Multi-User Aware Cloud Storage

Multi-User Aware Cloud Storage

Master's Thesis

Oleksiy Syrotkin
<soleksiy@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Dr. Michael Nebeling, Matthias Geel

Global Information Systems Group
Institute of Information Systems
Department of Computer Science

3rd October 2013

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

Collaborative work in shared folders is one popular use case of personal cloud storage services (e.g., Dropbox, Google Drive). In such collaborative scenarios, some of the following problems occur: poor awareness of team members' latest activities, difficulty in tracking moved and renamed files, lack of understanding of the origin (provenance) of files, reluctance to delete other users' files, and the need to use external communication channels to notify other team members. As part of this thesis, we developed MUBox, a web application that serves as a platform for experimenting with different features to help alleviate the problems. We implemented features to improve awareness (activity view), file traceability (shadow files and traces) and asynchronous collaboration (voting). We evaluated the use of the activity view and shadow files in a user study where participants had to perform realistic folder exploration tasks. The tasks were designed to make the users focus on the activities of the collaborators (workspace awareness) and operations performed on files (file traceability). The results of the study showed that the activity view has a significant effect on users' accuracy and confidence performing the awareness-related tasks. Shadow files significantly improve users' speed, accuracy and confidence in traceability tasks. We can therefore argue that the implemented features are useful for increasing workspace awareness and file traceability.

Acknowledgements

I would like to thank Prof. Dr. Moira C. Norrie for allowing me to write my thesis in the GlobIS group. I am sincerely grateful to my supervisors, Dr. Michael Nebeling and Matthias Geel, for all the time, guidance and advice they have given me. Many thanks to the anonymous users for agreeing to participate in the user study and providing useful feedback.

Contents

1 Introduction

This chapter discusses the motivation for our research focus, lists the objectives of the thesis, summarises the contribution of the work, and presents MUBox, a platform for our research, developed as a web interface for accessing cloud-based storage services.

1.1 Motivation

In recent years, the popularity of personal cloud storage providers, such as Dropbox[1] or Google Drive,[2] has been growing. Individuals have increasingly been using such services to synchronise their personal data among various devices as well as to share files with their friends or colleagues. As a relatively new and developing technology, that has already acquired a substantial user base, cloud storage services represent an interesting field for research.

This thesis focuses on collaborative work between users of cloud storage services, as opposed to, e.g., the use for personal data backup or file transfer. We are particularly interested in scenarios where users share folders with other people. We think that existing features of popular cloud storages services could be enhanced to make the user experience better, improve users' awareness of the activities of other collaborators and encourage users to participate more actively in managing the shared folder structure.

The following is a description of a few shortcomings of the existing cloud storage user interfaces. We reviewed the following providers: Dropbox, Google Drive, SkyDrive.[3] These are some of the most popular personal cloud services. Throughout the thesis, we will draw examples from our experience with these services. For example, when a user renames a file

[1]Dropbox. URL: `https://www.dropbox.com` Accessed: 18.09.2013
[2]Google Drive. URL: `https://drive.google.com` Accessed: 18.09.2013
[3]SkyDrive. URL: `https://skydrive.live.com` Accessed: 18.09.2013

in some of those services, another file with the new name appears, but the file with the old name can still be visible if the user chooses to display deleted files. The possible problems that collaborators may encounter in those services are, firstly, that the old file is marked as deleted, thus possibly confusing the rest of the team, and, secondly, that there is no apparent connection between the old and the new filenames. Users would need to open the files to see if the two entries actually have the same contents. Some personal cloud provider interfaces simply rename the file without retaining a copy with the old name, and users may still find it cumbersome to trace the renamed file back to the original one. Moved files may be easier to identify, but nothing about the appearance of the file's original folder indicates that the file has been moved. A user would either have to navigate to a few other folders or use a search function.

Given the aforementioned shortcomings, we decided to experiment with the mechanisms that we thought could improve folder sharing experience and make awareness of the other users' actions and file traceability easier to achieve.

On the one hand, powerful awareness features exist in source control software, such as Git and Apache Subversion, and their various GUI clients. On the other hand, document versioning systems support collaboration by imposing a specific workflow on the users, by providing access-control mechanisms or introducing complex role management. However, such features, while usual for professional information workers, may be difficult to understand for an average user without special training. Therefore, in this work we focus on the services that cater to mainstream users. We aim for a lightweight solution that could resemble the existing personal cloud user interfaces and be easy to start working with. Our objective is to implement features designed to increase individuals' workspace awareness and file traceability without making the users adhere to a specific workflow.

Some existing cloud storage providers already offer solutions optimised for use by teams. However, one problem with such a vendor-specific solution is that it is difficult to migrate the collaboration settings to a different provider. In cases when the same person is part of multiple teams, each using different service providers, this user has to become familiar with the peculiarities of each provider's user interface. This could be time-consuming and unproductive. Therefore, an important part of this research is to create a front-end solution that could be extended to various cloud service providers. To the best of our knowledge, there has not been a study concerned with the implementation of lightweight features to support workspace awareness and artefact traceability in collaborative work using personal cloud storage providers.

1.2 Goals

We aim to improve users' awareness of the activities of collaborators in shared folders in personal cloud storage services. We also want to improve file traceability by exposing move or rename relationships between files. Another goal is to increase user participation in maintenance tasks on the shared folder structure both by making the users' changes more visible and also by letting the other users agree on the changes. We aim for a solution that adds features to a common personal cloud storage interface without imposing a specific workflow.

We also want to make our solution provider-independent. First, we plan to experiment with a few features. Then, to validate the results and evaluate the utility of the most promising added features, we plan to conduct a user study.

1.3 Contribution

We developed MUBox (Multi-User Box), a web application that lets users access the functionality of cloud storage providers. One of the important features in our interface is an activity view. The activity view is a stream of file operations for a given folder and subfolders. This view aims to increase a user's awareness of the actions of the people, with whom the folder is shared. We provided an implementation of shadow files with traces that may point to an older or newer copy of the file. This way, traces create links between otherwise not visibly related files and thus contribute to file traceability. Further, we developed a voting infrastructure with notifications as a way to encourage users to modify the shared folder structure, particularly delete files, without fear that someone may need the file later and will not be aware of the deletion. In addition, we defined a cloud service abstraction that describes the necessary features a personal syncing and sharing service needs to possess to be compatible with our web interface. This was done to demonstrate that the solution is independent of a particular cloud storage provider. Currently, we have implementations for Dropbox and Google Drive. Having defined an abstraction layer, we made it a more straightforward task for subsequent developers to extend the implementation to other cloud providers if needed. Our user study evaluates two of the implemented mechanisms: shadow files with traces and the activity view. According to the study, shadow files significantly improve users' confidence, accuracy and speed in understanding file-to-file relationships. The activity view significantly increases people's confidence and accuracy in understanding the actions of collaborators, but it does not have a significant impact on users' speed of grasping the essence of the activities.

1.4 Thesis Overview

In Chapter 2, we will introduce our perspective on personal cloud storage and discuss related work in the fields of awareness, file provenance and users' behaviour with cloud storage services. In Chapter 3 we will describe the areas where the proposed solution aims to make an improvement and discuss how the solution will achieve the goals we set forth. Chapter 4 describes the design of MUBox, our web application and a platform for experimenting with new features. The chapter also presents our choice of technologies and implementation details of the solution. We will discuss the organisation of the user study and present the related results in Chapter 5. Chapter 6 will briefly review the thesis and discuss the limitations and possible improvements.

2 Background

This chapter describes the research areas where the thesis aims to make a contribution, and reviews the definitions and previous work in the fields of users' behaviour in file sharing and cloud storage systems, awareness in collaborative work, and file provenance.

In this thesis, we focus on personal cloud storage services. We use this term to refer to service providers that store people's personal data and settings and make them available online. Examples include Dropbox, Google Drive, Microsoft SkyDrive. Many services offer free registration and a limited storage quota and are open to a wide variety of users. We specifically concentrate on such personal data storage as opposed to other cloud-based service providers, e.g., cloud computing platforms where users can develop and host web applications, such as Google App Engine[1] or Amazon Web Services.[2]

Many personal cloud storage services enable users to share their folders with friends or co-workers. For a long collaborative project, sharing folders is a welcome alternative to emailing files as attachments because these services usually offer versioning capabilities and some of them also let users synchronously edit documents online. Furthermore, files can automatically be synchronised across the user's multiple devices.

Asynchronous collaboration tools allow people to work on the same team, possibly at different times [19]. Xu et al. [19] provide a review of tools used for asynchronous collaboration between users. The researchers point out four main categories of tools: communication, information sharing, electronic calendar and project management. Information sharing applications include file sharing systems, discussion boards, wikis and polls. Cloud-based storage services would naturally fall under the category of information sharing, and for purposes of classification can be thought of as file sharing systems.

[1]Google App Engine. URL: `https://appengine.google.com` Accessed: 18.09.2013
[2]Amazon Web Services. URL: `http://aws.amazon.com` Accessed: 18.09.2013

Insufficient visibility of activities of collaborators has been named as one of the problems in sharing scenarios. Voida et al. [18] reports that the lack of notifications and lack of visibility were some of the biggest problems in file sharing systems.

In the cloud storage domain, awareness of the activities in the workspace remains a problem. Voida et al. [17] investigates user experience with services such as Dropbox and Google Docs[3] (the document management and editing aspect of Google Drive) across multiple collaboration scenarios. The interviewed users complained about insufficient visibility of collaborators' activities and difficulties in being aware what parts of the shared folder structure one is responsible for.

There is a growing amount of research on personal cloud storage and shared information repositories and associated users' behaviour. Marshall and Tang [12] contend that users experience problems understanding the functionality of cloud providers because users have inaccurate conceptual models of the cloud. The authors propose design guidelines that cloud services should adhere to in order to become more accessible to the users: notifying the user when a given file was last accessed; adding visual status indicators of syncing; phrasing user messages and labels accurately and unambiguously. The authors also argue that vendors need to work on a new conceptual model of cloud storage instead of trying to exploit a familiar paradigm, e.g., a file system or a web-based collaboration application, in a new setting. Rader [14] has investigated users' behaviour in group information repositories to find out that users mostly edit the files they themselves have created and are reluctant to delete the files that could be useful to someone else in the future. Such behaviour results in clutter and frustrates users.

We think that some of the usability problems and misconceptions could be tackled with proper awareness and asynchronous collaboration mechanisms that would notify the other users if an important file has been deleted with the ability to revert the deletion.

2.1 Awareness In Collaboration Scenarios

Omoronyia et al. [13] provides an extensive review of different techniques used to increase workspace awareness and context awareness in distributed software engineering projects. The techniques aim to enhance a person's understanding of the activities of other collaborators, as well as the knowledge about tasks and artefacts.

Although various kinds of awareness exist, e.g., informal awareness, group-structural awareness, social awareness, the review emphasises workspace awareness, "knowledge of collaborators' interactions with a shared workspace and its artefacts". It also argues that context awareness ("the evolving internal and external state information that fully characterises the situation of each entity in a shared environment") is particularly important to distributed collaboration.

The review mentions various techniques that have traditionally been used to increase awareness within teams. Many of these are text-based: email, mailing lists, chats (instant messaging or group chats), RSS feeds and wikis. Collaborators frequently use version check-in logs in

[3]Google Docs. URL: `http://docs.google.com` Accessed: 25.09.2013.

configuration management systems and commit logs in revision control systems to familiarise themselves with the actions other people take. There are also examples of non-textual solutions, such as media spaces — persistent video and audio connections between remote offices. Users' presence indicators provide informal awareness.

The review goes on to focus on the research and commercial products that have implemented more elaborate and innovative awareness features. Noteworthy examples include the use of tagging (Jazz,[4] TagSEA [3], CASS [9], Augur [7]). Tags are used to help developers navigate the code and to enable them to subscribe to the notifications on the topic of interest.

Other research projects were dedicated to exploring the relations between software project entities and visualising them, such as Rational Team Concert [2]. The visualisations provide a view of the project that one can quickly grasp, thereby improving traceability of software components. The authors argue that this, in turn, helps to increase developers' awareness.

Another approach to awareness is to track developers' interactions, e.g., FASTDash [1] provides information about which class and method a given developer currently has checked out and is working on, signalling potential conflicts.

Information about developer interactions and artefact relations can be combined, e.g., Team Tracks [4] marks more frequently accessed parts of the code as more important. Furthermore, a notion of time can be added, so that more recently accessed artefacts weigh more than the less recently accessed ones. This is the approach Mylyn [11] takes.

Workspace awareness has been shown to be an important topic in other systems, not necessarily concerned with software development. One of the earlier studies of awareness [5] focused on workspace awareness as a productivity factor for designers.

Some of the techniques for improving awareness are applicable to the domain of personal file sharing. Google Drive has presence indicators, chatting capabilities, an activity stream. Dropbox allows users to send an email to the collaborators without leaving the web interface and also provides a list of events in all of the user's workspace or in a specific shared folder. The events essentially represent collaborators' operations on files.

2.2 File-To-File Relationships

Provenance of a file has been described as the history of ownership and operations performed on the file ([16], [8]). A new file can be generated as a result of copying an entire file or parts of an existing document into a new document, attaching files to email messages or using the "Save As" menu in office applications.

Various systems that capture and manage provenance information have been developed, some of them application-specific. Chimera [6] was developed for use with large scientific data repositories. It stores data derivation procedures and derived data and lets users query the data and provenance metadata using the Virtual Data Language (VDL). FiPS [16] is an example of an application-independent provenance system. It is a file system that not only manages files, but also captures and manages file provenance information. It is implemented as an

[4]Jazz Community Site. URL: `https://jazz.net` Accessed: 14.09.2013.

operating system kernel module. By intercepting file system calls it collects metadata that lets it recreate the history of a file. It was designed as a portable system that could be added on top of any conventional file system.

