INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Smart Briefcases
Sincronização de Ficheiros Replicados

Tiago Ferreira Nogueira Leite

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Professora Maria dos Remédios
Orientador:	Professor Paulo Ferreira
Co-Orientador:	Professor Luís Veiga
Vogal:	Professor Alberto Silva

Outubro 2010

Acknowledgements

Começo por agradecer ao professor Paulo Ferreira que me orientou ao longo desta dissertação. As suas ideias, sugestões e críticas foram indispensáveis para o desenvolvimento deste trabalho.

Quero também agradecer a todos os meus colegas e amigos do IST e Inesc-ID que me ajudaram de alguma forma no trabalho ou simplesmente me deram companhia e apoio sem o qual teria sido mais difícil concluir este trabalho. Agradeço em especial ao Ivo Anjo que me esclareceu algumas dúvidas e me ajudou em alguns pormenores de implementação. Mesmo tendo eu usado C#.

Um agradecimento especial para os meus pais e para a minha irmã Sofia que sempre me apoiaram e ajudaram ao longo da minha vida estudantil. Sem eles nunca poderia ter chegado onde cheguei nem seria quem sou hoje.

Um obrigado muito grande à minha namorada Telma Oliveira. Pelo seu incansável apoio e pela motivação que me deu. Ela foi o meu pilar de força ao longo de todo o trabalho e puxou por mim nos bons e nos maus momentos. Obrigado por toda a paciência para aturar os meus discursos intermináveis sobre este trabalho. Consegues sempre fazer-me sorrir. Obrigado!

Por fim agradeço a todos os meus amigos que me apoiaram, que me ajudaram a descontrair durante os tempos livres e que não se safaram de me ouvir falar do trabalho. Agradeço especialmente ao Eduardo, à Rita, ao meu primo Pedro e ao João. Se me esqueci de alguém peço desculpa.

Finalmente agradeço a todas as pessoas que experimentaram o Smart Briefcases e contribuíram com as suas sugestões e a toda a gente que de alguma forma contribuiu para este trabalho.

Obrigado a todos!

Lisboa, November 14, 2010
Tiago Ferreira Nogueira Leite

Aos meus pais

Resumo

Nos últimos anos, os dispositivos computacionais tornaram-se de tal forma acessíveis que passou a ser comum um utilizador possuir telefones móveis, PDAs, computadores portáteis e computadores de secretária. O utilizador poderá usar estes dispositivos, tanto para fins de entretenimento como para realizar o seu trabalho em qualquer lugar.

Devido a este facto, é esperado que um utilizador armazene versões distintas dos mesmos ficheiros em vários dos seus dispositivos. Este factor aumenta o desafio de manter as diferentes versões dos ficheiros consistentes e reconciliar dados modificados concorrentemente com sucesso.

Esta dissertação descreve o sistema Smart Briefcases, um sincronizador de ficheiros transparente para as aplicações, baseado em abordagens optimistas. O objectivo do Smart Briefcases é ajudar um utilizador que possui diversos dispositivos computacionais a manter os seus ficheiros replicados consistentes, aplicando mecanismos que permitem a detecção de conflitos e ajudam o utilizador a resolver os mesmos.

Quando o Smart Briefcases detecta conflitos, o sistema deve fornecer todas as informações relevantes para ajudar o utilizador a resolve-los manualmente. Para possibilitar isto, o sistema utiliza as propriedades semânticas dos ficheiros e monitoriza o comportamento do utilizador enquanto este modifica os seus dados.

Abstract

In recent years computational devices have become affordable to the point where is common for a user to own mobile phones, PDAs, Laptops and Desktops. He may use these devices both for entertaining purposes or in order to perform his work anywhere.

Due to this fact it is expected that a user stores different versions of the same files throughout his devices. This rises the challenge of maintaining the different versions of files up to date and reconciling concurrently modified data.

This dissertation describes Smart Briefcases, a file synchronizer transparent to applications, that is based on optimistic approaches. The goal of Smart Briefcases is to help a single user who owns several computational devices maintain all replicated files consistent by applying mechanisms that detect conflicts and help the user resolve said conflicts.

When Smart Briefcases detects conflicts, the system must provide all the relevant information to help the user to manually resolve the conflicts. In order to achieve this, the system uses the semantic properties of files and monitors the user behavior while he is modifying files.

Palavras Chave
Keywords

Palavras Chave

Replicação Optimista

Sincronização de Ficheiros

Resolução de Conflitos

Consistência

Keywords

Optimistic Replication

File Synchronization

Conflict Resolution

Consistency

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, more and more people use several computational devices in their daily life, either for entertaining purposes or in order to perform their work. They own mobile phones, PDAs, Laptops and Desktops so that they can keep working continuously, even while disconnected from a network. Due to this fact, it is expected that in some situations the same files will be copied between these devices.

For example, a user who is currently traveling abroad may take a great number of photos using his mobile phone during the trip. On each day, when he arrives at the hotel he copies the photos to his laptop where he can retouch or edit them at will. When the trip ends and the user gets home he may finally move some of the photos to his desktop in order to create customized albums. Finally, the user can transfer the albums to his media center. This way he is able to watch photo slide-shows from his vacations in his television with his friends when they come over to his house. This means that versions of the same photos, i.e. replicas, exist in four different devices.

Another example of file dissemination is when a user is writing a report and must leave his office for a long period of time. He may be interested in taking on his laptop all the files needed in order to keep working on the ongoing task elsewhere. So, when he finally comes back to the office he can send the finished document to his desktop.

Nevertheless, problems begin to arise when the user, for whatever reason, modifies a file on two different devices, e.g. both on his laptop and on his desktop. This means that different versions of the same file will exist in both devices creating a consistency problem. The user will then have to manually check each file for changes and decide which modifications he wants to keep or he risks losing data by overwriting some of his work.

Obviously, a better solution would be to have a tool which would synchronize the files on different devices, informing the user of what files had been modified, their changes and respective location. Ideally, such a tool would even solve all these issues automatically so that the user wouldn't even be aware that a conflict existed between files in the first place.

1.1 Objectives

Smart Briefcases is a system that focuses on assisting a single user who owns several computational devices and wants to have the same files replicated throughout those devices. The main goal of Smart Briefcases is to help the user in maintaining all the files consistent throughout his devices by applying mechanisms that identify and resolve conflicts and propagate updates between devices. Also, Smart Briefcases informs the user about which files have been changed when the computational devices get synchronized.

In some cases the merging of modified replicas is not simple. Sometimes an automatic solution is impossible to find. If a user modifies the same file in both replicas, it will be impossible to automatically decide (without feedback from the user) which of the files should be kept or if a better solution exists. In this case the only option the system has is to inform the user that a conflict exists and provide all the

Figure 1.1: Nowadays a user has several computational devices.

relevant information so that the user can solve the conflict manually. This information must be relevant and easily understandable, as it is easy to overwhelm the user with too much data, or display information that is not required.

Another goal of Smart Briefcases is that no application in the user's device should be modified in order to use the system. A user should be able to operate the unmodified applications he already uses. Also, Smart Briefcases should support all operations performed to directories inside monitored folders, such as creations, renames and deletions. There should be no difference between accessing a folder monitored by Smart Briefcases and a normal Windows folder. This fact makes the system much easier to use as the user does not have to learn new interfaces or tools.

Smart Briefcases should be supported by the Windows Operating System. Moreover, Smart Briefcases should allow a user to modify any file in any computer anytime. This should not be affected by not being connected to the Internet or not being connected to other replicas. Disconnected operations must be supported. Also, when synchronizing replicas, Smart Briefcases should not require a connection to the Internet or any type of central service. This is particularly important for reasons of cost, availability and security.

Smart Briefcases does not slow down or interrupts the user without need. If conflicts do not occur a user does not even realize that the system exists. Also, when a conflict occurs, the system helps the user in a fast and effective way, by providing him with easily understandable information and not confusing the user so that he can continue his work as soon as possible.

In summary, the goals of this work are the following: i) help a user maintain files replicated and consistent between different devices, ii) allow a user to modify his replicated files in any computer, iii) do not require a connection to the Internet or a central service to perform synchronization, iv) in case of conflicts, provide all the relevant information to help the user to manually resolve the conflicts, v) the system must run without any modifications to the user's applications and iv) does not interrupt the user's work without need and is not invasive.

1.2 Challenges

The creation of a system like Smart Briefcases involves certain challenges; Some of them are common to applications that deal with synchronization and conflict resolution:

1) Smart Briefcases must monitor a user's behavior when he is accessing the files that will need to be synchronized in the future. The system must collect all kinds of relevant data so that, if needed, it can inform the user of what has been changed and how he can solve the conflicts. This must be achieved in a way that does not slow down the system or creates log files with a large size.

2) The information collected from the user behavior comes from various sources. The user may be interested in replicating several different types of files, such as, text documents, spreadsheets, presentations, images, or other unknown file formats. Smart Briefcases must be able to collect data from the applications that deal with these distinct files and be able to abstract the information presented to the user as, for example, lines in a document, or slides in a presentation.

3) The system must be able to detect conflicts if they occur. When this happens the system must find out, using the collected information, if the conflict can be resolved automatically. Otherwise, the user must be presented with all the relevant information that will help him solve the conflicts manually.

4) It is also important that the propagation of modifications between devices is efficient. This means that the amount of data shared needed to synchronize and resolve conflicts must be kept to a minimum.

1.3 Shortcomings of Current Solutions

This paper presents Smart Briefcases, a tool aimed at helping a single user to synchronize files between his multiple computers by offering them assistance when conflicts occur. There are several commercially distributed file synchronizers that already allow a user to synchronize files between different devices. Some popular examples are:

1) Dropbox [10] and Live Mesh [23] are online file synchronizers that use cloud computing to enable users to store and share files and folders between computers using the Internet.

2) Active Sync [25] and its successor Windows Mobile Device Center [1] enable the synchronization of files and other data between a computer and a mobile device i.e. PDAS and smart phones.

3) Microsoft's Briefcase technology [22] and SyncToy [26, 28] are offline file synchronizers.

Dropbox and Live Mesh and most online file synchronizers support only single-master data updates. This means that reconciliation between replicas is done in a single replica to which the user does not have access to. To synchronize files or submit an update a user must be connected to the Internet which is not always possible. Also, complete copies of the user's files and folders are stored in a repository elsewhere which can raise some privacy issues as some users are not comfortable with this.

Microsoft's Briefcase technology is probably the system that most resembles Smart Briefcases, as it is able to synchronize files between two devices with a Windows Operating System installed. The problem with Briefcase is that it does not offer a sophisticated and intelligent file synchronization. When a conflict occurs the user is only presented with a window showing that both versions of the files have been modified. Briefcase does not inform the user of what has been modified in the files and how he should proceed in order to solve the conflicts. In fact, if the user wants to resolve the conflicts, he has to open both versions of a file and compare them manually.

This is also true, in some way, for all the presented technologies, as none of them, employ an intelligent and automatic way of resolving conflicts or provide a user with the information to help him do so himself.

Having in mind all the mentioned shortcomings of current solutions, this thesis focuses on verifying if it is viable to add additional mechanisms, to a file synchronizer, in order to provide a better conflict detection and resolution while providing the relevant information to users to help them resolve the conflicts themselves. This was accomplished by creating a file synchronizer named Smart Briefcases.

[1] http://www.microsoft.com/windowsmobile/en-us/downloads/microsoft/device-center-download.mspx

1.4 Solution

To accomplish the goals presented in section 1.1 the best solution is to implement a middleware that monitors all file operations performed by the user and keeps a log of all the operations performed as metadata. The collected metadata is used to successfully synchronize replicas and detect and resolve conflicts. The middleware monitors the user while he creates, copies and deletes files and folders inside the folders managed by Smart Briefcases. With this system, the user is able to resort to every application he already uses, in order to open and modify the files inside the managed folders.

When a user is modifying one of these files, the middleware knows exactly what file is being changed and what kind of modification is being performed. Be it a deletion, a rename or a modification of the file's contents. Metadata, regarding the performed operations, is stored to represent each of these modifications along with additional information that will be used during future synchronizations and conflict resolutions.

When a user wants to synchronize files between devices, the middleware analyzes the metadata stored and discovers which modifications need to be propagated. If a file has been renamed or deleted in one replica these actions are repeated in the other replica. Files are transfered from one replica to the other if their contents were modified or if they were created since the last synchronization. A conflict occurs when the same file in two different replicas, has been changed or if special scenarios occur. For example, if two folders/files have been renamed or if one file has been renamed in one replica and the corresponding file was deleted in the other replica.

When this happens, there is no method of achieving a consistent state without the user's intervention. The only solution is to inform the user that a conflict exists and provide all the relevant information so that he can solve the conflict manually. This is the same information collected by the middleware while the while the user modified the files.

A system with the characteristics described above accomplishes many of the goals and tackles most of the challenges described in the sections above.

1.5 Roadmap

The rest of the paper is organized as follows: Chapter 2, treats some related technologies giving a succinct review of what has been accomplished in the fields of replication, consistency and data synchronization. Chapter 3 presents the architecture and main algorithms used throughout the implementation of Smart Briefcases while explaining how the system works. The Implementation is then described in chapter 4 while presenting the evaluation of the system's performance in various criteria. This evaluation is both quantitative and qualitative. Finally in chapter 5 it is presented the conclusion of the work developed.

Chapter 2

Related Work

Users who alternately use multiple devices to perform their work will sooner or later face the problem of keeping their files synchronized. To help users with this task some approaches can be employed. This section presents an overview of some approaches, by focusing on the following themes: Pessimistic and Optimistic Replication, Distributed File Systems, Data Sharing Middleware and File synchronizers.

2.1 Basic Concepts and Terminology

This section presents some of the basic concepts and terminology that pertain to Optimistic Replication and are used throughout the paper.

Replication of data is a technique employed in order to increase data availability. In a replicated system data is replicated. A single unit of this data is called an object. An object can be a simple file, a database or fragments of information. When a copy of an object is stored in a computer or location within the system it is called a replica. The locations where replicas are stored are called replica managers and perform operations on them by users' request. These operations can usually be differentiated in read or update operations.

When two different replicas of the same object receive different operations, they become inconsistent and need to be synchronized. Data synchronization, then, is the process of making two sets of data look identical [40].

2.2 Pessimistic Replication

The main goal of Pessimistic Replication [36] techniques is to maintain data shared as a consistent single-copy throughout the system. A replica locks its resources and does not give access to more than one replica manager at a time.

The operations performed in a system that uses this approach are handled synchronously. Due to this, they perform reasonably well in local-area networks where latency is small and failures are uncommon. This does not remain true for wide-area networks such as the Internet. This happens mainly for three reasons:

1) The Internet is a network that is both slow and unreliable. Unlike in local-area networks, failures are more frequent and network partitions constitute a problem. This fact could prevent a user from connecting to a replica manager and fetch the latest version of a replica.

2) Unlike with the optimistic approach, the pessimistic algorithms don't scale well. The system's availability and throughput suffer when the number of replicas in the system increases. This makes it impracticable to create large networks using a pessimistic approach.

3) Some human activities cannot be achieved by using a pessimistic model. For example in cooperative software development it is of the interest of users to edit files concurrently to maximize productivity. If only one user was able to access a file, others would have to wait until that particular user finished working, in order to edit the file. This would be impracticable in a development environment. A preferable solution is to relax the consistency requirements and allow an asynchronous collaboration between users. Conflicts naturally occur and are resolved by the users manually. However, some infrequent conflicts are still preferred to the scenario above. This solution is used for example in CVS [7].

Despite these drawbacks, this approach is still commonly used, mainly in systems in local networks where consistency is an important requisite. One example of one such system is Deno [16]. The subsection below, introduces a strategy used by some pessimistic solutions.

2.2.1 Primary Copy

In primary copy algorithms [39] one replica is elected the leader from a collection of replicas. The leader has the responsibility of managing all the accesses to the replicated objects. When a request for update is received, the leader synchronously propagates the changes throughout all the secondary replicas. The secondary replicas apply the updates in the same order as the primary replica. This keeps the data consistent throughout all replicas as they are all kept in the same state.

However, there are two shortcomings with the primary copy approach. First, as only the primary copy can receive requests, it can easily become a performance bottleneck of the system. Second, if the primary copy fails, the whole system becomes unavailable. This second issue can be resolved by electing one of the secondary replicas to take the place of the failed primary copy. Still, during the time it takes to detect the failure and elect another leader the system will not be able to receive requests.

2.3 Optimistic Replication

Optimistic replication [36] is a group of techniques employed with the goal of sharing data efficiently. They propose that a system relaxes consistency requirements in order to achieve better availability and performance. Unlike the pessimistic approach, in which a replica synchronously coordinates with other replicas in order to spread updates, in the optimistic approach it is assumed that conflicts will rarely occur. Therefore, the replicas let users access data without previous synchronization.

