DooSo6: Easy collaboration over shared projects

Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli

DooSo6: Easy collaboration over shared projects

Claudia-Lavinia Ignat, Gérald Oster and Pascal Molli

LORIA, INRIA Nancy-Grand Est, Nancy Université, France
{ignatcla,oster,molli}@loria.fr

Abstract. 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 favourite editors and impose them to use a certain co-authorship application. In this paper, we propose 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 favourite editors. User authentication is achieved by means of a capability-based mechanism.

Keywords: collaborative systems, version control, capability-based security

1 Introduction

Collaboration is a key requirement of teams of individuals working together towards some common goal and hence of importance to any organisation - be it business, science, education, administration or social. A great deal of information central to the operation of an organisation is held in documents and therefore support for sharing and collaboration over documents has to be offered.

In spite of the existence of specialised collaborative writing tools, very often when people want to collaboratively author a document they edit locally the document and then they send emails to the other members of the group transmitting them their version of the document as an attachment [12]. Users have then to manually integrate the changes done in parallel, which might become a difficult task.

Studies have shown that the most desired features users need for supporting their group work are control over document versions and concurrent access to the shared documents [12]. Concurrent access means that while a user is editing a document, the other users have the possibility to consult the document or edit it. However, existing tools that offer support for collaborative parallel work feature some complexities of use. The software must be correctly installed at the site of all users that collaborate independently of users operating system, users need to accept this software and it must be usable by all users. As shown in [8], installing and using software can pose several problems. Installation of software

might be a difficult task for certain users. Moreover, very often learning how to use a tool requires a great deal of effort and users might easily abandon that tool. Furthermore, for certain users it might be a problem renouncing at their favourite word processors in order to use a coauthorship application. Moreover, existing tools for parallel work require that either each client opens an account or the project administrator creates accounts for each client and assigns clients to the collaboration group.

In this paper, we propose a collaboration tool over shared projects that offers the basic functionalities of a version control system for supporting parallel work and overcomes the problems mentioned above. Each user maintains locally a copy of the shared project. Users can perform modifications on their local copy and publish their changes on the repository containing the shared project at a later time. The collaboration tool requires no installation and no accounts for the users. Using the tool is very easy and users do not need learning complex commands necessary for tool functionality. Moreover, users can edit their local copy of the project with their favourite editor and are not obliged to use an imposed editor. Furthermore, as group awareness is a very important feature of a collaborative system [5], we integrate some basic aspects that provide information about group members activity.

Concerning the aspect of easiness of use of our tool we were inspired by the shared agenda called Doodle [4] that helps finding suitable dates and times for group events such as an appointment, a conference call or a family reunion. Doodle is an online tool that does not require any software installation and no account creation for using a shared agenda. However, Doodle is limited to the use of a shared agenda and cannot be used for collaboration over shared projects.

For dealing with authentication issues we use a capability-based security [9] for restricting actions of updating the local workspace and publishing changes to the shared repository only to the members of the group that possess these rights.

The paper is structured as follows. In section 2 we start by presenting by means of an example the limitations of existing systems that offer support for sharing and collaboration over a project. In section 3 we present our DooSo6 collaborative system for supporting parallel work and we show that it is very easy to use. In section 4 we give some details about our synchronisation mechanism. Concluding remarks and directions for future work are presented in section 5.

2 Motivating Example and Related Work

Consider the example of two researchers of two different universities that wish to collaborate on a paper. They have several approaches for sharing the source documents of the paper. A first solution would be to ask the system administrators of their universities to create accounts for the remote user. However, this solution imposes an administrative burden that does not scale with a large number of users and projects and needs a careful analysis of the rights given to the external user. Often, the latency for opening an account is large and users

cannot accept it. Moreover, sometimes institutions do not allow opening guest user accounts for people that do not work for that institution.

The two users could use individual word processors for editing the document and then sending each other emails containing as attachment versions of the document that integrate their changes. This work mode requires a good planning of activities. Integration raises no difficulties if people work sequentially. Manual integration in the case of parallel work can be easily performed by a coordinator if the document is well segmented and the different document segments are assigned to different authors. However, if the document decomposition is not possible and the number of collaborators is large, manual integration of parallel changes becomes complex. The number of exchanged document versions grows proportionally with the number of collaborators and therefore user changes in the exchanged versions are more difficult to be tracked.