Provenance data have also been used in the field of personal information management. Karlson et al. [10] proposes the concept of a versionset — a set of files connected by provenance relationships — as a unit of information management. The researchers developed a prototype that hooked into the operating system and office applications to capture what they refer to as "copy events", such as SaveAs, Attach, Copy/Paste, etc. Using this information, it was possible to visually represent a versionset in the file explorer in a way that made the newest version of the file the most prominent one and hid some of the oldest versions. The prototype also showed the users a representation of the versionset as a directed graph whose nodes were files and edges were copy events, or provenance relationships, between the files. The interviewed users found that using versionsets reduced clutter. They thought the new folder visualisation was especially useful to make sense of unfamiliar data, e.g., their co-worker's folder.

Since some personal cloud-based storage providers use the familiar folder and file paradigm, identifying provenance relationships between the files in shared folders in the cloud is also relevant. Tracking renamed or moved files becomes more difficult in such sharing services because of potentially a large number of people who have access to the folder and can make modifications. We hypothesise that some form of capturing and displaying file provenance information could be applied to the contents of shared cloud folders. This way, users may find it easier to track the origin of new files in shared folders and to identify which files are merely renamed copies of previously existing files.

3 Approach

This chapter presents possible problems with the current state of affairs in personal cloud-based storage systems, defines the terminology used throughout the thesis, as well as conceptually describes the proposed solutions in the fields of workspace awareness, file traceability and facilitating asynchronous collaboration.

The central focus area of the thesis is collaboration in shared folders in personal cloud storage services. Many cloud services let a user share an entire folder with one or more other people. The folder becomes visible in the other people's folder hierarchies, and they can add or remove files from it. Although some differences exist in how various cloud services treat the original owner of the file and the rest of the team, we are specifically interested in the case where there is no explicit role management and any user can delete or rename a file. When the number of collaborators grows, it may become increasingly more difficult to develop an understanding of the ongoing activities in the shared folder.

We draw our experience from the Idea Garden[1] EU FP7[2] project where a few researchers from our institution participate. The project aims to develop a creative learning environment to support designers throughout the creative process. More than 30 people from 8 organisations are members of the project. The team uses a Dropbox shared folder hierarchy as a repository for documentation, milestones and deliverables. Because of the large number of users, the level of activity in the shared folder is high, and it is difficult to be aware of the actions of the other team members. Some project members also have other work or collaborate with other teams; therefore, not everyone tracks the evolution of the shared folder very closely. For the users who rarely log in, it may be especially challenging to understand the changes that

[1]Idea Garden: An Interactive Learning Environment Fostering Creativity. URL: `http://idea-garden.org` Accessed: 02.10.2013.

[2]European Commission: CORDIS: FP7 : Home. URL: `http://cordis.europa.eu/fp7/home_en.html` Accessed: 02.10.2013.

occurred since the last login. Based on this project experience, we can identify the following problems the users might face:

1. Lack of overview of the activities of particular people in the team. Having such an overview would be beneficial to get up to speed with the work of the teammates.
2. Tracking the origin of files. When a new file appears in a shared repository, it may not be clear to the user whether the file has been uploaded (created), copied from another subfolder of the same shared folder or it is a renamed version of a previously existing file. Some cloud storage providers list the user who was the last to modify the given file, which may give some clues, although it may still not be clear what the particular file operation was.
3. Tracking files that disappear. If a particular file is not in the folder anymore, it may have been deleted, moved or renamed. It is not always obvious what exactly happened.
4. Hesitation to delete or otherwise interfere with the colleagues' files. In case every collaborator is entitled to delete any file, some people still have reservations and prefer not to cause any damage and limit their activities only to their own files, which over time may result in an unwieldy folder structure [14].
5. Need to use external communication channels (most likely, email) to alert the collaborators to one's new activity.

Our goal is to address problem 1 by raising collaborators' workspace awareness. Our approach to awareness is to present the changes in the shared folder in a systematic way to ensure that it is possible to identify the activities of a given collaborator. We plan to tackle problems 2 and 3 by introducing shadow files and traces. Shadow files are placeholders for the files that have been moved or renamed, so that no files appear missing. Backward traces will point from a file to a previous copy of this file, if such a copy exists. Thus, tracking the origin of files should become easier. We will address problems 4 and 5 by improving asynchronous collaboration via the use of voting. Firstly, the voting system will notify the collaborators of pending changes. Secondly, no files should be deleted without the other users' consent. The rest of the chapter conceptually describes the proposed features in detail.

3.1 Awareness

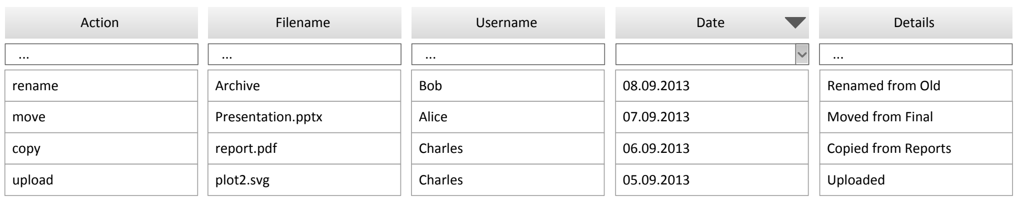
Users' workspace awareness is integral and critical to successful collaborative work [5, 13]. In our research we will not try to distinguish between different kinds of awareness. Instead, we will use the following definition of awareness in Omoronyia et al. [13] as a guideline:

> "an understanding of the activities of others, which provides a context for your own activities".

We have specifically decided to focus on the awareness of file operations the users perform. In a file repository, changes to files are some of the most important activities carried out.

Our attempt at improving awareness includes the design of an activity view as shown in the mockup in Figure 3.1. This view is designed to capture the following file operations:

creating a folder, uploading a file, copying, renaming, moving, deleting a file or a folder. The view must make the important file operation details available to the user: the name of the operation, the affected file, the user who made the change, the time of the change, and any relevant details. In case of a rename, move or copy, the details would include either the source or the destination folder/file. This way, a user may find it easier to track file movement.



Action	Filename	Username	Date ▼	Details
...	...	...		...
rename	Archive	Bob	08.09.2013	Renamed from Old
move	Presentation.pptx	Alice	07.09.2013	Moved from Final
copy	report.pdf	Charles	06.09.2013	Copied from Reports
upload	plot2.svg	Charles	05.09.2013	Uploaded

Figure 3.1: Activity view user interface mockup

We have decided to make our solution text-based to allow for filtering. We think that a grid could work better than a list. A grid would have one row per operation and columns dedicated to the properties of an operation, e.g., there would be an *Action* (operation name) column and a *Filename* column. The search fields (or a date picker in case of the *Date* column) will let the user filter the grid by each column and concentrate only on, e.g., delete operations or only on the operations concerning the file `Paris.png`. On the other hand, a list would probably only have a textual description of the operation, such as "`Bob renamed Paris.png to Eiffel Tower.png on 01.01.2013`", which is not as flexible for searching as a grid, and would require the use of the browser search function. In addition, we will implement sorting of each column. Sorting could be useful to find only the most recent or the oldest changes. Figure 3.1 shows the view sorted by the date, with the most recent dates first.

We have decided to present the activity information for any given folder and its subfolders. We think this is a good trade-off as opposed to showing the activity for the whole folder structure or limiting the view to a single folder. If the user wants to see the latest activities of the collaborators regardless of the folder, they should be able to see them from the root folder. The deeper in the folder hierarchy the user descends, the more specific the activity view becomes.

3.2 Traceability as Exploring Provenance

We define traceability as the user's ability to establish a provenance relationship between files. We are concerned with the coarse-grained file-level traceability where traces point to the older and newer versions of the files. In our approach, we only focus on immediate provenance, going one step back and forward. Summarising the history of a file or identifying all provenance relationships is outside of the scope of this thesis. We are mostly interested in delete, rename and move provenance relationships. These operations are the focus of our work because, in our view, they are more disruptive to the user's normal workflow than copying or modifying a file. We take a simplified view of provenance whereby we only consider a move or rename of the whole file, and do not try to track the history of files created by combining parts of other files.

To achieve traceability of deleted files, we propose leaving a placeholder, “deleted file”, in the folder where the file was deleted. We think there should also be an option to get a previous version of the file. Our approach to traceability of moved or renamed files is to introduce shadow files and forward and backward traces. From the user’s point of view, a shadow file is a file that used to reside in a certain folder, but has evidently disappeared. Essentially, this means the file was renamed or moved to a different folder. In our review of existing cloud storage providers, we saw such files labelled as deleted, and we think this misrepresents the information and may lead to confusion. Therefore, we explicitly decided to distinguish between deleted and shadow files. The goal of such a naming convention is to make it clear to the user that a given file is not deleted and may in fact still exist somewhere in the folder hierarchy.

Traces are closely related to shadow files. Figure 3.2 illustrates the two concepts. Forward traces are attributes of a shadow file that point to the file with the new name (in case of a rename operation) or at the new location (in case of a move). Backward traces are links from a current file to the source file. The source file may have been moved, renamed or copied. Thus, using our terminology, a backward trace is either a connection between a regular and a shadow file (in case of moving or renaming) or between two regular files (in case of copying).

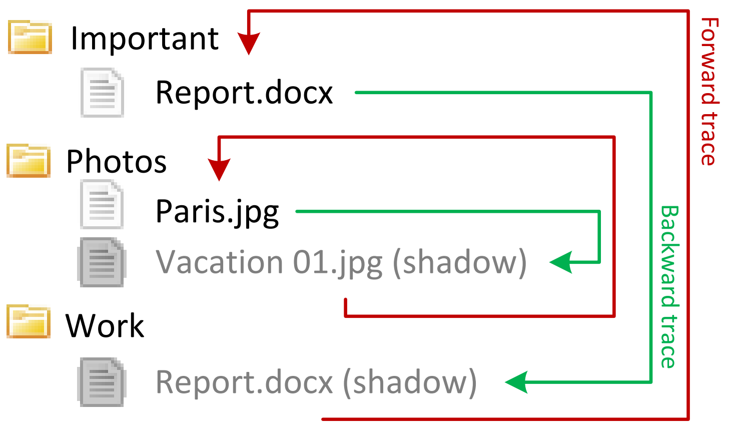


Figure 3.2: Shadow files and traces. Shadow files are grayed out. Forward traces are marked red and backward traces are marked green. The file `Vacation 01.jpg` was renamed to `Paris.jpg`, and a shadow file `Vacation 01.jpg` was created, whose forward trace points to `Paris.jpg`. The file `Report.docx` was moved to the folder `Important`. A backward trace from the new copy of `Report.docx` points back to the shadow file `Report.docx`.

We decided for both a forward trace and a backward trace to provide the user with as complete information as possible. Entering a folder and glancing at the file information should make it clear where new files might have originated from if the files contain backward traces. On the other hand, if the user is going through a list of shadow files and wants to find the files’ next incarnation, having a forward trace next to the file could be handy. Therefore, preserving both kinds of traces as opposed to only backward traces, makes navigation within the workspace more flexible.

3.3 Asynchronous Collaboration

This thesis aims to improve asynchronous collaboration and help users to collaborate more actively. Under asynchronous collaboration features we understand any feature that lets users work together while not requiring them to work at the same time. We propose voting capabilities with an integrated notification system as a contribution to asynchronous collaboration. As mentioned in chapter 2, users tend to limit their actions to the files they own, which leads to a growing number of files and existence of multiple copies of a document in the same folder and, as a result, reduced productivity of the team. In the context of collaboration, we consider move, delete and rename disruptive changes. In our opinion, these changes affect the collaborators' actions the most. It has been shown that users are hesitant to delete other people's files [14], and we also think that moving and renaming files can similarly deter users' actions because these two operations are most like deletion. Moving or renaming a file in some web interfaces (e.g. Dropbox) results in two files, a deleted one and a new one. This suggests that some people may interpret moving and renaming operations as consisting of a deletion and a creation, and will try to avoid performing these operations. We propose voting as a way to help users to carry out the operations they intend without the fear of interfering with the teammates' work. Our vision of the voting system is as follows. When a user makes a disruptive change (move, delete, rename) to a file in a shared folder, the other users will receive a notification and must approve the change. Otherwise, the change is reverted. We believe that introducing such voting process as a safeguard will not only increase the visibility of collaborators' actions, but also help users be more active in carrying out folder maintenance tasks, without having to wait for the original owners to clean up their files. We will refer to the process when the collaborators' votes are collected and evaluated as the "voting process".

Voting Strategies

A voting process is regulated by a voting strategy. We will experiment with the following two strategies:

1. Perform an operation subject to voting.
2. Suggest an operation.

Strategy 1 performs an operation immediately. The system notifies other users, and a voting process ensues. If the other teammates vote against the operation, it is reverted. Strategy 2 is to let the user suggest an operation. The other users are notified, but the operation is not performed until agreed on. With this strategy, no reverting is necessary because the operation may never take place at all.

From our experience with the Idea Garden project, we know that team leaders may accept the roles of guardians and maintainers of shared folders. However, other team members may be less frequent contributors. When such users perform folder maintenance tasks, they may need reassurance that they do not delete important files without the more senior members noticing it. We think of strategy 2 as a less intrusive way to perform an operation. This strategy was designed with ordinary team members in mind, as opposed to project leaders or gatekeepers. Knowing that the change will not actually happen until the participating users agree on it may help the user be more active in suggesting changes. Strategy 1 is meant for team leaders who may want to see the effect of the change immediately. We decided to limit our list of strategies to two. We do not want to impose an explicit workflow; therefore, we do not want to make the users select a strategy. Having only two strategies allows us to easily use one or the other strategy implicitly, based on the users' roles in the shared folder. Alternative voting strategies could give the vote initiator the power to override the voting safeguard and proceed with the vote.

