Decentralized documents authoring system for decentralized teamwork

Frédéric Merle, Aurélien Bénel, Guillaume Doyen, Dominique Gaïti

Decentralized Documents Authoring System for Decentralized Teamwork: Matching Architecture with Organizational Structure

Frédéric Merle, Aurélien Bénel, Guillaume Doyen and Dominique Gaïti

Abstract

While systems for collaborative distributed works focus on enhancing distributed work group productivity, little attention has been paid to their architecture. In fact, most of these systems rely on centralized ones for both user communications and data hosting. These architectures raise issues about the administrative control, maintenance and management of the central entity. In this paper, we present a new architecture based on peer-to-peer (P2P) model driven by user relationship. In our architecture, users choose the trusted co-workers they are connected with. Thus, only the most trusted users manage to obtain a high number of connections which grant them a relative authority inside the system.

1 Introduction

Over the past twenty years, numerous new collaborative systems have been developed that greatly improve the ability of distributed groups to work together. While human-computer interaction and new features have attracted much attention over the past few years, these system architectures have not undergone much improvement. In fact, most of them are highly centralized and do not attempt to adjust themselves to group organizations. Indeed, virtual communities, like those in open source projects, are far from being as centralized as the systems they use to work together [4]. Most of these communities adapt themselves to the system architectures by granting some system administration responsibilities to the most trusted members.

Nevertheless, many biases remain due to the mismatch between system architecture and group organization. Firstly many systems are hosted by a third party (service provider or institution) which dictates its own policies to users. Such policies have to be taken into account by users and can alter group organizations. Secondly, the administration of the systems grants to its user an authority over other group members. Such authority driven by architectural considerations can lead to internal tensions inside a community. Finally, cen-

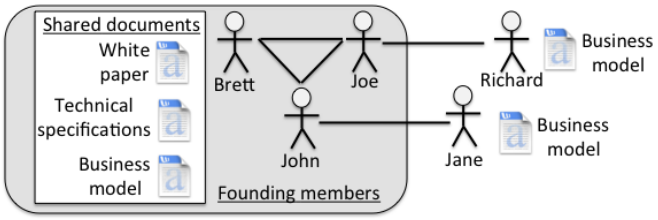

Figure 1: Social organization of the group

tralized architectures lack reliability. Many hazards like technical failures, service closures or changes in policies of use, can affect the system access for users. This kind of situation is even more harmful for collaborative works as most of the time, data is directly hosted inside the system's central entity.

In this paper, we present the use of the peer-to-peer (P2P) model for collaborative systems. Unlike centralized architectures or other P2P systems proposals [6], our aim is to fit the social organization to the working group level. Although, fitting the architecture of an information system to the organizational structure of a company has already been thoroughly studied [1], to the best of our knowledge nothing has yet been undertaken for smaller and more changeable structures like working groups. Thus, we use the relationships between group members to drive the topology of our architecture. To do so, users are connected to co-workers only if they trust them. Thus, the number of connections of a user reflects the trust of his co-workers. Such architecture adapts itself to the changes inside the social organization of the group that can occur. Furthermore, it has the same basic advantages as P2P systems such as scalability and fault-tolerance.

In section 2, we present a realistic scenario of a group of users who collaboratively author documents. Section 3 gives an overview of existing solutions to our scenario. Then, in section 4, we present a system using our socially driven architecture. We analyze its advantages regarding our scenario in section 5.

2 Scenario

We consider a group of users that collaboratively work on a set of documents with a document authoring application. For simplicity, the group has only five members and they work on three documents. Among members, three are the founding members who started the project and have access to every document and the two others are in charge of only some of the documents. Each of those remaining members was asked to help by a founding member. Moreover, only the founding member who invites a new member already knows him/her (Figure 1).

The authoring of these documents is a background task for group members and they do it in their free time. During these moments, they can have access to Internet or not. No specific workflow has been defined for group members.

Thus, everyone can freely edit the documents without any control by other members. However, all users have to be able to keep track of every modification made to a document.

3 Existing solutions

Different solutions are already used for collaborative document authoring, such as online word processors, file synchronization systems or revision control systems.

The most used online word processor is Google Docs.[1] In this system, every document is hosted by Google and can be authored by any authorized users with the web interface. It keeps track of every revision made to a document which allows a user to undo modifications previously made by another user. Its main advantage is to allow co-writers to synchronically edit a document. But this advantage comes with a major drawback (for users): the constant need of an Internet connection.