Optimistic algorithms have several practical applications. By applying an optimistic approach it is possible to create large distributed systems along wide-area networks [32]. Likewise, since this approach does not handle requests synchronously, applications do not block even when communication errors happen or the network connection is poor. These factors make this solution perfect for mobile environments. Finally, since several users may have access to the same replicas at the same time, it is the ideal solution for usage in applications that manage cooperative work [7].

In addition to the already discussed advantages, optimistic algorithms offer some more added value. First, they are flexible. Even in wide-area networks where not all the sites are known or where the communication can be unreliable, it is still possible to epidemically spread updates. Second, since the synchronization between replica managers is kept to a minimum, these algorithms scale well with the number of entities added to the system. Finally, they allow users to remain autonomous. A user is able to disconnect from the system. While offline, the user updates the shared objects acquired and when he is able to reconnect to the system, he does so, committing the changes performed. This is especially useful in mobile environments where communication links are unreliable.

However, there is a trade-off to be considered when employing optimistic replication. While this approach improves availability, performance and fault tolerance the consistency of data becomes harder to maintain. Where a pessimistic algorithm waits, an optimistic one speculates [36]. It detects and repairs conflicts as they occur. This is a better solution only for applications that can tolerate occasional conflicts and can

handle inconsistent data. However, this approach is much better than trying to prevent conflicts, as this is exceedingly expensive (in terms of delay or availability) [13].

The process of synchronizing files using optimistic algorithms can be divided in five distinct phases. An overview of each phase is presented in the following sections:

2.3.1 Operation Submission

In the first phase of data synchronization a user submits an update to a site. Different systems handle this phase differently. There are some design choices to contemplate. One of these design choices is the number of possible writers. In other words, the number of replicas which can apply updates. The other design choice is in how an operation is viewed by the system. Each choice is explained now with further detail:

Number of Writers: This choice regards to where updates can be submitted. In Single Master Systems only one replica manager can receive updates. It will then propagate the updates to other replicas. The problem with this scheme is that its availability is limited especially in times of numerous updates. However, for simpler systems this is the best solution.

In Multimaster systems, updates can be submitted to any replica manager. This allows for more flexible and scalable systems. However, since any replica can apply updates, conflicts become more common. Therefore, this approach requires more complex algorithms to detect and resolve the inconsistencies that eventually arise.

Operations: The way operations affect an object can be interpreted differently. In state-transfer systems each object has a state. A modification however small changes the object's state and is seen as an update. When this happens, the whole object is sent to other replica managers and overwrites their stored replicas of the object. This approach has some advantages. It is simple to maintain consistency, since only sending the more recent replica to other sites is needed. However, sending the whole object every time a bit has been changed is not efficient and wastes bandwidth especially when synchronizing large files. Also, this solution is easy to integrate with frequently used applications without having to modify them. The only requirement is that a modification in a file is detected. Microsoft's Briefcase [22] and ActiveSync [25] use this approach.

Operation-Transfer systems on the other hand, keep a history of operations or user actions performed to each replica. This history can be kept in a log or a database. This approach may be (in certain cases) more efficient than state-transfer as only the operation history collected since the last successful synchronization needs to be sent to other replicas. Additionally, this allows for a much more flexible conflict resolution since the semantic information provided by the stored operations allows for a better fine-grain control.

2.3.2 Propagation

In the second phase, updates are propagated to other replicas. There are two options for a system to disseminate its updates:

Pull-based Approach: In this approach a replica polls the other replicas for updates. This action can be initiated manually by a user or automatically, where a replica polls the others from time to time. Rumor[14] and Roam [35], are examples of systems where a replica can poll any other replicas to request updates. The polled replica sends not only its updates but also all the updates previously received.

Push-based Approach: When a replica acquires new updates it epidemically sends the changes to other replicas. This can reduce the propagation delay and eliminates the polling overhead [36]. A simple technique to achieve this approach is to blindly flood other replicas with the updates. However, there must be a way to detect when a duplicate of an already received update arrives.

The timestamp Matrices technique [45] solves this problem by only sending the operations still missing in other replicas. This technique is more efficient, however it can be very complex and does not work well in networks where sites enter and leave regularly.

2.3.3 Scheduling

Scheduling refers to the process in which replicas agree in an order in which the updates will be applied. This order will be maintained throughout all replicas. There are two policies that can be used in this phase: syntactic and semantic.

Syntactic: This policy applies operations based on a predetermined order. This order can be based on where, when and by whom operations were submitted [36]. Timestamps are usually employed in this scheduling process in order to create a temporal order between updates. While syntactic procedures are simpler to implement, unnecessary conflicts may arise. This happens because updates which are semantically ordered by happens-before may still be semantically commuting [4].

Semantic: Semantic policies on the other hand, examine the history of operations and user actions performed previously. Then, they exploit semantics and user intent to create a consistent final state. These actions try to minimize the number of conflicts and try to merge replicas even in cases where simply applying updates in a predetermined order would cause a conflict.

The problem with these systems is that they are complex and a naive implementation can suffer from a combinatorial explosion if it explores all possible orderings. There are however some proposals to solve this problem [17, 43]. Another drawback of this approach is the difficulty of integration with existing applications without further instrumentation.

Some of these systems must collect the operations performed on objects by applications. This cannot be easily achieved without adapting the applications. Finally, one important thing to note is that state-based systems cannot apply a semantic policy. They are not capable of considering individual operations and therefore, cannot extract the required information.

2.3.4 Detecting and Resolving Conflicts

How to detect conflicts?: To determine if an operation can be applied to a replica, each operation must have a precondition. If a precondition is unfulfilled the operation is in conflict. Different systems detect conflicts by using different methods. One possible method is not detecting conflicts at all. The system simply applies operations in the order agreed by the schedule phase. If conflicts occur, nothing is done in order to resolve them or prevent them.

Other systems where a syntactic policy is employed, apply operations in the order in which they happened. A conflict occurs if two operations were applied at the same time or concurrently to one another.

Systems that take a semantic approach, as explained above, use semantic properties and user's actions to detect conflicts. These systems also use preconditions to detect if two operations are in conflict. These preconditions vary from system to system. In Xmiddle[46] for example, conflicts are detected based on updates done in the same line of the file by different users. Icecube [17], on the other hand, allows a user or the application to establish the preconditions.

Resolving conflicts: When conflicts are detected a system must decide how to behave. Some systems [22, 46, 2] decide to do nothing and let the user discover what is wrong and how to solve the conflicts. Other systems [10, 7], apply the conflicting operations and present two different files to the user, letting him decide which one is the more correct. There are also systems, like Rumor [14], that call resolvers when conflicts are detected. Often the conflict can be automatically resolved, but when it cannot the user is notified of the conflict by email along with instructions on how to solve it.

2.3.5 Commitment

When a system successfully performs the commit phase, the previously applied operations cannot be rolled back anymore. From this point on, the objects are in a stable state and if needed a system can safely delete the previous history of operations.

2.4 Distributed File systems

A distributed file system (DFS) [19] has the purpose of allowing users who own several physically distributed computers interconnected by a communication network to share data by using a common file system.

In a DFS, data is spread across several independent storage devices and usually there can be multiple autonomous clients and servers. However, to a client, the DFS should look like a conventional centralized file system at all times. A user should not be able to distinguish between local and remote files. This is also true for applications since they must access shared files in a transparent manner.

With this in mind, consider a user who possesses several devices that share a DFS. He will be able to access the same file system using any of his devices, accessing remote files and modifying them. When he changes from one device to another, he will be able to see all the modifications performed previously.

In this section, some examples of distributed file systems will be presented from the point of view of a user who owns several devices and wants to access the same data in all of them.

2.4.1 Coda

NFS and Andrew File System (AFS) are distributed file systems with a client server architecture. In these file systems data is replicated to the clients in order to maintain performance. Both NFS and AFS use a pessimistic approach. Due to this fact they can only be applied in systems where the server is always accessible to clients. If a connection with the server fails all the replicated files in the client's hard drive become inaccessible. This fact makes it impossible to employ any of these file systems in mobile networks.

The Coda file system [37] is based on AFS and introduced consistent availability of data in the case of intermittent connectivity of devices due to data communication service disruptions. Therefore, Coda is able to handle client's disconnections [18]. These disconnections can happen because of network failures or voluntarily in order to conserve the battery of the client's mobile device.

In Coda, each client has a cache manager called Venus. Venus, guarantees that a user sees the system as a single UNIX file system. Venus is always in one of three phases: hoarding, emulating or reintegrating.

When a client is connected to a server, it is in the hoarding phase. The goal in this phase is to prepare the client for an eventual disconnection. To allow a client to keep working while disconnected, all critical objects must be replicated to the client's local cache. However, the system must first decide which objects are critical to the user's work. In order to better make this decision the system receives information from two different sources. First, Venus keeps a log in which the history of all files accessed is stored. This log works as a least recently used (LRU) cache algorithm. Second the user can explicitly tell the system the pathnames that are more important to him. This information is stored in a database inside Venus.

When a client disconnects from the servers it enters the emulating phase. In this phase the user works accessing the replicated objects stored locally. If a file is not present, there is a cache miss. This is seen by the user or application as a failure that prevents further operations to be applied to the missing objects. All the modifications successfully applied to objects, during this phase, are stored in a log named client modification log (CML).

Finally, when a client reconnects to the server, the reintegrating phase begins. In this phase the client propagates the CML to the set of servers that are currently accessible by a client. This set is called accessible Volume Storage Group (AVSG). The AVSG verify the correction of the operations performed during the

emulation phase and check if there are any conflicts. If all operations are correct the servers apply the updates sent by the user.

In Coda a conflict occurs if two disconnected users have modified the same files. The detection of conflicts is divided in two phases in order to maintain efficiency. In the first phase a purely syntactic approach is employed. The server checks the version of updates through vector clocks [20] to ensure that no updates are concurrent. Only if this mechanism detects the existence of conflicts, is a semantic mechanism invoked. This way performance is maintained without incurring in the overheads introduced when using a purely semantic approach.

Although Coda is a mature and robust system there are still some drawbacks. First the performance of writes compared to a traditional file system is slower. Second, when a client disconnects, although a cache is kept with the important objects, a user can still be unable to perform his work if the file he needs was not successfully hoarded to his mobile device.

2.4.2 Roam

Roam [35, 34] is an optimistic replicated file system that uses a peer-to-peer model. It was developed to be used in mobile devices and was built as an extension of Rumor[14]. Its main focus is on scalability. Unlike Coda, that has a client-server architecture, in Roam there is no need to access a centralized server to submit operations. Any two replicas can directly synchronize with one another. This greatly improves the availability of the system since it is not affected by failures of servers.

Each replica is grouped in a Ward (Wide Area Replication Domain). A Ward is a collection of peers that are close to each other. The proximity between replicas is decided based on geographic location, bandwidth, latency, etc. Inside a Ward, one replica is chosen to be the Ward master and acts in a similar manner to a server in a client-server architecture, albeit with some conceptual differences. The Ward Master belongs to two distinct wards: the local ward and a higher level ward where all ward masters reside. It exchanges information with other ward masters to keep consistency between wards. To accomplish this, each ward master is required to store the names of all objects stored in its corresponding ward.

To detect if new updates exist, each file has a dynamic version vector [35] associated. Each replica is responsible to check if new updates exist within it. The update detection is done exclusively through the analysis of modification times. This prevents the use of logs which allows to keep the memory usage relatively low.

In the reconciliation phase there are two distinct processes: the recon process and the server process. When new updates are detected, the recon process selects a replica to communicate with. The replica selected depends on the adaptive ring topology. The recon process receives meta-information from the remote replica. Then, it checks if the remote replica has data more recent than the one stored in the local replica. Finally, it requests the server to fetch the files from the remote replica and merge the files. Only the local replica is updated since the communication is done only in one direction.

When a certain file has been updated in both replicas, a conflict occurs. If the conflicting file has well known semantic properties and a resolver associated, the conflict can be automatically resolved. Otherwise, the user is notified of the existing conflict by email.

In conclusion, Roam is a relatively robust mobile system that accomplishes its scalability requirements through its simple reconciliation techniques, the use of a peer-to-peer architecture with epidemic propagation. However, an important drawback is that when an application issues a read request, the data delivered is only tentative and some systems require more rigorous correctness criteria.

2.4.3 Haddock-FS

Haddock-FS [5, 3] is a peer-to-peer replicated file system built for Windows CE.Net collaborative applications. It was built to tackle the memory and bandwidth constraints, inherent to mobile devices. To accomplish this, Haddock-FS reduces the size of update logs stored at each device, as well as of update data

to be transferred during replica reconciliation. This is done through an approach that was previously used in Low-Bandwidth File System (LBFS) [29].

Some conflicts can be automatically resolved by Haddock-FS. Namely, certain directory conflicts since they are system defined objects whose semantic is well-known. These conflicts are: remove/update conflicts, update/update conflicts and rename/rename conflicts.

To handle file updates, Haddock-FS, follows a strict causal consistency while ordering updates. This means that if a file update is issued in one replica no other update, issued by the same replica or another, will precede this update. This forces that only one of two concurrent updates will be applied at a certain replica. The author states that this implementation choice is preferred to ordering the concurrent updates since there is no semantic information available to make a suitable ordering.

However, if a causal conflict is detected no action is taken and the replica manager is notified.

Haddock-FS still has some drawbacks. The author refers that in the prototype version of Haddock-FS, every data structure kept by the file system is stored only in main memory. In the case of loss of battery or if the user hard resets the device all the file system data is lost. Another drawback, also found in the prototype version of Haddock-FS, is the fact that in case of conflict between two concurrent file versions, the system automatically corrects the conflict and deletes the divergent updates without offering the user with a back-up of the divergent object.

2.4.4 Conclusion

In Table 2.1 shown in page 11 is presented a summary of the systems described above.

	Coda	**Roam**	**Haddock-FS**
Topology:	Client-Server	Ward model	Peer-to-Peer
Platforms:	Linux, Windows	Linux	Windows CE.Net
Operations:	State-based Transfer (files), Operation Transfer (directories)	State-based Transfer	State-based Transfer
Propagation:	Hybrid	Pulling	Pulling
Scheduling:	Syntactic for files, Semantic for Directories	Syntactic for files, Semantic for Directories	Syntactic for files, Semantic for Directories,
Detecting Conflicts:	vector-clock/Semantic	vector-clock	Dynamic Version Vectors, Version Matrices
Resolving Conflicts:	Manually by user	Resolved automatically in files with known semantic properties. Otherwise, resolved by users.	Resolved automatically in some cases, In case of causal conflicts nothing is done

Table 2.1: Comparison between the studied Distributed File Systems

2.5 Data-Sharing Middleware

Data-sharing middleware hides the details of replication mechanisms. This allows applications and developers to transparently use these mechanisms without further changes to the applications code. Some examples of data-sharing middleware are presented now.

2.5.1 Semantic Chunks

Semantic Chunks [42] is an adaptive middleware that uses documents' semantic regions relevant to applications as a way to gather the appropriate information and enforce consistency. It was designed with cooperative work in mind.

The authors of Semantic Chunks defend that both update-based and operational-based approaches employed to ensure consistency in current solutions, are not efficient or flexible enough. They suggest an alternative solution that eliminates the drawbacks from these two approaches while keeping the advantages of each. Thus, their solution increases concurrency, it is transparent w.r.t. applications and reduces the number of conflicts.

This middleware divides documents in semantically relevant regions. The fragments of data that compose these regions are called semantic chunks and are different depending on file type and application semantics. For example, a semantic chunk may be a paragraph in a text document, a slide in a presentation or a cell area in a spreadsheet. When a file is modified and requires synchronization with another device, only the semantic chunks related to the modifications need to be propagated.

Chunks were used in previous solutions such as LBFS [29] and Haddock-FS [5, 3] and allow for savings both in bandwidth and storage. However, they do not possess consistency mechanisms by themselves. Semantic Chunks handles this omission by applying the chunk division in a semantic context dependent from applications and based on file types. This way, a semantic chunk may be a paragraph in a text document, or a slide in a presentation. This division offers an increased concurrency in file modifications and reduces the number of conflicts. XML is also used in order to encode meta-data regarding to consistency.

When conflicts occur there are several user based schemes that can be used in order to resolve them. Some of them are: i) voting schemes, where users can insert annotations to inform others which update they want to keep. ii) Authoritative messages, where a user with more privileges can force his update to be accepted. iii) user leases, that are time periods in which a certain user hopes to generate another update; and iv) custom hint messages, that are used to spread certain information to other users concerning a specific update.

In conclusion, Semantic Chunks is a middleware used for collaborative work. It establishes a middle-ground between update-based and operational-based approaches. By doing this it increases concurrency, it is transparent w.r.t. applications and reduces the number of conflicts. It uses Semantic Chunks which are smaller than a file, thus reducing network and memory usage and allowing for a fine-grain control over updates to files.

