Oct 31st, 9:00 AM - 10:30 AM

Multithreaded Approach for Using Blockchain as an Automated Versioning Tool

Arber Kadriu
University for Business and Technology - UBT, ak37429@ubt-uni.net

Dijar Kadriu
University for Business and Technology - UBT, dk37669@ubt-uni.net

Donjet Salihi
University for Business and Technology - UBT, ds36387@ubt-uni.net

Edmond Jajaga
University for Business and Technology, edmond.jajaga@ubt-uni.net

Multithreaded Approach for Using Blockchain as an Automated Versioning Tool

Arber Kadriu[1], Dijar Kadriu[2] ,Donjet Salihi[3] and Edmond Jajaga[4]

[1]ak37429@ubt-uni.net, [2]dk37669@ubt-uni.net, [3]ds36387@ubt-uni.net

[4] edmond.jahjaga@ubt-uni.net

University for Business and Technology, 10000 Prishtine, Kosovo

Abstract. Blockchain is a new and growing technology with a bright future ahead. It can be implemented in many different ways and different industries like banking, cryptocurrencies, health information systems etc. Its powerfulness in security and systematic tracking of different items makes it one of its kind and attractive to work with. This technology is also a suitable environment for multithreaded programming to be adopted in, while using blocks and items to be tracked by single threads. This paper presents an innovative approach of using blockchain as a versioning control tool for personal or commercial usage.

Keywords: Blockchain, Versioning Control, Multithreaded Programming

1 Introduction

Users, in general, are always looking for a method to back-up the files, so they can go to the older versions if they changed something that they shouldn't have. There are ways to track the time or the person who has made the changes but there's no solution to keep a copy of the file and revert it to that actual state. There is a high demand in the industries to keep a log of the user who changed the file, the old version and when did the user change it. We implemented this high demanding requirement by using blockchain and its advantages. Blockchain is a new and very promising technology and its usage in technology is growing rapidly.

A blockchain is essentially a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties [1]. It is believed that this is the solution because, as we know, blockchain can't be tempered with and since it will add the block automatically to the chain after closing the changed file without the need of the user to manually do it.

A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread [2].

Basically, it allows the process to have multiple paths of execution, so each thread can focus on a given task. This method allows us to have an improved application performance, since each thread has its task and it broadens the usability of the given application. In traditional settings i.e.1 thread for 1 process, the UI and the program has the tendency to be frozen because the program would be stuck on infinite loop because of the Watch method.
The paper is organized as follows. Section 2 describes the application's conceptual design. Implementation details are presented in Section 3 through pieces of code extracted from the source code. In Section 4 takes place the validation process conducted in a personal computer. Section 5 shows our future plans. Related works take place on Section 6 and finally the paper closes with the conclusion notes.

2 Conceptual design

The approach in building our versioning tool was pretty simple. We tried to make life easier to the user by minimizing, as much as we can, the interaction between the user and the application. The user only specifies the folder to watch and the folder where to keep the backup files. This procedure is shown in the flowchart depicted in Figure 1.