Voting Schemes

In addition to the voting strategies, we devised various voting schemes. A voting scheme is a rule that governs the evaluation of users' votes and decides whether a voting process can be closed and the change accepted. Based on our experience, we conceived the following voting schemes:

1. Majority
2. Majority with a time constraint
3. Percentage
4. Percentage with a time constraint
5. Veto with a time constraint

The majority and the percentage schemes are very similar. We decided for the inclusion of the percentage scheme because we think it can be more manageable in a large team than a majority scheme. Having to wait for half the number of users to vote could take a long time and be unproductive. By contrast, having a few active users vote on a change and cause its acceptance could speed up the process. Veto power could be useful for teams with a designated leader. It is the person who is in charge of maintaining the folder structure; therefore, being able to revert a change with only one vote is a useful privilege for this person.

Conceptually, the schemes revolve around only three basic ideas. The first idea is to ensure that a certain percentage of people agrees on a change. This percentage could be 50%, hence the existence of the "majority" scheme. The second idea is to introduce a time constraint. If no agreement has been reached by the predefined time in the future, the change is reverted. The next idea is the power of veto. If a person with the power of veto rejects the proposal, only this one vote can be enough to stop the change from occurring. In our design, when voting scheme 5 is used, any user has the veto power, and can vote against a change. We have decided to make the veto scheme work only in conjunction with a time constraint. If no user votes against a change by a certain time, the change is performed. Compared to the other schemes we have designed, the veto scheme has conceptually the least overhead. The implicit vote of the initiator who suggested performing the operation can eventually make the change happen if nobody objects. Thus, in this scheme only a single user's vote may be enough.

We name the user who has originally shared the folder the owner of this folder. This user remains the owner for the topmost folder and its subfolders. We will discuss the implementation details of different schemes and what it means to accept a vote in section 4.2.

The following chapter will provide a description of the functionality our application, MUBox, offers, discuss the implementation details of MUBox and specifically our proposed solutions in the areas of awareness, traceability and asynchronous collaboration. Chapter 5 will describe the user study we ran to evaluate the introduced features.

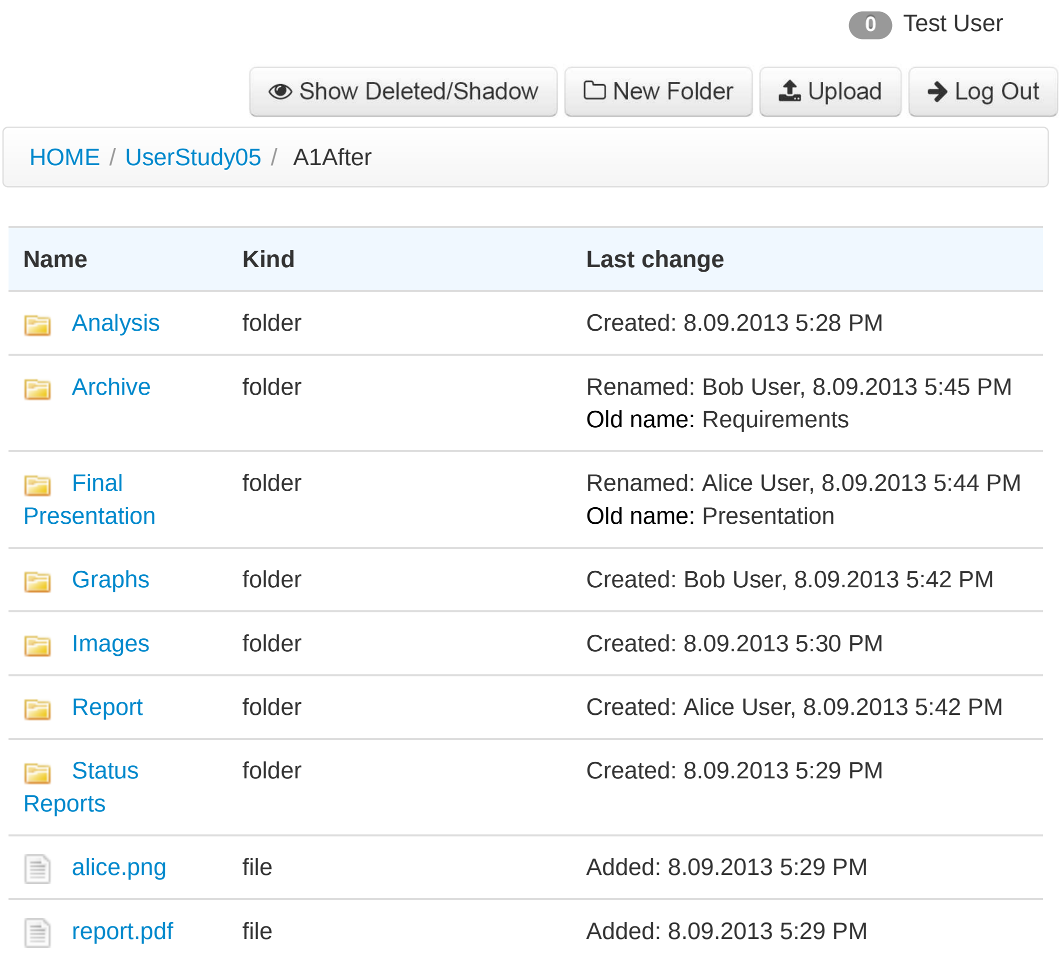
4 MUBox

We designed MUBox (Multi-User Box), a web application through which one can access their Dropbox and Google Drive accounts. The interface can be extended to other cloud storage providers. Currently, the Dropbox implementation is more mature. We stared working on it earlier, and dedicated more attention to it than to the Google Drive implementation.

MUBox is the platform that lets us achieve the goals set forth in this thesis. We implemented awareness, traceability and asynchronous collaboration (voting) features. The features important to experimentation and evaluation can be turned off and on individually.

After a user logs in, they are presented with the folder view, seen in Figure 4.1. The view is inspired by the Dropbox user interface. A grid with three columns displays the folder structure. The first column, *Name*, contains a list of folders and files ordered alphabetically. For each file, the second column specifies the file *Kind*. This could be simply a file or folder, a shadow or deleted file or folder, or a shared folder. The *Last change* column can display different information depending on the application settings. The minimum functionality is to display the username of the user who made the last change to the respective entry (file or folder). If shadow folders and traces are enabled, the column also displays the last operation performed on the file (e.g., “Created”, “Added”), and may also show a backward or a forward trace.

If the user clicks on a file row in the grid, the grid header displays a row of buttons signifying file operations. Alternatively, a user may right-click the file row and display a context menu listing the same operations. Usual operations for a file are: download, show versions, delete, rename, cut, copy. Cutting or copying a file followed by pasting let the user move or copy the file to a different folder. For a folder, the operations are: share folder, delete, rename, cut, copy. Clicking a hyperlink displaying the filename starts a download of the file. Clicking a folder name makes the user navigate into the folder.



Name	Kind	Last change
Analysis	folder	Created: 8.09.2013 5:28 PM
Archive	folder	Renamed: Bob User, 8.09.2013 5:45 PM Old name: Requirements
Final Presentation	folder	Renamed: Alice User, 8.09.2013 5:44 PM Old name: Presentation
Graphs	folder	Created: Bob User, 8.09.2013 5:42 PM
Images	folder	Created: 8.09.2013 5:30 PM
Report	folder	Created: Alice User, 8.09.2013 5:42 PM
Status Reports	folder	Created: 8.09.2013 5:29 PM
alice.png	file	Added: 8.09.2013 5:29 PM
report.pdf	file	Added: 8.09.2013 5:29 PM

Figure 4.1: MUBox folder view.

4.1 MUBox Concepts

4.1.1 Shadow Files and Traces

We decided to visually separate the shadow and deleted files from the regular files. The *Kind* column does indicate whether a given file is a regular file (simply “file”), a shadow or deleted file. However, to make it easy to see the difference at a glance, we also decided to gray out shadow and deleted files. The font and the file icon colour are both shades of gray. The *Show Deleted/Shadow* button, seen in Figure 4.1, lets the user show or hide deleted and shadow files. Currently, if shadow and deleted files are displayed, there is no limit on the age of the shadow files shown. For folders with long interaction histories, choosing to display deleted files may therefore result in some clutter.

A trace is visualised simply as text and a hyperlink in the *Last change* column that tells the user the old or the new location of a file. Forward traces, illustrated in Figure 4.2, are displayed in the same row as the shadow file they correspond to. Backward traces (Figure 4.3) are displayed next to the files that were created as a result of a rename or a move operation. Tracking a deleted file is somewhat different. A textual description in *Last change* column tells the user the file has been deleted. Clicking the hyperlink with the filename navigates to the *Revisions* page where a user can download a previous version of the file. We rely on the cloud provider to keep track of file revisions.

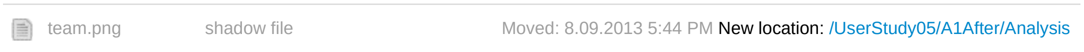

team.png shadow file Moved: 8.09.2013 5:44 PM New location: /UserStudy05/A1After/Analysis

Figure 4.2: A shadow file and a corresponding forward trace.

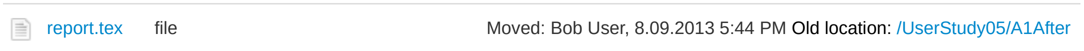

report.tex file Moved: Bob User, 8.09.2013 5:44 PM Old location: /UserStudy05/A1After

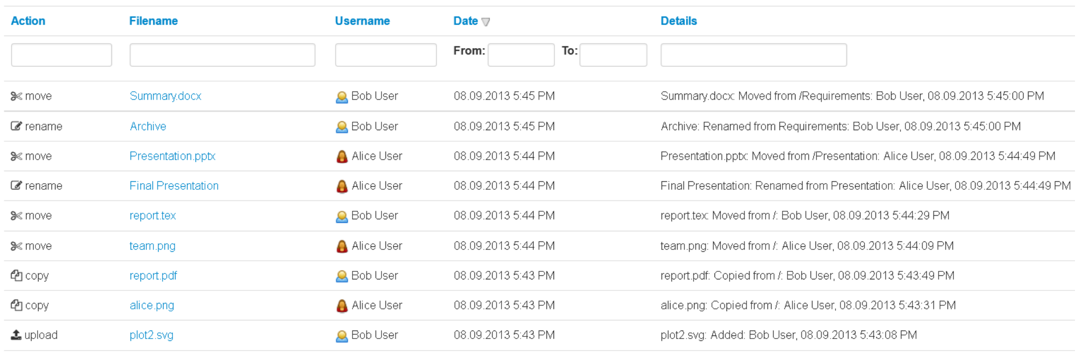
Figure 4.3: A file created as a result of a move operation and a corresponding backward trace.

4.1.2 Activity View

From each folder, a user can navigate to the activity view, a screen that lists file operations and the users who performed them. Figure 4.4 shows a screenshot of the view. It is represented as a grid where a single row corresponds to a file operation. The grid lists the name of the file and provides information about the file operation, the user who made the change, the time the operation was performed, as well as the details — a summary of the change that also lists the old location of the file, if the file was moved, or the old filename, if it was renamed.

The grid has the following columns: *Action*, *Filename*, *Username*, *Date*, *Details*. The operations listed in the *Action* column could be: “newfolder”, “upload”, “move”, “copy”, “rename”, “delete”. In the *Details* column, the verbs referring to the operation names could be the following: “Created” (for creating a folder), “Added” (for uploading a file), “Moved”, “Copied”, “Renamed”, “Deleted”. The naming of the operations was inspired by Dropbox.

To make the activity view useful in practical scenarios, we implemented sorting the columns and filtering the results by each column. When the user hovers the mouse cursor over the filename, a tooltip displays the path to the file at the time the operation was completed. The path is relative to the current folder.

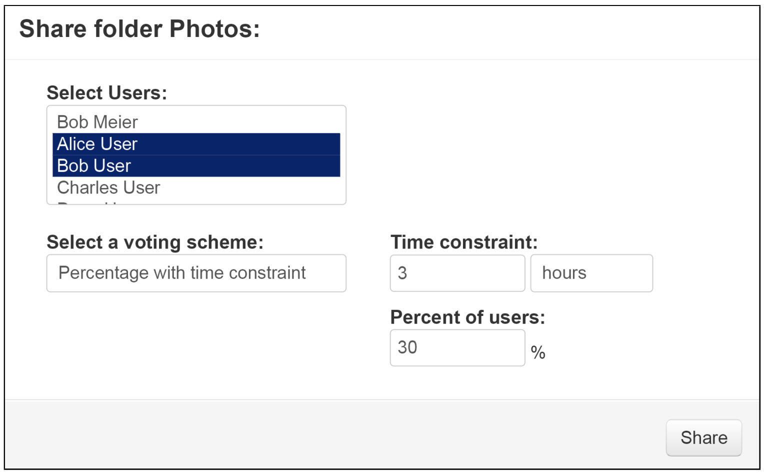


Action	Filename	Username	Date ▽	Details
			From: To:	
move	Summary.docx	Bob User	08.09.2013 5:45 PM	Summary.docx: Moved from /Requirements: Bob User, 08.09.2013 5:45:00 PM
rename	Archive	Bob User	08.09.2013 5:45 PM	Archive: Renamed from Requirements: Bob User, 08.09.2013 5:45:00 PM
move	Presentation.pptx	Alice User	08.09.2013 5:44 PM	Presentation.pptx: Moved from /Presentation: Alice User, 08.09.2013 5:44:49 PM
rename	Final Presentation	Alice User	08.09.2013 5:44 PM	Final Presentation: Renamed from Presentation: Alice User, 08.09.2013 5:44:49 PM
move	report.tex	Bob User	08.09.2013 5:44 PM	report.tex: Moved from /: Bob User, 08.09.2013 5:44:29 PM
move	team.png	Alice User	08.09.2013 5:44 PM	team.png: Moved from /: Alice User, 08.09.2013 5:44:09 PM
copy	report.pdf	Bob User	08.09.2013 5:43 PM	report.pdf: Copied from /: Bob User, 08.09.2013 5:43:49 PM
copy	alice.png	Alice User	08.09.2013 5:43 PM	alice.png: Copied from /: Alice User, 08.09.2013 5:43:31 PM
upload	plot2.svg	Bob User	08.09.2013 5:43 PM	plot2.svg: Added: Bob User, 08.09.2013 5:43:08 PM

Figure 4.4: Activity view.