2.5.2 Xmiddle

Xmiddle [46] is a mobile computing middleware designed to help building mobile applications that use both replication and reconciliation over ad-hoc networks.

Xmiddle uses eXtended Markup Language (XML) [6] to represent data and information. To manage this data Xmiddle resorts to Document Object Model (DOM) [44] . The information stored in mobile devices is structured in hierarchical tree structures.

To manage updates, Xmiddle assigns a version number to each node of a modified tree inside the mobile device. Besides this number it is also stored the identity of the host who performed the modification. When two mobile hosts have successfully reconciled two replicas, it is stored a new version number and the IDs of the hosts that were involved in the merging procedure. This way, this update is differentiated from another update bearing the same version number but processed by a different pair of hosts.

The goal of reconciliation in XMiddle is to create two identical trees with the same version numbers in two different mobile hosts. The reconciliation process is handled in two phases: At first, Xmiddle applies application independent techniques. Namely, XML tree comparisons and merging. When two hosts, H1 and H2 connect, they check if they share some branches from the trees they are storing. If they do, H2 sends the history of modifications performed to H1. H1 compares the received history with its own and marks all the differences to be sent to H2. When H2 receives the differences it generates a new tree that is compared to the one H2 already stores and merges the two. Finally, it sends to H1 all the modifications it has to perform in order to create this merged tree. When this process ends successfully, both H1 and H2 will store the same tree with the same version.

However, when both hosts have modified the same files a conflict that cannot be resolved through simple merging processes arises. These type of conflicts can only be resolved through application dependent

techniques. Xmiddle offers developers the ability to specify reconciliation policies through XML schema. This allows the specification of behaviors that will be triggered to handle conflict resolution.

In conclusion, Xmiddle provides a middleware designed to help developers to build mobile applications that use replication and reconciliation over ad-hoc networks. It uses data structures based on XML to represent and share data. It enforces consistency by taking advantage of application-specific information to achieve acceptable performance, usability and scalability.

2.5.3 IceCube

IceCube [17] is a reconciliation middleware platform that can be used by arbitrary (synchronization-aware) application programs. It is operation-based and uses logs to store the update information.

IceCube acts as a semantic scheduler. Its goal is to obtain the log of user actions from two or more replicas. Then, it merges them while ascertaining an order of updates that minimizes conflicts and respecting the applications semantics and user intent. However, exploring all possible update orders takes too much time.

In order to avoid a combinatorial explosion, IceCube, employs both static constraints and dynamic constraints. A static constraint has to do with the order in which operations can be applied to a shared object to achieve a consistent final state. It does not take into account the state of objects; Only the order in which it is safe to apply two different operations.

The dynamic constraints can be preconditions or operations. A precondition checks if the state of an object is valid. In case it is not, the execution of updates fails. An operation is a method that may modify the shared objects and returns a boolean to indicate success or failure. If false is returned the execution is stopped.

To help create better constraints, IceCube allows developers to establish pre and post-conditions or application specific policies in order to help the middleware reach a better ordering decision.

During execution, IceCube is in one of two phases: isolated execution phase or in reconciliation phase. A replica is in the isolated phase when in normal operation. In this phase the replica applies several updates to the shared objects by request of the user. All these updates are stored in a local log and are marked as tentative.

When two or more distinct replicas need to merge their local logs, they enter in the reconciliation phase. The goal of this phase is to bring all shared objects throughout the involved replicas to a consistent state. This phase is split in three stages.

The scheduling stage, in which combinations of updates stored in the logs are arranged into schedules. A schedule is a sequence of actions that meet all the static constraints and end in a state considered correct. This way a combinatorial explosion is avoided since only the valid combinations of updates are taken into account in future stages.

Next, comes the simulation stage in which all schedules are applied to local copies of the shared objects. In this stage IceCube tests if the dynamic constraints are fulfilled. If it is not, the schedule is aborted and the next schedule is tested.

Finally, IceCube enters the selection stage in which all schedules that delivered a correct and valid execution during the simulation stage are compared and graded. Only the schedule that ended with the best final state is chosen. After these phases conclude, each replica can apply the chosen schedule, each ending with the same consistent final state.

IceCube, is fairly efficient in reaching a chosen schedule. The authors state that a simulation of 10 000 schedules takes less than three seconds. However, simulation times are proportional to the number of simulated schedules and in other cases the results may not be as good. To limit the number of combinations static constraints, together with policy and dynamic constraints are required.

2.5.4 Microsoft Sync Framework

Microsoft Sync Framework [24] is a synchronization platform that allows developers to build solutions capable of synchronizing, roaming and sharing any data type on any platform, application or device. It also supports the synchronization of data modified offline.

A Participant is the name given to a location that stores the information that is to be synchronized. Any location capable of storing data can be a participant, such as, a web service, a laptop or a USB thumb drive. However participants are broken in three categories. The Devices that allow developers to create applications and new data stores directly on the device are called full participants. Full participants are able to run a Microsoft Sync Framework Runtime component within and are the ones that handle the synchronization process. Partial participants are devices that can store data within, such as, USB thumb drives or SD cards. Finally, Simple Participants are devices that are only able to provide information when requested, such as, RSS feeds.

In order to be able to synchronize different participants a developer must create a provider. A provider varies depending on the the data source that needs to be synchronized. Although the framework supports the creation of custom providers, several of the most common data sources are already provided. This includes database synchronization providers, file and folder synchronization providers or web synchronization providers (used, for example, to synchronize RSS feeds).

The developer also needs to specify the data source that contains the information that needs to be synchronized. Each Sync Provider accesses its respective data source and guarantees its consistency. The data source can be composed of any data type. From databases to media files. The Sync Framework also allows the developer to define custom data types in order to support all data needed by the developer.

The provider stores information regarding the objects located within a data source, with respect to their state and changes performed to them. This information is stored within a metadata store which can be stored within the system in a location specified by the developer. Although the developer can implement his own metadata store, the Sync Framework comes with an already implemented metadata store.

The Sync Provider, Data Source and Metadata Store are the three modules needed in order to perform the synchronization process. A developer is free to create an application that makes requests to the Sync Provider and is therefore able to to perform synchronization or sharing of data between other replicas. The Sync Provider will also handle the communications with other Providers located in other Participants.

Microsoft Sync Framework also comes with already implemented mechanisms that detect conflicts and resolve them. These mechanisms supports only a pre-established policy for automatic conflict resolution. For example, in order to resolve concurrent updates the synchronizer always keeps the last write. However, the Sync Framework allows developers to implement their own conflict handling mechanisms as they see fit.

In conclusion, Microsoft Sync Framework is a synchronization platform that provides developers with a flexible framework that allows them to build applications or collaboration tools while preventing them from handling the complexity of the communication between replicas, the synchronization process or conflict resolution themselves. However, if needed it also provides the tools for developers to create their own solution built for their specific requirements.

2.5.5 Conclusion

In Table 2.2 in page 15 is presented a summary of the data-sharing middlewares described above.

2.6 File Synchronizers

A file synchronizer is a user level tool that allows a user with data replicated throughout different devices to maintain that data updated and consistent. Creating a simple file synchronizer is not difficult. However, building one that works fast, that deals correctly with the details of file system semantics, and that operates

	Semantic Chunks	**Xmiddle**	**IceCube**	**Sync Framework**
Operations:	Hybrid	Operation Transfer	Operation Transfer	Update Transfer
Propagation:	Push	Pull	Manual	Manual
Scheduling:	Semantic	Syntactic	Semantic	Syntactic
Detecting Conflicts:	Vector-clock	Versioning mechanism / application dependent policies / Semantic	Semantic / preconditions	Configurable
Resolving Conflicts:	Voting schemes, Authoritative messages, user leases and custom hint messages	Uses defined applications dependent policies. When not possible, resolved manually by user	analyses user's intent, application's policies and dynamic constraints	Configurable

Table 2.2: Comparison between the studied Data-Sharing Middleware

robustly under a range of failure scenarios is much more challenging [1]. These tools can be divided into two categories: traditional File Synchronizers and Online File Synchronizers.

When using file synchronizers in general, a user keeps his files stored in each of his devices. Eventually, the user will face the problem of keeping his files synchronized. A traditional file synchronizer is able to detect if conflicts exist between different file versions and propagate the merged and consistent file throughout devices.

However, traditional file synchronizers face a problem: the different file versions contained in different devices must be merged without a master copy or a central repository. Before merging these files they may have to be moved both ways and checks have to be made to ensure that changes previously performed aren't lost. Some examples of these tools are Microsoft's Briefcase [22] or SyncToy[26].

Online file Synchronizers, like Dropbox [10] or Microsoft's Live Mesh [23] on the other hand, keep the user's files in a central repository. When a user needs to synchronize his files, a request will be sent to this repository. These systems offer several advantages to users compared to traditional solutions. They can be used as an online backup storage so that, even if the user moves to a new device, which he has never used before, he could still fetch his data from the backup server. Some solutions, even allow a user to accomplish this using a simple web-browser.

However, complete copies of the user's files are stored in these repositories which can arise some privacy issues as some users are not comfortable with this. Moreover, the fact that these systems require Internet access may prevent their usage by some devices, such as digital cameras or external hard drives.

In this sections, some case studies of already existing file synchronizers will be presented, being described from a user's point of view:

2.6.1 Microsoft's Briefcase

Microsoft's Briefcase [22] was created as part of Windows 95. Though better solutions exist nowadays (some built by Microsoft), new versions of Windows kept this utility for users who had used it in earlier versions of Windows and wanted to continue using it to synchronize their files [27].

Using Briefcase is relatively simple. A user creates a Briefcase as he would create a normal folder. Then, he just needs to drag or copy his files into the Briefcase and copy the briefcase to another device (by using a USB pen drive, for example).

If changes are performed to files inside the briefcase, the user must press the update button in order to initiate the synchronization process. This summons a window with information regarding the updates

detected and the default actions that need to be performed in order to synchronize the briefcases in both devices.

Other options when synchronizing files include: i) the selection of individual files to be synchronized, ii) the selection of which version of a file the user wants to keep in each replica or iii) not synchronizing a file at all.

The process of synchronizing files mainly consists of sending the newer version to the device that stores the older version, while overwriting it in the process. By the end of the synchronization process both replicas store the same versions of the files.

Despite working reasonably well when faced with simple situations, Briefcase has several limitations. File synchronization in Briefcase is neither sophisticated nor intelligent. If a file inside a Briefcase is renamed or moved to a sub-folder, Briefcase is not able to detect that it is still the same file. It splits the file from the original, rendering it an orphan, which prevents the file from being synchronized in the future.

Another limitation is the fact that, when a conflict occurs i.e.both of the files in both replicas have been modified, Briefcase does not provide the user with any information regarding the conflict whatsoever. The user is presented with the default option to do nothing and must open both versions of the conflicting file in order to discover the reason for the conflict and resolve it. This is not always an easy task.

In conclusion, Briefcase is a solution that only works well when a user modifies a file in each replica at a time, without performing more complex operations inside the Briefcase, such as, renaming or moving files. Likewise, when a conflict occurs, it is unable to inform the user of the cause and how to resolve it.

2.6.2 Unison

Unison [30] is an open-source file synchronizer built for UNIX operating systems and Microsoft Windows as well. The focus of this tool is on portability, robustness, and smooth operation across different OS and file system architectures. It uses the rsync algorithm [41] to prevent the need of sending whole files through the network while synchronizing. It is an update-based system.

This tool has several interesting features. First, it allows the synchronization of files between different platforms, allowing synchronization, for example, between a Windows laptop and a UNIX server. However, a user must be careful when using filenames that are legal in a platform and illegal in the other. Second, it lets a user restore a past version of a document, by saving backups of files. This is limited to a certain number of backups, specified by the user, in order to save disk space. Finally, although a user can create a folder and copy into it all the files he wants to synchronize, as he would do with Microsoft's Briefcase, there is also the option of a user choosing a folder as his root synchronization folder. All the files in that folder will be ready to be synchronized with other devices.

During the implementation of Unison, its authors decided to formally specify the whole system (minus the graphical user interface). This was performed in order to better deal with system failure during synchronization, detecting updates and resolving conflicts.

In Unison, as with Briefcase, an update is detected when a file or directory is in a different state than it was since the last synchronization. If two files have been updated in two different replicas and their contents are not identical a conflict is detected. When this happens, Unison does nothing to reconcile both replicas. The user is informed that a conflict has occurred and that these files cannot be synchronized. However, Unison is able to invoke external programs to merge conflicting versions of a file. Still, this is complex as all arguments must be given to Unison from a shell and only administrators will be able to use this feature.

There are some disadvantages when using Unison. The graphical user interface is single-threaded; This means that if Unison is performing some long-running operation, the display will not be updated until it finishes. The user can be confused by this and may try to end the process or other actions while Unison is in the middle of detecting changes or propagating files. Also, Unison gives the user the option of using ssh or sockets to propagate updates between devices. Although this is true, a user is encouraged to always use ssh as the socket method is insecure. The changes are transmitted over the network in unprotected form and it is also possible for anyone in the world to connect to the server process and read out the contents of the file system.

2.6.3 SyncToy

SyncToy [26, 28], is a file synchronizer built by Microsoft for Windows XP and Vista. While SyncToy can be used to move, copy or synchronize folders containing regular documents and files, this application was built with the synchronization of large sets of photos across different computers in mind. The goal of SyncToy is to synchronize large volumes of files and folders even when some of them have been renamed or deleted.

SyncToy, is a successor of Microsoft Briefcase and works in much the same way. If a file is modified, it copies the whole file to the device where the older version is stored, overwriting it in the process. However, SyncToy is a much more capable synchronizer than its predecessor. It can handle conflicts in several different scenarios. When files stored in different replicas are renamed, SyncToy can still recognize that they are the same files and that it will not be required to copy each file.

When files are deleted from one replica and renamed in the other, SyncToy remembers that these are the same files. And more importantly, SyncToy can also handle the case in which a file is renamed in one replica and modified in the other. Many of these scenarios in Briefcase would generate files that could no longer be synchronized.

When a user wants to synchronize two folders, he is presented with several options on which action he wants to perform. The actions available are Synchronize, Echo and Contribute. Synchronize, copies and updates files both ways propagating renames and deletes of files. Echo, copies, updates files and propagates deletes and renames from the left folder to the right folder. Contribute does the same as Echo but does not propagate deletes, only renames.

During synchronization, the user has an option to preview all the actions that SyncToy will take and what files will be affected. The user can unselect any action before the synchronization starts. This way, unwanted behaviors, like loss of data, are prevented.

Another feature of SyncToy is the ability to save snapshots of each folder. This snapshot contains information about each file such as size, date/time of the last synchronization and hashes of file contents. These snapshots are used to help SyncToy make better decisions during synchronization. With the snapshots it has enough information to tell what changes have been performed.

Nevertheless, SyncToy, like Briefcase, does not feature an intelligent conflict resolution and in cases where files are modified in both replicas it does nothing and cannot provide the user with the needed information to resolve the conflict.

2.6.4 ActiveSync

ActiveSync [25] is a data synchronization tool developed by Microsoft. It is available for Windows and uses Infrared [1], Bluetooth [15] or USB [2] to connect devices. If used together with Microsoft Exchange Server [3] a mobile device can also be updated through a wireless network.

It allows users to synchronize documents, contact lists, calendars, emails and several types of media such as, photos, videos and music between a PC or a server and a mobile device that supports ActiveSync's Protocol. It also allows a user to backup files stored in his mobile device to other devices.

Data synchronization is initiated the moment two devices are connected together. Conflicts occur when a shared object has been modified in each device since the last synchronization. When this happens the user is alerted and ActiveSync presents options to resolve the conflict. The default behavior taken by ActiveSync in this situation is similar to Microsoft's Briefcase. The user selects one of the two conflicting versions to be kept while the other is overwritten and lost.

However, the user can configure ActiveSync to resolve conflicts automatically, although the resolution options available are few. There is only the choice of which version will be automatically kept. This is a rather poor configuration, especially if a user is used to modify his files on both devices.

[1] `http://science.hq.nasa.gov/kids/imagers/ems/infrared.html` accessed in 06/01/2010
[2] `http://www.usb.org/home` accessed in 06/01/2010
[3] `http://www.microsoft.com/exchange/2010/en/us/default.aspx` accessed in 06/01/2010

Conflicts also happen when ActiveSync is unable to copy objects to the mobile device without user intervention. This can happen, for example, if a file is too large to fit in the mobile device's memory.

Nowadays, ActiveSync can only be used with Windows XP since it has been replaced with Windows Mobile Device Center in the newer versions of Windows. Windows Mobile Device Center, works similarly to ActiveSync but is built using the Microsoft Sync Framework which allow for some new features.