Another solution would be that users put their documents on the web which is unacceptable as documents are often confidential. There exist some online document sharing services that offer solutions for a secured space which can be used as a virtual drive. Unfortunately, these solutions do not deal with concurrent modifications of the same document. Therefore, users have to manually perform versioning and merging of their shared documents.

Another option for the two users is to use real-time collaborative editors such as SubEthaEdit [14] to edit shared documents. These editors offer an awareness mechanism about the activity of the other group members as it shows the users that are editing a document at the same time. However, this solution requires that the two users install these tools and use these editors that they are not familiar with. Very often projects contain documents other than textual ones for which the real-time editors might not offer support such as latex, code source or XML documents. These documents need to be compiled with special tools and they need to be kept in a stable state that prevents continuous integration of concurrent changes. Therefore, users need to make a copy of the shared documents each time they need to compile them. If, for resolving issues at compilation time, the users modify the local documents, they have to manually re-integrate their changes into the real-time editor. Manual integration is necessary as real-time editors do not offer support for work in isolation, i.e. users cannot work in their private workspaces and synchronise their changes with a shared repository. Moreover, these editors synchronise in real-time changes done on the same document, but do not synchronise changes targeting shared directories.

The burden of tool installation is eliminated in the case of web-based collaborative editors such as GoogleDocs [7] and ZohoWriter [16] at the price of necessity of creation of user accounts. All the other disadvantages previously mentioned for real-time editors hold for web-based collaborative editors.

Version control systems are popular systems for supporting parallel work. Centralised version control systems such as CVS [1] and Subversion [2] require that a server is installed by an administrator which has to create also user accounts for all project collaborators. Users have then to install the client applications. Distributed version control systems such as Git [6], Darcs [3] and Mercurial

[10] eliminate the need of a server, but require that users install locally the client applications. In distributed version control systems users maintain locally their data and their changes and can push their changes to different channels. Other users that have granted rights may pull these changes from these channels. These channels may be hosted on users' hardware. However, for maintaining channels permanently available for allowing authorized users to pull changes from those channels and for avoiding firewall problems, users of distributed version control systems often push their changes on dedicated servers that have to be installed for hosting their projects. Therefore, all version control systems require an installation process that might be difficult to be achieved by certain people. Moreover, these tools are quite difficult to be used by non developers as they require an understanding of how the system functions and a good knowledge of the set of commands.

This section proved the need of a collaborative system that offers support for parallel work, requires no installation and no burden for users to create accounts and log on for using the system. Moreover, the collaborative system should be easy to use. Users should be able to use their favourite editors for editing changes on the project and do not be obliged to use an imposed editor. Furthermore, group awareness mechanism about the activity of the other members of the group should be offered. In the next sections we present our approach that offers all these features.

3 DooSo6: easy collaborative system

The basic methods that have to be supplied by a collaborative system that supports the parallel work of a set of users are checkout, commit and update. A checkout operation creates a local working copy of the project from the repository. A commit operation creates in the repository a new version of the project based on the local copy, assuming that the repository does not contain a more recent version of the project than the local copy. An update operation performs the merging of the local copy of the project with the last version of that project stored in the repository. In what follows we describe all the phases necessary for setting up and using the DooSo6 framework.

3.1 Project setup

For setting up a project, the initiator of the collaboration has to create first a repository as shown in the Figure 1. The project initiator is then required to fill in a title of the project together with a short description. As shown in Figure 2, the user is then provided with a link for the administration of the project such as deletion of the project, a link for committing changes to the repository and a link for updating changes from the repository.

These links represent system capabilities. A capability is defined to be a protected object reference which, by virtue of its possession by a user, grants that user the capability to interact with the object in certain ways. For instance, the