File synchronization systems, like Dropbox,[2] are used to synchronize files and folders on two or more different computers. These systems use centralized architecture to host the canonical version of each shared folder. Every modification to a shared folder is pushed to this canonical version and applied to every other user's folder synchronized with it. The servers of these systems are always hosted by a company which is subject to policy changes or sometimes even service closures. Furthermore, only the last revision of a folder is replicated on user computers, which means that only the last revision will be replicated. These systems have an embedded a revision control feature which can only be used from the canonical folder web interface which is hosted by the server. Thus, any recovery of older versions of files needs a connection to the server.

Revision control systems are widely used by software developers to collaboratively edit source code and documentation. Like file synchronization systems, a centralized revision control system, like Subversion,[3] uses a server to host the canonical version of the project documents. Thus, any revision control operations or modification submissions need a connection with the server. In decentralized revision control systems, like Git,[4] each user hosts a copy as well as the complete history of revision for every document of the project. Users work on their own version of a project and periodically synchronize it with other users. This synchronization can be done in P2P fashion. However, without any central server, the synchronization becomes quickly unmanageable for more than three users on the same project. Thus, these systems are mainly used with a central server. What is more, these systems are not at all user-friendly as different commands are needed to simply share or update documents. Finally, the merging tools provided with these systems are usually for plain text rather than for office

[1] Google Docs, http://www.google.com/google-d-s/documents/
[2] Dropbox, http://www.dropbox.com/
[3] Apache Subversion, http://subversion.apache.org/
[4] Git, the fast version control system, http://git-scm.com/

documents.

None of those systems are really suitable for our scenario. They all use centralized architectures which place a heavy constraint on users. In this kind of architecture, server administrators have an overwhelming authority over users. Thus, users always have to agree to the policies of the service providers or the IT service which hosts the server. Furthermore, the service providers can close the service or remove the server for internal reorganization or strategical reasons. In addition, the founders of the group are usually given the ability to grant or revoke membership. Even if users are now accustomed to them, these computer models mimic very poorly the social organization and dynamics of a group.

4 Our proposal

We propose to extend a decentralized revision control system with an automated decentralized synchronization feature and a user friendly interface. We chose Git as the decentralized revision control system because of its great popularity. Furthermore, it is built as a set of basic features warped together with scripts, which eases its integration. To automatize the synchronization of the user's works, we use a P2P model based on group member relationships. These systems use a virtual architecture, called 'overlay', to connect user's computers ('peers'), with each other. We use the mutual trust between users to drive the overlay construction, like in a Friend-to-Friend system [8]. Thus, our system can follow the evolution of the social organization of the user's group. Furthermore, this model allows users to collaboratively manage the group by trusting or distrusting other members. To do this, we propose an interface for user relationship management. Then, we use another abstraction of social mechanisms for document synchronization, called 'epidemics algorithm' or 'gossip protocols' [5, 2]. These algorithms broadcast information with low overheads for each peer and high guarantees about information reception for every group member as time goes by.

4.1 Overlay design driven by social organization

In this model, two peers share a connection only if their two owners agree to do so. Consequently, a user can unilaterally break a connection with another peer. Thus, the more a peer is connected with other user peers, the more the user is trusted by the other group members. If a peer does not share a connection with any other member peer, its user is evicted from the group. As a corollary, any user peer who shares a connection with another member peer is also a member of this group. Therefore, any member of the group can invite a new user inside on their own by creating a connection with his/her peer.

The data about every peer inside the system is shared by all of them. This is mandatory in order to allow users to create a new connection between their peers and another member's peer at any time. We extend this data to the list of every peer a peer is connected with. This data is translated to users into the

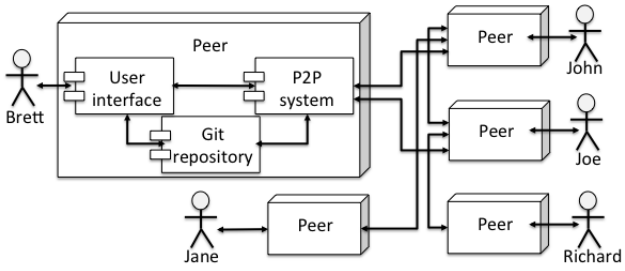

Figure 2: Deployment diagram of our application over the group define into our scenario

trust relationship between them which allows any group member to evaluate the reputation of any other member inside the group. With our design, the overlay structure tends to fit in with the social organization of a group (Figure 2). Thus, mutual agreement between members can create any type of overlay structure.