In conclusion, ActiveSync allows to successfully keep files replicated between a computer and several mobile devices. However, the conflict resolution options available are few and rather poor. Also as with other synchronizers, no information about the conflicts are presented to users.

2.6.5 HP QuickSync

HP QuickSync is a file synchronizer that comes preinstalled with HP mini netbooks. It can be used to synchronize photos, videos, documents, e-mails, contacts and calendar between several devices that have HP QuickSync installed including mobile phones. Devices can be synchronized only if they are connected to the same local network. An option can be selected in order to allow devices to be automatically updated whenever a modification is performed.

In general, HP QuickSync, works the same way as Microsoft Briefcases does. However it has several drawbacks that prevent HP QuickSync from being a good synchronization solution for the casual user.

HP QuickSync can only be acquired and used by buying a HP Mini netbook. To be able to install the application on another device, and allow the two machines to synchronize, a user has to allow HP QuickSync to copy some files onto a USB thumb drive. Then, the user must take the thumb drive and install the program on the other device. If the user has other devices which he wants to synchronize with the netbook he must purchase an additional license from HP for each of the devices. This fact prevents users from taking full advantage of the software since they are not able to synchronize information throughout all the devices they own.

The program can be downloaded from the HP website [4], however it can only be installed in supported devices which are HP netbooks. Casual users will not be able to install and use the program in their machines.

Unfortunately, since it can only be used by someone that acquired a HP Mini netbook, the efficiency of the synchronization process, the ease of use and the conflict resolution could not be evaluated.

2.6.6 Sync Center

Sync Center [21] is a file management system developed by Microsoft. It is available both for Windows Vista and Windows 7.

The main goal of Sync Center is to provide a central location inside the OS for users to keep files, stored in their different devices, up-to-date. Users are able to sync laptops, mobile phones, PDAs, portable music players and digital cameras with a primary computer. The ability to keep all these different devices synchronized is the main advantage of Sync Center.

Sync Center does not provide synchronization mechanisms by itself. It uses the synchronization protocols already employed by the devices. For example, if Sync Center is synchronizing a Windows Mobile device with a desktop PC it uses Windows Mobile Device Center. What Sync Center provides is a central platform where a user can access all devices and files stored within them, schedule synchronization times, manually synchronize his devices or resolve conflicts. The file management and the conflict resolution is exclusively performed by Sync Center.

A major feature of Sync Center is the use of "Offline Files". This is a feature that first appeared in Windows 2000 and was called "Offline Folders". Offline Files allows a user to mark shared folders in

[4] http://h71036.www7.hp.com/hho/us/en/pclc/articles/quicksync-software.html

another device to be available offline. While the devices are connected, the marked folders are automatically synchronized every time a file inside them is modified. When a user disconnects he is able to access and modify the local cached copies of the offline files. Therefore, when he reconnects the devices, the changes that have been made offline are automatically synchronized with the online version.

During the synchronization process, Sync Center checks if a file has been modified and overwrites the older version with the newer version. If both versions have been modified since the last synchronization a conflict occurs. The user is allowed to choose which version he want to keep and which version he wants to update. This process is similar to Briefcase. However, Sync Center also allows the user to keep both versions in order to manually resolve the conflict.

In conclusion, Sync Center provides a tool that unifies all the synchronization activities across different devices. Although it works well with the supported devices it still suffers from the same problem as the other solutions. It does not offer any information as to what has been updated inside a file in case of conflicts and it is unable to merge files. It can only keep one version of each modified file. This limitation may result in the loss of data.

2.6.7 DropBox

Dropbox [10] is an online storage utility that allows a user to backup and access his files from anywhere where a computer with an Internet connection is available. It is available for Windows, Linux, Mac OS X and Iphone. This application provides 2GB free storage for each user to upload files. A paying customer can get up to 100GB of storage.

The usage of Dropbox is fairly simple. A user has in his computer a "Dropbox folder" that is managed by a background process. When a user modifies one of the files inside this folder the file is automatically uploaded to the Server. When a user moves to another one of his devices with Dropbox installed, the file is immediately fetched from the remote server and is synchronized with the local copy available. If new files are uploaded, the user is notified and they are propagated to all of the user's devices.

This application can also be used by several users as a collaboration tool as a user can define permissions to allow others to access specific folders inside his Dropbox folder.

Another feature of interest is the versioning system of Dropbox. It saves older versions of files and a user can access them if he needs to. Moreover, depending on the type of certain files, Dropbox is able to propagate only the modifications performed saving time and bandwidth during synchronizations.

Although Dropbox works very well and is able to successfully synchronize folders across different devices connected to the Internet, it still has a problem: when a user modifies a file in two or more devices without first synchronizing, Dropbox detects a conflict. It copies the two versions of the files to the user's computer and informs the user that a conflict occurred. However, no more information is presented and the user must find what is causing the conflict and resolve it himself.

2.6.8 Live Mesh

Live Mesh [23], is a new technology developed by Microsoft that is in many ways similar to Dropbox. Its purpose is to backup files online and to keep them synchronized. Like in Dropbox, files are stored in a Server. Each user has 5GB of free storage space.

Although Live Mesh is similar to Dropbox in many of the features it provides, it has some novelties. In Live Mesh a user is able to access all the devices that are currently online and fetch files from the shared folders through the browser. Moreover, in browsers that support activeX a user can remotely access other devices which are online at that moment. It allows a user to copy and paste files between the remote computer and his local computer.

Also, it provides a live feed that informs the user of new updates to folders or new devices that just connected. This feature is useful when using Live Mesh as a collaboration tool. A user is also able to access and synchronize his files from a mobile device with windows mobile installed.

However, Live Mesh also has some drawbacks. Some of them come from the fact that it is a relatively new service. The synchronization through the Internet is slow and live Mesh does not offer a progress indicator. Also, there are still some bugs that can crash the application.

2.6.9 Conclusion

Table 2.3 in page 21 presents a summary of the file synchronizers described above.

2.7 Distributed Collaboration Software

Distributed collaboration software enables teams to collaborate and perform concurrent work even if they are working remotely or belong to different organizations. Collaborative software provides tools that help communicating ideas and brainstorming. Additionally, it should support project management functions, such as task assignments, time-management with deadlines and shared calendars.

However, there are some challenges when creating distributed collaboration software. It is important to maintain the replicas consistent while allowing concurrent updates and merging. Also, since the Internet is used for communication it is important that the dissemination of information is efficient without wasting network bandwidth. Finally, it should provide security during communications and provide a way to manage a user's permissions inside each project.

By providing all the described tools, collaborative software supports the individuals that make up a team and the interactions between them. Hence, accelerating the work and facilitating the management aspects of each project.

2.7.1 Microsoft Office Groove 2007

Microsoft Office Groove 2007 [8] is a collaboration tool included in Microsoft Office 2007. It allows a user to form a team by creating collaborative virtual workspaces and share data with said team. Each workspace contains documents, images or other files to which each user will be able to access the project and modify it. Groove ensures that all users see the latest versions of the items contained in a workspace.

To create a team a user simply sends a workspace invitation to other colleagues. If they accept they receive a copy of the workspace and after the transmission is over they are able to start editing the workspace. From that moment on, each modification performed by the user to the workspace is automatically propagated to each member of the team.

Groove allows a user to also modify the workspace while disconnected. He sends the modifications he performed when he reconnects to the internet. Also he receives all modifications that were performed while he was offline. To enable this functionality Groove is divided in two different components: Office Groove 2007, that is a client that allows a user to access and edit the workspace and Office Groove Server 2007 a centralized server. The server stores modifications only if users are disconnected while they are being performed. Otherwise, modifications are sent directly to all users.

In order to make communication more efficient, each modification to the content inside the workspace creates a delta that represents a certain modification to a file. It could represent a modified entry on a calendar, a member that was added to the workspace, or the binary differentials between a changed file and the previous version. In order to synchronize each replica, Groove first modifies the local replica where the modification took place. Then, the delta is sent to all the other replicas. If they are online at the moment the delta is immediately applied to them. However, if a replica is offline the delta is saved is that replica's Groove server. When the replica connects to the workspace the server must apply the modification at that moment.

	Briefcase	**Unison**	**SyncToy**	**Active Sync**	**HP QuickSync**	**Sync Center**	**DropBox**	**Live Mesh**
Platforms:	Windows	Windows and Linux	Windows	Windows	Windows	Windows	Windows, Linux, Mac OS X and Iphone	Windows and Windows Mobile
Type of Synchronizer:	Offline	Offline	Offline	Offline	Offline	Offline	Online	Online
Correctly Handles Renaming / Deletion:	No	Yes, according to specification	Yes	May recreate already deleted files or generate duplicates	Yes	Yes in some cases	Yes	Only in some cases
Online Backup Service:	No	No	No	No	No	No	Yes	Yes
Syncs Mobile Devices:	No	No	No	Yes	No	Requires ActiveSync/ WMDC installed	Yes (Iphone)	Yes (Windows Mobile)
Detecting Conflicts:	Based on modifications of files	Based on modifications of files	Based on modifications of files	Based on modifications of files	Based on modifications of files	Based on modifications of files	Based on modifications of files	Based on modifications of files
In case of Conflicts:	Alerts users and does nothing	Alerts users and does nothing	Alerts users and does nothing	May be configured to automatically resolve conflicts	Not evaluated	May be configured to automatically resolve conflicts	Recreates two conflicting versions of a file so the user can resolve the conflict manually	Creates a folder with backups of all conflicting file versions

Table 2.3: Comparison between the studied File Synchronizers

Since disconnected operations are allowed, conflicts can occur if a user is offline and is modifying a certain document while other users who are online are concurrently modifying the same file. When this happens at the moment the user establishes a connection with the workspace he will receive a notification that a conflict has been found and a copy of the conflicting file is created with the name of the user who modified it as its title. To resolve the conflict user's intervention is required.

Data within each Groove workspace is always protected by 192-bit Advanced Encryption System (AES) encryption. To help the management of each team Groove allows the assignment of roles and permissions to each member of the team. Also, if a member is removed from a certain team, the next time he connects to Groove all of the data stored in his local replica is automatically deleted and he is not able to connect to the workspace.

Groove also includes built-in member presence awareness, workspace chat, messaging, and integration with Microsoft Office Communicator 2005 and 2007.

However, Microsoft Groove also has some drawbacks. In particular, for a single user who only wants to keep his files stored in several of his devices. The disadvantages of Microsoft Groove are:

- Groove is paid as it is a program included in Microsoft Office 2007. This detail makes Groove the most expensive system studied.
- Also, it is far too complicated to be used by a casual user. A server must be deployed and configured for each Groove Client. However, since Groove is mainly targeted towards enterprise environments this should not represent an obstacle for most teams.
- Groove does not have version control and in case of concurrent updates the last update always wins. However, it stores older revisions and notifies users when modifications are performed.
- Finally, as is the case with most studied solutions, when a conflict arises the user must manually search each of the conflicting versions to discover what differs between them. Since, Groove is a Microsoft product a diff application could be supplied at least for plain text and Microsoft Office files.

With that said, for teams distributed geographically it is a very good solution to share content and perform work concurrently.

2.8 Revision Control Software

Revision Control Software allows developers to manage and control the modifications of files within a certain project. Revision Control Software is mostly used in development environments in which a team of developers are working on the same project and possibly on the same files. Its focus is on development environments and an average user would not use this type of software to keep his personal folders in sync throughout several devices. However, a system called Git was studied since its emphasis on speed, efficiency and scalability are important characteristics for a successful synchronization tool.

2.8.1 Git

Git [11, 12, 38] is a free and open source, distributed revision control system. It was designed to be fast, efficient and scalable.

A central system such as Subversion uses a single repository for each project. The repository is located on a central location where all the project's history is stored. In order to checkout or commit an update, the user must access this central repository. With Git each user has his own project tree that contains its own repository. The tree can have local branches that were branches created in the local repository or remote branches which were created by other developers in other repositories. The fact that there are N repositories instead of a single central one ensures that no data will be lost if something happens to one of the repositories.

Since each developer has its own repository there is no need to be connected to the internet when tracking changes, merging branches or committing changes to the project. A user needs only to be online in order to pull or push data from a remote branch. When a developer finally pulls a remote branch from another repository, Git provides tools to visualize and navigate through the development history. This allows the developer to merge only what he wants and possibly prevent possible conflicts.

Git does not use delta storage system like other source control solutions. These solutions keep the differences between one commit and the next. Git, on the other hand, stores a snapshot of the state of all files present in the tree structure each time a commit is performed. This is done in order to save space and make the system more efficient. The metadata information stored for each snapshot is kept in a single folder located in the root of the project. This information allows Git to enforce consistency and to display to users the modifications performed from one commit to the next. The metadata kept by Git is also composed of configuration files, all the project's objects (commits, trees, blobs, logs) and some other files.

Git also detects conflicts and provides tools to help users visualize the differences between files and merge files in conflict, resolving the conflicts. However, since Git is mostly used for software development the included tools can only show differences between plain text files. Therefore, Git gives the user the option of choosing another differencing tool to be used by Git.

In conclusion, Git is a fast, efficient and scalable distributed revision control tool that greatly reduces the complexity of managing the modifications performed by a development team.

2.9 Discussion

In this section several solutions were presented. Each one performs replication of files and folders between different devices while enforcing consistency. In this subsection we study how each of these solutions fit to our previously presented goals. As stated in section 1.1 the main goals of Smart Briefcases are:

Goal 1) help a user maintain files replicated and

Goal 2) consistent between different devices by minimizing the number of conflicts during the synchronization process,

Goal 3) in case of conflicts, provide all the relevant information to help the user to manually resolve them,

Goal 4) the system must run without any modifications to the user's applications

Goal 5) the system must be efficient

Goal 6) and user-friendly.

Table 2.4 and table 2.5 in pages 24 and 25 display how each of the studied solutions achieves each one of these goals.

As can be observed, none of the solutions successfully accomplish all the established goals. Goal 3 in particular is not correctly accomplished by these solutions. In fact Haddock-FS and Dropbox are able to accomplish all the goals and they are still unable to accomplish goal 3. This fact, further substantiates the implementation of Smart Briefcases.

	Coda	**Roam**	**Haddock-FS**	**Semantic Chunks**	**Xmiddle**	**IceCube**	**Microsoft Sync Framework**	**Briefcase**	**Unison**	**SyncToy**
Help a user maintain files replicated:	+	+	+	+	+	+	+	+	+	+
Keep files consistent by detecting conflicts:	-	+/-	+	+	+/-	+	+	-	-	-
In case of conflicts, provide all the relevant information to help the user to manually resolve them:	-	-	-	+/-	-	-	+/-	-	-	-
Run without modifications to the user's applications :	+	-	+	+	-	-	+	+	+	+
The system must be efficient :	+/-	+	+	+	+	+/-	+	+/-	+	+/-
The system must be user-friendly:	+/-	+	+	N/A	N/A	N/A	N/A	+	N/A	+

Table 2.4: Part1: Comparison between the studied solutions and the goals of Smart Briefcases. The '+' sign represents that the solution successfully accomplishes the goal. '+/-' represents that the solution accomplishes the goal in some cases. The '-' sign represents that the goal is not accomplished by the solution. 'N/A' means that the goal is not applicable in this case.

	Active Sync	**HP QuickSync**	**Sync Center**	**DropBox**	**Live Mesh**	Microsoft Office Groove 2007	Git
Help a user maintain files replicated:	+	+	+	+	+	+	+
Keep files consistent by detecting conflicts:	-	N/A	-	+	+/-	+	+
In case of conflicts, provide all the relevant information to help the user to manually resolve them:	-	-	-	-	-	-	+
Run without modifications to the user's applications :	+	+	+	+	+	+	+
The system must be efficient :	+/-	N/A	+/-	+	+/-	+/-	+
The system must be user-friendly:	+/-	N/A	+	+	+	+/-	+/-

Table 2.5: Part 2: Comparison between the studied solutions and the goals of Smart Briefcases. The '+' sign represents that the solution successfully accomplishes the goal. '+/-' represents that the solution accomplishes the goal in some cases. The '-' sign represents that the goal is not accomplished by the solution. 'N/A' means that the goal is not applicable in this case.

Chapter 3

Architecture

This chapter presents the architectural approach used in Smart Briefcases, the main algorithms used and explains how the system works. The solution was designed having in mind all the goals presented in chapter 1, which are:

1) Smart Briefcases monitors the user's behavior when he is accessing files that will need to be synchronized in the future. This must be achieved in a way that does not slow down the system or creates large log files.

2) It must identify and resolve conflicts automatically when possible.

3) If it is not possible to resolve a conflict, Smart Briefcases must provide all the relevant information so that the user can easily resolve the conflict manually.

4) It must be efficient in terms of memory, performance and bandwidth usage. Data propagated must be kept to a minimum both in number of transfers and in size.

