Distributed Access Control for Collaborative Applications using CRDTs

Pierre-Antoine Rault, Claudia-Lavinia Ignat, Olivier Perrin

Distributed Access Control for Collaborative Applications using CRDTs

Pierre-Antoine Rault
Université de Lorraine, CNRS, Inria
LORIA, F-54000 Nancy, France
pierre-antoine.rault@loria.fr

Claudia-Lavinia Ignat
Université de Lorraine, CNRS, Inria
LORIA, F-54000 Nancy, France
claudia.ignat@inria.fr

Olivier Perrin
Université de Lorraine, CNRS, Inria
LORIA, F-54000 Nancy, France
olivier.perrin@univ-lorraine.fr

Abstract

Distributed applications are part of our everyday lives, but too often their good operation depends on central servers, all potential points of failure and performance bottlenecks. Designing systems for fully distributed communications however still requires porting common mechanisms needed for feature-rich modern applications such as user rights differentiation, multiple administrators, and end-to-end encryption. We propose a distributed access control mechanism for collaborative applications by relying on conflict-free replicated data types (CRDT), and design an access control policy CRDT able to support Google Docs and POSIX file systems as example of distributed applications. To enforce that policy, we outline a generic data model, examine different conflict resolution strategies at the data and policy levels, and consider a novel approach towards conflicts between data and policy operations.

CCS Concepts: • **Computing methodologies → Distributed algorithms**; • **Security and privacy → Access control**; • **Human-centered computing → Synchronous editors**; **Asynchronous editors**.

Keywords: distributed algorithms, access control, CRDT (Conflict-free Replicated Data Type), real-time collaborative editors, POSIX

1 Introduction

Collaborative applications are ubiquitous. Despite the varied execution environments required by modern usage, their operation often relies on a handful of centralized services, limiting their resilience upon the failure of one of these services. To alleviate that risk, systems need to be designed for decentralized operation. CRDTs [9] provide a framework to design data structures that can be replicated across multiple replicas and updated independently and concurrently without coordination to converge to a given state.

A distributed application, be it a distributed file system or a collaborative editor, needs to provide mechanisms to prevent unauthorized access or modification – namely, access control. In order to achieve high availability and low latency at a global scale, access rights have to be replicated. Modifications on the data can be done concurrently with policy modifications, hence requiring an access control CRDT that deals with both data and policy modifications.

The dynamic nature of a group, with administrators and members changing over time, is likely to produce conflicts in the definition of the rights assigned to a user for a given resource. Most systems that deal with concurrent data and policy modifications are so far limited to a single, fixed administrator per document [3]. In [10] authors explores concurrent policy and data operations conflict resolution in the presence of multiple administrators for a data store by preserving two properties: all policy operations are applied on all replicas before applying the subsequent data operations ; and in the presence of concurrent policy modifications, the most restrictive policy is kept. However, it does not consider scenarios in which the order of execution of policy and data operations might lead to a divergent data state. In [12] authors examine potential data leakage (rights conflict resolution favoring accessibility) or undue loss of access (resolution favoring confidentiality) introduced by concurrent access policy changes. They outline the need not only for flexible data models, but also for flexible resolution strategies. However, they do not further explore concurrent data and access policy changes.

Automatic conflict identification and repair tools [2] often do not take into account implicit relations between the multiple CRDTs involved for each of the data types exchanged within complex applications. More critically, they do not consider an overarching access control policy CRDT that integrates with the implicit dependency of data operations to the access policy in which they are created.

In this paper we illustrate by means of examples the challenges faced by replication algorithms for obtaining consistency over both data and access control policies in distributed applications such as Google Docs and POSIX file systems. We provide a CRDT-based solution for consistency maintenance over access control policies. We give an overview of our proposed solution for the composition of the CRDT on access control policies with a CRDT on data.

The paper is organised as follows. In Section 2 we start by modeling access control policies in collaborative applications such as Google Docs. In Section 3 we present our solution to resolve conflicts between policy operations. Next, in Section 4 we present our solution for maintaining consistency over both data and access control policies. Section 5 shows

how our solution can be adapted for POSIX file systems. Finally, in Section 6 we present concluding remarks and some directions for future work.