Technically, to create a connection between two peers, each of them creates a new account for the other with an access to the local project repository to allow SSH connections. The peer also registers the other peer local repository as a Git remote repository. To break a connection, a peer deletes this peer account and removes its project repository from the Git remote repository list. To keep track on any group member, each peer stores two lists: the connected peers list and the peers list. The first one only refers to data about connected peers. The second list contains data about every user peer of the group.

Every part of the overlay management is controlled through a contact management interface (Figure 3). We use the address book metaphor to facilitate user adoption of the system. We enhance this kind of system with two functions. The first is group management. This feature allows a user to create a new group of users with the "add" button at the bottom of the first pane. If he does, a window will ask them for a project name and the path to the synchronized folder. A user can also invite someone from outside the group with "add" button at the bottom of the second pane. If he does, the system will ask for this user peer address in order to send it an invitation to join the group. This user is notified of the invitation through a notification interface. When this user joins the group, he automatically shares a trusted relationship with the user who invited him. The second functionality is a trust relationship management. It allows a user to modify her relationship status with a group member. With the "modify trusted status" button, a user can change the relationship status with the group member displayed on the third pane. Every modification to the status is notified to this user through a notification interface. As the trusted relationship inside the system is bidirectional, this member become a trusted one only after he accepts the invitation. Thus, the trusted relationship reflects both mutual trust between users and their will to work together.

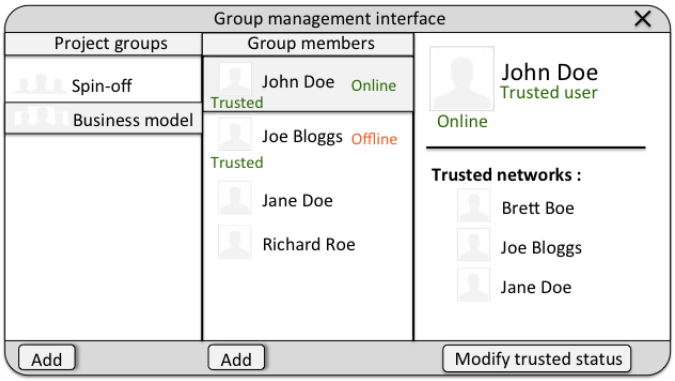

Figure 3: Brett contact management interface (mockup)

4.2 Synchronization process

The system synchronizes both the Git local repository and the user list of every group member peer. To achieve this synchronization, we combine two different epidemic algorithms [5] like in [2]: *rumour mongering* and *anti-entropy*.

The rumour mongering algorithm broadcasts immediately new data with a low guarantee that it reachs every peer. To enhance it, the anti-entropy algorithm periodically synchronizes the latest data known by two peers to fetch any missed rumours. This process automatically starts after a certain amount of time or when a peer returns online.

Git features are very well suited to implement these algorithms: sending new data can be carried out by a simple 'push' and the synchronization of the anti-entropy process done by a 'fetch'. Furthermore, with Git server-side 'hooks', the system can be notified of these modifications and process them. For this reason, we have choosen to use Git for the users list to manage updates. To limit the size of the user's list modification history, all modifications are made inside a 'branch' which is weekly pruned.

For the synchronization management, the user interface is divided into two parts: an icon on the system tray and another which can be found on the files and folders contextual menu. The system tray icon of the system works as a instant messenger one as it notifies the user about the state of their connection with other members and their notifications. If the system cannot find a trusted team member online, this icon is shaded. Under the icon, we have a counter which shows the number of unread notifications. By a click on this icon, the user opens a contextual menu with an access to the contact management interface as its first entry. The other entries are notifications which can be removed by ignoring them or, for invitations, by accepting them.