4.1.3 Sharing Folders

As sharing folders is central to collaborative work, "Share Folder" is one of the context menu choices if a folder is selected. At the time of sharing, a user has to choose the voting scheme that will be used for reconciling the changes during collaboration on the shared folder. Figure 4.5 shows the modal dialog presented to the user at the time of sharing the folder "Photos". The figure shows the user's selection of Alice User and Bob User as team members. The selected voting scheme is "Percentage with time constraint". At least 30% of the users have to agree on the change within 3 hours. The person who initially shared the folder (the owner) can later change the voting scheme and add or remove collaborators from the folder. The other collaborators do not have such privileges.



Share folder Photos:

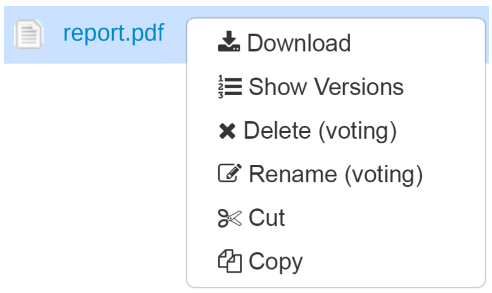
Select Users:
- Bob Meier
- Alice User
- Bob User
- Charles User

Select a voting scheme: Percentage with time constraint

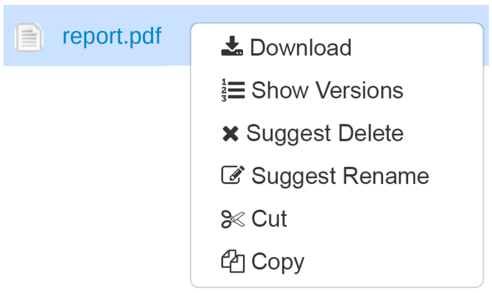
Time constraint: 3 hours

Percent of users: 30 %

Figure 4.5: Voting scheme selection while sharing a folder.



(a) Context menu for folder owners



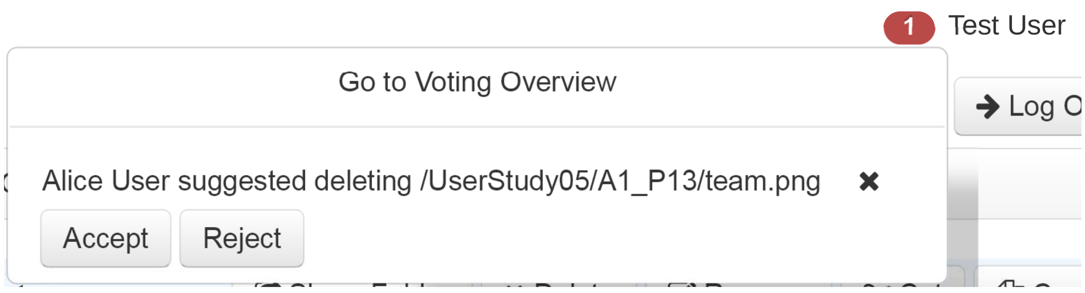
(b) Context menu for the rest of the team

Figure 4.6: Context menus for the folder owner and the rest of the team.

4.1.4 Voting

The selection of the voting strategy to use is based on whether the current user is the owner of the shared folder hierarchy. For owners, the "operation subject to voting" strategy is applied, and for non-owners, the strategy is "suggest operation". So, if a file is to be renamed, one of the context menu options the shared folder owner sees is "Rename (voting)". The rest of the team see the "Suggest rename" menu item instead. Figure 4.6 shows the difference in the context menus the owner and the other team members see. Outside of shared folders, the menu item is simply "Rename". The other file operations that are treated differently depending on the voting strategy are delete and move ("Cut" followed by "Paste").

After a user initiates the voting process, the user's vote is saved as a vote in favour of the change, and notifications are sent to the other collaborators. If they log into MUBox during an open voting process, a red notification badge in the upper right corner of the screen alerts them to the ongoing voting. The user can open the notification dropdown and choose to accept or reject the operation, as seen in Figure 4.7. Depending on the voting scheme (e.g. majority), the voting can be considered closed even if not all collaborators have voted. After the change is accepted or rejected, a notification is sent to the vote initiator. The web interface also has a voting overview screen that lists all voting processes, in which the current user participates, and which have not been closed yet.



Alice User suggested deleting /UserStudy05/A1_P13/team.png

Figure 4.7: Notification badge and a voting alert dropdown.

4.2 Implementation

This chapter describes implementation details of MUBox, as well as the details of the mechanisms aimed at increasing workspace awareness, file traceability and encouraging asynchronous collaboration, presented in Chapter 3. We also discuss the cloud abstraction, the data model and the specifics of file metadata management.

MUBox is implemented as a web application, with the client-side JavaScript part developed with the AngularJS framework[1] and the Bootstrap HTML5 and CSS framework.[2] The client typically issues AJAX calls to the server, and receives JSON data that it formats and displays to the user. The server-side code is based on Spark,[3] a Java "micro web framework". The database used is MongoDB.[4] Figure 4.8 illustrates the architecture of our system. In the labels in Figure 4.8, we use the term "metadata" for the folder structure and information about files, such as the filename, whereas "data" stands for the actual files, e.g., in case of downloading a file.

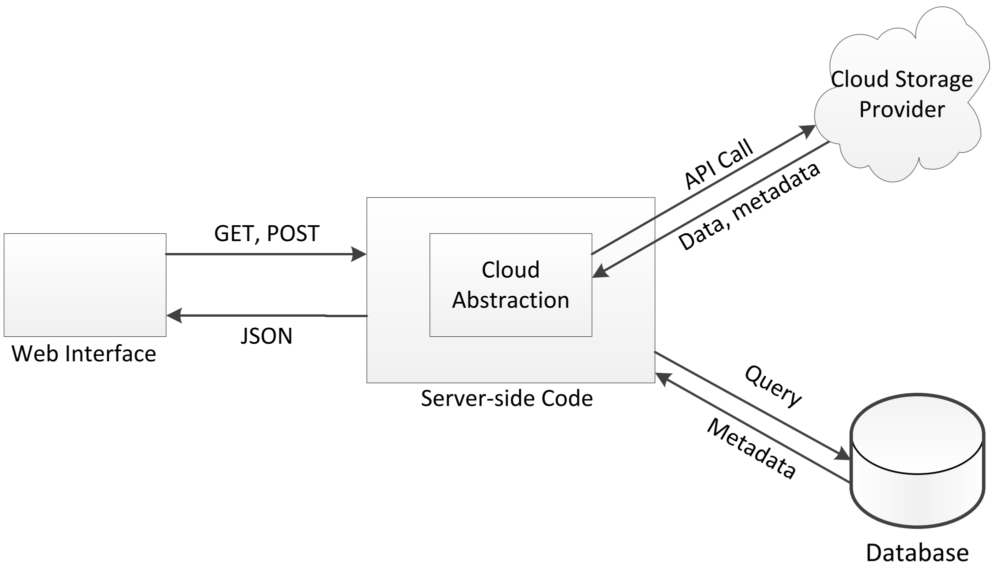


Figure 4.8: MUBox Architecture.

We chose AngularJS with the intent of achieving higher productivity writing client-side code. AngularJS provides built-in support for two-way data binding to user interface elements, which eliminates much of the boilerplate code and, in our opinion, makes AngularJS a good front-end development alternative. A wide selection of JavaScript frameworks is available. However, at the time of writing AngularJS has been voted one of the most adoption-

[1]AngularJS — Superheroic JavaScript MVW Framework.
URL: `http://angularjs.org` Accessed: 22.09.2013.
[2]Bootstrap. URL: `http://getbootstrap.com/2.3.2` Accessed: 26.09.2013.
[3]Spark. URL: `http://www.sparkjava.com` Accessed: 22.09.2013.
[4]MongoDB. URL: `http://www.mongodb.org` Accessed: 22.09.2013.

ready JavaScript MVC frameworks.[5] The approach of AngularJS is to add new attributes directly into the HTML markup. These new attributes are called directives.[6] A directive has JavaScript code associated with it. When the HTML is parsed and the DOM is compiled, the directives are matched, and AngularJS emits the code for proper rendering of the page and links the DOM to the variables defined in the JavaScript code, thereby establishing data binding. Bootstrap provides stylesheets that make it easy to create a layout that works in major browsers using just a few CSS classes. The framework also offers cross-browser user interface components, such as modal dialogs. To be able to work with AngularJS, some of the interface widgets (date picker, popover, dropdown) require the use of AngularJS directives provided by the AngularUI suite.[7]

Spark is one of the frameworks inspired[8] by Sinatra, a domain-specific language for creating web applications in Ruby with minimal effort.[9] More mature "minimal effort" frameworks exist in other languages, and we chose Spark mostly because of our prior experience with Java and because cloud storage providers usually have Java APIs available. Spark is easy to start working with, and for the purposes of the thesis it has all the necessary functionality. Spark lets its users develop RESTful web services using the concept of a route. Routes are mappings of HTTP methods (most frequently, GET and POST in our system) and URL patterns to the code to be executed when a request is issued to a specific URL. Listing 4.1 demonstrates a sample route. Spark can also be used as a general-purpose server-side web framework. It runs on an embedded Jetty web server, can serve static files and provides exception handling. Spark wraps around the Java Servlet API, but lets developers access the underlying API if needed, e.g., to obtain the request URL.

MongoDB is a document-oriented NoSQL database. When discussing the details specific to the database, we will use the MongoDB terminology and call a record stored in the database a document, and a grouping of documents a collection.[10] MongoDB uses a query language based on JavaScript, but drivers and client libraries for other languages exist, including Java. MongoDB allows for fast prototyping because one does not need to define the schema of a collection to start using it. As we were experimenting with the data model, having a flexible schema let us test our assumptions quickly. Moreover, data in the database are stored in the BSON format, a binary representation of JSON documents. This also helped speed up the development. The MongoDB Java driver has classes `BasicDBList` and `BasicDBObject`, which are implementations of a BSON list and BSON object respectively. Since our client side application accepted JSON data, on some occasions we could pass the string representations of these BSON objects to the client and let the client display them. However, BSON defines more data types that are not present in JSON, such as `data_date`. When we needed to convert BSON dates to JSON, we sent the timestamp as the number of milliseconds

[5]Top JavaScript MVC Frameworks.
URL: `http://www.infoq.com/research/top-javascript-mvc-frameworks` Accessed: 22.09.2013.

[6]AngularJS: Directives. URL: `http://docs.angularjs.org/guide/directive` Accessed: 03.10.2013.

[7]AngularUI for AngularJS. URL: `http://angular-ui.github.io/` Accessed: 26.09.2013.

[8]Sinatra. Wikipedia. URL: `http://en.wikipedia.org/wiki/Sinatra_%28software%29#Frameworks_inspired_by_Sinatra`

[9]Sinatra: README. URL: `http://www.sinatrarb.com/intro.html` Accessed: 28.09.2013.

[10]Glossary — MongoDB Manual 2.4.6. URL: `http://docs.mongodb.org/manual/reference/glossary` Accessed: 28.09.2013.

and converted it to a date on the client side. When we needed to implement custom logic, we defined data model Java classes, such as `FileEntry`, a representation of a file metadata document, which extends `BasicDBObject`.

Listing 4.1: Spark route that returns a JSON object with 3 fields in response to a GET request to the URL pattern "`/settings`".

```java
Spark.get(new Route("/settings") {
            @Override
            public Object handle(Request request, Response
                response) {
                NoWarningJSONObject result = new
                    NoWarningJSONObject();
                result.put("disableActivityView", settings.
                    disableActivityView);
                result.put("disableShadow", settings.
                    disableShadow);
                result.put("disableVoting", settings.
                    disableVoting);
                return result;
            }
        });
```

4.2.1 Cloud Abstraction

Since we aim for a provider-independent solution, we developed the `CloudStorage` interface, a cloud storage abstraction for accessing the required functionality of a cloud service. A few noteworthy methods are listed in table 4.1. So far, the interface has two implementations: `Dropbox` and `GoogleDrive`.

Design Decisions and Challenges

One of the methods where the two implementations differ greatly is `shareFolder`. It is only implemented for Google Drive, and not for Dropbox. At the time of writing, Dropbox does not have folder sharing API available to the public. There is no API call that lets a third-party user share a given folder. Furthermore, folder metadata retrieved via the Dropbox `metadata` REST API method does not provide the information whether the folder is shared or who the folder users are. Therefore, we had to use workarounds, such as share folders manually via the Dropbox web interface and then mark those folders as shared in MUBox. However, Google Drive API does support sharing folders, and `GoogleDrive` has the proper implementation of the `shareFolder` method. Another limitation of the Dropbox API is the inability to undelete a folder. To restore a folder, the workaround is to recursively restore the files that originally were in this folder and its subfolders.