2 Access control policies model

Depending on the application that needs to be modeled, the data model and conflict resolutions vary in complexity. We can however define a baseline with collaborative editing applications like Google Docs, as they govern limited data: a document, and possibly comments. It features dynamic members and administrators. Administrators (called *editors* with the exception of the fixed *owner*) hold all rights over the data. Other users can be given a *write* access, but otherwise have a *read* access.

In our baseline data model, access control lists (ACLs) are represented by a Map of users to a Set CRDT of their roles, following a Role-Based Access Control [8] model. This Map is responsible of managing a user's rights and transitions from a set to another upon each policy modification. Since there is a total order among roles being able to modify the managed data or its access control policy, a simple modelization can be given mapping resolution priority to role hierarchy. This allows to define transition semantics where, in the above example, a user being restricted from *read* would also lose all other rights (including *admin*), since it is the lowest in the total order of roles.

This diverges from using a Set of object-user or object-object relations represented as relation tuples, as seen in projects like Google Zanzibar [7]. It is already an opinionated data model, with groups, objects and namespaces – arguably making a complex ableit conveniently flexible proposition. However, Zanzibar's reliance on Spanner, a database that provides external consistency and snapshot reads with bounded staleness assigning each ACL write a microsecond timestamp through True Time API [4], makes this proposition unsuitable for our purpose.

When receiving a third-party operation – and depending on the nature of the action and CRDT concerned – the implementer can check against the Access Control Policy whether this operation can be locally executed. Namely it compares the roles needed for the operation execution with the ones currently possessed by the user emitting the operation in the user Map, for instance by using a function analogous to *hasRights* in the OR-Set CRDT Algorithm 1 in appendix.

3 Conflicts resolution for policy operations

Like other CRDTs, our Access Control CRDT relies on conflict resolution strategies between two operations on that datatype. We chose a simplification of potential conflict resolutions, by assuming an operation emitted by a user with a higher role will prevail. This relies on roles being totally ordered, or otherwise mapped on a linear priority scale in order to decide between merge conflicts. In the use case of Google Docs notably, there can be found a total order that could be mapped to a priority in conflict resolutions of policy modifications, as roles bearing administrator rights are the *owner* and *editor* roles:

$$\underbrace{owner > editor}_{\text{administrators}} > \underbrace{writer > commenter > viewer}_{\text{users}}$$

The *owner* role is unique to the data initial creator and is immutable, while *editor* can be assigned by any administrator to any user. Two concurrent policy updates performed by different administrators might incur conflict (see Example 3.1). In the rights representation above, we could expand the implicit roles of an administrator to contain all those of a regular user. In this case administrator operations would conflict with regular user ones when any policy change occurs on a same target.

In case of conflicting operations by two users of the same priority, a finer strategy must be devised for the system to converge. Typically, a strategy favors either accessibility (by resolving in favor of opening rights that give access to data), confidentiality (by resolving in favor of restricting these rights) while preserving integrity - preventing unauthorized modifications, as shown late in Example 3.2.

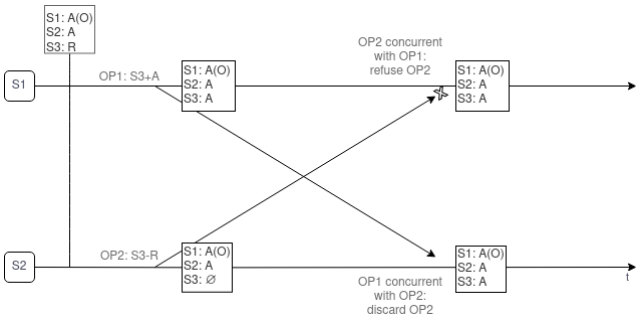
Notations used in the following examples and their figures are found in Table 1.