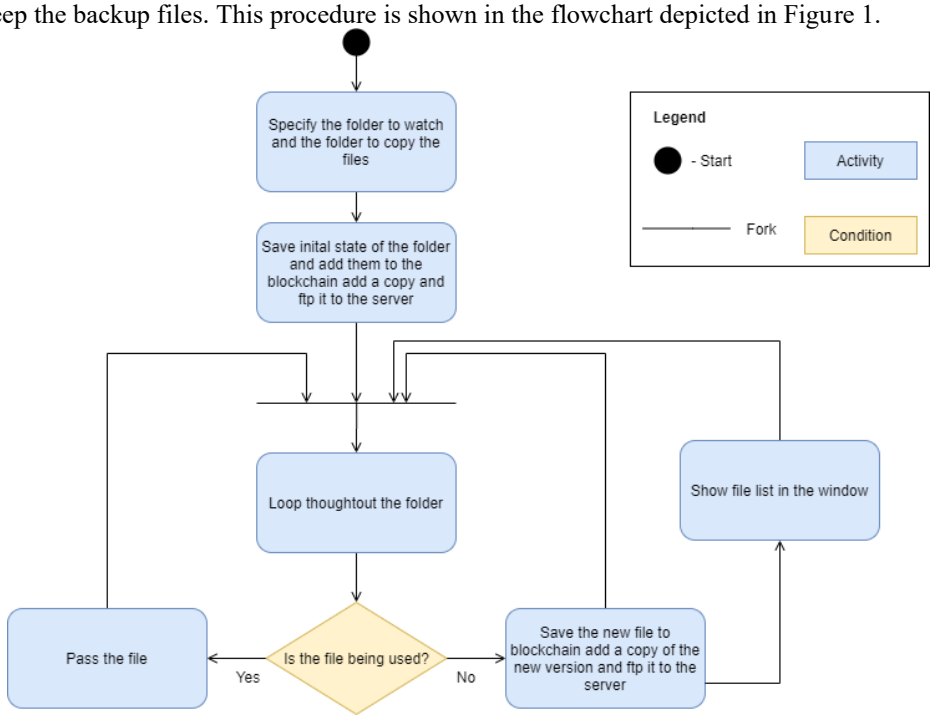

Fig. 1. Activity Diagram

The architecture comprises from 2 modules: the UI and the core module. The UI contains what the user sees, while the core module contains all the models of the application and their respective methods and helpers, as can be observed on Figure 2.

Both components are well designed, in order to reach highest form of cohesion and low coupling.
Global variables are placed in the helper's namespace, which is used to specify file's location. Functions are methods without a base class which are used to help deciding a file's name, state, time accessed and copying the file to the specified folder.
The blockchain sub-component has the implementation of the whole chaining algorithm and structure. The application also contains an FTP protocol configuration, so it could be a perfectly connected to an FTP Server which holds the other part of the chain structure of a specific file. Using FTP Server is considered as a useful and effective practice on preserving physical resources needed for the chain.
The only two models include the FileModel and Block model. Since they have different usage and completely different relation to the software, they are separated into two different folders. This helps on increasing the cohesion level.
The software's another main feature are threads. In this application, threads are used as core part of the software. First and foremost, the UI module is run by a thread itself. Secondly, another thread keeps observing the folder specified for any possible changes in the files. Finally, as soon as there is a change in the file, \another thread is used to apply those changes, while still keeping observed the actual folder. This implementation is considered as the solution for large folders and multiple users applying changes to the files. Using threads in this case prevents a bottleneck situation, Namely, one thread creates other threads, thus breaking the dense tasks into smaller ones and performing them in parallel fashion.