5) Be user-friendly by allowing a user to keep working with all the applications he usually uses and by displaying the information pertaining conflicts in an easy to understand manner.

Section 3.1 presents an overview of Smart Briefcases and a usage scenario of how a user may use Smart Briefcases daily. Section 3.2 presents the architecture of Smart Briefcases and describes how each module of the system works in general. Section 3.3 presents a brief description of each module that compose Smart Briefcases. Sections 3.4 to 3.10 provide a more detailed description of each module, explaining what is the purpose of each and how they fit together. Finally, section 3.11 explains some advantages and disadvantages of using file system watchers to monitor the user's behaviour.

3.1 System Overview

Smart Briefcases is an application that allows a user to keep data replicated and consistent throughout all his computers. To accomplish this, the application is built on top of a middleware that observes the user actions and maintains relevant data to be used during the synchronization process.

Consider a user who wants to keep several of his files and folders replicated between his computers. To accomplish this, the user, installs the Smart Briefcases application on his computers. After the installation process is over the user is able to start Smart Briefcases which becomes immediately ready to be used. The user interface of Smart Briefcases is simply a tray icon from which a menu can be accessed if the user right clicks on it. The menu gives three options to the user:

1) Create a New Briefcase: A briefcase is a folder created and monitored by Smart Briefcases. This options allows the user to create a briefcase in a location of his choice.

2) **Synchronize Briefcases:** This option allows a user to choose which pair of replicas he wants to synchronize. He also has the option to synchronize all the available pairs of briefcases at once.

3) **Close:** Exits from the application while saving all the metadata collected until that point.

The first step the user must perform when using Smart Briefcases is selecting the "Create a New Briefcase" option from the menu and selecting the location where the application creates the briefcase. When the briefcase is created, Smart Briefcases starts monitoring the folder for any changes, such as creations, renames or deletions of files or folders, that may happen within the briefcase. From this moment on, the user may copy any content that he wants to have replicated with other computers to the Briefcase. He can also create new files and folders. When this is done Smart Briefcases collects all information regarding what files and folders are stored within that specific briefcase.

Next, the user must copy the briefcase to another computer which forms a pair between the two replicas in each computer. The user has several options on how to transfer the briefcase to the other machine:

1) **Using a USB thumb drive:** The user can copy the briefcase to a USB thumb drive in one computer. Then, he connects the USB thumb drive to the other computer and transfers the copied briefcase to a location of his choice.

2) **Sending through the network:** The user simply accesses the other computer through the network and copies the briefcase to the desired location

3) **By email:** If the contents within the briefcase have a reduced size the user may create a zip file with a copy of the briefcase within and send the zip file by email. When accessing the other computer the user may download the file from his email and unzip the briefcase to a location of his choice.

If the two computers are connected by a local network and Smart Briefcases is running in both computers, Smart Briefcases will detect the creation of a new briefcase and will establish a new synchronization pair. To do this the instances of Smart Briefcases communicate with each other to share the already collected metadata and other information that will allow them to perform future synchronizations. In the end the two machines will contain a pair of briefcases that contain the same data and can be synchronized whenever the user requires.

A user is able to modify each isolated replica even is he is not connected to the local network or does not have an Internet connection available. When the user modifies a file or makes other changes to the Briefcase in each computer, the respective instance of Smart Briefcases stores metadata for all the changes performed.

When the user is finally able to reconnect the two computers through a local network, he may choose to synchronize the two briefcases. This process uses the metadata previously collected to detect all the modifications performed, synchronize the replicas and detect conflicts if there are any. Smart Briefcases is able to handle many different situations of file and folder's modifications. The several situations and how they are handled by Smart Briefcases is presented in table 3.1, for files and in table 3.2 for folders.

In case conflicts are detected during the synchronization process, Smart Briefcases informs the user and provides relevant information to help him resolve the conflicts and achieve a consistent state. A form detailing what caused the conflict is displayed and the user decides how he wants to resolve the conflict. In the end if the user resolves all conflicts the synchronization process ends and both replicas are left in an identical state. However, the user also has the choice of not resolving the conflicts immediately. He may end the synchronization process with conflicts unresolved. This conflicts will be detected again the next time the user requests the synchronization of this pair of briefcases.

This is the process through which a user is able to successfully maintain two replicas consistent in two of his computers. By using Smart Briefcases the user has several advantages compared to using other available solutions. Some of these advantages are:

1) Smart Briefcases does not need to be connected to the Internet or to a central service during the synchronization process. All replicas are peers that can be modified independently. This fact improves the availability of the shared contents.

Action Performed in Replica 1	Action Performed in Replica 2	How Smart Briefcases handles it
Rename file	No change to file	File in replica 2 is renamed
Rename file “A” with name “B”	Rename File “A” with name “B”	Nothing to be done since they have the same name.
Rename file “A” with name “B”	Rename File “A” with name “C”	**Conflict detected**. User decides which name to keep
Rename file “A” with name “B” and delete.	Rename file “A” with name “C” and delete.	Resolved automatically since they have been deleted.
Rename file “A” with name “B”	Delete file “A” and create file called “A”	File “A” in replica 2 is renamed to “B”
Rename file “A” with name “B”	Delete file “A”	**Conflict detected**. User decides whether he wants to delete file “A” from replica 1 or keep it in replica 2 with name “B”.
Create file with name “A”	Do nothing	File “A” is created in replica 2
Create file with name “A”	Create file with name “A”	**Conflict detected**. User decides which file to keep.
Create file with name “A”	Create file with name “B” then rename to “A”	**Conflict detected**. User decides which file to keep.
Create file with name “A”	Create file with name “A” then rename to “B”	File “A” is created in replica 2 and file “B” is created in replica 1
Delete file “A”	No change to file “A”	File “A” is deleted in replica 2
Delete file “A”	Delete file “A”	Nothing to be done since they have been deleted.
Delete file “A”	Delete file “A” and then create file named “A”	File “A” is created in replica 1
Modified file “A”	No change to file “A”	File “A” from replica 1 overwrites. File “A” from replica 2
Modified file “A”	Modified file “A”	**Conflict detected**. User decides which file to keep.
Modified file “A”	Rename File “A” with name “B”	File “A” from replica 1 overwrites. File “A” from replica 2. File in replica 1 is renamed to “B”.
Modified file “A”	Delete file “A”	**Conflict detected**. User decides whether he wants to delete file “A” from replica 1 or keep it in replica 2.

Figure 3.1: The operations that the user is able to perform to files and how the system handles these operations during synchronization.

2) As displayed in tables 3.1 and 3.2, Smart Briefcases is able to handle a lot of different scenarios when a user is changing a briefcase. This flexibility is not present in other solutions such as Microsoft's Briefcase which does not support, for example, the renaming of folders.

3) The graphical user interface is simple to use. In most cases a user is only required to press a single button in order to synchronize all the replicas.

4) Smart Briefcases is able to detect different types of conflicts. Moreover, when conflicts occur the application is able to present all the information the user requires to correctly resolve them. Most solutions do not inform the users that conflicts have occurred. The ones that inform the user of conflicts do not provide information on what caused the conflict or how to resolve it.

5) Smart Briefcases offers several differencing tools that allow a user to detect differences between the contents of files in conflict. This is true not only for plain text files but also for Microsoft Office files. This feature is not offered by traditional file synchronizers.

Action Performed in Replica 1	Action Performed in Replica 2	How Smart Briefcases handles it
Rename folder	No change to folder	Folder in replica 2 is renamed
Rename folder “A” with name “B”	Rename Folder “A” with name “B”	Nothing to be done since they have the same name.
Rename folder “A” with name “B”	Rename Folder “A” with name “C”	**Conflict detected**. User decides which name to keep
Rename folder “A” with name “B” and delete.	Rename folder “A” with name “C” and delete.	Resolved automatically since they have been deleted.
Rename folder “A” with name “B”	Delete Folder “A” and create folder called “A”	Folder “A” in replica 2 is renamed to “B”
Rename folder “A” with name “B”	Delete Folder “A” and create folder called “B”	**Conflict detected**. User decides if he wants to delete folder in replica 1 or not.
Rename folder “A” with name “B”	Rename subfolder of “A” with name “C”	Folder “A” in replica 2 is renamed to “B”. Subfolder in replica 1 is renamed to “C”
Rename folder “A” with name “B”	Delete Subfolder of “A”	Folder “A” in replica 2 is renamed to “B”. Subfolder in replica 1 is deleted.
Rename folder “A” with name “B”	Create Subfolder of “A”	Folder “A” in replica 2 is renamed to “B”. Subfolder in replica 1 is created.
Create folder with name “A”	Do nothing	Folder “A” is created in replica 2
Create folder with name “A”	Create folder with name “B”	Folder “A” is created in replica 2 and folder “B” is created in replica 1
Create folder with name “A” (No files inside)	Create folder with name “A” (No files inside)	Nothing to be done since they have the same name.
Create folder with name “A” (File with name “C” inside)	Create folder with name “A” (File with name “D” inside)	Folders remain with same name. File C is copied to replica 2. File D is copied to replica 1.
Create folder with name “A” (File with name “C” inside)	Create folder with name “A” (File with name “C” inside)	**Conflict detected**. User decides which file to keep.
Create folder with name “A”	Create folder with name “B” then rename to “A”	Nothing to be done since they have the same name.
Create folder with name “A”	Create folder with name “A” then rename to “B”	Folder “A” is created in replica 2 and folder “B” is created in replica 1
Delete folder “A”	No change to folder “A”	Folder “A” is deleted in replica 2
Delete folder “A”	Delete folder “A”	Nothing to be done since they have been deleted.
Rename folder “A” with name “B”	Delete folder “A”	**Conflict detected**. User decides whether he wants to delete folder “A” from replica 1 or keep it in replica 2 with name “B”.
Delete Folder “A”	Delete a subfolder of “A”	Delete folder “A” in replica 2
Delete folder “A”	Delete Folder “A” and then create folder named “A”	Folder “A” is created in replica 1
Delete folder “A”	Rename file “B” inside folder “A”	Folder “A” is deleted in replica 2
Delete folder “A”	Modify file “B” inside folder “A”	Folder “A” is deleted in replica 2

Figure 3.2: The operations that the user is able to perform to folders and how the system handles these operations during synchronization.

3.2 Architecture

To facilitate the implementation, maintenance and modifiability of Smart Briefcases the system is divided in different modules. The responsibility of each module, along with the main algorithms used, is explained in general below and will be described in more detail in the following sections. Figure 3.3 presents the architecture of the system.

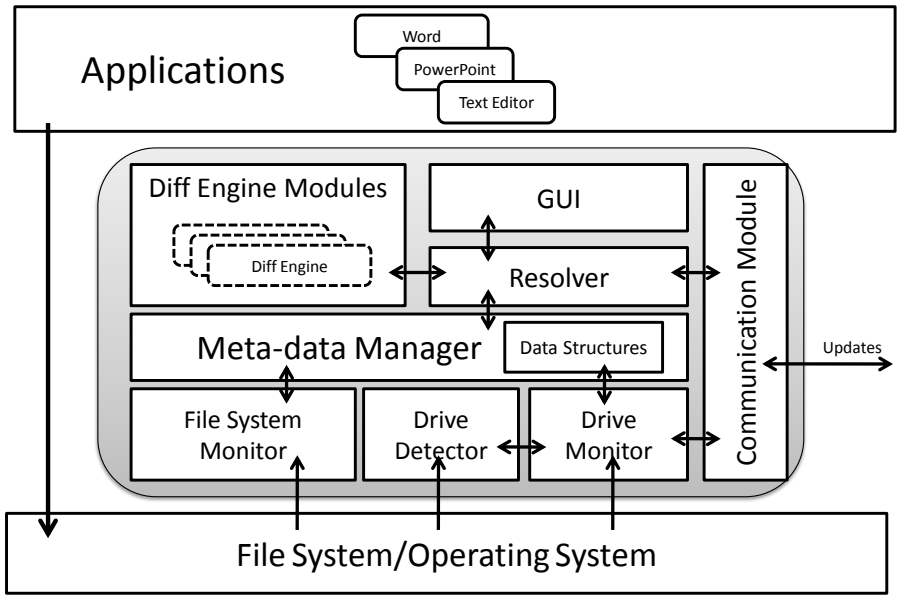

Figure 3.3: The modules that constitute the Smart Briefcases' Architecture.

3.2.1 File System Monitor

When a new briefcase is created by the user a File System Monitor is created and associated with that briefcase. The File System Monitor's goal is to monitor all the modifications a user performs to the files and folders stored inside a briefcase.

Every time a file or folder is renamed, modified, deleted or created within a briefcase, the file system monitor that is associated with that briefcase is informed by the operating system and receives information regarding the modification performed. This information is sent to the Metadata Manager to be stored in the respective data structures. The collected metadata will eventually be used during the synchronization process to detect all modifications that were performed while the replicas were disconnected.

3.2.2 Metadata Manager

The metadata manager is the component responsible for storing all the metadata that is used during the synchronization process. The data structures kept by this module are presented below.

1) **SynchronizationFolderPairs:** This structure maintains a list of all briefcases that exist in the user's computer and their respective pairs. This information is important to maintain an association between briefcases located in different computers. It is also used, before synchronization, to allow a user to select which pair he wants to synchronize.

2) **Directory Tree:** This structure represents the Directory Hierarchy of a briefcase. A representation of a Directory Tree is displayed in figure 3.4. A Directory Tree is represented by a tree data structure in which each node is an object that represents a folder. This object is called a FolderStruct and

stores information vital to keep a folder consistent in all replicas. The root of the Directory Tree is a FolderStruct that represents the briefcase. Its children are FolderStructs that represent the folders inside the briefcase.

The Directory Tree stores information regarding to which files and folders are stored inside a briefcase and what changes have been performed to them. During the synchronization process the Directory Trees from each replica are compared, by the Resolver, in order to discover what has been modified. A more detailed description of how the Directory Tree works is presented in section 3.4.

3) **Briefcase_Modified_Information:** When a briefcase is deleted or renamed in one replica, the replica pairing with this briefcase must be informed. With this information the unmodified replica breaks the synchronization pair, in case the other briefcase was deleted or updates its internal information, in case the other briefcase was renamed.

   If, at the time of the modification, the replicas are connected by a local network the unmodified replica is informed immediately. However, when the replicas are disconnected, the information pertaining to the changes performed must be stored somewhere until the replicas are able to reconnect. This is the goal of the Briefcase_Modified_Information structure.

   Briefcase_Modified_Information stores metadata regarding to which briefcase has been modified and how it was modified. When the replicas connect and the modifications have been performed the information stored in the structure can be deleted as it is no longer needed.

3.2.3 Drive Monitor

The Drive Monitor's goal is to monitor the creation of new synchronization pairs and the creation, renaming or deletion of briefcases. Smart Briefcases associates a different Drive Monitor to each drive (Hard disk drives and USB flash drives) inside the user's computer. Each Drive Monitor watches for modifications performed in its respective Drive.

Every time a folder is created, renamed or deleted inside a drive, the Drive Monitor that watches that drive receives information through the operating system regarding the modification performed. This is used to detect when a briefcase is created, renamed or deleted and apply the necessary mechanisms in order to handle the modification performed.

In order to allow the Drive Monitor to differentiate between a briefcase and a regular Windows folder a special hidden file is created inside each briefcase. This file is called Settings.ini and after its creation the file is set as a hidden system file which makes it completely invisible to users.

Settings.ini contains important information such as: the name and path of the briefcase; the IP address and port used by the executing instance of Smart Briefcases to receive requests from other replicas; a number that identifies the briefcase in this computer; and a global unique ID that identifies this specific briefcase in all the user's computers. The global unique ID is composed by the number of this folder and the IP and port previously mentioned. An example of a global unique ID is "2:192.168.1.3:8080".

Using the Settings.ini file the Drive Monitor is able to correctly identify a briefcase and update all structures using the information of the modifications applied to that briefcase. For example, if a folder located in Drive D: is renamed, the drive monitor that monitors drive D: receives this information. He verifies if a Settings.ini file exists within the folder. If the file does exist, the folder is identified as a briefcase. The Drive Monitor then proceeds to update all the relevant metadata structures, stored in the Metadata Manager, with the new name. When this process is finished the renaming of a briefcase has been correctly handled.

The Drive Monitor is also able to detect when a user copies a briefcase from one computer to another with the intention of forming a synchronization pair between the two computers. In this case, the Drive Monitor must use the IP and port stored in the Settings.ini file to connect to the computer from where the new briefcase was copied. The Drive Monitor sends and receives all the information required for the synchronization pair to be formed. The network information required for future connections between the two replicas is stored in the NetworkInfo structure kept by the Communication Manager.

A more detailed description of how the Drive Monitor performs the required actions to handle the modifications performed inside a drive is detailed in section 3.5.

