. June 2006. "Supporting Customised Collaboration over Shared Document Repositories". In: *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*. Luxembourg, Grand Duché de Luxembourg, pp. 190–204. DOI: 10.1007/11767138_14. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCAiSE06.pdf.

[48] **Claudia-Lavinia Ignat** and Moira C. Norrie. Sept. 2006. "Flexible Collaboration over XML Documents". In: *Proceedings of the International Conference on Cooperative Design, Visualization and Engineering (CDVE 2006)*. Mallorca, Spain, pp. 267–274. ISBN: 3-540-44494-7. DOI: 10.1007/11863649_33. HAL: inria-00139708.

[49] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2006. "Draw-Together: Graphical Editor for Collaborative Drawing". In: *Proceedings of the International Conference on Computer Supported Cooperative Work (CSCW 2006)*. Banff, Alberta, Canada, pp. 269–278. DOI: 10.1145/1180875.1180917. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCSCW06.pdf.

[50] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2006. "Flexible Definition and Resolution of Conflicts through Multi-level Editing". In: *Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*. Georgia, Atlanta, USA, pp. 1–10. DOI: 10.1109/COLCOM.2006.361869. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCollaborateCom06.pdf.

[51] Stavroula Papadopoulou, **Claudia-Lavinia Ignat**, Gérald Oster, and Moira C. Norrie. Nov. 2006. "Increasing Awareness in Collaborative Authoring through Edit Profiling". In: *Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*. Georgia, Atlanta, USA, pp. 1–9. DOI: 10.1109/COLCOM.2006.361864. HAL: inria-00109093.

[52] **Claudia-Lavinia Ignat** and Moira C. Norrie. June 2005. "Operation-based Merging of Hierarchical Documents". In: *Proceedings of the CAiSE'05 Forum, 17th International Conference on Advanced Information Systems Engineering*. Porto, Portugal, pp. 101–106. ISBN: 972-752-078-2. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCAiSEForum05.pdf.

[53] **Claudia-Lavinia Ignat** and Moira C. Norrie. May 2004. "Extending real-time editing systems with asynchronous communication". In: *Proceedings of the 8th International Conference on CSCW in Design (CSCWD 2004)*. Xiamen, P.R. China: IEEE Press, pp. 528–533. ISBN: 7-5062-6209-6. DOI: 10.1109/CACWD.2004.1349246. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCSCWD04.pdf.

[54] **Claudia-Lavinia Ignat** and Moira C. Norrie. June 2004. "CoDoc: Multi-mode Collaboration over Documents". In: *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004)*. Riga, Latvia: Springer, pp. 580–594. ISBN: 3-540-22151-4. DOI: 10.1007/978-3-540-25975-6_41. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCAiSE04.pdf.

[55] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2004. "Grouping in Collaborative Graphical Editors". In: *Proceedings of the International Conference on Computer Supported Cooperative Work (CSCW 2004)*. Chicago, Illinois, USA: ACM Press, pp. 447–456. ISBN: 1-58113-810-5. DOI: 10.1145/1031607.1031682. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCSCW04.pdf.

[56] **Claudia-Lavinia Ignat** and Moira C. Norrie. Sept. 2003. "Customizable Collaborative Editor Relying on treeOPT Algorithm". In: *Proceedings of the 8th European Conference on Computer-supported Cooperative Work (ECSCW 2003)*. Helsinki, Finland, pp. 315–334. ISBN: 1-4020-1573-9. DOI: 10.1007/978-94-010-0068-0_17. URL: https://dl.eusset.eu/handle/20.500.12015/2436.

Articles in proceedings of peer-reviewed international workshops

[57] Hoai Le Nguyen and **Claudia-Lavinia Ignat**. Feb. 2017. "Parallelism and conflicting changes in Git version control systems". In: *IWCES'17 - The Fifteenth International Workshop on Collaborative Editing Systems*. Portland, Oregon, United States. HAL: hal-01588482.

[58] Quang-Vinh Dang and **Claudia-Lavinia Ignat**. May 2016. "Performance of real-time collaborative editors at large scale: User perspective". In: *Internet of People Workshop, 2016 IFIP Networking Conference*. Proceedings of 2016 IFIP Networking Conference, Networking 2016 and Workshops. Vienna, Austria, pp. 548–553. DOI: 10.1109/IFIPNetworking.2016.7497258. HAL: hal-01351229.

[59] Luc André, **Claudia-Lavinia Ignat**, and Gérald Oster. Feb. 2012. "Collaboration over Wiki Content". In: *The Twelfth International Workshop on Collaborative Editing Systems, CSCW 2012, IEEE Distributed Systems online*, pp. 1–21. ISSN: 1541-4922. HAL: hal-00726833.

[60] Hien Thi Thu Truong, **Claudia-Lavinia Ignat**, and Pascal Molli. Feb. 2012. "Securing Logs in Operation-based Collaborative Editing". In: *The Twelfth International Workshop*

on Collaborative Editing Systems, CSCW 2012, IEEE Distributed Systems online, pp. 1–6. ISSN: 1541-4922. HAL: hal-00761058.

[61] Hien Thi Thu Truong, Mohamed-Rafik Bouguelia, **Claudia-Lavinia Ignat**, and Pascal Molli. June 2011. "Collaborative Editing with Contract over Friend-to-Friend Networks". In: *Atelier Protection de la Vie Privée / Géolocalisation et Vie Privée, APVP2011*, pp. 1–6. HAL: inria-00636184.

[62] **Claudia-Lavinia Ignat**, Gérald Oster, Pascal Molli, and Hala Skaf-Molli. Nov. 2007. "A Collaborative Writing Mode for Avoiding Blind Modifications". In: *Ninth International Workshop on Collaborative Editing, GROUP 2007, IEEE Distributed Systems online*. ISSN: 1541-4922. HAL: inria-00182424.

[63] Stavroula Papadopoulou, **Claudia-Lavinia Ignat**, and Moira C. Norrie. Nov. 2007. "Awareness Model to Overview Modifications in Collaborative Graphical Authoring Tools". In: *Ninth International Workshop on Collaborative Editing, GROUP 2007, IEEE Distributed Systems online*. ISSN: 1541-4922. HAL: inria-00182427.

[64] **Claudia-Lavinia Ignat**, Moira C. Norrie, and Gérald Oster. Nov. 2006. "Handling Conflicts through Multi-level Editing in Peer-to-peer Environments". In: *Eighth International Workshop on Collaborative Editing, CSCW 2006, IEEE Distributed Systems online*. ISSN: 1541-4922. HAL: inria-00109098.

[65] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2005. "Flexible Merging of Hierarchical Documents". In: *Seventh International Workshop on Collaborative Editing, GROUP 2005, IEEE Distributed Systems online*. ISSN: 1541-4922. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCEW05.pdf.

[66] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2004. "Operation-based versus State-based Merging in Asynchronous Graphical Collaborative Editing". In: *Sixth International Workshop on Collaborative Editing, CSCW 2004, IEEE Distributed Systems online*. ISSN: 1541-4922. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCEW04.pdf.

[67] **Claudia-Lavinia Ignat** and Moira C. Norrie. Sept. 2003. "Grouping/Ungrouping in Graphical Collaborative Editing Systems". In: *Fifth International Workshop on Collaborative Editing, ECSCW 2003, IEEE Distributed Systems online*. ISSN: 1541-4922. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCSCW04.pdf.

[68] **Claudia-Lavinia Ignat** and Moira C. Norrie. Nov. 2002. "Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems". In: *Fourth International Workshop on Collaborative Editing, CSCW 2002, IEEE Distributed Systems online*. ISSN: 1541-4922. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCEW02.pdf.