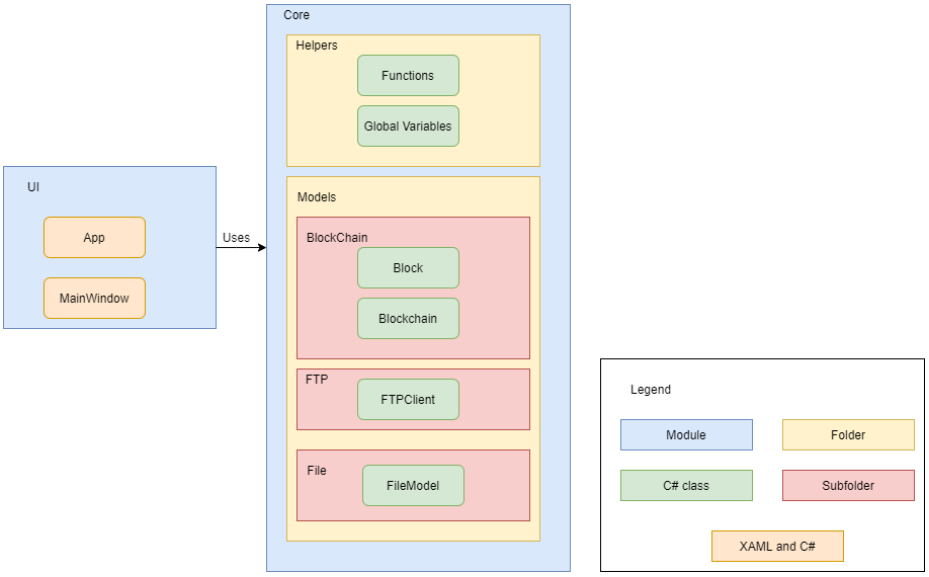

Fig. 2. The architecture of the application

3 Implementation

Watch is the main method of the application because is always checking if the files in the folder are modified. After every execution, the method waits for 100 milliseconds. If the file was modified and it is not being used anymore, the new state of the file is saved in the blockchain. Moreover, it is copied into the specified folder and then it is sent to a different server using FTP as a backup method on the backed-up file.
As shown in the following code fragment, a dedicated thread has the purpose to check if the file is in use and copy it to the main *CopyFiles* folder:

```
    public async Task Watch(Blockchain chain){
1. string path = "";
2. FileModel fm;
3. while (true){
4. fm = new FileModel();
5. string date = DateTime.Now.ToString().Replace(':', '-').Trim();
6. List<FileModel> newFiles = fm.PopulateFilesList();
7. newFiles.RemoveAll(f => f.FileName.StartsWith("~$"));
8. for (int i = 0; i < newFiles.Count; i++){
9. if (!fm.IsFileinUse(new FileInfo(newFiles[i].FullPath))){
10. if (!chain.Chain.Any(f => f.FileName == newFiles[i].FileName && f.FileExtension
      == newFiles[i].FileExtension)){
11. path = CopyFiles(newFiles[i].FileName + date, newFiles[i].FileExtension,
      newFiles[i].FullPath);
12. Block block = new Block(DateTime.Now, ""){
                FileExtension = newFiles[i].FileExtension,
                FileName = newFiles[i].FileName,
                FullPath = path,
                LastEdited = newFiles[i].LastEdited,
                LastEditedBy = newFiles[i].LastEditedBy,
                LastEditedForCheck = newFiles[i].LastEdited,
                FileNameForList = newFiles[i].FileName + date};
13. System.Windows.Application.Current.Dispatcher.Invoke((System.Action)delegat
      e {
14. chain.AddBlock(block);});}
15. else {
16. var block = chain.Chain.SingleOrDefault(f => f.FileName == newFiles[i].FileName
      && f.FileExtension == newFiles[i].FileExtension);
17. if (block.LastEditedForCheck != newFiles[i].LastEdited){
18. string lastFilePath = ReturnPathOfLastFile(newFiles[i].FileName,
      newFiles[i].FileExtension);
19. if (!FileEquals(lastFilePath, newFiles[i].FullPath)){
20. block.LastEditedForCheck = newFiles[i].LastEdited;
```

```
21. path = CopyFiles(newFiles[i].FileName + date, newFiles[i].FileExtension,
      newFiles[i].FullPath);
22. Block blockToAdd = new Block(DateTime.Now, ""){
                FileExtension = newFiles[i].FileExtension,
                FileName = newFiles[i].FileName + date,
                FullPath = path,
                LastEdited = newFiles[i].LastEdited,
                LastEditedBy = newFiles[i].LastEditedBy,
                LastEditedForCheck = newFiles[i].LastEdited,
                FileNameForList = newFiles[i].FileName + date};
23. System.Windows.Application.Current.Dispatcher.Invoke((System.Action)delegate
      {
24. chain.AddBlock(blockToAdd);
      });}}}}}}
25. await Task.Delay(100);}
```

Fig. 3. Watch method.