The contextual menu of a synchronized folder or document is extended with two entries: "Share" and "History". With "Share", the user pushes his revision to every member of the team. It opens an email-like interface (Figure 4) which

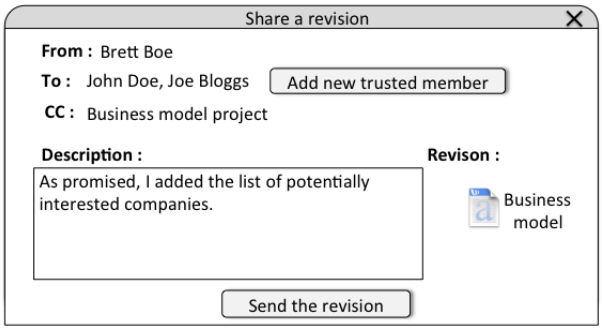

Figure 4: Share interface (mockup)

is pre-filled with the trusted members as message recipients and all the group as a copy recipient. The choice of this kind of interface is driven by the constraints similar of committing a revision and sending an attachment by email. Any revision's commit needs a brief description of the modification as would have been made in the body of an email. Moreover, as trusted members of the group, only message recipients receive a notification of this revision. Finally, the user can add a new trusted member from this interface, as he would do if he added a new recipient for an email. Thus, this metaphor turns ordinary email practice into on of advanced revision control. "History" opens the revision history of a folder or document. The interface of this history is out of the scope of this document but could resemble a simplified version of GitX[5] or SmartGit[6] revision tree visualization.

5 Advantages of our system

Regarding the scenario that we have proposed, our system fulfils most of the requirements. Document authoring and revision control can be carried out by any member of the group during both online and offline periods. The sharing of the documents between founding members and the two remaining ones can be done by having two distinct project folders. A global one shared only between founding members and a sub-folder shared with the entire group which contains only documents for the two other members. Every modification in the organization can be made by mutual agreement between members. Thus, a founding member can be evicted from the project only if the two others agree to it. However, any of the remaining members can be evicted by the founding member who invited them to join the project. Finally, our system is fully managed by the users with no third party policies to restrict the system use. The limit of our system, compared to centralized ones, is that members work is synchronized only if two trusted members are online at the same time. However mutual online

[5]GitX, http://gitx.frim.nl/seeit.html
[6]SmartGit, http://www.syntevo.com/smartgit/features.html

periods are very likely, e.g. during office hours, and this limit only applies to small groups.

Our model can be used for many different community organizations or applications. Indeed, our social-driven construction of the overlay is scalable and can develop the same properties as structured overlays have [3]. Thus, most of the P2P systems previously proposed can be used with our overlay construction [6, 7]. Thus, our model is very versatile and can easily support large communities as well as a large range of features.

6 Conclusion

In this work, we have proposed a new kind of overlay construction for collaborative P2P systems. Our system uses social relationships between users to drive the overlay construction. Thus, it fits the social organizational structure of the group. With this overlay construction, we avoid organizational limits of centralized architectures like third party policies, or the overbearing authority of any server administrator. Furthermore, as a P2P model, our system provides higher scalability and reliability, as well as wider offline features than centralized systems. We have illustrated these advantages over existing solutions on a realistic scenario of collaborative document authoring. This work could be the first step towards a fully decentralized infrastructure for various collaborative software.

7 Acknowledgement

The authors are thankful to Peter Jacobs for his comments about a preliminary version of this article.

References

[1] Y. Chan and B. Reich. IT alignment: what have we learned? *Journal of Information Technology*, 22:297–315, September 2007.

[2] F. Cuenca-Acuna, C. Peery, R. Martin, and T. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 236, June 2003.

[3] M. Dell'Amico. Mapping small worlds. In *Peer-to-Peer Computing*, pages 219–228. IEEE, 2007.

[4] D. Demazière, F. Horn, and M. Zune. La dynamique de développement des 'communautés' du logiciel libre: conditions d'émergence et régulations des tensions. *Terminal, technologie de l'information, culture et société*, (97-98):71–84, 2006.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, August 1987.

[6] P. Gotthelf, A. Zunino, and M. Campo. A decentralized middleware for groupware applications. In *international conference on Groupware: design implementation, and use*. Springer-Verlag, September 2007.

[7] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *Conference on Computer Supported Cooperative Work*. ACM, November 2006.

[8] M. Rogers and S. Bhatti. How to disappear completely: A survey of private peer-to-peer networks. In *International Workshop on Sustaining Privacy in Autonomous Collaborative Environments*. IFIP, July 2007.