[69] **Claudia-Lavinia Ignat** and Moira C. Norrie. June 2001. "Framework supporting Rapid Information Modelling". In: *Eighth Doctoral Consortium in Conjunction, CAiSE 2001*. DOI: 10.17169/refubium-23076. URL: https://members.loria.fr/CIgnat/files/pdf/IgnatCAiSEDC01.pdf.

Posters and demos in proceedings of peer-reviewed international conferences

[70] Hoang-Long Nguyen, **Claudia-Lavinia Ignat**, and Olivier Perrin. Apr. 2018. "Trusternity: Auditing Transparent Log Server with Blockchain". In: *Companion of the The Web Conference 2018*. Lyon, France, pp. 79–80. DOI: 10.1145/3184558.3186938. HAL: hal-01883589.

[71] Matthieu Nicolas, Victorien Elvinger, Gérald Oster, **Claudia-Lavinia Ignat**, and François Charoy. Aug. 2017. "MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor". In: *ECSCW 2017 - 15th European Conference on Computer-Supported Cooperative*

Work. Proceedings of 15th European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos 3. Sheffield, United Kingdom: EUSSET, pp. 1–4. DOI: 10.18420/ecscw2017_p5. HAL: hal-01655438.

Research and technical reports

[72] Afshin Moin and **Claudia-Lavinia Ignat**. Feb. 2014. *Hybrid Weighting Schemes For Collaborative Filtering*. Research Report. Inria Nancy-Grand Est. HAL: hal-00947194.

[73] **Claudia-Lavinia Ignat**, Gérald Oster, Meagan Newman, Valerie Shalin, and François Charoy. Dec. 2013. *Measurement of Remote Response Delay in Multi-Synchronous Collaborative Editing*. Research Report RR-8419. Inria Nancy-Grand Est. HAL: hal-00917317.

[74] Hyun-Gul Roh and **Claudia-Lavinia Ignat**. Aug. 2012. *Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation*. Research Report RR-8044. Inria Nancy-Grand Est, p. 33. HAL: hal-00724232v2.

[75] Hien Thi Thu Truong and **Claudia-Lavinia Ignat**. Dec. 2010. *A Log Auditing Approach for Trust Management in Peer-to-Peer Collaboration*. Research Report RR-7472. Inria Nancy-Grand Est, p. 16. HAL: inria-00542852.

[76] **Claudia-Lavinia Ignat**, Gérald Oster, Pascal Molli, and Hala Skaf-Molli. May 2007. *Gasper: A Collaborative Writing Mode for Avoiding Blind Modifications*. Research Report RR-6204. Inria Nancy-Grand Est, p. 20. HAL: inria-00150013v3.

[77] **Claudia-Lavinia Ignat**, Gérald Oster, Pascal Molli, Michèle Cart, Jean Ferrié, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. Sept. 2007. *A Comparison of Optimistic Approaches to Collaborative Editing of Wiki Pages*. Research Report RR-6278. Inria Nancy-Grand Est, p. 19. HAL: inria-00169395v2.

References

[78] *Practical Meta-Analysis Effect Size Calculator* (2017a). http://campbellcollaboration.org/escalc/html/EffectSizeCalculator-SMD10.php.

[79] *Git.* Fast version control system (2005a). http://git.or.cz/.

[80] *IkiWiki* (2015a). http://ikiwiki.info/.

[81] *Linux Kernel* (2015b). http://kernel.org/.

[82] *Ruby on Rails. The popular MVC framework for Ruby* (2015c). http://rubyonrails.org/.

[83] *Samba. Opening Windows to a Wider World* (2015d). http://www.samba.org/.

[84] *TeamEdit. A collaborative text editor* (n.d.[a]). http://teamedit.sourceforge.net/.

[85] Logilab.org. *hgview*. http://www.logilab.org/project/hgview.

[86] OpenJDK. *OpenJDK*. http://openjdk.java.net.

[87] Nuno M. Preguiça, Carlos Baquero, and Marc Shapiro. 2019a. "Conflict-Free Replicated Data Types CRDTs". In: *Encyclopedia of Big Data Technologies*. Springer, Cham. DOI: 10.1007/978-3-319-63962-8_185-1. URL: https://doi.org/10.1007/978-3-319-63962-8%5C_185-1.

[88] Carlos Baquero, Paulo Sérgio Almeida, Alcino Cunha, and Carla Ferreira. 2017c. "Composition in State-based Replicated Data Types". In: *Bulletin of the EATCS* 123. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/507.

[89] Hussein A. Abbass, Garrison W. Greenwood, and Eleni Petraki. 2016a. "The N-Player Trust Game and its Replicator Dynamics". In: *IEEE Trans. Evolutionary Computation* 20.3, pp. 470–474.

[90] Roy J. Lewicki and Chad Brinsfield. 2015e. "Trust research: measuring trust beliefs and behaviours". In: *Handbook of research methods on trust*. Edward Elgar Publishing, pp. 46–64.

[91] Cody Buntain and Jennifer Golbeck. 2015f. "Trust transfer between contexts". In: *Journal of Trust Management* 2.1, 6. DOI: 10.1186/s40493-015-0017-1.

[92] Rigzin Angmo and Mukesh Sharma. 2014a. "Performance evaluation of web based automation testing tools". In: *Proceedings of the 5th International Conference Confluence, The Next Generation Information Technology Summit (Confluence)*, pp. 731–735.

[93] Ed Coyne and Timothy R. Weil. 2013a. "ABAC and RBAC: Scalable, Flexible, and Auditable Access Management". In: *IT Professional* 15.3, pp. 14–16. DOI: 10.1109/MITP.2013.37.

[94] Hien Thi Thu Truong. 2012a. "A Contract-based and Trust-aware Collaboration Model. (Un modèle de collaboration basé sur les contrats et la confiance)". PhD thesis. University of Lorraine, Nancy, France. URL: https://tel.archives-ouvertes.fr/tel-00769076.

[95] Tom Hvitved, Felix Klaedtke, and Eugen Zalinescu. 2012b. "A trace-based model for multiparty contracts". In: *Journal of Logic and Algebraic Programming* 81.2. Formal Languages and Analysis of Contract-Oriented Software (FLACOS'10), pp. 72–98. ISSN: 1567-8326.

[96] Dimitri Dubois, Marc Willinger, and Thierry Blayac. 2012c. "Does players' identification affect trust and reciprocity in the lab?" In: *Jour. of Econ. Psychology* 33.1, pp. 303–317.

[97] Anupam Das and Mohammad Mahfuzul Islam. 2012d. "SecuredTrust: A Dynamic Trust Computation Model for Secured Communication in Multiagent Systems". In: *IEEE Trans. Dependable Sec. Comput.* 9.2, pp. 261–274.

[98] Giangiacomo Bravo, Flaminio Squazzoni, and Riccardo Boero. 2012e. "Trust and partner selection in social networks: An experimentally grounded model". In: *Social Networks* 34.4, pp. 481–492.

[99] Carlos Molina-Jimenez, Santosh Shrivastava, and Massimo Strano. June 2012. "A Model for Checking Contractual Compliance of Business Interactions". In: *IEEE Transactions on Services Computing* 5.2, pp. 276–289. ISSN: 1939-1374.

[100] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011a. "Conflict-free replicated data types". In: *Proceedings of the 13th international conference on Stabilization, safety, and security of distributed systems*. SSS'11. Grenoble, France: Springer-Verlag, pp. 386–400. ISBN: 978-3-642-24549-7.