Namely, in line 6 the thread fetches all the files within the folder and removes the temp files of the opened files in line 7. The temp files are removed from the list since we are not interested in copying the operating system temporary files. In line 9 the thread checks each file if it is not in use i.e. opened. Afterwards, in line 10 it checks if the file exists in the folder. If the file has the same name and extension it means that there is the same file in the folder and skips to line 15. If the file which the thread is trying to copy it to the destination folder, create the node and then with the help of dispatcher in line 13 will add it to the blockchain and notify the UI thread. If the file already exists, in line 17 it will be checked if the file is edited by using the date and time of the last edit, then if the file is edited the last copy of that file is taken and saved and in line 19 check if they are the same or something changed. In line 21 the file is copied to the destination then create the node and in line 23 it is added and the UI thread is notified for the changes.One of the most important methods in the application *CopyFiles* is described in the following code fragment:

```
    public string CopyFiles(string name, string extension, string fullPath){
1.string path = CalculatePath(name, extension);
2.File.Copy(fullPath, path);
3.TransferFilesViaFTP();
4.return path;}

    private void TransferFilesViaFTP(){
5.FtpClient fTPClient = new FtpClient();
6.fTPClient.upload(fileName, path);
    }
    private string CalculatePath(string name, string extension){
7.return GlobalVariables.CopiedFilePath + name + extension;}
```

```
    private string ReturnPathOfLastFile(string fileName, string extensions){
8.  List<string> filesPaths =
    Directory.GetFiles(GlobalVariables.CopiedFilePath).ToList();
9.  string file = filesPaths.Where(c => c.Contains(fileName) &&
    c.Contains(extensions) && !c.Contains("~$")).OrderByDescending(c => c).First();
10. return file;}

    private bool FileEquals(string lastFilePath, string newFilePath){
11. byte[] lastFile = File.ReadAllBytes(lastFilePath);
12. byte[] newFile = File.ReadAllBytes(newFilePath);
13. if (lastFile.Length == newFile.Length){
14. for (int i = 0; i < lastFile.Length; i++){
15. if (lastFile[i] != newFile[i]){
16. return false;}}
17. return true;}
18. return false;}

    public bool IsFileinUse(FileInfo file){
19. FileStream stream = null;
20. try {
21. stream = file.Open(FileMode.Open, FileAccess.ReadWrite, FileShare.None);}
22. catch (IOException){
23. return true;}
24. finally {
25. if (stream != null)
26. stream.Close();}
27. return false;}
```

Fig. 4. Important methods

As indicated in line 1, it calls another method named *CalculatePaths*, which returns the string of the destination path (line 7). Once the path is calculated it copies the files, then it transfers them via FTP and returns the path of the copied file.

The `FTP method TransferFilesViaFTP is called when the system copies the files, which in turn are considered as new blocks in the chain. In order to make the system perform faster and save clients physical resources, FTP Server is used to to store the backup files. In line 5 the FTPClient gets created, then in line 6 the files get transfered.

In line 8, are placed methods which are called by watch methods to get the last saved copy of the file that is currently checking. In line 8 all the files paths are taken from the directory that are being watched. Then, they are filtered by name, the temp files get removed and finally order them descending by the last time they were edited. The method returns the last edited path.

Line 11 and 12 captures the functionality of FileEquals method. This method reads all the bytes of the files that need to be checked. In line 13 they are firstly checked if they have the same length of bytes. If method returns false then the files are not equal. After

this check completes (lines 14 and 15) an iteration through the array of bytes gets performed and compare them. If one iteration is not equal then the files are not equal. The FileInUse method is also very important, because we don't want to save the copy of the file before the user has finished editing. In line 21 the file to be checked is tried to open and in line 22 eventually an IOException gets caught. If an exception arises then it means that the file is being in use. Otherwise, the user has done editing and the stream gets closed.

4 Validation

As a proof of work, we have implemented the program in our own computers and used it. The results were as expected.