Method Name	Description
`upload`	Uploads a file.
`createFolder`	Creates a folder.
`shareFolder`	Shares a folder.
`copyOrMove`	Copies or moves a file/folder depending on the arguments.
`rename`	Renames a file/folder.
`delete`	Deletes a file/folder.
`undelete`	Restores a file/folder.
`moveImpersonated`	Moves a file/folder on behalf of a given user (for voting scenarios).
`renameImpersonated`	Renames a file/folder on behalf of a given user (for voting scenarios).
`deleteImpersonated`	Deletes a file/folder on behalf of a given user (for voting scenarios).
`getDeltaData`	Retrieves a list of changes for a given user.

Table 4.1: Cloud abstraction methods

The first time the `getDeltaData` method is called, it queries the cloud for all the changes in a given user's account since the beginning of use. It notes this point by saving a token (delta cursor in Dropbox, change ID in Google Drive), which will be passed to the subsequent calls to get only the changes that have taken place since the last call. The changes are integrated into the local copy of the cloud storage folder structure.

We access file metadata by a combination of a path and a user UID. Paths are unique by design in Dropbox and are used as identifiers. In Google Drive, paths are not unique, and another field, `id`, is used to identify files. When we work with Google Drive file metadata, we still access them by paths in our system. To be able to make API calls, a Google Drive-specific document in the database stores the path alongside the file ID, which is passed to the API calls. Requiring unique paths for each user creates a few limitations. A file can only be in one folder, and there cannot be more than one file with the same name in a given folder. Since we need as general a solution as possible, and some providers enforce unique paths, we consider such limitations acceptable. Deleting a file means calling the appropriate cloud provider API method and marking the file as deleted in the database. Renaming or moving a file consists of calling the appropriate cloud API method (`update` in Google Drive, `move` in Dropbox), marking the original file metadata as deleted and creating new file metadata with a different path.

4.2.2 Data Model

Table 4.2 lists the most important database collections and their roles. File metadata are stored as documents in MongoDB. The properties of files we will refer to are also properties of documents in MongoDB. We store a local copy of the metadata in the database. Whenever the folder structure is modified via the MUBox web interface, we update the metadata in the database and call the cloud API to make the corresponding update in the storage provider. We do not access the files in the cloud storage unless there is a request to download a file.

Collection Name	Description
`users`	User information, including authorisation tokens.
`filedata`	File metadata for each user.
`sharedfolders`	Shared folder information: the collaborators and the local path.
`voting`	Open voting processes.
`votingusers`	Users' votes for each voting process.

Table 4.2: Most important database collections used by MUBox.

Figure 4.9 shows the relationships between the entities in different collections. A user can have multiple files and any number of shared folders. Shared folders reference paths stored in the `filedata` collection. A user can be an initiator of a voting process, therefore there is a one-to-many relationship between `users` and `voting`. A user also needs to cast a vote in the voting process where they participate, which explains the many-to-many relationship between `users` and `votingusers`. A voting process needs to have at least one vote stored, therefore there is a relationship between `voting` and `votingusers`.

An important consideration when using MongoDB is whether to normalise the data — the decision between embedding and referencing. MongoDB design guidelines suggest using referencing for complex many-to-many relationships.[11] We decided for referencing because, in our opinion, it represents the involved logical entities better, and this way, we found it easier to reason about the relationships between them. For a production application, we could consider denormalising the data, e.g., embedding the `filedata` documents inside the `users` documents, for better read performance.

To model hierarchical folder structures, we decided to use materialised paths.[12] For each `filedata` document, we store the complete path with '/' as separators, as well as the filename. We only need one MongoDB query using a regular expression to select only the files in a given folder or all the descendants of a given folder. During normal interaction with a cloud storage interface, users spend considerable time navigating the folder hierarchy. Therefore, querying file metadata is one of the most frequent database queries. Since materialised paths made querying simple, they are a good alternative. Using this strategy also let us use the path as a whole, without having to construct it. Therefore, we could easily use the path both in cloud API calls and in database queries. It was also easy to store the cloud file metadata as is, with the complete path, when we received metadata updates from the cloud provider. Other possible ways to store hierarchical data could be to store parent or child references in each document. Parsing the paths would be required in these cases. Also, inserting a new

[11]MongoDB Manual 2.4.6. URL: `http://docs.mongodb.org/manual/core/data-modeling/` Accessed: 29.09.2013.

[12]Model Tree Structures with Materialized Paths. URL: `http://docs.mongodb.org/manual/tutorial/model-tree-structures-with-materialized-paths` Accessed: 29.09.2013.

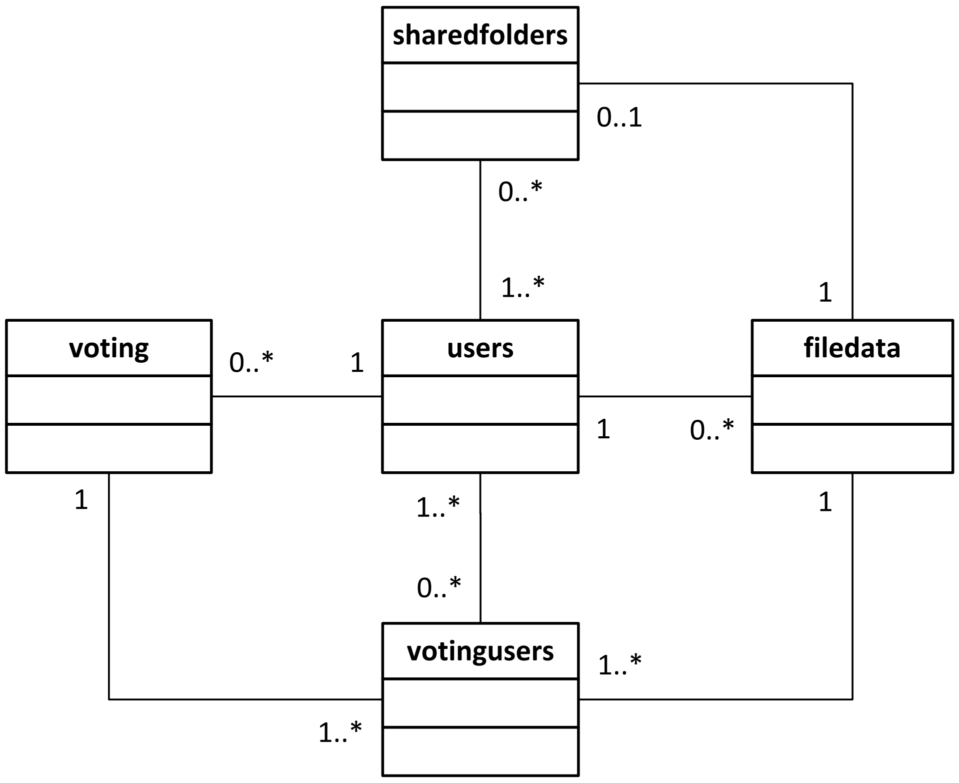


Figure 4.9: Relationships between entities.

`filedata` document into the database would either require querying for its parent's ID (if we were to store parent IDs) or updating the parent with the ID of the newly inserted file (in case of storing child IDs).

File metadata are case-sensitive. However, Dropbox sends its delta data (the updates since the last time) using lowercase paths. To be able to update the file metadata correctly, we needed an ability to query the files by lowercase paths. A case-insensitive regular expression is expensive in this case.[13] Therefore, we decided to store lowercase paths alongside the regular paths for all file metadata. When receiving the delta data, we query the `filedata` collection using the lowercase paths.

If a folder is shared, the whole shared hierarchy is copied as many times as there are users who share the folder. This approach made it easy to experiment with having a different view of the folder structure for each user if necessary. We use this capability in combination with voting. When the folder owner renames a file in their MUBox account, only the metadata in the owner's view change. The file name in the cloud provider and the other users' views in the database remain unchanged until the rename operation is voted on. Despite giving us a way to experiment with folder views, our approach to file metadata management has the following problems. Firstly, it takes more space than simply having the same file metadata

[13]Stack Overflow. URL: `http://stackoverflow.com/questions/1863399/mongodb-is-it-possible-to-make-a-case-insensitive-query` Accessed: 29.09.2013.

with pointers to multiple users. Secondly, any change has to be propagated to the views of all users, which is costly.

Implementing shared folders

The database collection `sharedFolders` contains properties of shared folders. It stores the users who are sharing the folder and the path to the shared folder in the each user's account. The path may be different for different users. Our assumption about the paths is the following. If User A shares a folder with User B, the newly shared folder will appear in the root folder of User B's hierarchy. However, the original owner (User A) may have the shared folder deep in their folder hierarchy. This leads to different paths for different users. In addition, the `sharedFolders` collection stores the voting scheme for the given shared folder.

4.2.3 Experimental Features

Shadow Files and Traces

Shadow and deleted files have similar representations. They are both marked as deleted — the `isDeleted` field is set to true. However, the file's `deletionAction` property is set to the specific action that has taken place. Possible choices are: `delete`, `move`, `rename`, `moveshared`, `renameshared` and `deleteshared`. The value `delete` means that the respective file is indeed deleted, and the rest of the actions mean that it is a shadow file. `moveshared`, `renameshared` and `deleteshared` are used in voting scenarios. Entities representing file metadata also have the following properties: `oldParent`, `oldFileName`, `newParent`, `newFileName`. `oldParent` contains the path to the original folder. It is set at the time a file is moved, renamed or copied. `newParent` is set when a file is renamed or moved. `oldParent` together with `oldFileName` constitute a backward trace, and `newParent` with `newFileName` form a forward trace. Listing 4.2 demonstrates the difference between a deleted file and a shadow file that appeared as a result of a rename operation. It also shows the renamed file with a backward trace. Some fields have been omitted for better readability. The deleted file has its `newParent` and `newFileName` fields set to null (they could also remain not set). By contrast, a shadow file has `deletionAction` set to "rename", and has both `newParent` and `newFileName` set — this is a forward trace. The last file in the Listing, a newly renamed file, has the fields `oldParent` and `oldFileName` set — a backward trace.

Using traces allowed us to implement undo operations as opposed to restoring files. Restoring shadow files could lead to having multiple copies of the same file. For example, if a file `Report.docx` is renamed to `Report (Final).docx`, choosing to restore the shadow file `Report.docx` leaves the user with two files with the same content. By contrast, an undo operation, available for rename and move, deletes the newly created file in addition to restoring the shadow file. This way, the user is left with one current copy of the file, which is probably the user's intention. In case of delete, undo and restore semantics are identical. After the renamed/moved file changes (e.g. a new version of this file is uploaded), the undo operation is not available anymore.

Listing 4.2: BSON representation of a deleted file, a shadow file and a renamed file.

```
// deleted file:
{ "_id" : ObjectId("522c993b5f27ccf43793d104"),
    "uid" : "172128502",
    "path" : "/UserStudy05/S1After/Business/Tech",
    "filename" : "Tech",
    "creationAction" : "newfolder",
    "isDeleted" : true,
    "deletionAction" : "delete",
    "newFileName" : null,
    "newParent" : null,
    "oldParent" : null }

// shadow file with a forward trace:
{ "_id" : ObjectId("522c97bb5f27ccf43793d07d"),
    "uid" : "172128502",
    "path" : "/UserStudy05/A1After/Requirements",
    "filename" : "Requirements",
    "creationAction" : "newfolder",
    "isDeleted" : true,
    "deletionAction" : "rename",
    "newFileName" : "Archive",
    "newParent" : "/UserStudy05/A1After",
    "oldParent" : null }

// renamed file with a backward trace:
{ "_id" : ObjectId("522c9b7d5f27ccf43793d17f"),
    "uid" : "172128502",
    "path" : "/UserStudy05/A1After/Archive",
    "filename" : "Archive",
    "creationAction" : "rename",
    "isDeleted" : false,
    "oldParent" : "/UserStudy05/A1After",
    "oldFileName" : "Requirements",
    "newParent" : null }
```

Our implementation of shadow files suffers from the following drawback. Shadow files and traces do not expire, and there is no way to show only the most recent traces. A folder with a long interaction history will keep all the traces since the beginning of use. Having many traces visible in the same folder may actually make traceability worse. Improvements should include more control over which traces to display and hide.

Activity View

Each file records its creation date and deletion (also moving and renaming) dates, which allows us to retrieve lists of creations and deletions. To display the activities, first we get two lists: 100 recent deletions and 100 recent creations, both sorted descending by the date. We then merge the two lists in such a way as to preserve the sorted order. The result is formatted as changes as opposed to files and presented to the user. Since we merge two lists, the same file could be listed twice, once at its creation time and the second time at its deletion time. Since we represent move and rename operations as deletions followed by creation of new files, we decided only to display the creation of the new (renamed or moved) file to avoid clutter. Our test users seemed to understand our intention and demonstrated the ability to use the interface to learn the folder history.

We need to discuss a few limitations and possible issues with the activity view implementation. The activity view still requires the user to leave the folder view to get a more complete picture of the collaborators' actions. If the user does not realise a change has happened or the activity view needs examining, the user may not even visit the activity view. It is an example of a pull, rather than a push mechanism. An alternative would be to alert the user to the changes right at the time of the login. Our experimentation with notifications is an example of such a feature. However, notifications only provide information about a limited set of operations (delete, rename, move). We did, however, experiment with another awareness feature. If the user enters a folder where other users have added new files, the background of the newly added files flashes green. However, we did not evaluate this feature because we felt it was not substantial enough on its own, and it would be difficult to isolate the effects of this feature alone.