[101] Daniel Le Métayer. 2011b. "Formal Methods as a Link between Software Code and Legal Rules". In: *Proceedings 9th International Conference on Software Engineering and Formal Methods, SEFM 2011*, pp. 3–18.

[102] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 2011c. "Proactive Detection of Collaboration Conflicts". In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE '11. Szeged, Hungary: ACM, pp. 168–178. ISBN: 978-1-4503-0443-6.

[103] Hyun-Gul Roh, Myeongjae Jeon, Jinsoo Kim, and Joonwon Lee. 2011d. "Replicated abstract data types: Building blocks for collaborative applications". In: *Journal of Parallel and Distributed Computing* 71.3, pp. 354–368. ISSN: 0743-7315.

[104] Noel D Johnson and Alexandra A Mislin. 2011e. "Trust games: A meta-analysis". In: *Journal of Economic Psychology* 32.5, pp. 865–889.

[105] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Dec. 2011. "Depot: Cloud Storage with Minimal Trust". In: *ACM Trans. Comput. Syst.* 29.4, 12:1–12:38. ISSN: 0734-2071. DOI: 10.1145/2063509.2063512.

[106] Du Li and Rui Li. 2010a. "An Admissibility-Based Operational Transformation Framework for Collaborative Editing Systems". In: *Computer Supported Cooperative Work* 19.1, pp. 1–43. DOI: 10.1007/s10606-009-9103-1. URL: https://doi.org/10.1007/s10606-009-9103-1.

[107] Daniel Le Métayer, Manuel Maarek, Valérie Viet Triem Tong, Eduardo Mazza, Marie-Laure Potet, Nicolas Craipeau, Stéphane Frénot, and Ronan Hardouin. 2010b. "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*, pp. 135–144.

[108] Stéphane Weiss, Pascal Urso, and Pascal Molli. 2010c. "Logoot-Undo: Distributed Collaborative Editing System on P2P networks (to appear)". In: *IEEE Transactions on Parallel and Distributed Systems*.

[109] Ted Wobber, Thomas L. Rodeheffer, and Douglas B. Terry. 2010d. "Policy-based access control for weakly consistent replication". In: *Proceedings of the 5th European conference on Computer systems*. EuroSys '10. Paris, France: ACM, pp. 293–306. ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755943.

[110] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. 2010e. "Usage control in computer security: A survey". In: *Computer Science Review* 4.2, pp. 81–99.

[111] Krishna P.N. Puttaswamy, Catherine C. Marshall, Venugopalan Ramasubramanian, Patrick Stuedi, Douglas B. Terry, and Ted Wobber. June 2010. "Docx2Go: Collaborative Editing of Fidelity Reduced Documents on Mobile Devices". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys 2010*. San Francisco, CA, USA: ACM Press, pp. 345–356. ISBN: 978-1-60558-985-5. DOI: 10.1145/1814433.1814467.

[112] Rodrigo Rodrigues and Peter Druschel. Oct. 2010. "Peer-to-peer Systems". In: *Commun. ACM* 53.10, pp. 72–82. ISSN: 0001-0782. DOI: 10.1145/1831407.1831427. URL: http://doi.acm.org/10.1145/1831407.1831427.

[113] Di Ma and Gene Tsudik. 2009a. "A new approach to secure logging". In: *TOS* 5.1.

[114] Keshnee Padayachee and Jan H. P. Eloff. 2009b. "Adapting usage control as a deterrent to address the inadequacies of access controls". In: *Computers & Security* 28.7, pp. 536–544.

[115] Attila Altay Yavuz and Peng Ning. 2009c. "BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed Systems". In: *Proceedings of the 2009 Annual Computer Security Applications Conference*. ACSAC '09. Washington, DC, USA: IEEE Computer Society, pp. 219–228. ISBN: 978-0-7695-3919-5.

[116] Gordon J. Pace and Gerardo Schneider. 2009d. "Challenges in the Specification of Full Contracts". In: *Proceedings of the 7th International Conference on Integrated Formal Methods*. IFM '09. Düsseldorf, Germany: Springer-Verlag, pp. 292–306. ISBN: 978-3-642-00254-0.

[117] Scott A. Crosby and Dan S. Wallach. 2009e. "Efficient Data Structures For Tamper-Evident Logging". In: *USENIX Security Symposium*, pp. 317–334.

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

[119] Federico Stagni, Alvaro Arenas, Benjamin Aziz, and Fabio Martinelli. 2009g. "On Usage Control in Data Grids". In: *IFIPTM*, pp. 99–116.

[120] Alexander Pretschner, Florian Schütz, Christian Schaefer, and Thomas Walter. 2009h. "Policy Evolution in Distributed Usage Control". In: *Electr. Notes Theor. Comput. Sci.* 244, pp. 109–123.

[121] Ragib Hasan, Radu Sion, and Marianne Winslett. 2009i. "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance". In: *FAST*, pp. 1–14.

[122] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. 2009j. "Towards a theory of accountability and audit". In: *Proceedings of the 14th European conference on Research in computer security*. ESORICS'09. Saint-Malo, France: Springer-Verlag, pp. 152–167.

[123] Avner Ben-Ner and Louis Putterman. 2009k. "Trust, communication and contracts: An experiment". In: *Jour. of Econ. Behav. & Organization* 70.1, pp. 106–121.

[124] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Lec tia. June 2009. "A commutative replicated data type for cooperative editing". In: *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009*. Montréal, Canada, pp. 395–403. DOI: http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.20.

[125] Stéphane Weiss, Pascal Urso, and Pascal Molli. June 2009. "Logoot : A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks". In: *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009*. Montreal, Quebec, Canada: IEEE Computer Society, pp. 404–412. DOI: http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.75.

[126] Alberto Montresor and Márk Jelasity. Sept. 2009. "PeerSim: A Scalable P2P Simulator". In: *Proceedings of the 9th International Conference on Peer-to-Peer, P2P'09*. Seattle, WA, pp. 99–100.

[127] R. Accorsi. Sept. 2009. "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*, pp. 94–110.

[128] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. 2008a. "Authenticated hash tables". In: *Proceedings of the 15th ACM conference on Computer and communications security*. CCS '08. Alexandria, Virginia, USA: ACM, pp. 437–448. ISBN: 978-1-59593-810-7. DOI: 10.1145/1455770.1455826.

[129] Steffen Altmann, Thomas Dohmen, and Matthias Wibral. 2008b. "Do the reciprocal trust less?" In: *Econ. Letters* 99.3, pp. 454–457.

[130] Robert Slonim and Ellen Garbarino. 2008c. "Increases in trust and altruism from partner selection: Experimental evidence". In: *Exp. Econ.* 11.2, pp. 134–153.

[131] Alexander Pretschner, Manuel Hilty, David A. Basin, Christian Schaefer, and Thomas Walter. 2008d. "Mechanisms for usage control". In: *ASIACCS*, pp. 240–244.

[132] Di Ma. 2008e. "Practical forward secure sequential aggregate signatures". In: *Proceedings of the 2008 ACM symposium on Information, computer and communications security*. ASIACCS '08. Tokyo, Japan: ACM, pp. 341–352. ISBN: 978-1-59593-979-1.

[133] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi S. Sandhu. 2008f. "Toward a Usage-Based Security Framework for Collaborative Computing Systems". In: *ACM Trans. Inf. Syst. Secur.* 11.1.

[134] Krukow Karl, Nielsen Mogens, and Sassone Vladimiro. Jan. 2008. "A Logical Framework for History-based Access Control and Reputation Systems". In: *Journal of Computer Security* 16.1, pp. 63–101. ISSN: 0926-227X.