3.2.4 Drive Detector

The Drive Detector's goal is to detect when a removable drive, i.e. USB thumb drives or external hard drives, is inserted or removed. This is used to allow a user to transfer a briefcase to another computer (in order to form a synchronization pair) using one of these devices. Therefore, it is required that Smart Briefcases detects when a USB drive is mounted or unmounted from the user's computer.

When a USB thumb drive is mounted on the user's computer the drive Detector is informed and requests the instantiation of a Drive Monitor to watch over the drive. This way, if a briefcase is copied to the USB flash drive, the Drive Monitor is informed and forms a synchronization pair between the original briefcase and the briefcase located inside the USB flash drive. This allows the user to modify contents inside the briefcases and synchronize them until the USB drive is unmounted from the computer.

Finally, when the USB flash drive is unmounted from the user's computer the drive Detector is informed, disposes of the Drive Monitor and breaks the synchronization pair. The user is now able to copy the briefcase located inside the USB flash drive to another of his computers. This action forms a synchronization pair between the original briefcase stored in the user's computer and the one that has just been copied.

A more detailed description of how the Drive Detector is able to detect the mount and unmount of USB flash drives is presented in section 3.6.

3.2.5 Resolver

The Resolver is the module responsible for the execution of the synchronization process between synchronization pairs. Also, the Resolver detects the conflicts that arise during the synchronization process and resolves them.

In order to detect what was modified between two replicas since the last synchronization the Resolver accesses the Directory Tree that stores information regarding the briefcase stored locally. To obtain the Directory Tree from the remote replica the Resolver uses the information stored in the communication manager. Afterwards, the Resolver compares the two Directory Trees and detects what has changed since the last synchronization. With that information the Resolver is able to perform all actions in order to make the replicas consistent.

If any conflicts are detected during the synchronization process the Resolver displays all the relevant information to the user in order to help him make an informed decision on how to resolve them. When a conflict is detected the Resolver stores information about the conflict inside a structure called conflictList. The Resolver finishes synchronizing all the cases that do not present conflicts and then presents the information stored inside the conflictList to the user. This structure stores the following information for each file or folder in conflict:

1) A copy of the structures that represent the files or folders that are in conflict. This structures are copied from the file or folder's respective Directory Tree.

2) A description of what caused the conflict. This description is shown to the user so that he has an idea on how to resolve the conflict.

3) The type of the conflict that occurred.

4) What the user has chosen regarding on how to resolve the conflict.

In order to better inform the user of what has been changed inside two replicas of a file that are in conflict the Resolver is able to access a set of differencing modules. These modules allow the Resolver to

display the contents of the two replicas in conflict side-by-side highlighting the differences in content. This functionality is available for plain text files, Microsoft Word files and Power Point files.

In the end of the synchronization process, if all conflicts have been resolved by the user, both replicas will be in a consistent state. The Resolver is explained in more detail in section 3.7.

3.2.6 Diff Engine Modules

The diff engine modules are used by the Resolver to display the contents of two replicas of the same file side-by-side. Colors highlight the differences between the lines where the files differ. This is useful to provide the information of what has been modified in each file when there is a modification conflict between two versions of the same file. With this information the user is able to make a better decision on how to resolve the conflict.

The differencing modules that are currently implemented in Smart Briefcases are able to identify differences between plain text files, Microsoft Word files and Power Point files. However, Smart Briefcases was created with the goal of being extensible in this regard. It is relatively easy to integrate a new difference engine with Smart Briefcases and allow the Resolver to use it. The use of Diff Engine Modules are explained in section 3.8

3.2.7 Communication Module

The communication module contains the structure that stores the information needed to communicate with other replicas through the network. The structure is called BriefcaseNetworkInfo and contains the following information:

1) The path of the briefcase stored locally.

2) The path of the remote briefcase.

3) Information that represents if the briefcase has a pair and if that pair is stored locally or remotely. The pair can be located in the same computer where the other replica is located or in another computer.

4) The IP address and port needed to communicate with the remote replica.

5) The number that identifies the folder locally.

Also, the communication module contains the implementation of the functions used by all modules to request or send information to other replicas. The communication module is explained in section 3.9.

3.3 File System Monitor

The file system Monitor's goal is to monitor all the modifications a user performs to the files and folders stored inside a briefcase. To accomplish this, each monitor has a tree structure, that represents all the folders and files stored in the Briefcase, and a File system Watcher. A file system watcher is a component present in the 2.0 .Net framework that triggers an event each time there is a modification performed inside the watched folder.

3.3.1 Initialization

When a new briefcase is created a new file system monitor is instantiated and associated with the briefcase. Also, an empty tree structure is created to store all the meta-data, pertaining to files and folders, that will be used during future synchronizations.

From this moment on, all modification performed to the files or folders inside the briefcase are monitored and stored by the system.

Each briefcase has its own file system monitor. To allow several file system monitors at the same time, each file system monitor is started in a different thread. Likewise when a briefcase is deleted, the file system monitor is disposed and the thread is stopped.

3.3.2 The process of storing modifications

When a file or folder, inside a monitored briefcase, is created, modified, deleted or renamed an event is triggered and a function is called to handle the change. Each of these functions checks whether it was a file or folder that was modified and sends the information to the Metadata Manager. This module will then update the tree structure of the corresponding folder with the received information.

In the end metadata is kept for the file or folder pertaining to the type of change performed. This information is used, by the Resolver, to make decisions during the synchronization process.

3.3.3 FileSystemEventHandlers used by the File System Monitor

FileSystemEventHandlers are handlers that are triggered when a folder being watched is modified.

1) **OnChanged:** This event is triggered when a file inside a briefcase is changed. When this happens a function is called by the event handler to handle the modification. The function receives, from the operating system, the name and the path of the file that was modified. With this information the file system monitor accesses the Directory Tree structure stored in the Metadata Manager and updates the information inside the structure of the changed file. A flag is changed inside the structure representing that the file was changed since the last synchronization.

2) **OnCreated:** The OnCreated Event is triggered when a new file or folder has been created inside a briefcase. The function called by the event handler receives the name and path of the file or folder created. Then, it accesses the Directory Tree and creates a structure that represents the newly created file or folder.

3) **OnDeleted:** The OnDeleted Event is triggered when a file or folder has been deleted inside a briefcase. The function called by the event handler receives the name and path of the file or folder deleted. The function accesses the Directory Tree and sets the structure that represents the deleted file or folder as deleted. From this moment on, the structure of the deleted file or folder represents a tombstone. A tombstone is used by the Resolver to know that the file or folder has been deleted since the last synchronization.

4) **OnRenamed:** The OnRenamed Event is triggered when a new file or folder is renamed inside a briefcase. The function called by the event handler receives the previous name and the current name of the renamed file or folder, and its path. Then, it accesses the Directory Tree and updates the structure that represents the renamed file or folder.

3.4 Metadata Manager

The metadata manager is the component responsible for storing all the information that will be used during the synchronization process. The structures kept inside the Metadata Manager are:

1) **Directory Tree:** The Directory Tree represents the structure of the folders and files inside each briefcase. These trees help keeping data that represents all the modifications that occur. This data is crucial to help the resolver make decisions during synchronization.

2) **SynchronizationFolderPairs:** The SynchronizationFolderPairs maintains a list of all briefcases stored inside the user's computer and their respective pairs. This information is used to maintain an association between briefcases located in different computers. It is also used, before synchronization, to allow a user to select which pair he wants to synchronize.

3) **Briefcase_Modified_Information:** This structure is used to maintain information regarding the renaming or deletion of a briefcase while the system was disconnected. When the user's computer is reconnected to the network where other briefcases are located, Smart Briefcases uses the information stored inside the Briefcase_Modified_Information structure and propagates it to other replicas. With this information the unmodified replica breaks the synchronization pair, in case the other briefcase was deleted or updates its internal information, in case the other briefcase was renamed.

How the Directory Tree structures store and update the information they store is explained in the following subsections.

3.4.1 Directory Trees

A directory tree is a structure composed by objects that represent folders and files stored inside each Briefcase folder. As discussed above each briefcase folder has a Directory Tree associated.

Each folder is represented by an object named FolderStruct. This object contains important information about the folder, such as, the full path and name of the folder, the date in which the folder has been modified, flags that represent the state of the folder (modified, created, deleted, renamed or synchronized) and if the folder has been previously renamed the FolderStruct maintains the name the folder had before.

The FolderStruct also maintains a Dictionary of objects that represent each of the files that exist inside the folder. These objects are named FileStructs and are similar to FolderStructs but they store information that only regards to files. A FileStruct shares most of the fields present in a FolderStruct with the exception of the list of files.

Finally, a FolderStruct stores other FolderStructs that represent each of the Folder's sub folders. This allows the Metadata Manager to iterate through the Tree as if it was traveling through the real folders. These FolderStructs are stored inside a Dictionary where the name of each folder is used as its key. This decision was made in order to make searching a certain folder in a Tree faster. This is important since the process of updating the tree and synchronizing two folders must be the fastest possible. The importance of this will become apparent in section 3.4.2.

A conceptual representation of a DirectoryTree is displayed in Figure 3.4.

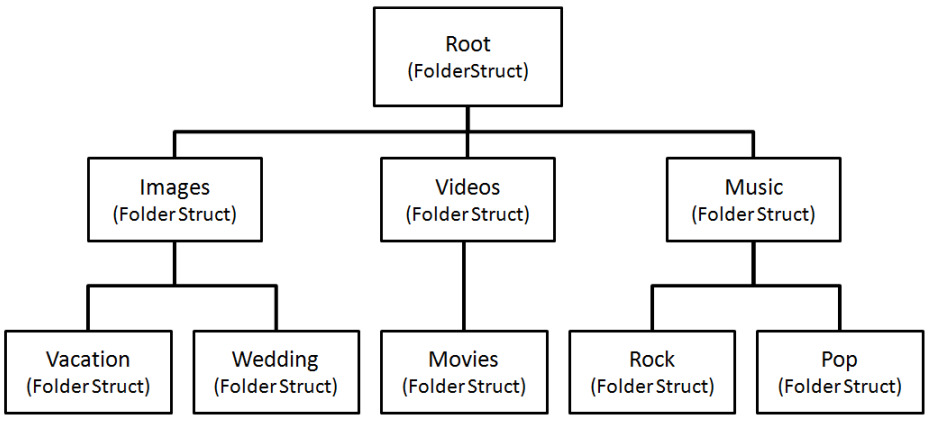

Figure 3.4: The conceptual representation of a DirectoryTree. In this image the Root folder represents a Briefcase Folder. The Root's FolderStruct has a list that contains three other FolderStructs, each for a different of its sub-folders (Images, Videos and Music). Similarly, each of these sub-folders also has a list that stores FolderStructs representing its sub-folders.

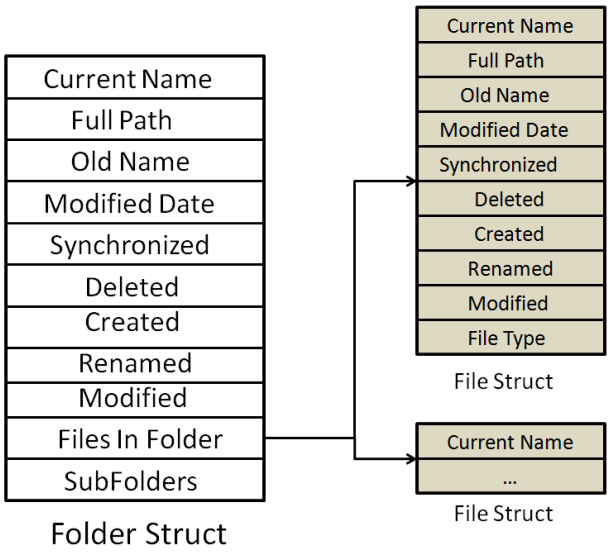

Figure 3.5: The image represents the FolderStruct and the FileStructs stored within. The FolderStruct is a structure that contains important information concerning a certain folder. A FileStruct on the other hand contains information concerning a file. This information is used by the Resolver during the synchronization process.

3.4.2 Searching for the modified files and folders structures

As mentioned above, when a file or folder is modified by a user the metadata manager receives data, regarding the modification, from the File System Monitor and updates the structure that represents the file or folder in question. However, before a structure can be updated the metadata manager must search the tree for the file or folder that triggered the event in the Monitor.

The data received from the File System Monitor contains the path of the folder where the modified file or folder is stored. To find the structure that represents this file or folder we simply iterate through the DirectoryTree going through all the FolderStructs that make up the received path.

Figure 3.6 displays the actions taken during the search algorithm. To better understand how the search is performed consider the following example: The folder "Pop", present in the DirectoryTree displayed in Figure 3.4, has been renamed to "PopRock". When this happens an event is triggered in the file system monitor. The file system monitor requests that the Metadata Manager applies the required modifications to the structure that represents the folder with the path: "C:\Root\Music\Pop".

The algorithm fetches the tree associated with the briefcase "C:\Root" and accesses the FolderStruct that represents the Root folder. It sets the folder Root as Unsynchronized and searches the Dictionay storing the subfolders for the FolderStruct with the key "Music". After finding the folder, the previous actions are performed again. The FolderStruct of the folder "Music" is set as unsynchronized and its Dictionary is searched for the FolderStruct that represents "Pop". Afterwards, since the FolderStruct that represents the folder "Pop" has been found the search algorithm is complete and the actions that reflect a rename are executed.

Now, consider that instead of a folder a file has been modified. For example a music inside the folder "PopRock" was renamed. The algorithm would search the tree in a similar way to the previous example until it discovers the FolderStruct of the folder "PopRock". Afterwards, the algorithm fetches from the Dictionary of FileStructs inside the FolderStruct of folder "PopRock", the object that has the key correspondent to the modified file's name. Finally, the algorithm updates the FileStruct to reflect the modification.

3.4.3 The state of Files and Folders

At all times a folder or file can be found in one of five different states that is now described:

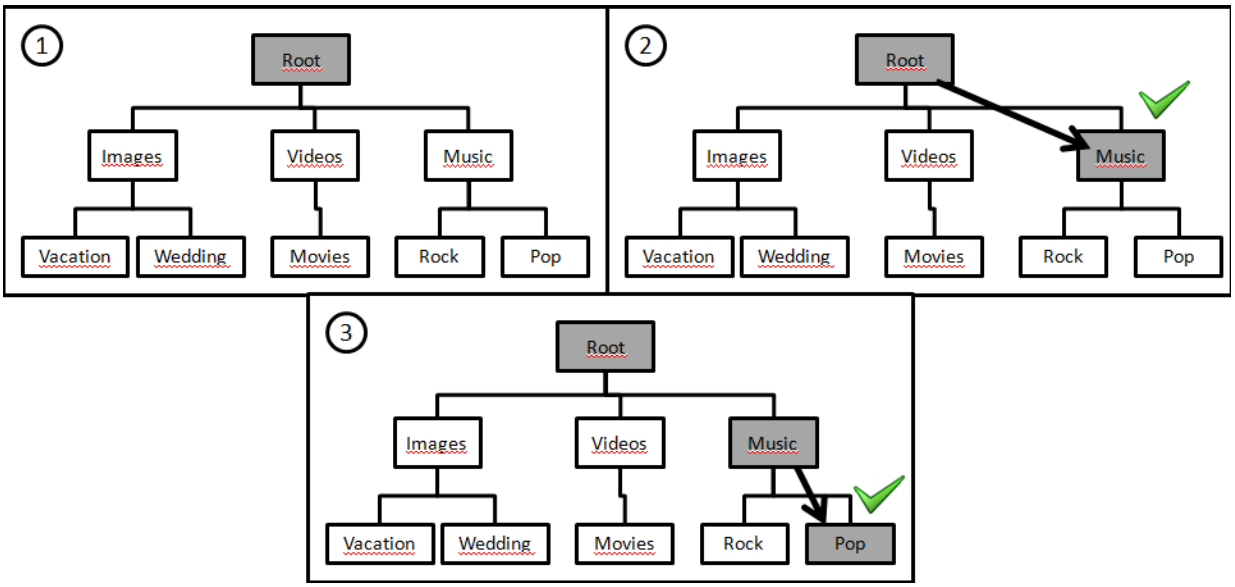

Figure 3.6: The search algorithm employed by the metadata manager to search for the structures that represent modified files and folders.

- When a file or folder is set as **Synchronized**, it means that this file or folder is currently in the same state in all of the replicas in which it is stored. At synchronization time, no action needs to be performed.
- The **Renamed** state implies that the name of a file or folder as been changed since the last synchronization.
- The **Deleted** state signifies that the file or folder has been deleted since the last synchronization. This information is kept as a tombstone representing the deleted file and maintaining a timestamp of the deletion time. This information is extremely important to correctly handle modification-delete and rename-delete conflicts.
- The **Created** state signifies that a file or folder has been created since the last synchronization. During synchronization these files and folders need to be replicated in the other replica.
- The **Modified** state is only applicable to files. It is used to mark a file which has had its content changed since the last synchronization.