The activity view as it is now may be confusing and not fully complete. For actions such as rename and move, we decided only to report the "creation" part of the action to avoid clutter. If a file is renamed, only one row is added to the activity view, and this row shows the renamed file. A problem occurs if a file is moved outside of the current folder. Then the activity view of the original folder does not list the moved file, and only the activity view of the destination folder does. This results in missing information and may lead to confusion. In such a case, shadow files could clarify the misunderstanding.

Voting

Two database collections are used to support the voting infrastructure: `voting` and `votingusers`. The former is used to manage voting processes and the latter stores users' votes on the respective voting processes. When a user initiates a voting process, two entries are created in the database: one for the voting process, and one for the initiating user's vote. This is a vote in favour. When a voting process is closed, the associated data are removed from the database.

At the time a user's vote is collected, we evaluate whether the voting process can be closed and, if so, whether it is approved or rejected. In case of time-constrained voting, the processes can be closed when the time expires and not at the point when someone submits a vote. To be able to evaluate such votes without a trigger from the user, we schedule a `TimerTask` on the server-side to run every 60s to decide whether there are any time-constrained votes and close them if necessary.

Sometimes in voting scenarios, we need to perform a cloud-based operation on behalf of a user. This happens when the last user to vote before the change is approved is different from the user who initiated the voting process. The other situation is when the timer task needs to close the voting process and execute the approved change. In these cases we retrieve the voting initiator's authorisation tokens from the database, request authorisation from the cloud provider and execute the change. Table 4.3 lists closing and approval logic for the implemented voting schemes.

Voting Scheme	**Closing Condition**	**Approval Condition**
Majority	The sum of votes in favour and against is greater than half the number of participating users.	The difference between the votes in favour and against is at least half the number of users.
Majority with time constraint	Same as above OR a predefined time period has elapsed.	Same as above.
Percentage	The sum of votes in favour and against is greater than a certain percent of the users.	The number of votes in favour is at least a certain percentage of the number of users.
Percentage with time constraint	Same as above OR a predefined time period has elapsed.	Same as above.
Veto with time constraint	There is at least one vote against OR a predefined time period has elapsed.	No votes are against.

Table 4.3: Logic to evaluate votes under different schemes

When there is an active user in the system, the client sends an AJAX request every 20 seconds to check if there are outstanding voting processes where the currently logged on user has not participated yet. The server also checks if there are any completed voting processes that the current user has initiated. The requests to vote and the vote completion alerts are combined in one list and displayed to the user as notifications. 20 seconds were chosen as the polling interval because it is short enough, so the users do not lose their patience waiting for a notification to arrive, and long enough not to overload the system. If we test the system with a large number of concurrent users, we could adjust the time interval to achieve the best trade-off.

As already mentioned, for shared folder owners, we use voting strategy 1, "perform an operation subject to voting". Initially we chose this strategy to contrast it with the "suggest an operation" strategy applied to non-owners. Our intention was to convey the lack of disruption a non-owner causes when they choose to make a change. Making it a suggestion, we hoped to encourage the ordinary team members to participate without restraint and without having to notify the owners explicitly. However, in retrospect, owners' strategy 1 seems more limiting than the other strategy. When an owner makes a change, they see the effect of the change immediately. However, at that point, the change has not taken place in the cloud repository because the voting process is still pending. We set the `deletionAction` property of the affected files to `renameshared`, `moveshared` or `deleteshared` to set them apart from the regular deleted or shadow files. Therefore, for some time, the cloud repository and the local file metadata are out of sync. Our attempt at minimising the damage from such a discrepancy is to disable the operations on the file until the voting process is over. Therefore, waiting for the teammates to vote becomes a drag, and the folder owner, who is supposedly the gatekeeper and enforcer of the shared folder policies, may not feel empowered.

5 Evaluation

5.1 Method

We conducted a user study to evaluate the usefulness of two of the features we developed: the activity view and shadow files/traces. Evaluation of the voting infrastructure is outside of the scope of this thesis because such an experiment would require a long-term deployment and a different design of the user study. Our study used a 2x2 design for the features (activity view, shadow files) and conditions (on, off). This amounted to 4 tasks in total. Each task used a different scenario. In the discussion of the user study, by the "on condition" we mean the task carried out when a feature (the activity view or shadow files) is turned on. If the feature is not available to the user, we call the scenario the "off condition". We rotated and counterbalanced the order of the scenarios and conditions to reduce carryover effects.

Each task required the participant to familiarise themselves with a shared folder structure, which was different for each task. Before beginning the task, the participants received a printout with a diagram of the folder structure. They were also shown an incomplete folder structure in the MUBox interface and asked to complete it so that it matched the structure on paper. In each task, completing the structure meant they had to create one empty folder and upload two files. The files to upload were provided. The folder hierarchy completion was not part of the task per se, but it was our attempt to familiarise the users with the system and each of the shared folders. The folder structures in the 4 tasks were similar, but not identical. We especially strived to make them as similar as possible for the two tasks that pertained to the same feature. Each folder hierarchy had 14 files, but could have different numbers of folders (5 in the activity view tasks and 6 in the shadow folder tasks). The users were told to assume that filenames were unique within each task. Then the actual study started. Assuming that the printout of the folder structure contained the initial hierarchy, the user was demonstrated what this folder structure looked like after two collaborators made changes to some files and folders. The collaborators were always two users with the names *Alice User* and *Bob User*. To save time, we pre-created the folder structure with the changes made. After the participants saw the modified folder structure, they were given a questionnaire, and the timing started. The participants needed to use the available features in MUBox to fill in the questionnaire that tested their understanding of exactly what changes took place in the shared folder. When the users reported completion of the task, the timing stopped.

We had 16 participants (5 female), aged between 23 and 31 (median age: 25.5), all with computer science backgrounds. They were either students, assistants or PhD researchers. All of them had Dropbox experience, and 12 of them were also Google Drive users. The goals of the study were to verify whether the features improved workspace awareness and file traceability. Our approach was to test the time it took to perform the task, the accuracy of the responses and the users' own estimates of their confidence, which they provided in post-task questionnaires. Because of the way the tasks were designed, we could argue that the time it takes to perform the task, as well as the user's confidence and accuracy while performing it, can be signs of whether the user is aware of the activities of the teammates and whether file history is traceable. Although we could obtain such indirect evidence, we cannot claim that the features actually improve awareness or traceability. Our approaches to testing the impact of the features on awareness and traceability were different. We will discuss awareness first.

Evaluating the Effect of the Activity View on Awareness

Our expectation was that the activity view helped users be more aware of the activities of the teammates in the shared folders. We expected that the use of the activity view would let the participants do the task in less time, answer more questions correctly and be more confident in their answers. One of the two awareness task questionnaires is the A1 questionnaire in Appendix A. The questionnaire contained the names of the collaborators and asked to list the file operations each collaborator performed. According to the study design, each collaborator performed the following actions: creating a folder, uploading a file, renaming a folder with one file in it, copying a file and moving another file. The participants' answers were considered correct if they specified 5 correct operations per user, including the name of the operation (e.g., "copy") and the destination if applicable (for copying, moving, renaming). If the user listed other changes, they were ignored. The users received partial points if parts of the answers were correct. Shadow files were disabled for the duration of the two tasks. In the off condition for the activity view, the user had to make educated guesses. For example, if a file with a new name appeared in the shared folder and no other changes were apparent, one could assume that the file was uploaded. If a file with the same name appeared in a different folder, the best guess would be that the file was copied. In the on condition, the participants could navigate to the activity view and see the list of changes. All the participants chose to consult the activity view at some point during the task.

Evaluating the Effect of Shadow Files/Traces on Traceability

The other two tasks concerned the utility of shadow files/traces. This time, our goal was to test whether the feature improves file traceability and lets the user understand where a file comes from. If this is the case, enabling shadow files should reduce the time it takes to do the task, increase the number of correct answers the users give, and boost the users' confidence in their own correctness. The two test collaborators (*Alice* and *Bob*) each made the following changes: creating a folder, uploading a file, copying a file, moving a file, renaming a file and deleting a folder with two files in it. Since we designed these tasks with traceability in mind, the questionnaires used for these tasks were different. Questionnaire S1 in Appendix A is one of the two questionnaires used for these tasks. The questions concerned particular files and asked the user what happened to those files. There were 5 questions per questionnaire. The questions were about the two files that were renamed, two files that were moved, and the deleted folder. The users had to select the operation performed on the file and write the name of the user who performed it. In case of a move or a rename operation, they were also asked for the final name or location of the file. The participants could get partial credit if parts of answers were correct. The activity view was disabled for the duration of the two tasks. In the off condition, it was impossible to say conclusively who deleted a given file. Therefore, we accepted "It was deleted" as a correct answer even if the username was not specified.

5.2 Results

Figure 5.1 illustrates the average values and standard deviations of the time it took the participants to do the tasks across all the tasks and conditions. As seen in the figure, there is a larger difference in the average times for the tasks involving shadow files than the tasks evaluating the activity view. Later in this section, we will describe significance testing of the times in the off and on conditions.

Figure 5.2 shows the median accuracy of the users for all the tasks. The median as opposed to the mean was used because accuracy is a discrete measure. It is the number of points the user got for their answers. The maximum is 16 for the activity view tasks and 9 for the shadow files tasks. We are not comparing the accuracy across all the tasks, but only across the tasks for the same feature.

Figure 5.3 shows the median confidence of the users for all the tasks. In our experiments, confidence was a discrete value, from 1 (least confident) to 7 (most confident). The differences between the on and off conditions in shadow files tasks look more pronounced for both accuracy and confidence than in the activity view tasks. Significance analysis will follow.

For significance testing, we used built-in R[1] functions to evaluate the results. The null hypothesis was that the measurements with the feature enabled and without it are not significantly different. The alternative hypothesis was that the measurements are significantly different. The independent variable was the presence of a particular feature. The dependent variables were the time it takes to do the task, the participant's accuracy, and the participant's own subjective estimate of their confidence. For ordinal quantities, such as accuracy and confidence, we ran the Friedman rank sum tests. If the p-value obtained in the test was less than 0.05 and therefore significant, we ran the Wilcoxon signed rank test to verify the significance of the result. Since the time it takes to do the task is a continuous measure, we ran the repeated

[1]The R Project for Statistical Computing. URL: `http://www.r-project.org` Accessed: 24.09.2013.

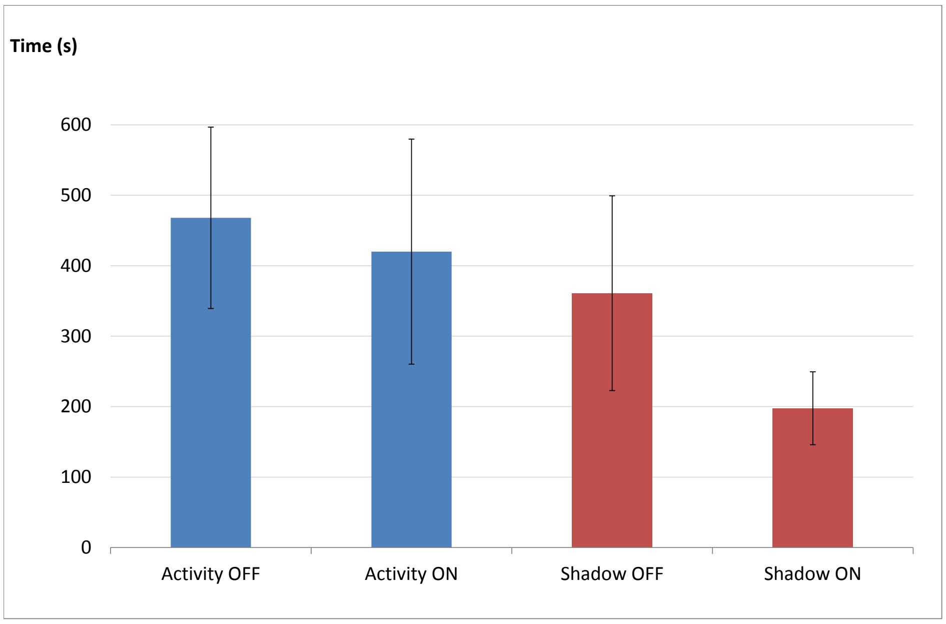


Figure 5.1: Average time it took to do the task across different conditions: activity view on, activity view off, shadow files on, shadow files off. The error bar lengths are one standard deviation in both directions.

measures ANOVA test as pairwise t-tests with the Holm correction. A p-value < 0.05 meant the result was significant.

For the activity view-related tasks, the results are as follows. There is no significant effect on the time ($p > 0.22$), however there are significant effects on confidence ($p < 0.003$) and accuracy ($p < 0.009$). Presence of shadow files and traces does appear to have a significant effect on the time it took to do the tasks ($p < 0.001$). In addition, the effects on confidence ($p < 0.007$) and accuracy ($p < 0.002$) are also significant.

Based on the results, we can conclude that the activity view significantly increases people's accuracy and confidence while performing the task we designed. However, the activity view does not have a significant effect on the time it takes to do the task. Shadow files/traces also increase participants' accuracy and confidence when doing the task. Furthermore, use of shadow files significantly reduces the time it takes to do the task. We think it is safe to assume that if the complexity of the tasks is comparable, and the time to perform the task decreases, this is an evidence of the increase in the user's speed while doing the task. Users generally agreed that the tasks were somewhat realistic (6 median scores on a 1-to-7 Likert scale for both the activity view and shadow files tasks). This leads us to believe that the results of the study could also have an impact on real-life shared folder maintenance needs.