Term	Usage
Sx	Subject/group member of reference x
A(O)	Right to administer, corresponds to *owner* role
A	Right to administer, corresponds to *editor* role
W	Right to write, corresponds to *writer* role
R	Right to read , corresponds to *viewer* role
OPx	Operation of reference x
d	data
$\rightarrow \times$	Operation got discarded on arrival
ϵ_i	*epoch* of reference i

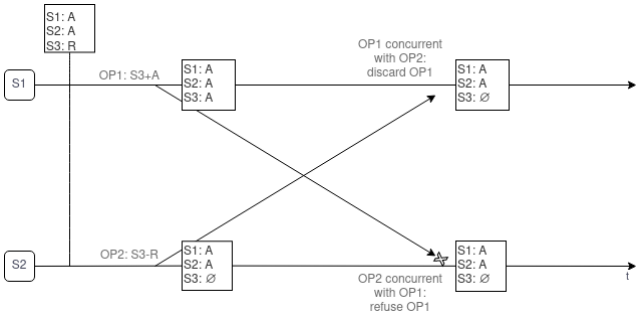
Table 1. Nomenclature of symbols used in figures

Example 3.1. In Figure 1a, $OP1$ and $OP2$ are concurrently emitted respectively by $S1$ and $S2$, two administrators as stated by the policy part of the initial state box on the top left hand corner. Since they are both targeting $S3$'s rights, and since rights in our use case are totally ordered and contain the rights below them, $OP1$ (setting the *editor*/A role to $S3$) and $OP2$ (setting off the *reader*/R role to $S3$) are clearly in conflict. As $S1$ is an *owner* among administrators, $OP1$ will take priority over $OP2$, and thus $OP2$ is discarded.

Example 3.2. In Figure 1b, $S1$ is not an *owner* anymore, but a regular *editor*. As such, a conflict resolution strategy reliant on other attributes must be used. In the confidentiality-favoring strategy used here, the lesser of two right sets is kept – that is to say, $OP2$.



(a) resolution based on emitter role priority



(b) resolution based on confidentiality

Figure 1. Comparison of policy conflicts, with different strategies, in the context of Google Docs's policy model.

4 Interactions between policy and data operations

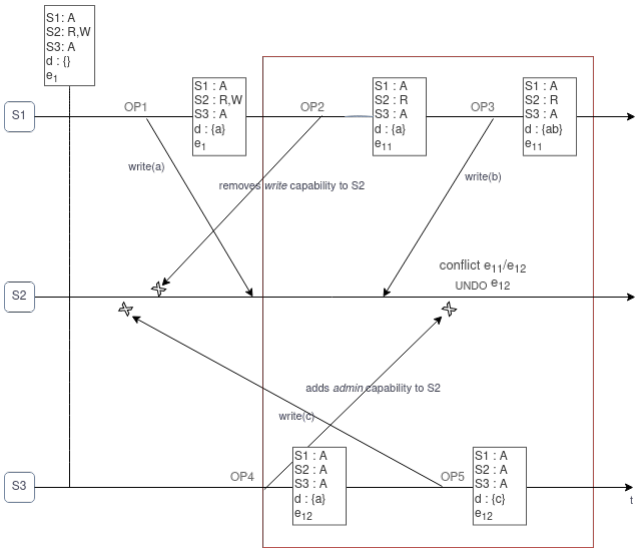
Conflicts between data operations can be resolved by means of a data specific CRDT such as LogootSplit for sequence-based data underlying real-time collaborative editing [1]. Conflicts between policy-modifying operations can be resolved by means of the CRDT proposed in Algorithm 1 in appendix. However, combining CRDTs for data with CRDTs for policies raises several challenges. Conflicts between two concurrent operations based on diverging policies cannot be safely resolved. Both policy operations and data operations need to rely on the result of a conflict resolution of the closest policy operation before them (parent policy). Multiple mechanisms can freely be chosen from, but we devise a reference – named *epoch* [6] – to that parent operation. The reference is sent along each operation. We can then use it to resolve conflicts down the chain of dependent operations, based on the assumption that each operation can be undone and conversely redone. For this, each CRDT involved (policy and data) must possess two functions: (i) *undo* where descendants of a parent operation are also invalidated and their changes reverted, (ii) *redo* where operations ignored or undone but still kept within the log are reestablished along with their parent [13, 14]. This allows to reason about document operations without costly *vector clocks* whose size grows quadratically with the number of participants in a dynamic collaboration group. Instead, all operations are referring to an *epoch* created by the most recent previous operation that changed the access control policy. The *epoch* doesn't grow in size, but merely refers to a parent operation that last changed the policy.