3.4.4 Updating the state of files and folders

Every time the file system monitor detects a modification inside a briefcase, be it the deletion of a file or a folder being renamed, the event must be set in the corresponding tree in order to collect the information that will help the resolver to perform the right decisions during the synchronization process.

In this section, the process of updating the flags and structures of files and folders inside the tree is described with some detail. It is important to note that the structures are only updated if there is already a pair, in another replica, to the briefcase where the modification took place. This is done because if there is only one replica, there is no need for synchronization and therefore, no need to collect metadata.

The first step required to update a file or folder is to find its corresponding structure inside the tree structure. This is performed through the algorithm described in section 3.4.2. It is important to refer that each folder visited during the search process when iterating through the path where the modification took place is set as not synchronized. This is done to enhance the synchronization process' efficiency. This way, all folders that are set as synchronized, during synchronization, are ignored and only the paths that contain modifications are visited.

After the FolderStruct/FileStruct of the corresponding modified folder/file is found, this structure is updated depending on the type of modification:

- **A File or Folder is Renamed**

  When a folder/file is renamed the current name is updated to reflect the new name, the new path and the modified Date are updated and the folder is set as Renamed. Also, the old name is kept to identify the renamed file in other replicas, to resolve several different conflict situations and to give more information to the user if needed.

  Also, when a folder/file is renamed, the key of the structure inside the Dictionary must be updated with the new name. Therefore the structure must be removed from the Dictionary and stored again with the updated name as the new key.

  In case a folder/file is renamed several times all the modifications described above are applied. However, the original name that was stored during the first rename is kept since this is the name that identifies the renamed file in other replicas.

- **A File or Folder is Deleted**

  When a file or folder is deleted it is only required to set it as deleted and update the modified date.

- **A File or Folder is Created**

  When a file or folder is created its FileStruct/FolderStruct is created and initialized with the name and path of the file/folder and is set as created.

- **A File or Folder is Modified**

  When a file or folder is modified it is only required to set it as modified and update the modified date. No information is kept regarding to which part of the content of a file has been modified. The modifications made to the content of files are only detected during synchronization in case a conflict occurs and the user requests to see the differences between the two versions of the file.

However, when a file or folder already had a state assigned to it, different rules must be applied to prevent unnecessary actions to be executed. This rules are explained below:

- **A File or Folder is set as Renamed**
  - **The file/folder is modified:**
    When this happens, the file keeps its renamed state and is also set as modified. This is done because the rename and modification are seen as two different actions that must be handled differently and may originate different conflicts.
  - **The file/folder is deleted:**
    The FileStruct/FolderStruct is set as deleted and all other modifications are unmarked.
  - **The file/folder is renamed again:**
    Since the file/folder has already been renamed since the last synchronization it is only needed to update the current name, the path and the timestamp of the modification. The previous name (the original name that the file/folder had during the previous synchronization) is not updated since this is the name that is known by the other replicas.
  - **The file/folder is created:**
    Due to restrictions of the Operating System, it is impossible for a file/folder to be created in the same path of a file/folder with the same name. Therefore this case is not handled by Smart Briefcases.

- **A File or Folder is set Deleted**
  - **The file/folder is created:**
    In this case, the FileStruct or FolderStruct that existed previously and are currently set as deleted are reused. The structure is remarked as created and the timestamp is updated. During synchronization this will be handled has if this is a brand new file/folder.

- **A File or Folder is set Created**
  - **The file/folder is renamed or modified:**
    If the file/folder is set as created it has not yet been seen by other replicas. Therefore, there is no need to keep data for renames or modifications and update flags. Other replicas only need to see that this file/folder was created and replicate it when a synchronization process occurs.
  - **The file/folder is deleted:**
    In this particular case, it is possible to completely delete the structure from the tree since it has never been seen by other replicas. In the case of folders, all the structures of sub-folders are deleted along with the deleted FolderStruct since this is what happens to folders anyway.
- **A File or Folder is set Modified**
  - **The file/folder is renamed:**
    As already been mentioned renames and modifications are seen as two different changes. Therefore, the file keeps set as modified and it is also set as renamed.
  - **The file/folder is deleted:**
    The FileStruct/FolderStruct is set as deleted and all other modifications are unmarked.

3.5 Drive Monitor

As with the File System Monitor, the Drive Monitor is a component implemented using the File system Watcher component present in the .Net framework. The Drive Monitor, monitors all modifications inside each drive present in a machine and searches for briefcase creations, renames or deletions. This section describes how each Drive Monitor is able to detect the creation, deletion and renaming of briefcases and what actions are performed by the system to handle these operations.

3.5.1 Initialization

Every time Smart Briefcases is started, it initializes a Drive Monitor for each drive, of the type fixed or removable, present in the computer. This, filters the CD, DVD, floppy drives and other types of drives that are of no use to us, since briefcases are not supposed to be created inside these type of drives.

Each drive monitor is instantiated inside a different thread and watches for briefcase creations, briefcases being deleted or renamed. The Drive Monitors are only stopped when Smart Briefcases is shut down.

3.5.2 Briefcase Creation

Whenever a folder is created inside a hard drive or removable drive, an event is triggered inside the drive Monitor that watches that drive. However, not all folder creations are of our interest. The drive monitor only needs to handle cases in which a new briefcase pair is created or when a briefcase is copied from one folder to another inside the same computer. The whole process that handles the creation of briefcase is shown in the form of a flowchart in figure A.1 in page 90.

When a new synchronization pair is established, the newly created briefcase may have been copied from another replica or from the same machine where the original briefcase is stored. This two scenarios must also be distinguished by the Drive Monitor.

With this said, when a folder is created the Drive Monitor must first ascertain if the created folder is a briefcase or a regular folder. This is important since the former case can simply be ignored by the Drive Monitor. In order to accomplish this, the drive monitor verifies if inside the created folder exists a file called Settings.ini. The Settings.ini file has already been described in section 3.2.3. This file is what allows the drive monitor to differentiate a normal folder from a briefcase. In case this file does not exist, the created

folder is simply a regular folder created either by the user or the operating system. In this case the Drive Monitor takes no more actions.

In case it is a briefcase that is being created there are several tasks that must be executed. The Drive Monitor must first differentiate between the three scenarios in which a briefcase may be created. These scenarios are:

Scenario 1: The briefcase is a new briefcase created by the user through the Smart Briefcases' interface. In this case the drive monitor is not required to perform additional actions.

Scenario 2: The created briefcase is a copy of a local briefcase stored in the same computer. This functionality may be used, for example, by a user who wants to backup certain files and folders to an external hard drive or to another location inside the computer. In this case a synchronization pair will be formed between these two local briefcases.

Scenario 3: The created briefcase is a copy from a briefcase from a remote computer. The briefcase may have been created when the user was copying the remote briefcase through the network, transferring the copy of a briefcase through a usb flash drive or extracting the briefcase from a zip file sent by email. In this case a synchronization pair must be formed between the computer from where the original briefcase was copied and the computer that received the briefcase.

In order to differentiate between these two cases the drive monitor fetches the information stored inside the Settings.ini file. This information is important as it gives us details of the machine where the original briefcase resides. This information consists for example of the path where this briefcase was originally located. It also contains the IP address and port used by the instance of Smart Briefcases that created this briefcase. This information can be used to detect if the created briefcase was copied from a briefcase in the same computer or if it came from another computer. This allows the drive monitor to distinguish between the three scenarios described above.

The drive monitor compares the path and IP address taken from the Settings.ini file to the current path of the created briefcase and the IP address used by the local instance of Smart Briefcases, respectively. If they match, it means that the created briefcase was simply a new briefcase being created by the user. This is the scenario 1 described above. In this case, the drive monitor does not need to do anything else.

If the disc monitor does not recognize scenario 1 it means that the created briefcase was either copied from a local briefcase or from a remote replica. However, before taking more actions the disc monitor updates the Settings.ini file inside the created briefcase with: the new path of this folder, the local IP address, port and new ID. This step is not performed if the new briefcase is created inside a removable drive. This is done because a user may use the removable drive to copy a briefcase from one computer to another. When the briefcase is moved from the removable drive to the other computer the Settings.ini file included must provide all the information about the original briefcase to allow the two replicas to become a synchronization pair. If the information was updated with the path of the briefcase inside the removable drive, the original briefcase could not be found by the new replica.

Afterwards, using the information previously taken from the Settings.ini file the Drive Monitor infers if the created briefcase has been copied from an already existing folder in this same computer (scenario 2) or was copied from a folder that resides in another replica (scenario 3).

To better explain each of the actions taken in each one of these scenarios, they are described in the remaining subsections:

3.5.2.1 Scenario 2: Briefcase was copied from a local briefcase

In order to discover if the folder was copied from a briefcase stored in the same computer, the drive monitor determines if the path of the briefcase read from the Settings.ini file exists inside the local computer. If that is the case the drive monitor determines if the folder in that path has a Settings.ini file within and if the ID of that folder is the same from the one created previously. Finally, the drive monitor compares the IP address fetched from the Settings.ini to the IP address used by the running instance of Smart Briefcases.

If all these comparisons match, Smart Briefcases concludes that the original briefcase exists in the same computer where the new briefcase was created. The drive monitor will now form a synchronization pair between the newly created briefcase and the original briefcase. The drive monitor will also store information to allow future synchronizations between the synchronization pair. First, the Directory Tree of the original briefcase is copied and associated with the created briefcase. Since both briefcases store the same folders and files it is only required to update the path from the FolderStruct that represents the root of the DirectoryTree replica. No other modifications are needed.

A file system monitor is then created to monitor the created briefcase and to allow future updates performed inside the briefcase to be recorded. Finally, the original briefcase and the new briefcase are marked as a synchronization pair and some information is stored in order to allow future synchronizations between these two folders. This pair is marked as "Local" since the two folders are located inside the same machine.

With all these steps executed, the creation of a new synchronization pair in which the two briefcases are located in the same machine is concluded. Both folders can now be monitored for modifications and can synchronize with one another.

3.5.2.2 Scenario 3: Briefcase was copied from a remote replica

To ascertain if the pair of the newly created briefcase is stored in a remote replica the Drive Monitor compares the IP address fetched from the Settings.ini file with the IP address used by the local instance of Smart Briefcases. If they differ, the briefcase from which the created briefcase was copied is located in a remote replica.

In this case, the first task to be executed is to get the Directory Tree associated with the original briefcase located in the other computer. Since the briefcase is located in a remote computer, the drive monitor uses the Communication manager to establish a connection with the remote replica that stores the original briefcase. To do this successfully the Communication Manager uses the IP address and port previously read from the Settings.ini file.

After the communication between replicas is established, the drive monitor sends a request to the other replica asking for the Directory Tree of the briefcase stored in the path previously read from the Settings.ini. The remote replica responds by sending the requested Directory Tree.

The FolderStruct that represents the root of the Directory Tree is changed to mirror the location of the newly created briefcase. The Directory Tree is stored and associated with the newly created briefcase. Then, a new File System Monitor is created to watch over the briefcase and all the information required for future synchronizations between the two briefcases is stored.

With these steps performed, the creation of a synchronization pair in which the two briefcases are stored in two remote machines is concluded. Figure 3.7 shows the messages sent through the network between two remote replicas in order to create a new synchronization pair.

3.5.3 Briefcase Deletion

As is the case with briefcase creations, each time a briefcase is deleted in the user's computer an event is triggered inside the Drive Monitor. When this happens the drive monitor proceeds to clean all the information associated with the deleted briefcase.

First, the Directory Tree associated with the deleted briefcase is erased from the metadata manager. Then, the File System Monitor that monitored the deleted briefcase is stopped and disposed of.

If the deleted briefcase was part of a synchronization pair the replica where the pair is located must be informed in order to break the pair and clean all information pertaining to the deleted briefcase from the structures. However, if the replicas are not connected when a briefcase is deleted, the drive monitor stores this information inside the Briefcase_Modified_Information structure in the metadata manager. When the

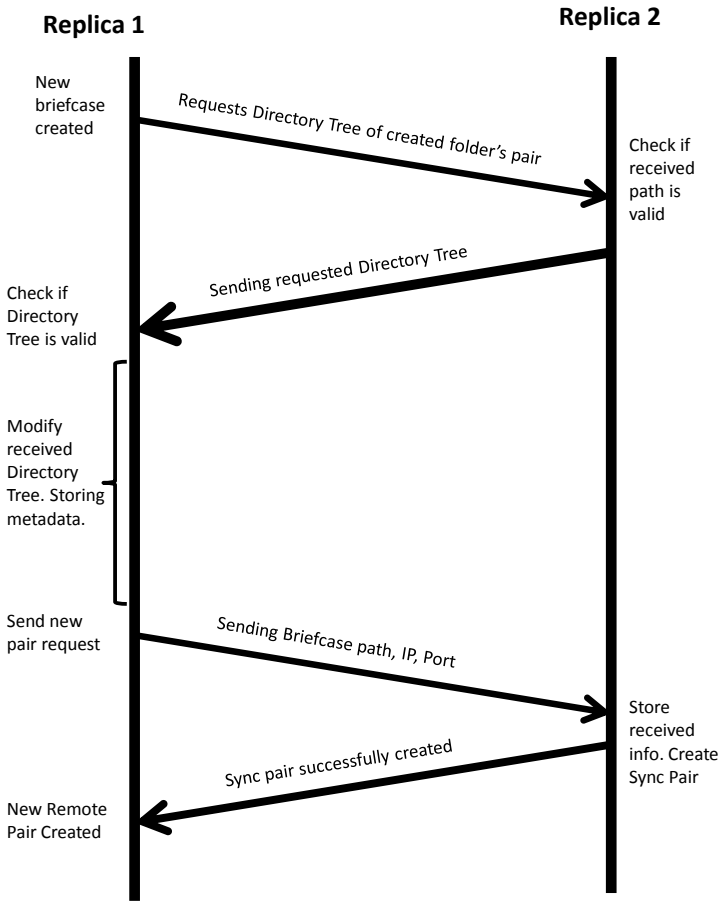

Figure 3.7: The messages sent through the network when creating a synchronization pair between two remote replicas.

replicas reconnect, the remote replica is informed of the deletion and takes the required actions. Finally, all the metadata stored that is related with the deleted briefcase is deleted.

This concludes, the actions performed when a briefcase is deleted.

3.5.4 Briefcase Rename

Again, every time a briefcase is renamed an event is triggered inside the drive monitor that will handle this modification.

First, the FolderStruct at the root of the DirectoryTree associated with this briefcase is updated to mirror the new path and name. Then, since the path of the briefcase changed, the File System Monitor that watches over this briefcase must be "restarted". This happens because when a briefcase changes its path its File Sytem Monitor stops receiving events. Therefore, the File System Monitor that was watching the briefcase is stopped and disposed of and a new one is created and initialized with the updated path.

Finally, the references to the renamed briefcase are updated in all the relevant structures kept by Smart Briefcases. In all cases only the path needs to be updated. These modifications are enough to ensure that the monitoring and synchronization of this briefcase keep performing normally.

3.6 Drive Detector

The Drive Detector's goal is to notify the application when a removable drive (such as a flash drive or external hard-drives) is inserted or removed. This is important since a briefcase can be transfered to another computer through one of these devices. Therefore, it is required that Smart Briefcases detects when a USB drive is mounted or unmounted from the user's computer.

The actual source code of the Device Detector class was downloaded from the Code Project website [9]. By adding the code to the project, the Drive Monitor allows Smart Briefcases to receive an event every time a device is mounted, unmounted or when a user tries to unmount it. When instantiated, the DriveDetector will create a hidden form, which it will use to receive notification messages from Windows. A developer is only required to implement handlers that are called every time one of these events is triggered.

The next sections describe how each one of these handlers is designed and used by Smart Briefcases.

3.6.1 DeviceArrived event

This event is triggered each time a removable drive is inserted in the machine. The DeviceArrived event's handler receives the path of the inserted drive and uses it to create a Drive Monitor to watch over this drive. This Drive Monitor is used to know if a briefcase is copied to the external drive or deleted before the external drive removal.

If a briefcase is indeed copied to the usb drive it becomes a synchronization pair with the original briefcase from which it was copied. Until the usb drive is unmounted or one of the briefcases is deleted a user can make changes to each of the two briefcases and synchronize them as he sees fit. This gives more freedom to the user in regard to him deciding when he wants to send the copy of the briefcase to other machines.

It is important to take into account that every removable drive inserted by a user whether it will be used to move briefcases to other machines or not will always have a drive monitor watching it.

3.6.2 OnQueryRemove event

When a user requests the operating system to unmount a removable drive the OnQueryRemove event is triggered. This happens before the drive has been unmounted. This is important because there is a Drive Monitor watching the drive. If the operating system tried to unmount the drive before the Drive Monitor is stopped it would not succeed since the removable drive is being used. Therefore, stopping and disposing of the Drive Monitor is the first step performed.