[135] Xin Luo, Yixian Yang, and Zhengming Hu. Dec. 2008. "Controllable Delegation Model Based on Usage and Trustworthiness". In: *Knowledge Acquisition and Modeling, 2008. KAM'08. International Symposium on*, pp. 745–749.

[136] Romain Robbes and Michele Lanza. 2007a. "A Change-based Approach to Software Evolution". In: *Electronic Notes in Theoretical Computer Science* 166, pp. 93–109. ISSN: 1571-0661. DOI: http://dx.doi.org/10.1016/j.entcs.2006.06.015.

[137] Cristian Prisacariu and Gerardo Schneider. 2007b. "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. Paphos, Cyprus: Springer-Verlag, pp. 174–189. ISBN: 978-3-540-72919-8.

[138] Sandro Etalle and William H. Winsborough. 2007c. "A posteriori compliance control". In: *Proceedings of the 12th ACM symposium on Access control models and technologies*. SACMAT '07. Sophia Antipolis, France: ACM, pp. 11–20. ISBN: 978-1-59593-745-2. DOI: 10.1145/1266840.1266843.

[139] Manuel Hilty, Alexander Pretschner, Christian Schaefer, and Thomas Walter. 2007e. "DUKE - Distributed Usage Control Enforcement". In: *POLICY*, p. 275.

[140] Di Ma and Gene Tsudik. 2007f. "Extended Abstract: Forward-Secure Sequential Aggregate Authentication". In: *IEEE Symposium on Security and Privacy*, pp. 86–91.

[141] Thomas Zimmermann. 2007g. "Mining Workspace Updates in CVS". In: *Proceedings of the Fourth International Workshop on Mining Software Repositories*. MSR '07. Washington, DC, USA: IEEE Computer Society, pp. 11–. ISBN: 0-7695-2950-X.

[142] Urs Fischbacher. 2007h. "z-Tree: Zurich toolbox for ready-made economic experiments". In: *Exp. economics* 10.2, pp. 171–178.

[143] Jan G. Cederquist, Ricardo Corin, Mendie A. C. Dekker, Sandro Etalle, Jerry I. den Hartog, and Gabriele Lenzini. Mar. 2007. “Audit-based Compliance Control”. In: *International Journal of Information Security* 6.2, pp. 133–151. ISSN: 1615-5262.

[144] Prasun Dewan and Rajesh Hegde. Sept. 2007. “ Semi-synchronous conflict detection and resolution in asynchronous software development”. In: *Proceedings of the European Conference on Computer-Supported Cooperative Work (ECSCW'07)*. Limerick, Ireland: Springer London, pp. 159–178.

[145] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Oct. 2007. “PeerReview: practical accountability for distributed systems”. In: *SIGOPS Oper. Syst. Rev.* 41 (6), pp. 175–188. ISSN: 0163-5980. DOI: http://doi.acm.org/10.1145/1323293.1294279.

[146] Michelle Cart and Jean Ferrié. Nov. 2007. “Asynchronous Reconciliation based on Operational Transformation for P2P Collaborative Environments”. In: *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2007*. White Plains, New York, USA: IEEE Computer Society, pp. 127–138. ISBN: 978-1-4244-1318-8. DOI: http://dx.doi.org/10.1109/COLCOM.2007.4553821.

[147] Carlos Baquero Daniel Machado and J. Legatheaux Martins. Nov. 2007. “VC^2 - Providing Awareness in Off-The-Shelf Version Control Systems”. In: *Ninth International Workshop on Collaborative Editing Systems, GROUP 2007, IEEE Distributed Systems online*, p. 6. ISSN: 1541-4922.

[148] Stéphane Weiss, Pascal Urso, and Pascal Molli. Dec. 2007. “Wooki: a P2P Wiki-based Collaborative Writing Tool”. In: *Web Information Systems Engineering*. Nancy, France: Springer.

[149] Guido Governatori and Zoran Milosevic. 2006a. “A Formal Analysis of a Business Contract Language”. In: *International Journal of Cooperative Information Systems* 15.4, pp. 659–685.

[150] G. Boella and L. van der Torre. 2006b. “A game theoretic approach to contracts in multiagent systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 36.1, pp. 68–79.

[151] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. 2006c. “A usage-based authorization framework for collaborative computing systems”. In: *Proceedings of the eleventh ACM symposium on Access control models and technologies*. SACMAT '06. Lake Tahoe, California, USA: ACM, pp. 180–189. ISBN: 1-59593-353-0. DOI: 10.1145/1133058.1133084.

[152] Antawan Holmes and Marc Kellogg. 2006d. “Automating Functional Tests Using Selenium”. In: *Proceedings of AGILE Conference (AGILE)*, pp. 270–275. DOI: 10.1109/AGILE.2006.19.

[153] Geraldine Fitzpatrick, Paul Marshall, and Anthony Phillips. 2006e. “CVS integration with notification and chat: lightweight software team collaboration”. In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'06)*. New York, NY, USA: ACM, pp. 49–58. ISBN: 1-59593-249-6. DOI: http://doi.acm.org/10.1145/1180875.1180884.

[154] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. 2006f. “Data Consistency for P2P Collaborative Editing”. In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Banff, AB, Canada, pp. 259–267. ISBN: 1-59593-249-6. DOI: http://doi.acm.org/10.1145/1180875.1180916.

[155] Nava Ashraf, Iris Bohnet, and Nikita Piankov. 2006g. “Decomposing trust and trustworthiness”. In: *Exp. economics* 9.3, pp. 193–208.

[156] Kevin Walsh and Emin Gün Sirer. 2006h. "Experience with an Object Reputation System for Peer-to-Peer Filesharing (Awarded Best Paper)". In: *NSDI*.

[157] Jim Engle-Warnick and Robert L. Slonim. 2006i. "Learning to trust in indefinitely repeated games". In: *Games and Economic Behavior* 54.1, pp. 95–114.

[158] Jason E. Holt. 2006j. "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. Hobart, Tasmania, Australia: Australian Computer Society, Inc., pp. 203–211. ISBN: 1-920-68236-8.

[159] Marti Sergio and Garcia-Molina Hector. Mar. 2006. "Taxonomy of Trust: Categorizing P2P Reputation Systems". In: *Computer Networks* 50 (4), pp. 472–484. ISSN: 1389-1286. DOI: http://doi.acm.org/10.1016/j.comnet.2005.07.011.

[160] Jon Loeliger. June 2006. "Collaborating with Git". In: *Linux Magazine*.

[161] James Tam and Saul Greenberg. July 2006. "A Framework for Asynchronous Change Awareness in Collaborative Documents and Workspaces". In: *International Journal of Human-Computer Studies* 64.7, pp. 583–598. ISSN: 1071-5819. DOI: http://dx.doi.org/10.1016/j.ijhcs.2006.02.004.

[162] Alexander Pretschner, Manuel Hilty, and David Basin. Sept. 2006. "Distributed Usage Control". In: *Commun. ACM* 49 (9), pp. 39–44. ISSN: 0001-0782.

[163] Hyun-Gul Roh, Jinsoo Kim, and Joonwon Lee. Oct. 2006. "How to Design Optimistic Operations for Peer-to-Peer Replication". In: *Proceedings of the Joint Conference on Information Sciences, JCIS 2006*. Kaohsiung, Taiwan: Atlantis Press. ISBN: 90-78677-01-5.

[164] Giovanni Mella, Elena Ferrari, Elisa Bertino, and Yunhua Koglin. Nov. 2006. "Controlled and cooperative updates of XML documents in byzantine and failure-prone distributed systems". In: *ACM Trans. Inf. Syst. Secur.* 9 (4), pp. 421–460. ISSN: 1094-9224. DOI: http://doi.acm.org/10.1145/1187441.1187443.