Via this backtracking mechanism, we can apply conflict resolution decisions to batches of operations *a posteriori*, as in Example 4.1, where operations based on later *epochs* are undone or kept in batches as their parent operation conflict gets resolved. In the scope of collaborative applications, ACL changes can be infrequent, but only conflicting policies can lead to lots of *undo*/*redo*, bounded by their *epoch*.

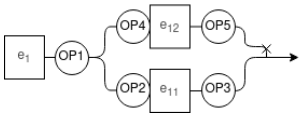
Example 4.1. In Figure 2, we show a typical situation in a distributed setting. In Figure 2a, messages emitted coherently by either $S1$ or $S2$ arrive in different order and thus bear a different final state once received on $S3$. The context in which authorization of each operation is checked is not consistent across sites $S1/S2/S3$. Using our proposed *epoch* mechanism (here added to the bottom of the state box), we expect to be able to backtrack the state of rights, thus recreating a context for authorization checking of operations. In our case, $OP3$ and $OP5$ are unaffected by the conflict, but still subject to trickle-down effects of the conflict resolution of their parent. We can represent it as a graph (see Figure 2b), with *epochs* done in parallel as branches that can be dealt with in batch.

While the order of operations on the document does not matter, these operations must be performed against a version of the access policy that matches the desired access control context. In practice, this translates into a limitation of commutativity requiring the operation that created the referenced *epoch* to be applied before applying the desired operation. And since operations are using their closest parent as *epoch* value instead of their most meaningful parent, it is not a semantically accurate description of an operation's access control context, but rather a lowest common denominator to solve conflicts across branches. Children operations are systematically affected by the resolution of their parent, even when they aren't reliant on its policy changes.

We note that while data and policies are expected to converge, *epochs* are not. Subjects are free to choose a parent *epoch* among the branches they aggregated at their replica. However, without a mechanism deciding which *epoch* to choose, replicas will not emit new operations with one specific *epoch*. A replica at which multiple concurrent – but



(a) multi-conflicting scenario



(b) corresponding operation hierarchy with *epochs*

Figure 2. Example of conflict between two concurrent policy modifications, and potential divergence on a third site without *epoch* policy-causality preservation.

non-conflicting – *epochs* exist needs to define its next policy-modifying operation on multiple *epochs*, i.e. the last ones of each branch.

5 Modeling complex systems: POSIX

A wide range of user applications expect some strong invariants to model key functionalities, and while we previously identified a baseline application (Google Docs) to model our initial CRDT, a more realistic degree of invariant complexity lies in the case of POSIX. We thus follow this standard to explore the flexibility of using our CRDT modelisation on systems which rely on strong invariants.

We consider a simplified POSIX model in Algorithm 2 in appendix, where a list of files, groups and users is administered by the access control policy CRDT, mimicking a file system without directories for the sake of a minimal example. We need to transform our data model to include the notion of groups, and switch our representation of users having roles to that of groups and files listing their users and their rights. To that end we use a Set of users, a Map of groups each pointing to a Set of users, and comprised at least of the `sudoers` group with at least one user. The list of files is represented as a Map to an *ad hoc* File CRDT, which lists each permission field (owner, group, mask, other), the entities behind the owner and group fields, and a data field. Contrary to simpler models, a part of the ACL is co-located with the data.