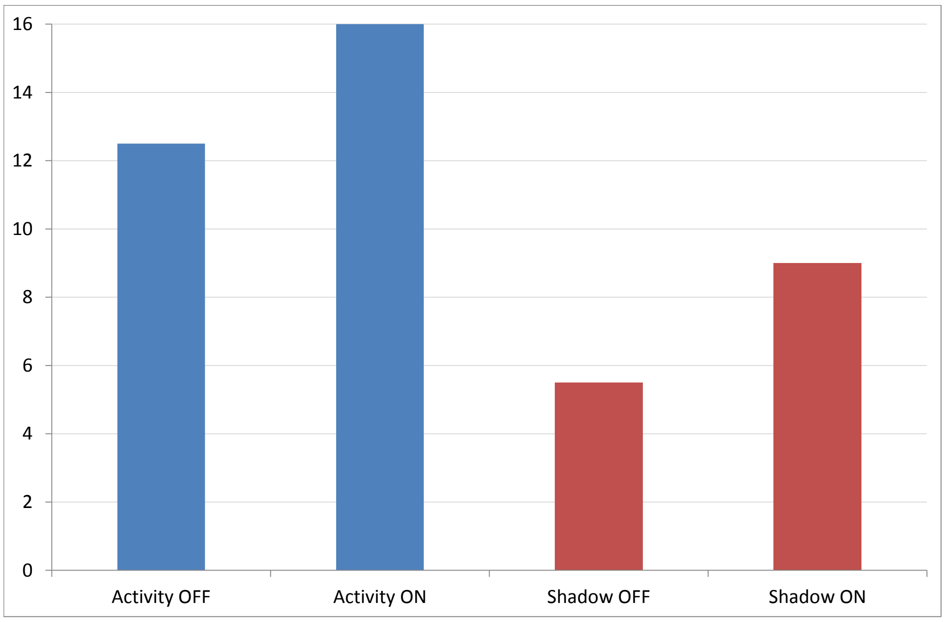


Figure 5.2: Median accuracy (number of correct answers) across different tasks and conditions.

We also asked the users for other subjective estimates in the post-task questionnaires. For example, the users tended to consider the activity view and shadow files effective features to help them do the tasks (6 median score for the activity view, 6.5 median score for shadow files). These effectiveness estimates contrast with the ones in the off conditions (4 median score for the activity view, 2.5 median score for shadow files). Similarly to the aforementioned experiment, the difference in the users' estimates is greater in case of shadow files. This can suggest that the use of shadow files did provide appreciable assistance to the users.

In post-study questionnaires, the users gave us generally positive feedback. Specifically, they found that both the activity view and shadow files were useful features (7 median scores for both features). They also generally agreed that existing cloud storage providers should implement features similar to the activity view and shadow files (6.5 median scores for both features).

5.3 Limitations

In this section, we will discuss some limitations of our study based on MUBox. We must bear in mind that the lack of diversity of the participants' backgrounds may have biased the results of the study. This thesis aims to address the needs of mainstream, untrained users. Therefore, evaluating only participants with a computer science background and previous

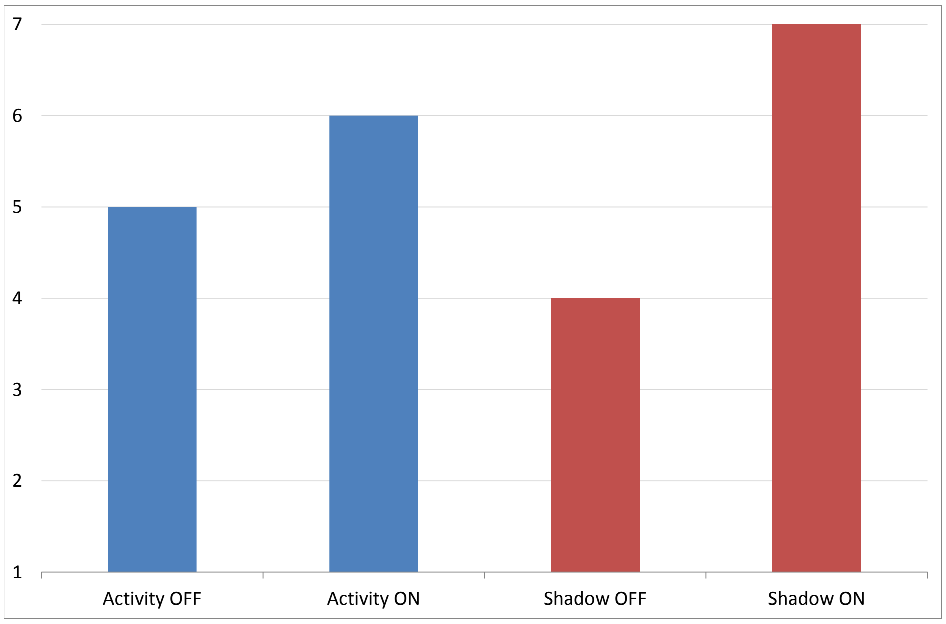


Figure 5.3: Median confidence (user's own perceived confidence) across different tasks and conditions.

Dropbox experience makes the results questionable. We think that a study with more diverse participants is required to achieve reliable results.

Another potential issue was the file structures across the tasks for the same feature. We tried our best to make the tasks as similar as possible, but they were still not identical. The complexity of the file structure had to be exactly the same to achieve unbiased results. However, a few filenames in shadow file scenario 2 (S2) were unfortunately very similar (`Building Web Apps With Lift, Lazy Loading.indd` and `Building Web Applications With Lift.indd`). While this is still a realistic scenario, it confused some users. Some noticed the filename similarity and reported it as a potential source of confusion, the other simply assumed that these two files were copies. Another shortcoming was the operation selection. In shadow file scenario 1 (S1), there was a folder `Tech` with 2 files in it. One file was copied from this folder elsewhere before the folder was deleted. By contrast, in shadow file scenario 2 (S2), there was a folder `Scala` with two files, both of which were copied from this folder elsewhere before the folder was deleted. Despite this discrepancy, we argue that rotating the order of the scenarios and the conditions minimised the effect of differences between the tasks. Even if one task was supposedly easier to do than the other one in the same category (same feature), alternating the on and off conditions as well as the order should have eliminated any handicap the users might have had.

If a file is moved or renamed, opening the file to examine its contents for any changes is one strategy to infer the operation performed on the file. When preparing the study, we decided to make all the files empty to avoid any attempts of opening the files by the participants to compare the contents. Such behaviour would lead to uncontrolled procedures that are hard to compare between subjects. Without the file contents, the participants had to rely on the uniqueness of file names anywhere in the shared folder hierarchy, which may not be the case in a real-life scenario. As mentioned before, we decided that evaluation of the voting features would require a longitudinal study, and did not assess these features due to the lack of time.

6 Conclusion

We identified existing shortcomings of collaborative work features in personal cloud storage providers. One such shortcoming is the lack of overview of the activities of a given collaborator. We classified this problem as insufficient workspace awareness. To tackle the problem, we designed the activity view. It is our solution for improving users' workspace awareness by systematically listing the changes to shared folders and subfolders and allowing the users to sort the changes and filter them by the operation, filename, username or date. Two other problems involved difficulties in tracking the origin of files, and being able to find deleted, moved and renamed files. These problems are related to poor file traceability. Based on our research, we decided to address the problems by capturing and displaying file provenance information. To this end, we designed shadow files and traces. The goal of these features is to make the origin and disappearance of files in shared folders easier to track by providing placeholders in place of deleted and renamed/moved files and links to and from them. Further problems with user experience in existing personal cloud services had to do with users' reluctance to make changes to the files they did not own, and a need of external communication channels (email) to notify the collaborators of the changes. We decided to deal with these problems by introducing voting safeguards for file deletion, renaming and moving. When such operations are performed, the voting system design ensures that the other users are notified, and have to agree on the change before the change is final.

Cloud abstraction is a further outcome of the research. The abstraction is our attempt at providing a vendor-independent solution. We recommend that user interfaces be developed against the abstraction and not be concerned with the implementations of specific personal cloud providers. Such a solution can be ported to a different provider, so that the users can use the same file management, awareness and traceability mechanisms across providers. The cloud abstraction is also a guideline for future developers. The abstraction lists the methods a personal storage service needs to provide to be compatible with our system. We have implemented the cloud abstraction for two personal cloud providers.

One of the goals of the thesis was a lightweight solution that could be built on top of existing personal cloud services. This way, we intended to facilitate its adoption by the existing cloud users. We developed MUBox, a web application and a research platform for implementing the aforementioned experimental features. Using MUBox, users can create and share folder hierarchies, upload files, copy, rename, move and delete files. Access to file revisions is also available, but it is handled by the underlying cloud storage providers. The MUBox web interface resembles that of existing personal cloud providers. We used MUBox to experiment with and finalise the implementations of shadow files/traces, the activity view and the voting system with notifications.

We evaluated two of the proposed features, the activity view and shadow files, using custom designed shared folder exploration scenarios. We can conclude that shadow files have a significant effect on the speed, accuracy and confidence of the user while the user is performing the task. The activity view significantly increases users' accuracy and confidence. The users generally thought the tasks were realistic. Therefore, we may argue that the features could help users explore shared folders not only in a test environment, but also in real-life situations. We designed the tasks with the focus either on awareness — the activities of collaborators — or file traceability — tracking operations performed on specific files. Given that the proposed features made the users more productive performing the tasks, we may contend that introducing the features indeed increases the user's awareness about the activities of others and makes file histories easier to track.

6.1 Possible Extensions or Improvements

Future work on MUBox could involve implementation of new experimental features, better integration of MUBox into the user's environment, continuous usability improvements, better visualisations, security and consistency of the system.

First of all, the implemented features have not been contrasted with other features in the same category (e.g., awareness or traceability) that could be comparable or even more powerful. One possible improvement is to experiment with other awareness and traceability features, evaluate them and choose the ones that work best according to study results. One obvious awareness feature to try is email notifications, indeed a popular awareness mechanism [13]. In order not to clobber their mailboxes, the users would need to have control over the granularity and frequency of notifications, e.g., being able to get notifications after every change, or only a daily digest, as well as being able to unsubscribe from the notifications for a particular file or folder.

Another step towards making MUBox a useful application, better integrating into the user's working environment, and increasing the adoption of our application is to develop operating system clients. One of the features that makes Dropbox an appealing cloud storage alternative is the presence of clients for various operating systems that require minimum configuration and are easy to start working with.

We have to continuously work on the usability of the system. During the user study, we received usability feedback. One source of comments was the activity view usability. The users complained that the activity view presents file change details in an unintuitive order and the text is too long, e.g. “Matlab: Renamed from Scripts: Bob User, 08.09.2013 5:48:31 PM”. Maybe it would make sense to simply report “Matlab: Renamed from Scripts” because the user and the time are already listed in the other columns of the same grid. Some users complained about the phrasing. The expression “renamed from” does not sound natural. Formulating the label this way: “Matlab: Scripts renamed to Matlab” could be more customary for the user. A further usability suggestion was to shorten long path names. A usual technique is to use an ellipsis if the path name is long, and display the complete name on a click or mouseover. The study participants also requested a search function. This would result in a considerable usability improvement. In fact, email has been praised for its flexibility partly because it allows the user to search for messages [13]. One source of confusion during the study was the naming of operations and their presentation in the activity view. Some users were not sure that “added” actually meant “uploaded” for the purposes of our system, and that the word “created” was reserved for folders only, and not files. More consistent naming would not use the label “added” at all and possibly describe the operation explicitly: “created a folder” or “uploaded a file”.

Other improvements include experimenting with visualisation techniques. Schmidt et al. [15] presents graph-based visualisations for interaction histories. Bar chart or graph visualisations of the activity view in MUBox could be a good addition and could provide useful information at a glance. Such information could graphically present the number of contributions per user in the past week, to single out the most active users.

Another area of improvement is security. We do not verify the origin of HTTP requests on the server. A user can forge a request with the correct body and circumvent the client-side checks by sending the request directly to the server. In case of voting, we disable the regular (non-voting) operations only on the client side. A request to the correct URL can still perform a rename or delete without a voting process. Server-side checks need to be implemented.

We need to strive to keep the system in a consistent state. For most file operations, we have to access the cloud storage via an API call, and we have to make changes in the database. Ideally, these two accesses should represent an atomic operation, and be rolled back in case of failure. We do try to minimise damage in some situations. For example, if the cloud operation fails, we do not proceed with the database operation. However, if the cloud operation succeeds and the database operation fails, we do not roll back the cloud operation, and the state of the database becomes inconsistent with the state of the cloud storage. Future work would need to define the meaning of a rollback operation in our system. Cloud storage providers we reviewed do not provide rollback API to developers. In order to roll back a cloud operation, we would need to reverse the effect of the operation, e.g. in case of renaming, rename a file back. We cannot achieve full atomicity simply because the underlying systems (cloud storage and MongoDB) do not provide such guarantees, but we can look for ways to keep the system as consistent as possible.

Bibliography

[1] J. Biehl, M. Czerwinski, G. Smith, and G. Robertson. FASTDash: A visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007.

[2] F. Calefato, D. Gendarmi, and F. Lanubile. Adding social awareness to Jazz for reducing socio-cultural distance between distributed teams, 2009. Eclipse-IT.

[3] L. Cheng, M. Desmond, and M.-A. Storey. Presentations for programmers by programmers. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, 2007.

[4] R. DeLine, M. Czerwinski, and G. Robertson. Easing Program Comprehension by Sharing Navigation Data. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC)*, 2005.