[165] Gérald Oster, Pascal Molli, Pascal Urso, and Abdessamad Imine. Nov. 2006. "Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems". In: *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006*. Atlanta, Georgia, USA: IEEE Computer Society.

[166] Gary E. Bolton, Elena Katok, and Axel Ockenfels. 2005b. "Cooperation among strangers with limited information about reputation". In: *Jour. of Pub. Econ.* 89.8, pp. 1457–1468.

[167] Jens Riegelsberger, Martina Angela Sasse, and John D. McCarthy. 2005c. "The mechanics of trust: A framework for research and design". In: *Int. J. Hum.-Comput. Stud.* 62.3, pp. 381–422.

[168] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Mar. 2005. "Access Control in Collaborative Systems". In: *ACM Computing Surveys* 37.1, pp. 29–41. ISSN: 0360-0300. DOI: 10.1145/1057977.1057979. URL: http://doi.acm.org/10.1145/1057977.1057979.

[169] Yasushi Saito and Marc Shapiro. Mar. 2005. "Optimistic Replication". In: *Computing Surveys* 37.1, pp. 42–81.

[170] Carl Gutwin, Reagan Penner, and Kevin Schneider. 2004a. "Group awareness in distributed software development". In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04*. New York, NY, USA: ACM, pp. 72–81. ISBN: 1-58113-810-5. DOI: http://doi.acm.org/10.1145/1031607.1031621.

[171] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. 2004b. "Leveraging Single-user Applications for Multi-user Collaboration: the CoWord Approach". In: *Proc. of CSCW'04*. Chicago, Illinois, USA, pp. 162–171.

[172] J. Park and R. Sandhu. 2004c. "The UCON$_{ABC}$ Usage Control Model". In: *ACM Transactions on Information and System Security*, 7(1):128–174.

[173] Darren Davis, Fabian Monrose, and Michael K. Reiter. 2004d. "Time-Scoped Searching of Encrypted Audit Logs". In: *ICICS*, pp. 532–545.

[174] Brian Ninham Shand. 2004e. *Trust for resource control: Self-enforcing automatic rational contracts between computers*. Tech. rep. UCAM-CL-TR-600. University of Cambridge.

[175] Francois Cochard, Phu Nguyen Van, and Marc Willinger. 2004f. "Trusting behavior in a repeated investment game". In: *Journal of Economic Behavior & Organization* 55.1, pp. 31–44.

[176] Marc Shapiro and Karthik Bhargavan. Mar. 2004. *The Actions-Constraints approach to replication: Definitions and Proofs*. Technical Report MSR-TR-2004-14, Microsoft Research.

[177] Brent ByungHoon Kang. Oct. 2004. "S2D2: A Framework for Scalable and Secure Optimistic Replication". PhD thesis. EECS Department, University of California, Berkeley. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2004/5917.html.

[178] Petros Maniatis and Mary Baker. 2003a. "Authenticated Append-only Skip Lists". In: *Acta Mathematica* 137, pp. 151–169.

[179] Colin Camerer. 2003b. *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press.

[180] Stephen V Burks, Jeffrey P Carpenter, and Eric Verhoogen. 2003c. "Playing both roles in the trust game". In: *Journal of Economic Behavior & Organization* 51.2, pp. 195–216.

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

[182] ByungHoon Kang, Robert Wilensky, and John Kubiatowicz. 2003e. "The Hash History Approach for Reconciling Mutual Inconsistency". In: *ICDCS*, pp. 670–677.

[183] Ravi S. Sandhu and Jaehong Park. 2003f. "Usage Control: A Vision for Next Generation Access Control". In: *MMM-ACNS*, pp. 17–31.

[184] Laurence Cholvya and Anthony Hunterb. Mar. 2003. "Merging requirements from a set of ranked agents". In: *Knowledge-Based Systems* 16.2, pp. 113–126.

[185] Anita Sarma, Zahra Noroozi, and André van der Hoek. May 2003. "Palantìr: Raising Awareness among Configuration Management Worspaces". In: *Proceedings of the International Conference on Software Engineering (ICSE'03)*. Portland, Oregon, USA: IEEE Computer Society, pp. 444–454. ISBN: 0-7695-1877-X. DOI: http://dx.doi.org/10.1109/ICSE.2003.1201222.

[186] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. May 2003. "The Eigentrust algorithm for reputation management in P2P networks". In: *Proceedings of the 12th International Conference on World Wide Web, WWW 2003*. Budapest, Hungary: ACM Press, pp. 640–651. ISBN: 1-58113-680-3.

[187] Abdessamad Imine, Pascal Molli, Gérald Oster, and Michaël Rusinowitch. Sept. 2003. "Proving Correctness of Transformation Functions in Real-Time Groupware". In: *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW 2003*. Helsinki, Finland: Kluwer Academic Publishers, pp. 277–293.

[188] Vital Anderhub, Dirk Engelmann, and Werner Güth. 2002a. "An experimental study of the repeated trust game with incomplete information". In: *Journal of Economic Behavior & Organization* 48.2, pp. 197–216.

[189] Chengzheng Sun and David Chen. 2002c. "Consistency maintenance in real-time collaborative graphics editing systems". In: *ACM Transactions on Computer-Human Interaction* 9.1, pp. 1–41. URL: http://doi.acm.org/10.1145/505151.505152.

[190] Zoran Milosevic, Audun Josang, Theo Dimitrakos, and Mary Anne Patton. 2002d. "Discretionary Enforcement of Electronic Contracts". In: *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference)*. EDOC '02, pp. 39–50. ISBN: 0-7695-1656-4. DOI: 10.1109/EDOC.2002.1137695.

[191] Aguido Horatio Davis, Chengzheng Sun, and Junwei Lu. 2002e. "Generalizing operational transformation to the standard general markup language". In: *Proceedings of ACM CSCW'02 Conference on Computer-Supported Cooperative Work*. Group editing algorithms, pp. 58–67. URL: http://doi.acm.org/10.1145/587078.587088.

[192] Brian Shand and Jean Bacon. 2002f. "Policies in Accountable Contracts". In: *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 5-7 June 2002, Monterey, CA, USA*. IEEE Computer Society, pp. 80–91. ISBN: 0-7695-1611-4.

[193] Petros Maniatis and Mary Baker. 2002g. "Secure History Preservation Through Timeline Entanglement". In: *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, pp. 297–312. ISBN: 1-931971-00-5.

[194] Jaehong Park and Ravi Sandhu. 2002h. "Towards usage control models: beyond traditional access control". In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. SACMAT '02. Monterey, California, USA: ACM, pp. 57–64. ISBN: 1-58113-496-7. DOI: 10.1145/507711.507722.

[195] Claudia Keser. 2002i. *Trust and reputation building in e-commerce*. Tech. rep. IBM Watson Research Center working paper.

[196] Anna Gunnthorsdottir, Kevin McCabe, and Vernon Smith. 2002j. "Using the Machiavellianism instrument to predict trustworthiness in a bargaining game". In: *Jour. of Econ. Psychology* 23.1, pp. 49–66.

[197] Mui Lik, Mohtashemi Mojdeh, and Ari Halberstadt. Jan. 2002. "A Computational Model of Trust and Reputation". In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, HICSS 2002*. Waikoloa, Big Island, Hawaii: IEEE Computer Society, pp. 2431–2439. ISBN: 0-7695-1435-9.

[198] Seth Gilbert and Nancy Lynch. June 2002. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: *SIGACT News* 33.2, pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601. URL: https://doi.org/10.1145/564585.564601.