Since all the information required to solve potential conflicts is present within the Access Control CRDT at merge time, the same aforementioned strategies can be developed. However, new *implicit conflict origins* need to be considered for resolution, as simply having divergent groups or owner field values is not enough. As exampled by [11], changing an owner and the permission associated to it concurrently is not typically located on the same field – and thus not dealt by the same CRDT. However our CRDT encompasses both, and should declare it a potential conflict, as the intent of changing the owner was done in a context where its permissions would have been different. The same can be said of group members, the group field, and its permissions.

6 Conclusion

By laying the base for a minimal Access Control CRDT, as well as mechanisms to deal with conflicts not only inter-policy CRDT, but also between policy and data CRDTs, our model proves flexible enough to replicate two archetypal cases for decentralized applications: Google Docs, and POSIX ACLs. We argue that our model thus encompasses a wider array of applications than *ad hoc* models.

We plan to enhance our model along multiple axes, first by providing the formal model for its convergence when dealing with *epochs*, but also by taking into account its fitness with other implicit conflicts arising from end-to-end encryption. We then intend to integrate it on our real-time collaborative editor MUTE [5].

References

[1] Luc André, Stéphane Martin, Gérald Oster, and Claudia-Lavinia Ignat. 2013. Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2013)*. ICST, Austin, Texas, USA, 50–59. https://doi.org/10.4108/icst.collaboratecom.2013.254123

[2] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, and Nuno Preguiça. 2018. IPA: Invariant-Preserving Applications for Weakly Consistent Replicated Databases. *Proceedings of the VLDB Endowment* 12, 4 (Dec. 2018), 404–418. https://doi.org/10.14778/3297753.3297760

[3] Asma Cherif, Abdessamad Imine, and Michaël Rusinowitch. 2014. Practical Access Control Management for Distributed Collaborative Editors. *Pervasive and Mobile Computing* 15 (Dec. 2014), 62–86. https://doi.org/10.1016/j.pmcj.2013.09.004

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. 2012. Spanner: Google's Globally-Distributed Database. In *OSDI*, Vol. 31. USENIX Association, Hollywood, CA, 261–264.

https://doi.org/10.5555/2387880.2387905

[5] Matthieu Nicolas, Victorien Elvinger, Gérald Oster, Claudia-Lavinia Ignat, and François Charoy. 2017. MUTE: A Peer-to-Peer Web-Based Real-Time Collaborative Editor. In *Proceedings of the 15th European Conference on Computer-Supported Cooperative Work (ECSCW 2017, Vol. 1)*. EUSSET, Sheffield, United Kingdom, 1–4. https://doi.org/10.18420/ecscw2017_p5

[6] Matthieu Nicolas, Gérald Oster, and Olivier Perrin. 2020. Efficient Renaming in Sequence CRDTs. In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. Number 9 in PaPoC '20. Association for Computing Machinery, New York, NY, USA, 1–8.

[7] Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christopher D. Richards, and Mengzhi Wang. 2019. Zanzibar: Google's Consistent, Global Authorization System. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19)*. USENIX Association, Renton, WA, 33–46. https://doi.org/10.5555/3358807.3358811

[8] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (Feb. 1996), 38–47. https://doi.org/10.1109/2.485845

[9] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. *A Comprehensive Study of Convergent and Commutative Replicated Data Types*. Report. Inria – Centre Paris-Rocquencourt ; INRIA.

[10] Mathias Weber, Annette Bieniusa, and Arnd Poetzsch-Heffter. 2016. Access Control for Weakly Consistent Replicated Information Systems. In *Proceedings of International Workshop on Security and Trust Management (STM 2016)*, Gilles Barthe, Evangelos Markatos, and Pierangela Samarati (Eds.). Springer International Publishing, Cham, 82–97. https://doi.org/10.1007/978-3-319-46598-2_6

[11] Elena Yanakieva, Michael Youssef, Ahmad Hussein Rezae, and Annette Bieniusa. 2021. Access Control Conflict Resolution in Distributed File Systems Using CRDTs. In *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '21)*. Association for Computing Machinery, New York, NY, USA, 1–3. https://doi.org/10.1145/3447865.3457970