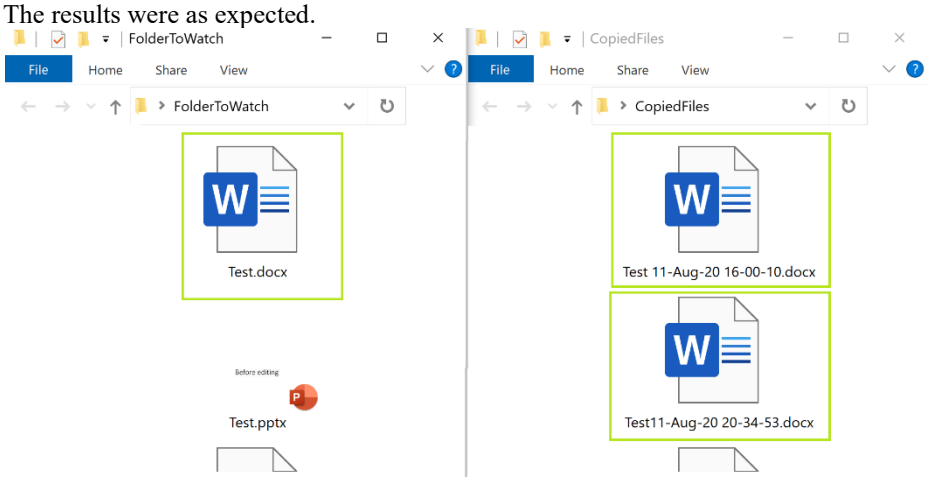

Fig. 5. Watch and Copied folders with their files

As can be observed from Figure 5, the original files resides on the left side of the figure while the copied files on the right side. The copied files have the name of the file and also the date and the time of its last edit (highlighted with green). Every file you open has the content that the file had in their respective period. The difference is seen in Figure 6, where in the left side resides the original file content and the modified content sits on the right side of the figure.

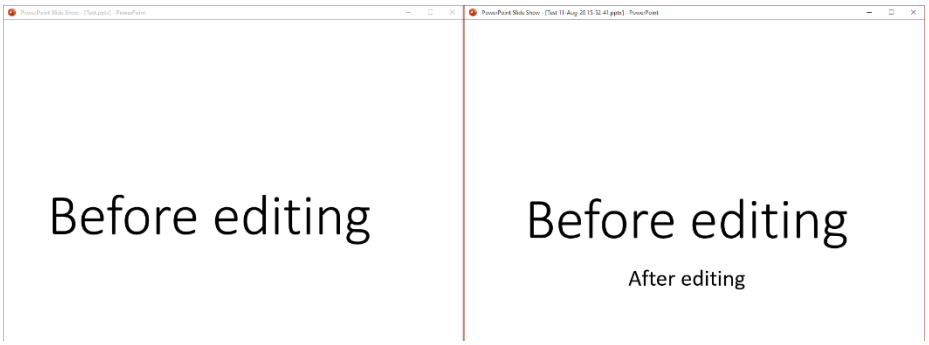

Fig. 6. Content of the same file but in different versions

Figure 7 depicts the GUI of the applications. One can observe the name of the file, user who edited the file and the location where of the edited file.

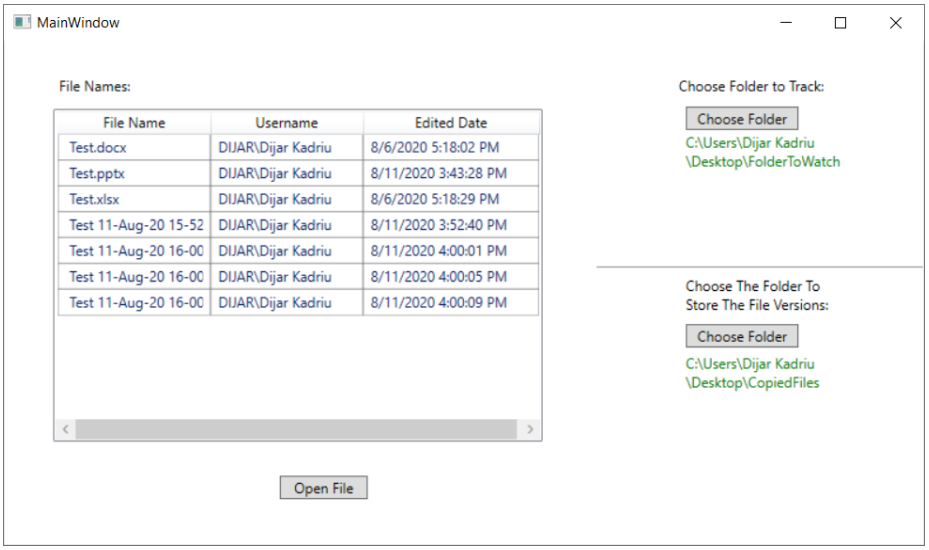

Fig. 7. GUI of the application.

5 Related Works

It is evident that blockchain has been implemented in cryptocurrency, gaming, digital identity, healthcare systems, etc. but to the best of our knowledge there is not any implementation as a version control tool for files or a so called a solution for automated management of document changes.Regarding the *version control* domain several types of applications are been developed using different methodologies and techniques. Such applications include Git, Mercurial and Bazaar.