[199] Pascal Molli, Hala Skaf-Molli, and Gérald Oster. June 2002. "Divergence Awareness for Virtual Team through the Web". In: *Integrated Design and Process Technology (IDPT'02)*. Pasadena, California, USA.

[200] Arun Kumar, Neeran Karnik, and Girish Chafle. July 2002. "Context sensitivity in role-based access control". In: *SIGOPS Oper. Syst. Rev.* 36 (3), pp. 53–66. ISSN: 0163-5980.

[201] Haifeng Shen and Chengzheng Sun. Nov. 2002. "Flexible Merging for Asynchronous Collaborative Systems". In: *Proceeding of the Conference on Cooperative Information Systems (CoopIS'02)*. Vol. 2519. Lecture Notes in Computer Science. Irvine, California, USA: Springer-Verlag, pp. 304–321. ISBN: 3-540-00106-9.

[202] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. 2001b. *Database Systems: The Complete Book*. Prentice Hall PTR. ISBN: 0130319953.

[203] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. 2001c. *Database Systems: The Complete Book*. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0130319953.

[204] Aspassia Daskalopulu, Theo Dimitrakos, and Tom Maibaum. 2001d. "E-Contract Fulfilment and Agents' Attitudes". In: *In Proceedings of the ERCIM WG E-Commerce Workshop on The Role of Trust in e-Business*.

[205] Michael T. Goodrich, Roberto Tamassia, and Andrew Schwerin. 2001e. "Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing". In: *DARPA Information Survivability Conference and Exposition,* 2, p. 1068. DOI: http://doi.ieeecomputersociety.org/10.1109/DISCEX.2001.932160.

[206] Ali A. Zafer, Clifford A. Shaffer, Roger W. Ehrich, and Manuel Perez. 2001f. *NetEdit: a Collaborative Editor*. Tech. rep. Technical Report TR-01-13, Computer Science, Virginia Tech.

[207] Dewayne E. Perry, Harvey P. Siy, and Lawrence G. Votta. 2001g. "Parallel changes in large-scale software development: an observational case study". In: *ACM Transactions on Software Engineering and Methodology* 10.3, pp. 308–337. ISSN: 1049-331X.

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

[209] Pascal Molli, Hala Skaf-Molli, and Christophe Bouthier. Sept. 2001. "State Treemap: an Awareness Widget for Multi-Synchronous Groupware". In: *Proceedings of the Seventh International Workshop on Groupware (CRIWG'01)*. Darmstadt, Germany: IEEE Computer Society, pp. 106–114. ISBN: 0-7695-1351-4. DOI: http://dx.doi.org/10.1109/CRIWG.2001.951823.

[210] Michel Abdalla and Leonid Reyzin. 2000a. "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)*. London, UK: Springer-Verlag, pp. 116–129. ISBN: 3-540-41404-5. URL: http://dl.acm.org/citation.cfm?id=647096.716987.

[211] Edward L. Glaeser, David I. Laibson, Jose A. Scheinkman, and Christine L. Soutter. 2000b. "Measuring trust". In: *Quarterly Jour. of Econ.*, pp. 811–846.

[212] Dean Povey. 2000c. "Optimistic security: a new access control paradigm". In: *Proceedings of the 1999 Workshop on New Security Paradigms*. NSPW '99. Caledon Hills, Ontario, Canada: ACM, pp. 40–45. ISBN: 1-58113-149-6. DOI: 10.1145/335169.335188.

[213] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. 2000d. "Reputation systems". In: *Communications of the ACM* 43.12, pp. 45–48.

[214] Eric A. Brewer. 2000e. "Towards robust distributed systems (abstract)". In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, p. 7. DOI: 10.1145/343477.343502. URL: https://doi.org/10.1145/343477.343502.

[215] Nicolas Vidot, Michèle Cart, Jean Ferrié, and Maher Suleiman. Dec. 2000. "Copies Convergence in a Distributed Real-Time Collaborative Environment". In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2000*. Philadelphia, Pennsylvania, USA: ACM Press, pp. 171–180. DOI: http://doi.acm.org/10.1145/358916.358988.

[216] Armando Fox and Eric A. Brewer. 1999. "Harvest, Yield, and Scalable Tolerant Systems". In: *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*. HOTOS '99. USA: IEEE Computer Society, p. 174. ISBN: 0769502377.

[217] Alfarez Abdul-Rahman and Stephen Hailes. 1998a. "A Distributed Trust Model". In: *Proceedings of the 1997 Workshop on New Security Paradigms*. NSPW '97. Langdale, Cumbria, United Kingdom: Association for Computing Machinery, pp. 48–60. ISBN: 0897919866. DOI: 10.1145/283699.283739. URL: https://doi.org/10.1145/283699.283739.

[218] Bruce Schneier and John Kelsey. 1998b. "Cryptographic support for secure logs on untrusted machines". In: *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*. San Antonio, Texas: USENIX Association, pp. 4–4.

[219] Prasun Dewan and HongHai Shen. 1998c. "Flexible meta access-control for collaborative applications". In: *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. CSCW '98. Seattle, Washington, United States: ACM, pp. 247–256. ISBN: 1-58113-009-0. DOI: 10.1145/289444.289499.

[220] Cheh Goh and Adrian Baldwin. 1998d. "Towards a more complete model of role". In: *Proceedings of the third ACM workshop on Role-based access control*. RBAC '98. Fairfax, Virginia, United States: ACM, pp. 55–62. ISBN: 1-58113-113-5.

[221] Maher Suleiman, Michèle Cart, and Jean Ferrié. Feb. 1998. "Concurrent Operations in a Distributed and Mobile Collaborative Environment". In: *Proceedings of the International Conference on Data Engineering - ICDE'98*. Orlando, Florida, USA: IEEE Computer Society, pp. 36–45. ISBN: 0-8186-8289-2. DOI: http://doi.ieeecomputersociety.org/10.1109/ICDE.1998.655755.

[222] Sun Chengzheng, Jia Xiaohua, Zhang Yanchun, Yang Yun, and Chen David. Mar. 1998. "Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems". In: *ACM Transactions on Computer-Human Interaction* 5.1, pp. 63–108. ISSN: 1073-0516.

[223] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Mar. 1998. "Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems". In: *ACM Transactions on Computer-Human Interaction* 5.1, pp. 63–108. ISSN: 1073-0516. DOI: http://doi.acm.org/10.1145/274444.274447.

[224] Prasun Dewan and Honghai Shen. Mar. 1998. "Controlling Access in Multiuser Interfaces". In: *ACM Transactions on Computer-Human Interaction* 5.1, pp. 37–62. ISSN: 1073-0516. DOI: http://doi.acm.org/10.1145/274444.274446.

[225] Chengzheng Sun and Clarence Ellis. Nov. 1998. "Operational transformation in real-time group editors: issues, algorithms, and achievements". In: *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*. Seattle WA, USA, p. 59. DOI: http://doi.acm.org/10.1145/289444.289469.

[226] Klaas Sikkel. 1997a. "A Group-based Authorization Model for Cooperative Systems". In: *Proceedings of the Fifth Conference on European Conference on Computer-Supported Cooperative Work*. ECSCW'97. Lancaster, UK: Kluwer Academic Publishers, pp. 345–360. ISBN: 0-7923-4638-6. URL: http://dl.acm.org/citation.cfm?id=1241980.1242003.

[227] Mike J. Spreitzer, Marvin M. Theimer, Karin Petersen, Alan J. Demers, and Douglas B. Terry. 1997b. "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. Budapest, Hungary: ACM, pp. 234–240. ISBN: 0-89791-988-2. DOI: http://doi.acm.org/10.1145/262116.262151.