[12] Elena Yanakieva, Michael Youssef, Ahmad Hussein Rezae, and Annette Bieniusa. 2021. On the Impossibility of Confidentiality, Integrity and Accessibility in Highly-Available File Systems. In *Proceedings of the International Conference on Networked Systems (NETYS 2021)*, Karima Echihabi and Roland Meyer (Eds.). Springer International Publishing, Cham, 3–18. https://doi.org/10.1007/978-3-030-91014-3_1

[13] Weihai Yu, Luc André, and Claudia-Lavinia Ignat. 2015. A CRDT Supporting Selective Undo for Collaborative Text Editing. In *Proceedings of the 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015)*, Alysson Bessani and Sara Bouchenak (Eds.). Springer International Publishing, Grenoble, France, 193–206. https://doi.org/10.1007/978-3-319-19129-4_16

[14] Weihai Yu, Victorien Elvinger, and Claudia-Lavinia Ignat. 2019. A Generic Undo Support for State-Based CRDTs. In *Proceedings of the 23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*. https://doi.org/10.4230/LIPIcs.OPODIS.2019.14

Appendix

We provide the modified state-based OR-Set CRDTs enhanced with support of user rights checks.

Algorithm 1 CRDT for Google Docs Access Control

1: payload set S, set T ▷ set of triples: element identifier e, unique tag u, rightset r
2: initial $\emptyset, \emptyset$
3: payload epoch ϵ
4: initial /

5: query *contains* (element e) : boolean b
6: let $b = (\exists u : (e, u, r) \in S)$

7: query *get* (element e) : rightset r
8: let $r = (r | \exists u_X : (e, u, r) \in S, S := \{u_1, u_2, ...u_X\}, u_{X-1} \Rightarrow u_X)$

9: update *add* (element e, rightset r)
10: pre *hasRights*() sufficient
11: let $u = unique()$ ▷ *unique*() returns a unique value
12: $S := S \cup \{(e, u, r)\}$

13: update *remove* (element e)
14: pre *contains*(e)
15: pre *hasRights*() sufficient
16: let $R = \{(e, u, r) | \exists n : (e, u, r) \in S\}$
17: $S := S \setminus R$
18: $T := T \cup R$

19: query *hasRights* (role = admin) : boolean b
20: pre *contains*(e')
21: let $r' = get(e').r$
22: let $b = r' \geq$ role ▷ by default, only admins can modify the policy

23: compare (A, B) : boolean b
24: let $b = ((A.S \cup A.T) \subseteq (B.S \cup B.T)) \cap (A.T \subseteq B.T)S$

25: merge (B) :
26: $S := (S \setminus B.T) \cup (B.S \setminus T)$
27: $T := T \cup B.T$

Algorithm 2 CRDT for POSIX Access Control, extending Algorithm 1

payload map S, set T_s ▷ map of triples: element identifier e, unique tag u, rightset r
    initial $\emptyset, \emptyset$
payload map G, set T_g ▷ map of groups, each listing users' element identifier
    initial $\{sudoers\}, \emptyset$
payload epoch ϵ
    initial /

update *addUser* (element e, rightset r)
    pre *hasRights*($sudoers$) sufficient
    let $u = unique()$ ▷ *unique*() returns a unique value
    $S := S \cup \{(e, u, r)\}$
update *removeUser* (element e)
    pre *contains*(e)
    pre *hasRights*($sudoers$) sufficient
    let $R = \{(e, u, r) | \exists n : (e, u, r) \in S\}$
    $S := S \setminus R$
    $T_s := T_s \cup R$

update *addGroup* (group g)
    pre *hasRights*($sudoers$) sufficient
    let $u = unique()$
    $G[g] := \{u\}$
update *removeGroup* (group g)

update *addUserToGroup* (element e, group g)
    pre *hasRights*($sudoers$) sufficient
    let $u = unique()$
    $G[g] := G[g] \cup \{(u, e)\}$

update *changeFileOwner* (file f, element e)
update *changeFileGroup* (file f, group g)
update *changeFileOwnerPermission* (file f, int i)
update *changeFileGroupPermission* (file f, int i)
update *changeFileOtherPermission* (file f, int i)