Git is a distributed versioning-control system, mainly designed to track changes in source code during software development. It uses the tree and nodes technique, where a tree can have multiple blobs or subtrees eventually other files are connected to it.
SHA1 is used as an index to keep track between the changes of the file. System uses this index to recognizes that the file has been changed, added or removed [2]. Git also checks if the *modified* property is enabled, so that it helps for faster recognition of the modified files. Unlike Git we use hashing to define the connection between chains and maintain a persistent chaining continuity between files but we do not hash the entire file. . Hashing process helps us maintain the integrity of the files stored as a backup. In addition, our prototype uses *modified* state to define if the file has been changed, which is a similar approach taken by Git.
Mercurial in the other hand, is a distributed versioning-control system, which rivals Git in terms of application, but with differences in its internal parts and configurations. In Mercurial, the approach to determine the changes of a file and also keeping track of them, is performed using metadata objects called *filelogs*. Each *filelog* maps to an entry and contains information to reconstruct one revision of the file that is being tracked. A *filelog* usually contains two kind of information: the revision data and the index which helps Mercurial to determine the revision efficiently. Mercurial doesn't create a specific copy of the file that has been changed, added or removed. It uses the *delta* mechanism, which stores only the changes needed to transform an older revision into the new revision [4]. This kind of approach in storing the changes of a single file differs from our prototype. In our case, as previously mentioned, we store a single copy of the modified file, every time a change occurs in the specified file. Although it is memory inefficient, we have overcome the problem of binary data or patterns mismatch when it comes to retrieving the older files.
Bazaar is another versioning control tool based on trees and snapshots. Every time a commit happens, Bazaar takes a snapshot of the current state and stores it as a revision file, which can reconstruct the files and folders completely [1]. Unlike our approach, snapshots are indexed and saved, but they do not have a relation to one another. Moreover, our methodology on saving changes differs from Bazaar. Namely, as previously mentioned, we save a physical copy of each modified file.

6 Conclusion

Blockchain is an up and coming technology with all different kinds of implementations. It is a revolutionary technique in technology as we know it, and its usage is still being discovered. In this paper we presented a novel approach by applying blockchain for a versioning control tool. It is a promising tool for personal usage to keep versions of the folders, respectively files, but it can also be used as a security policy in companies that used shared folders. This, because we can take evidence on how the file was before editing and after it being edited. Moreover, it maintains record for who and when the changes were made. The blocks can also be encrypted as part of security company policies. Blockchain is also a suitable environment for threads. The chaining methodology is simple and an effective way to be implemented by a single thread, which makes it a fast, efficient and secured technology.

Taking into the consideration that we try to intertwine multithreaded programming and blockchain as a powerful automated versioning tool, there are many ways that the application can be further improved. Three major upgrades are envisioned, as described in the following:

1. Add more threads to check the files. It would be a challenge to dynamically add and adopt threads to check one or more files for changes. In order to avoid overhead a specific number of threads will be created where one thread would dynamically be optimized for checking more than one file with a fair distribution of files and folders.
2. Another big upgrade for this program would be adding more security to the files after they are copied. In this manner, no one would be able to delete them and they could only be opened by the program.
3. And lastly save only the modified part of the file, not the whole file. And when the user modifies it, then it is displayed a before and after state of the file.

Our tool is open source and available on GitHub[1] .

References

1. Bazaar. (2013). *Documentation for Bazaar - The Adaptive Version Control System.* Retrieved 7 27, 2020, 1from Bazaar: http://doc.bazaar.canonical.com/en/
2. Chacon, S., & Straub, B. (2014). *Pro Git.* APRESS.
3. Crosby, M., Nachiappan, Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). BlockChain Technology: Beyond Bitcoin. *Applied Innovation Review*, 5-20.
4. O'Sullivan, B. (2009). *Mercurial: The Definitive Guide.* O'Reilly Media.
5. Silberschatz, A., Galvin, P. B., & Gange, G. (2013). *OPERATING SYSTEM CONCEPTS.* Wiley.

[1] https://github.com/dijarkadriu/BlockChain