[228] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. 1997c. "Flexible update propagation for weakly consistent replication". In: *Proceedings of the sixteenth ACM symposium on Operating systems principles*. SOSP '97. Saint Malo, France: ACM, pp. 288–301. ISBN: 0-89791-916-5.

[229] Maher Suleiman, Michéle Cart, and Jean Ferrié. 1997d. “Serialization of concurrent operations in a distributed collaborative environment”. In: *Proceedings of the international ACM SIGGROUP conference on Supporting group work (GROUP ’97)*. New York, NY, USA: ACM Press, pp. 435–445. ISBN: 0-89791-897-5. DOI: http://doi.acm.org/10.1145/266838.267369.

[230] Prakash Ravi, Raynal Michel, and Singhal Mukesh. Mar. 1997. “An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments”. In: *Journal of Parallel and Distributed Computing* 41.2, pp. 190–204. ISSN: 0743-7315. DOI: http://doi.acm.org/10.1006/jpdc.1996.1300.

[231] Martin Roscheisen and Terry Winograd. 1996a. “A Communication Agreement Framework for Access/Action Control”. In: *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. SP ’96. Washington, DC, USA: IEEE Computer Society, pp. 154–. ISBN: 0-8186-7417-2.

[232] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Feb. 1996. “Role-Based Access Control Models”. In: *IEEE Computer* 29 (2), pp. 38–47. ISSN: 0018-9162.

[233] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. Nov. 1996. “An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors”. In: *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW’96)*. Boston, Massachusetts, USA: ACM Press, pp. 288–297. ISBN: 0-89791-765-0. DOI: http://doi.acm.org/10.1145/240080.240305.

[234] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. 1995a. “High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System”. In: *Proceedings of the 8th annual ACM symposium on User interface and software technology - UIST ’95*. Pittsburgh, Pennsylvania, USA: ACM Press, pp. 111–120. ISBN: 0-89791-709-X. DOI: http://doi.acm.org/10.1145/215585.215706.

[235] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. 1995b. “Managing update conflicts in Bayou, a weakly connected replicated storage system”. In: *SOSP ’95: Proceedings of the fifteenth ACM symposium on Operating systems principles*. Copper Mountain, Colorado, United States: ACM, pp. 172–182. ISBN: 0-89791-715-4. DOI: http://doi.acm.org/10.1145/224056.224070.

[236] Thomas P. Moran, Kim McCall, Bill van Melle, Elin Pedersen, and Frank Halasz. 1995c. “Some design principles for sharing in tivoli, a whiteboard meeting-support tool.” In: *Groupware for Real-Time Drawings: A designer’s Guide*, pp. 24–36.

[237] Paul Dourish. 1995d. “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. Stockholm, Sweden: Kluwer Academic Publishers, pp. 215–230. ISBN: 0-7923-3697-6.

[238] R William Soukoreff and I Scott Mackenzie. 1995e. “Theoretical upper and lower bounds on typing speed using a stylus and a soft keyboard”. In: *Behaviour & Information Technology* 14.6, pp. 370–379.

[239] Joyce Berg, John Dickhaut, and Kevin McCabe. 1995f. “Trust, reciprocity, and social history”. In: *Games and economic behavior* 10.1, pp. 122–142.

[240] Zoran Milosevic. Oct. 1995. “Enterprise Aspects of Open Distributed Systems”. PhD thesis. The Department of Computer Science, The University of Queensland.

[241] Peter Reiher, John Heidemann, David Ratner, Greg Skinner, and Gerald Popek. 1994. “Resolving file conflicts in the ficus file system”. In: *In Proceedings of the Summer Usenix Conference (USTC’94)*, pp. 183–195.

[242] Ronald M. Baecker, Dimitrios Nastos, Ilona R. Posner, and Kelly L. Mawby. 1993a. "The User-Centred Iterative Design of Collaborative Writing Software". In: *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*. Meetings and Collaborative Writing, pp. 399–405. URL: http://www.acm.org/pubs/articles/proceedings/chi/169059/p399-baecker/p399-baecker.pdf.

[243] Alain Karsenty and Michel Beaudouin-Lafon. May 1993. "An Algorithm for Distributed Groupware Applications". In: *Proceedings of the 13th International Conference on Distributed Computing Systems*. Ed. by Robert Werner. Pittsburgh, Pennsylvania, USA: IEEE Computer Society Press, pp. 195–202. ISBN: 0-8186-3770-6.

[244] HongHai Shen and Prasun Dewan. 1992a. "Access Control for Collaborative Environments". In: *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*. Building Real-Time Groupware, pp. 51–58.

[245] Paul Dourish and Victoria Bellotti. 1992b. "Awareness and Coordination in Shared Workspaces". In: *CSCW '92, Proceedings of the Conference on Computer Supported Cooperative Work, Toronto, Canada, October 31 - November 4, 1992*, pp. 107–114. DOI: 10.1145/143457.143468. URL: https://doi.org/10.1145/143457.143468.

[246] Greenberg Saul, Roseman Mark, Webster Dave, and Bohnet Ralph. 1992c. "Human and Technical Factors of Distributed Group Drawing Tools". In: *Interacting with Computers* 4.3, pp. 364–392.

[247] Saul Greenberg, Mark Roseman, Dave Webster, and Ralph Bohnet. 1992d. "Human and Technical Factors of Distributed Group Drawing Tools". In: *Interacting with Computers*. Special Issue on CSCW: Part 1 4.3, pp. 364–392.

[248] David F. Ferraiolo and D. Richard Kuhn. 1992e. "Role-Based Access Controls". In: *Proceedings of the 15th National Computer Security Conference*. Baltimore MD, USA, pp. 554–563.

[249] Geoffrey Keppel. 1991a. *Design and analysis: A researcher's handbook* . Prentice-Hall, Inc.

[250] R. E. Newman-Wolfe and Harsha K. Pelimuhandiram. 1991b. "MACE: A Fine Grained Concurrent Editor". In: *Conference on Organizational Computing Systems*. Collaboration, pp. 240–254. URL: http://www.acm.org/pubs/articles/proceedings/cocs/122831/p240-newman-wolfe/p240-newman-wolfe.pdf.

[251] Saul Greenberg. 1991c. "Personalisable Groupware: Accommodating Individual Roles and Group Differences". In: *Proceedings of the Second European Conference on Computer Supported Cooperative Work, 24-27 September 1991, Amsterdam, The Netherlands*, pp. 17–31.

[252] E. Chang, R. Kasperski, and T. Copping. Jan. 1991. "Group coordination in participant systems". In: *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*. Vol. 3, pp. 589–599. DOI: 10.1109/HICSS.1991.184191.

[253] Clarence A Ellis, Simon J Gibbs, and Gail Rein. Jan. 1991. "Groupware: Some Issues and Experiences". In: *Commununications of the ACM* 34.1, pp. 39–58. DOI: 10.1145/99977.99987.

[254] Walter F. Tichy. Nov. 1991. *RCS - A System for Version Control*. en. URL: http://citeseer.ist.psu.edu/111753.html;%20http://www.cs.yorku.ca/~brecht/4321/handouts/rcs.ps.

[255] Jozef Paul Chris Lauwers. 1990a. "Collaboration Transparency in Desktop Teleconferencing Environments". AAI9102296. PhD thesis. Stanford, CA, USA: Stanford University.

[256] Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok, and James H. Morris. 1990b. "Issues in the Design of Computer Support for Co-Authoring and Commenting".

In: *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work*. CSCW '90. Los Angeles, California, USA: Association for Computing Machinery, pp. 183–195. ISBN: 0897914023. DOI: 10.1145/99332.99354. URL: https://doi.org/10.1145/99332.99354.