[5] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work (CSCW)*, 1992.

[6] I. Foster, J. Vöckler, M. Wilde, and Y. Zhao. Chimera: AVirtual Data System for Representing, Querying, and Automating Data Derivation. In *Proceedings of 14th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2002. DOI: 10.1109/SSDM.2002.1029704.

[7] J. Froehlich and P. Dourish. Unifying artefacts and activities in a visual tool for distributed software engineering teams. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, 2004.

[8] C. Jensen, H. Lonsdale, E. Wynn, J. Cao, M. Slater, and T. G. Dietterich. The life and times of files and information: a study of desktop provenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010.

[9] M. Kantor and D. Redmiles. Creating infrastructure for ubiquitous awareness. In *Proceedings of the International Conference on Human Computer Interaction (INTERACT)*, 2001.

[10] A. K. Karlson, G. Smith, and B. Lee. Which version is this?: improving the desktop experience within a copy-aware computing ecosystem. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011.

[11] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, 2006.

[12] C. Marshall and J. Tang. That syncing feeling: early user experiences with the cloud. In *Proceedings of the Designing Interactive Systems Conference (DIS)*, 2012.

[13] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood. A review of awareness in distributed collaborative software engineering. *Softw: Pract. Exper.*, 2010. DOI: 10.1002/spe.1005.

[14] E. Rader. Yours, mine and (not) ours: social influences on group information repositories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009.

[15] B. Schmidt, S. Doeweling, and M. Mühlhäuser. Interaction history visualization. In *Proceedings of the 30th ACM international conference on Design of communication (SIGDOC)*, 2012.

[16] S. Sultana and E. Bertino. A file provenance system. In *Proceedings of the third ACM conference on Data and application security and privacy (CODASPY)*, 2013.

[17] A. Voida, J. S. Olson, and G. M. Olson. Turbulence in the clouds: challenges of cloud-based information work. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013.

[18] S. Voida, W. K. Edwards, M. W. Newman, R. E. Griner, and N. Ducheneaut. Share and Share Alike: Exploring the User Interface Affordances of File Sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006.

[19] J. Xu, J. Zhang, T. Harvey, and J. Young. A Survey of Asynchronous Collaboration Tools. *Information Technology Journal*, 2008. DOI: 10.3923/itj.2008.1182.1187.

A Questionnaires

The following are some of the questionnaires we developed for the user study. Figure A.1 shows the questionnaire A1, used in one of the activity view tasks, scenario 1. The questionnaire focuses on two users' activities, *Alice User* and *Bob User*. Scenario 1 was run 8 times in the on condition, and 8 times in the off condition. According to this scenario, the following changes took place:

Alice User:

- Created the folder `Report`
- Uploaded the file `plot1.svg`
- Copied the file `alice.png` to the folder `Analysis`
- Moved the file `team.png` to the folder `Analysis`
- Renamed the folder `Presentation` to `Final Presentation`

Bob User:

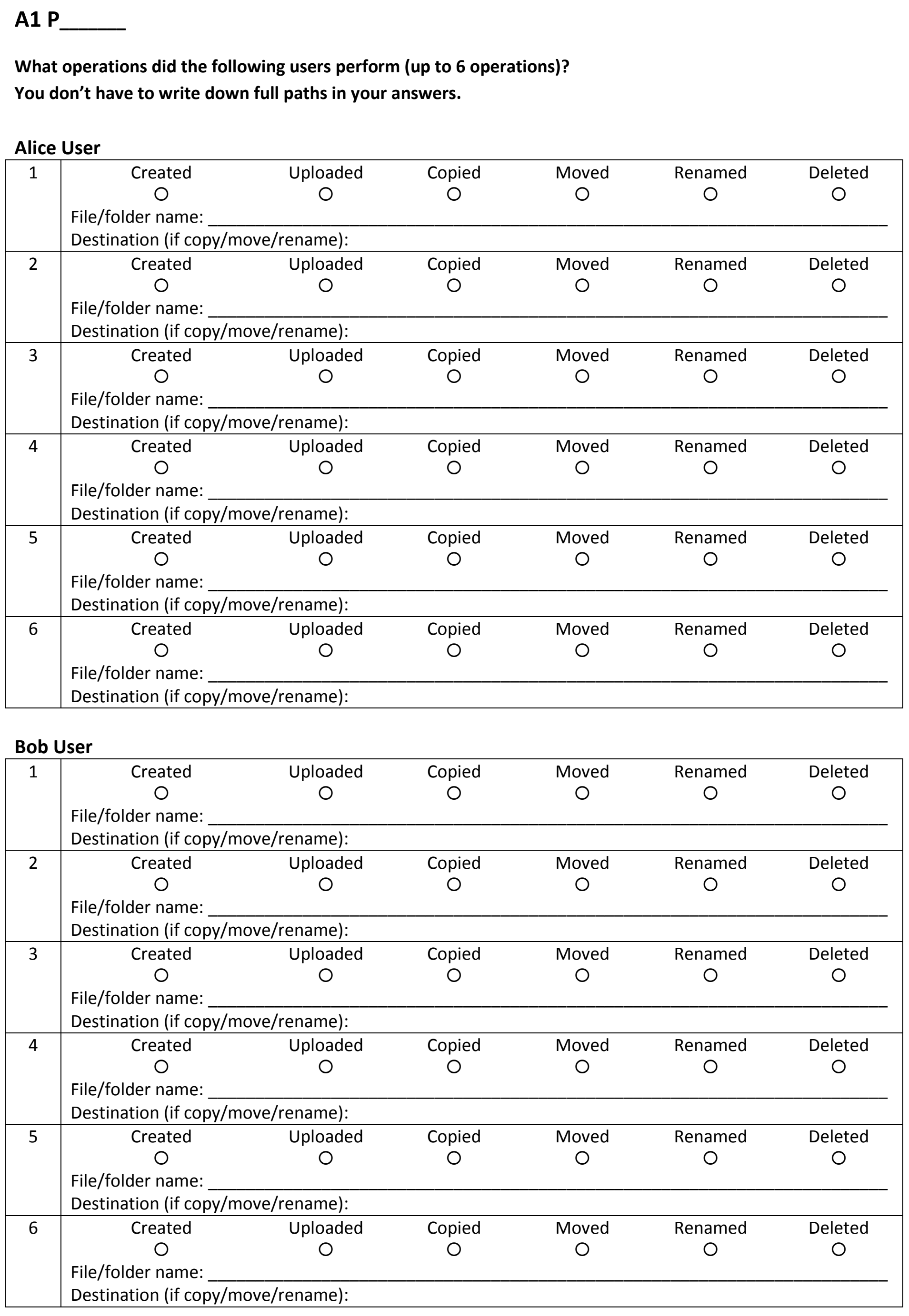
- Created the folder `Graphs`
- Uploaded the file `plot2.svg`
- Copied the file `report.pdf` to the folder `Report`
- Moved the file `report.tex` to the folder `Report`
- Renamed the folder `Requirements` to `Archive`

There is a total of 10 operations. The users could receive a maximum of 16 points. They could receive 2 points for each of the 2 copy, rename and move operations if they correctly specify both the operation and the destination. The other operations could bring the user one point each. A complete answer would correctly indicate both the operation and the file. We accepted "created" as a correct operation with respect to files, although our intention was to use "create" only for folders, and "upload" for files.

Figure A.2 presents the questionnaire S1, used in one of the shadow files tasks. The scenario was run 8 times in the on condition, and 8 times in the off condition. The correct list of actions in this scenario is the following:

- `Smartphone Sales Beat Feature Phones.indd` renamed to `The Death Of The Dumb Phone Is Near.indd` by *Alice User*
- `Curiosity Rover Celebrates 1 Year.indd` renamed to `A Year on Mars.indd` by *Bob User*
- `A Year After Curiosity's Landing.indd` moved to the folder `Science` by *Alice User*
- `Oldest Rock Art in North America Revealed.pdf` moved to the folder `Archive` by *Bob User*
- Folder `Tech` deleted by *Alice User*

The maximum number of points a user could get is 9. The rename and move operations could bring the user 2 points each if the user correctly specified both the operation and the person who performed it. For the delete operation, it only mattered if the user specified that the folder was deleted because in the off condition it was impossible to detect who deleted a given file.



A1 P______

What operations did the following users perform (up to 6 operations)?
You don't have to write down full paths in your answers.

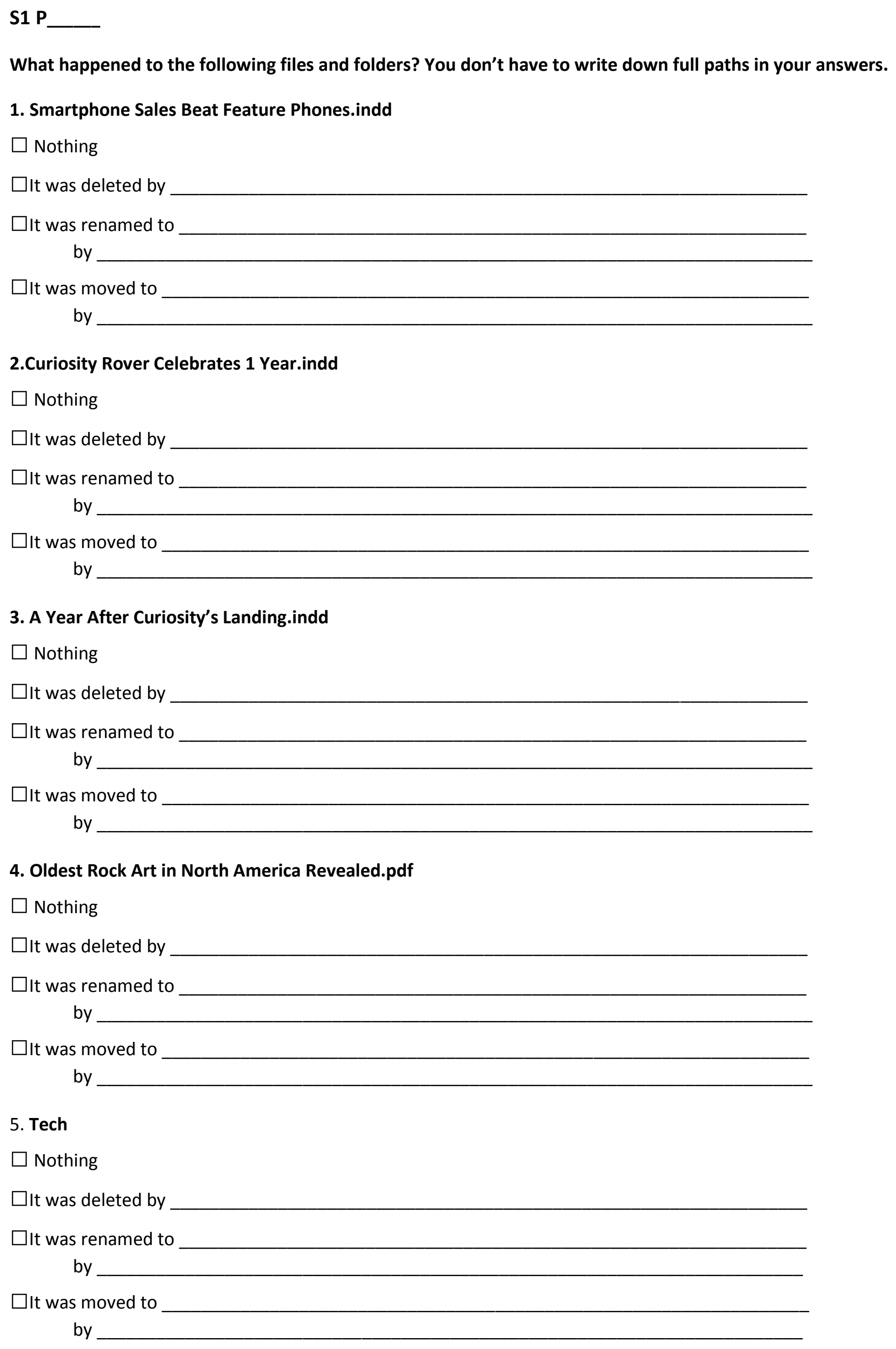
Alice User

1	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
2	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
3	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
4	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
5	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
6	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					

Bob User

1	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
2	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
3	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
4	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
5	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					
6	Created ○	Uploaded ○	Copied ○	Moved ○	Renamed ○	Deleted ○
	File/folder name: ______					
	Destination (if copy/move/rename):					

Figure A.1: Activity view questionnaire 1.



S1 P_____

What happened to the following files and folders? You don't have to write down full paths in your answers.

1. Smartphone Sales Beat Feature Phones.indd

☐ Nothing

☐It was deleted by ____________________

☐It was renamed to ____________________
by ____________________

☐It was moved to ____________________
by ____________________

2.Curiosity Rover Celebrates 1 Year.indd

☐ Nothing

☐It was deleted by ____________________

☐It was renamed to ____________________
by ____________________

☐It was moved to ____________________
by ____________________

3. A Year After Curiosity's Landing.indd

☐ Nothing

☐It was deleted by ____________________

☐It was renamed to ____________________
by ____________________

☐It was moved to ____________________
by ____________________

4. Oldest Rock Art in North America Revealed.pdf

☐ Nothing

☐It was deleted by ____________________

☐It was renamed to ____________________
by ____________________

☐It was moved to ____________________
by ____________________

5. **Tech**

☐ Nothing

☐It was deleted by ____________________

☐It was renamed to ____________________
by ____________________

☐It was moved to ____________________
by ____________________

Figure A.2: Shadow files questionnaire 1.