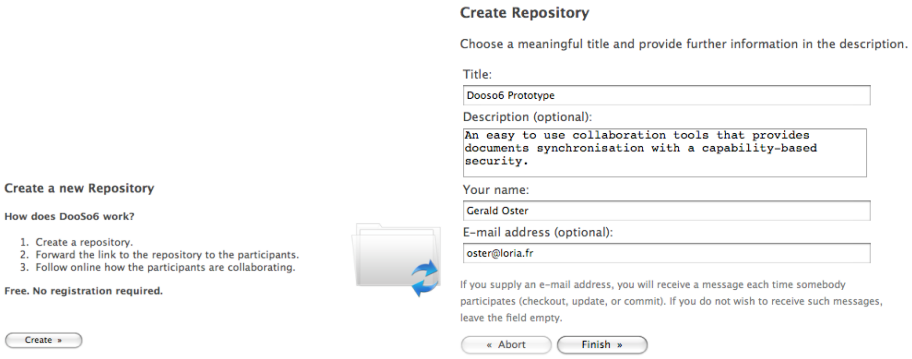

Fig. 1. Creation of a repository

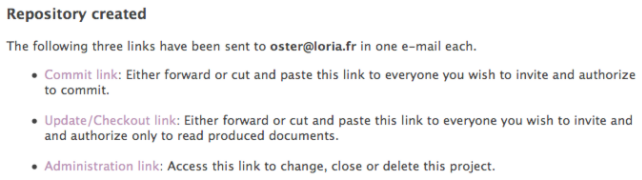

Fig. 2. Links that represent system capabilities

`Administration link` in Figure 2 is a key that gives the initiator the right of administration of the project. The initiator is supposed to do not distribute the key to other users. On the contrary, the initiator should forward the key for committing to the repository and updating changes from the repository to the other members involved in the project. The commit and update keys offer users that are in their possession the capabilities of updating and committing respectively to the repository. It is possible that the key for updating is distributed to some users and these users do not receive the key for committing. It means that these members of the group have only the right to update changes from the repository and do not have the right of committing changes to the repository.

3.2 Check-out/Update of the project in the local workspace

For performing a check-out or an update a user has to follow the update link, being provided with the interface shown in Figure 3.

The user can see the list of the other members of the group that already checked out the project and started working on it. Each member is associated with his up-to-date status, i.e. the percentage of the number of commits in the repository that he integrated in his local workspace. If the user has not yet performed a checkout, his name does not appear among the members of the group. If, however, the user previously performed a checkout, the name that he

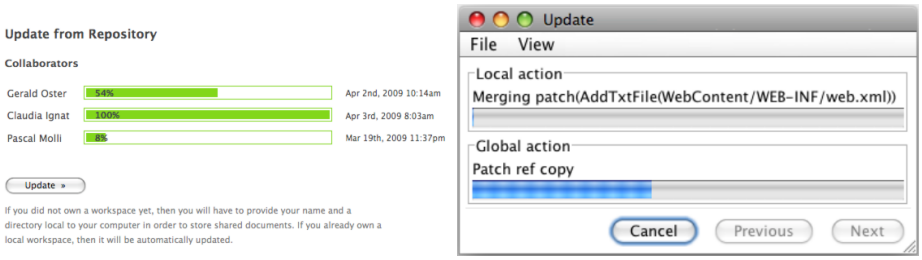

Fig. 3. Updating from the repository

provided appears in the list of names together with his up-to-date status. Note that user names are used only for awareness purposes and not for authentication.

When the `Update` button is clicked, the client application is downloaded and launched. This is achieved by using the Java web start technology. Server sends the update capability to the client application. If `Update` button is clicked for the first time, i.e. a checkout is performed, user is asked to provide a name and a path to a local directory where his copy of the project will be stored. This path is stored by the local file system and will be used for next updates. Afterwards, the local workspace is synchronised with the latest version of the project from the repository.

3.3 Committing changes to the repository

If the user wants to commit changes that he performed locally he is presented with almost the same interface as the one shown in Figure 3 with an additional `Commit` button. If `Commit` button is pressed and the user is not up-to-date, he is advised to perform first an update. If, however, the user is up-to-date, he can commit the changes he performed locally. Afterwards, the client application is started and commit procedure executed. The system automatically detects the patch of operations performed locally since the last update and sends this patch to the repository.

3.4 Why collaboration is easy with DooSo6?

As we use a capability-based security mechanism according to which users that own a key have the right of updating or committing to the repository, we do not need to authenticate users by means of logins and passwords. Therefore, users do not need to create accounts and logins for using the system.

The facility of non-installation of software is achieved by means of technology. The system is written in Java which is a platform independent language and therefore we do not have problems of incompatibility between operating systems. By using the Java Web Start technology, our Java software application can be started by clients directly from the Internet simply by using a web browser.

Our collaborative system is easy to be used as users do not need to learn and type commands for publishing and synchronising their changes. Users have basically two buttons for activating the actions of updating their local version of the project and for publishing the changes to the repository.

Changes on the local copies of the shared project can be edited using any editor. At the moment when synchronisation is performed between the local project and the shared one in the repository, directories and files belonging to the project are synchronised according to their content (textual, XML or binary).