[257] S. Greenberg. 1990c. "Sharing Views and Interactions with Single-user Applications". In: *Proceedings of the ACM SIGOIS and IEEE CS TC-OA Conference on Office Information Systems*. COCS '90. Cambridge, Massachusetts, USA: ACM, pp. 227–237. ISBN: 0-89791-358-2. DOI: 10.1145/91474.91546. URL: http://doi.acm.org/10.1145/91474.91546.

[258] Brian Berliner. Jan. 1990. "CVS II: Parallelizing Software Development". In: *Proceedings of the USENIX Winter Technical Conference*. Washington, District of Columbia, USA, pp. 341–352.

[259] C. A. Ellis and S. J. Gibbs. June 1989. "Concurrency control in groupware systems". In: *SIGMOD Record (ACM Special Interest Group on Management of Data)* 18.2, pp. 399–407. ISSN: 0163-5808.

[260] Clarence A. Ellis and Simon J. Gibbs. June 1989. "Concurrency control in groupware systems". In: *SIGMOD Record (ACM Special Interest Group on Management of Data)* 18.2, pp. 399–407.

[261] Mattern Friedemann. Oct. 1989. "Virtual Time and Global States of Distributed Systems". In: *Proceedings of the International Workshop on Parallel and Distributed Algorithms*. Ed. by Michel Cosnard et al. Chateau de Bonas, France: Elsevier Science Publishers B. V., pp. 215–226.

[262] J. J. Garcia-Luna-Aceves, E. J. Craighill, and R. Lang. 1988a. "An open-systems model for computer-supported collaboration". In: *Proceedings of the 2nd IEEE Conference of Computer Workstations*. Santa Clara, pp. 40–51.

[263] Marilyn Mantei. 1988b. "Capturing the Capture Concepts: A Case Study in the Design of Computer-supported Meeting Environments". In: *Proceedings of the 1988 ACM Conference on Computer-supported Cooperative Work*. CSCW '88. Portland, Oregon, USA: ACM, pp. 257–270. ISBN: 0-89791-282-9. DOI: 10.1145/62266.62287. URL: http://doi.acm.org/10.1145/62266.62287.

[264] Sunil Sarin and Irene Greif. 1988c. "Computer-Based Real-Time Conferencing Systems (Reprint)". In: *Computer-Supported Cooperative Work: A Book of Readings*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 397–422. ISBN: 0934613575. URL: http://dl.acm.org/citation.cfm?id=49504.49519.

[265] Robert Johansen. 1988d. *GroupWare: Computer Support for Business Teams*. New York, NY, USA: The Free Press. ISBN: 0029164915.

[266] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1987a. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley. ISBN: 0-201-10715-5.

[267] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987b. "Epidemic algorithms for replicated database maintenance". In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. PODC '87. Vancouver, British Columbia, Canada: ACM, pp. 1–12. ISBN: 0-89791-239-X.

[268] Irene Greif and Sunil Sarin. Apr. 1987. "Data Sharing in Group Work". In: *ACM Trans. Inf. Syst.* 5.2, pp. 187–211. ISSN: 1046-8188. DOI: 10.1145/27636.27640. URL: http://doi.acm.org/10.1145/27636.27640.

[269] David R. Jefferson. July 1985. "Virtual Time". In: *ACM Transactions on Programming Languages and Systems* 7.3, pp. 404–425. ISSN: 0164-0925. URL: http://www.acm.org/pubs/toc/Abstracts/0164-0925/3988.html.

[270] Reid G. Smith. Dec. 1980. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". In: *IEEE Transactions on Computers* C-29.12, pp. 1104–1113. ISSN: 0018-9340. DOI: 10.1109/TC.1980.1675516. URL: http://dx.doi.org/10.1109/TC.1980.1675516.

[271] Ralph Charles Merkle. 1979. "Secrecy, authentication, and public key systems." AAI8001972. PhD thesis. Stanford, CA, USA: Stanford University.

[272] Leslie Lamport. July 1978. "Times, Clocks, and the Ordering of Events in a Distributed System". In: *Communications of the ACM* 21.7, pp. 558–565. ISSN: 0001-0782.

[273] Paul R. Johnson and Robert H. Thomas. Jan. 1975. *Maintenance of duplicate databases.* RFC 677. http://www.ietf.org/rfc/rfc677.txt. Internet Engineering Task Force, p. 10.

[274] Butler W. Lampson. 1971. "Protection". In: *Proc. Fifth Princeton Symposium on Information Sciences and Systems.* Princeton University, pp. 437–443.

[275] Georgii Maximovich Adelson-Velskii and Evgenii Mikhailovich Landis. 1962. "An Algorithm for the Organization of Information". In: *Doklady Akademii Nauk SSSR* 146. (English translation in Soviet Mathematics Doklady, vol 3, pp. 1259–1263), pp. 263–266.

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

[277] Georg Henrik Von Wright. Jan. 1951. "Deontic Logic". In: *Mind* 60.237. Oxford University Press, pp. 1–15.

Abstract

Most commonly used collaborative systems are provided by large service providers. While these collaborative services offer very interesting functionalities, they feature certain limitations. Most of the platforms hosting these collaboration services rely on a central authority and place personal information in the hands of a single large corporation which is a perceived privacy threat. Users must provide and store their data to vendors of these services and have to trust that they will preserve privacy of their data, but they have little control over the usage of their data after sharing it with other users. Moreover, these systems do not scale well in terms of the number of users and their modifications. Furthermore, user communities cannot deploy these kind of service applications since they generally rely on costly infrastructures rather than allowing sharing infrastructure and administration costs. My research work aims to move away from centralized authority-based collaboration towards a large scale trust-based peer-to-peer collaboration where control over data is given to users who can decide with whom to share their data. The main advantages of peer-to-peer collaborative systems are high scalability, resilience to faults and attacks and a low deployment barrier for new services. My contributions are structured around two axes of research: *collaborative data management* and *trustworthy collaboration*.

In order to provide efficient data availability, data is typically replicated and users are allowed to concurrently modify replicated data. One of the challenges is to develop optimistic replication algorithms that maintain consistency of the shared data in the face of concurrent modifications. These algorithms have to be reliable, i.e. after the reception of all modifications the data copies have to converge. These algorithms also have to be scalable, i.e. have a complexity that does not depend on the number of users. These algorithms have also to be explainable, i.e. their decision must be comprehensible by users and therefore user intentions must be preserved. In the first part of this manuscript I present my contributions to the design of various optimistic data replication and their evaluation in terms of their complexities but also by means of experimental design with users. I also present my contributions on group awareness specifically on what information should be provided to users to prevent conflicting changes and to understand divergence when conflicts cannot be avoided.

In this large scale peer-to-peer collaboration where users cannot remember the interactions with all collaborators a main question is how to choose the trusted collaborators in order to share them the data. A main challenge is how to assess users trust according to their past behaviour during collaboration in order to be able to predict their future behavior. In the second part of this manuscript I present a contract-based collaboration model where contracts are specified by the data owners when they share the data and user trust is assessed according to the observation of adherence to or violation of contracts. I also present a hash chain based authenticators approach for ensuring integrity and authenticity of logs of collaboration. For testing the proposed trust-based collaboration model, I designed a user experiment employing trust game and relying on a computational trust metric according to user exchanges in this game.

I finally present my future research directions around secure and trustworthy collaborative data management.

Keywords: distributed collaborative systems, operational transformation, CRDT, group awareness, trust, contract-based collaboration, authenticated logs, trust game, user studies