An awareness mechanism is offered for tracking group activity – users can see their up-to-date status of the project and the delay with which users follow the project. From our first experimentation we observed that, due to this awareness visualisation, users try to provide a good impression to the group by contributing and keeping their local copies of the shared projects up-to-date.

4 The DooSo6 synchronisation mechanism

In this section we present a short overview of the DooSo6 synchronisation mechanism. The DooSo6 system contains two main components: the DooSo6 repository and several DooSo6 workspaces associated to different clients. The DooSo6 repository hosts a timestamper and a sequence of patches of operations. A patch of operations contains a list of operations representing changes performed by a certain user. When a user executes a commit, the list of all operations locally performed is sent as a single patch to the repository. The list of operations is obtained by executing a diff [11] algorithm between the last updated and the current local version of the project.

A DooSo6 workspace stores all documents shared by a user. Operations locally performed by that user are saved in a log of operations. The DooSo6 workspace also keeps the timestamp of the last delivered/received patch to/from the repository. Initially the local timestamp is set to 0. When a user wants to publish his changes into the DooSo6 repository, he needs first to request a timestamp. According to this received timestamp the user is allowed or not to save his changes. If the timestamp received equals to the value of the local timestamp, the user is allowed to save his changes. A patch of operations containing the operations locally generated since the last time the user committed is sent to the DooSo6 repository. If, however, the timestamp received from the DooSo6 repository is greater than the value of the local timestamp, the user needs first to update his local workspace. The DooSo6 repository sends the workspace the list of unconsumed patches. The client then merges the list of operations representing the unconsumed patches with the local operations and applies them to the workspace.

The merge algorithm used by DooSo6 is adapted from SOCT4 [15]. The algorithmic description of the checkout, update and commit procedures is similar to the one of the same procedures presented in the So6 system [13]. The main difference between the DooSo6 merge mechanism and the So6 merge mechanism is that DooSo6 works with patches of operations while So6 works with operations.

5 Conclusion

In this paper we presented a very easy to use system for the collaboration over a shared project. Users do not need to open accounts for having access to the shared projects and do not need to learn any commands for using the system. Moreover, they can use their favourite editor for performing changes to the project. We use a capability-based security for restricting actions of updating and committing changes to the shared repository only to members of the group.

Currently, our system uses a centralised repository. Besides issues of limited scalability, lack of shared administration costs and limited fault tolerance, data centralisation is an inherent threat to privacy. In order to overcome these issues, we investigate to store repositories containing patches of operations on a distributed hash table.

References

1. B. Berliner. CVS II: Parallelizing Software Development. In *Proceedings of the USENIX Winter Technical Conference*, pages 341–352, Washington, D. C., USA, Jan. 1990.
2. B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version control with Subversion*. O'Reilly & Associates, Inc., 2004.
3. Darcs. *Distributed. Interactive. Smart*. http://darcs.net/.
4. Doodle. *Easy scheduling*. http://www.doodle.com/.
5. P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, pages 107–114, Toronto, Ontario, Canada, 1992. ACM Press.
6. Git. *Fast version control system*. http://git.or.cz/.
7. GoogleDocs. *Create and share your work online*. http://docs.google.com.
8. A. Grasso, J. L. Meunier, D. Pagani, and R. Pareschi. Distributed coordination and workflow on the world wide web. *Journal of Collaborative Computing*, 6:175–200, 1997.
9. H. M. Levy. *Capability-Based Computer Systems*. Butterworth-Heinemann, 1984.
10. Mercurial. *A fast and lightweight Source Control Management system*. http://www.selenic.com/mercurial/.
11. E. W. Myers. An O(ND) Difference Algorithm and its Variations. *Algorithmica*, 1(2):251–266, 1986.
12. S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work - JCSCW*, 13(1):63–89, 2004.
13. G. Oster, H. Skaf-Molli, P. Molli, and H. Naja-Jazzar. Supporting Collaborative Writing of XML Documents. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS'07)*, pages 335–342, Funchal, Madeira, Portugal, June 2007.
14. SubEthaEdit. *Collaborative text editing. Share and Enjoy*. http://www.codingmonkeys.de/subethaedit/.
15. N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies Convergence in a Distributed Real-Time Collaborative Environment. In *Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, pages 171–180, 2000.
16. ZohoWriter. *Easy Online Word Processor*. http://writer.zoho.com/.