Afterwards, if a briefcase was indeed copied to the removable drive we must delete all references to that briefcase as if it had been deleted. Also, the original briefcase from which the briefcase was copied must remove its reference from its pair list. The pair between these two folders will be restored only if the briefcase inside the removable drive is copied to another computer. With these two steps performed the removable drive can be safely unmounted from the computer.

However, there is a downside to using this solution. If the removable drive is being used by another application, the drive cannot be unmounted. Although this is true, the OnQueryRemove event is still triggered and all information regarding the briefcase stored inside the drive is erased from Smart Briefcases. The drive remains mounted but all the modifications performed to the briefcase stored on the drive are ignored by Smart Briefcases. Unfortunately, currently there is no solution for this problem.

3.6.3 OnDriveRemoved event

This event is triggered after the removable drive has been unmounted. It can be used to perform some clean up operations. However, currently it is only being used for test purposes.

3.7 Resolver

This module is responsible for the whole synchronization process in which the pairs of briefcases selected by the user are made identical. It is also responsible for saving all the conflicts that may arise during the process and resolve them after receiving the user's input.

The synchronization process is comprised by several complex actions. To better explain how the Resolver works, the synchronization process if first explained as a whole and in the following subsections each step is described with more detail.

3.7.1 The Synchronization Process

The Synchronization Process is initialized by the user's request simply by pressing the Synchronization button located in the briefcase tray icon's menu. The user can select which pair of briefcases he wants to synchronize or synchronize all of the existing pairs at once. The replica where the synchronization process is initialized is where all the comparisons between folders, conflict detection and conflict resolution are performed. All the information that is needed to perform the process are the two Directory Trees from each briefcase that compose the synchronization pair. This is the main reason why only one computer is needed to perform all the computational work.

Therefore, when synchronizing all pairs at once, the computer where the process is initiated iterates through each of the existing pairs of briefcases, gets the Directory Trees from each of the folders of each pair and performs all the required actions to synchronize them.

The fact that the processing work is only performed by one computer is important to minimize the number of communications between the replicas. One computer performs all the work and discovers what actions are needed so that each briefcase reaches an identical state. The only information sent to the other computer are requests to perform actions like creating, renaming or deleting a file or folder or to modify the Tree structure of a briefcase.

In the end if there are no conflicts and the synchronization process was successful the pair of folders and their respective Directory Trees will be in an identical state.

The synchronization process is divided into two different phases: In the first phase only modifications performed to folders are handled; in the second phase modifications performed to files are resolved. It was chosen to divide the process in these two phases for three reasons. The first reason is simply to facilitate the implementation of the Resolver as it is easier to differentiate the resolution of folders from files.

The second reason is to handle the case in which files have been modified inside folders which have been renamed. By resolving folders first, at the time the Resolver resolves the file modifications, all the folders have already been renamed. If this was not the case, when resolving files the Resolver had to check if the renamed folders had already been renamed. It also had to keep the previous name of the renamed folder to allow the resolution. The problem would become even more complex if all the folders that compose the path of the file had been renamed. The additional mechanisms that had to be implemented to handle these scenarios would bring unnecessary complexity to the process.

Finally, the third reason is the fact that conflicts are also divided into two different types: folder conflicts and file conflicts. Only when folder's conflicts have been completely resolved can the Resolver start resolving files. This was the only solution found in order to allow the user to perform concurrent renames to folders and files in different replicas. These are the reasons why the synchronization was divided into two phases.

As explained before, in the first phase, the Resolver handles folder deletions, folder renames and folder creations by this specific order. Then, if there were conflicts detected, they are displayed to the user who must give his input in order to resolve them. In case the user decides for some reason not to resolve all the detected conflicts, the synchronization process ends without resolving file's modifications. The user is free to continue modifying his briefcases. The next time the user decides to synchronize these synchronization pairs the previously unresolved conflicts are shown again to the user along with new ones that might be detected. Again, only if the user resolves all the detected conflicts can the synchronization of files be performed.

During the next phase the Resolver synchronizes all files modified since the last synchronization. The actions performed to do this are similar to the ones employed during the previous phase. The Resolver handles file's deletions, renames, creation and modifications by this specific order and in the end displays all conflicts that were detected. If the user resolves all conflicts, all folders and files structures stored inside the Directory Trees are marked as synchronized.

This concludes the description of the synchronization process as a whole. In the end, each briefcase in each replica and their respective DirectoryTrees are identical.

3.7.2 How Conflicts are Stored

Every time a conflict is detected during the Synchronization Process, an object of the type Conflict is created. This object's goal is to store all the information that will allow the user to make a decision when resolving the conflict later.

An object of the type conflict stores: the FileStructs and the FolderStructs of both of the files or folders in conflict, a description of what was modified in each file/folder, the time at which the modification took place, the type of conflict that occurred and a flag that represents the choice of the user regarding to what modification he wants to keep when resolving conflicts. Some of these fields are explained in the following sections.

3.7.2.1 Types of Conflicts

The type of conflict stored in the Conflict object is a flag that represents the type of conflict that occurred. There are several types of conflicts that can occur:

1) Renamed: Both replicas of the same file or folder have been renamed in each machine. To resolve this conflict the user decides which name he wants to keep.

2) Delete-Renamed: A replica of a file or folder has been deleted in one computer while the other version of the same file or folder has been renamed in the other computer.

3) Creation: A replica of a file has been created in one computer while in the other computer a file has been created with the same name. When two folders are created with the same name it is not considered a conflict since the two folders can be merged by copying the files inside each one of them to the other. However, a file creation conflict occurs if files with the same name have been created inside these folders in each replica.

4) Modification: Both replicas of the same file have been modified in each computer. This conflict only occurs with files since it is important to detect if a file's contents have changed. Two modified folders on the other hand, are not viewed as a conflict.

5) Delete-Modification: A replica of a file has been deleted in one replica while the replica of the same file has been modified in the other computer.

3.7.2.2 User's Choices

The user's choice field is used to store the user's decision regarding on how he wants to resolve a specific conflict. When conflicts occur a Windows form is displayed to the user presenting all the collected information. The user must decide which folder/file he wants to keep by selecting the corresponding option presented in the interface. One of these Windows forms is shown in figure 3.8. The user is presented with three options :

1) LeftChoice: The user decides to maintain the modification performed to the folder/file displayed on the left side of the form.

2) **RightChoice:** The user decides to maintain the modification performed to the folder/file displayed on the right side of the form.

3) **None:** By default the user's choice field is marked as none. It represents the fact that no choice was made by the user regarding which modification to keep. When a user chooses this option, he wants to ignore the conflict and solve it at a later time. All the other conflicts where the user chose to keep the left or right file/folder are resolved and the conflicts marked with none are ignored.

   If at the end of the folders' conflict resolution there are conflicts marked as none the synchronization process ends without resolving the remaining modifications. The next time the user tries to synchronize this pair of replicas again he will be shown the same conflict form displaying all the conflicts previously marked as none. When he finally resolves them the synchronization process can continue and, if no more conflicts are found, terminate.

4) **Synchronized:** This value means that there is no need for the user to decide which file/folder he wants to keep. The files/folders are already identical and no action is needed. Normally, this state is achieved when a difference engine is used to resolve a modification conflict.

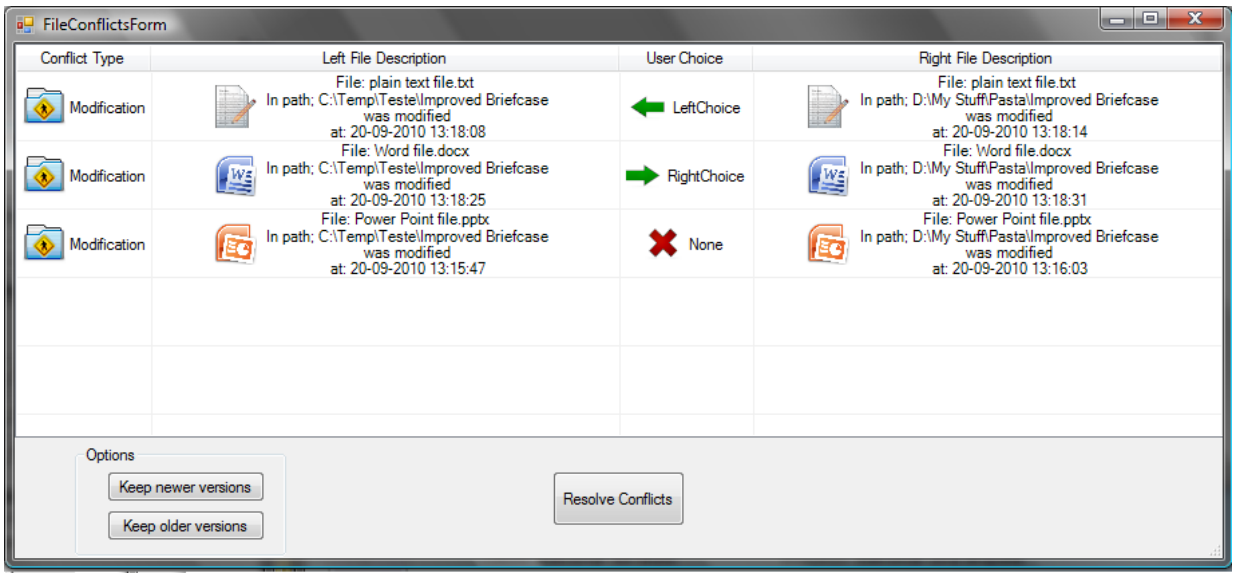

Figure 3.8: The Windows form displayed to the user when conflicts are detected between files.

3.7.3 How are modified folders synchronized?

The synchronization of folders is divided into three steps:

1) **Synchronization of Deleted Folders:** The first step is to handle all the deleted folders in each computer. In the end of this step all folders that were deleted in one computer and are not in conflict are deleted in the other computer.

2) **Synchronization of Renamed Folders:** The next step is to handle all the renamed folders in each computer. In the end of this step all folders that were renamed in one computer and are not in conflict are renamed in the other computer.

3) **Synchronization of Created Folders:** Finally, the Resolver handles all the folders that were created in each computer. In the end of this step all folders that were created in each computer and are propagated to the other computer along with the files stored within.

These three steps are described with more detail in the following sections.

3.7.3.1 Synchronization of Deleted Folders

The process detailed in this section is ilustrated in figure A.2 in page 91 to help the understanding of the following explanation.

The first step when resolving folders is to handle deletions. The reason why deletions are the first modification to be handled is to prevent the resolution of cases that do not need to be resolved. For example, consider that in one computer a folder A is deleted while in the other replica files and folders inside that same folder are modified. In this case it is only needed to delete the folder A without the need to resolve all the other modifications that happened inside the same folder in the other replica.

However, before the Resolver starts resolving deletions it must first analyze the Directory Trees of the briefcases in both computers in order to detect all the existing rename-rename conflicts. Then, the Resolver detects all the folders that were deleted each replica. With this information the Resolver is able to compare the paths of folders in a rename-rename conflict to the paths of folders that were deleted. All deleted folders that were in the path of a folder that is in a rename-rename conflict are filtered as they will not be handled in this step. Only after the rename-rename conflict is resolved can the deleted folders be handled.

This is done due to the fact that, the name of folders that are in a rename-rename conflict may be unknown to the other replica. In some cases the Resolver is unable to know beforehand what the folder is called in the other replica (and therefore its path). An example of this is the scenario where a deleted folder is inside a folder in a rename-rename folder which is itself also in a rename-rename conflict. Therefore, the Resolver is unable to successfully delete the folder in the other replica since the Resolver is unable to know the path the folder has. As a result, the deleted folders in this situation are filtered and will only be synchronized after the conflicts are resolved.

Then, the Resolver verifies if any of the deleted folders were deleted in both replicas. If this happened the folders can simply be deleted from both Directory Trees and continue the synchronization process, since the folders no longer exist in each replica.

After this is done, the Resolver can start synchronizing each of the deleted folders. To ultimately resolve a deletion the Resolver simply deletes the folder in the computer where the folder still exists and removes the pair's FolderStructs from each Directory Tree. To accomplish this, the first step is to discover the path of the folder that needs to be deleted and discover its FolderStruct inside the tree.

Next, the Resolver analyses the undeleted folder's FolderStruct to discover if this folder was previously marked as renamed or created in the other replica. If it was renamed, these two folders are in a delete-rename conflict and their FolderStructs are stored in an object that represents a conflict that will be handled in the future.

In case the folder is marked as created, it means that in the other replica the folder was first deleted and then created again with the same name. To resolve this, the FolderStruct of the deleted folder is simply removed its respective Directory Tree. Since the FolderStruct of the other folder is still marked as created the rest of the synchronization will be handled during the folder creation resolution. This way the created folder will simply be copied to this replica as if it had never existed before.

Finally, if no conflicts were detected for this pair of folders, the Resolver deletes the undeleted folder of the pair. Next, the FolderStructs of this pair of folders are removed from their respective Directory Trees. With these actions concluded the next deleted folder in the queue will be resolved.

When there are no more deleted folders to be resolved the Folder Deletion resolution is over and the Folder Renames resolution begins.

3.7.3.2 Synchronization of Renamed Folders

To synchronize a folder that was renamed, the Resolver fetches the current name from this folder and renames the corresponding folder in the other replica with this name. However, this process is usually not this simple since there are several special cases to detect and filter. The synchronization of folders that were renamed is illustrated in figure A.3 in page 92.

First the Resolver must find all conflicts or cases that can be ignored. To accomplish this the Resolver must detect all the folders that have been renamed. With this information the Resolver is able to verify if the same folder was renamed in both replicas. If this is indeed the case and the same folder was renamed with the same name in both replicas this modification can simply be ignored since the folders already have the same name. However, if they were renamed with different names instead, the folders are marked as a renamed-renamed conflict and are stored to be handled in the future during the conflict resolution phase.

After these cases are filtered, the Resolver will iterate through all of the renamed folders and will start renaming the corresponding folder stored in the other replica. To do this, it is required that the Resolver first discovers the path of the folder to rename, the path the folder will have after the renaming and the folder's FolderStruct.

Before the Resolver actually renames the folder it must first check if a FolderStruct with the path the folder will have after being renamed already exists. If a folder already exists in this path that means that since the last synchronization, the folder in the other replica has been deleted and then created with the name of the renamed folder. If this happened, the Resolver simply stops the synchronization of this folder and starts resolving the next folder in the queue since the folder will eventually be resolved during the folder creation resolution.

Finally, if none of the special cases described above were true for this folder the Resolver renames the folder. After the folder is renamed the Resolver will start the process again for the other renamed folders in the list. When no more folders remain to be handled the resolutions of renames ends and the resolution of created folders begins.

3.7.3.3 Synchronization of Created Folders

To synchronize created folders, the Resolver copies the newly created folder along with its contents to the other computer where the folder does not exist. However, it can happen that the same folder was created in both replicas with different contents stored within. The synchronization of created folders is shown in figure A.4 in page 93.

The first step when synchronizing created folders is to detect all folders which are marked as created. Then, the path from all created folders is compared to the path of all the folders that were already identified as being in conflict. If a path of a created folder is contained within the path of a folder in conflict, the created folder is filtered to be handled only after the conflicts are resolved.

Next, the Resolver iterates through the list of created folders and determines their respective path in the other computer where the folder will be created. After this is done, the folder, along with the contents within, is copied to the other computer recursively. The situation in which a folder already exists in the other computer is not considered a conflict since the content of the folders may differ. The contents of each folder are merged and a creation-creation conflict is only detected if the same file exists inside both replicas. All files inside the created folder that only exist in one of the computers are sent to the other computer.

After all folders and its contents have been copied to the other replica the folder creation resolution is concluded.

3.7.4 Folder Conflicts Resolution

After the deletions, renames and creations of folders have been resolved, the Resolver verifies if conflicts were detected. The conflicts that may happen during this phase are rename-rename conflicts and delete-rename conflicts. If conflicts were detected, a Windows form is loaded using all the information collected during the previous phases of the synchronization process. This form displays a brief description of the conflict along with its cause and all the information the user needs to decide on how to resolve it.

When shown the conflict's form the user must decide, based on the information displayed, which replica he wants to keep. The form presents information that is relevant to help the user deciding on how to resolve the conflict. For each conflict is presented the type of the conflict, the name and path of both modified

folders, the time at which each modification was performed and an icon that shows the choice made by the user regarding which modification he wants to keep.

For example, in the form presented in Figure 3.9, the user is shown a delete-rename conflict. He must choose if he wants to delete the renamed folder or if he wants to recreate the renamed folder in the replica where the folder was deleted. Likewise, in the rename-rename conflict shown the user must decide which of the names he wants to keep.

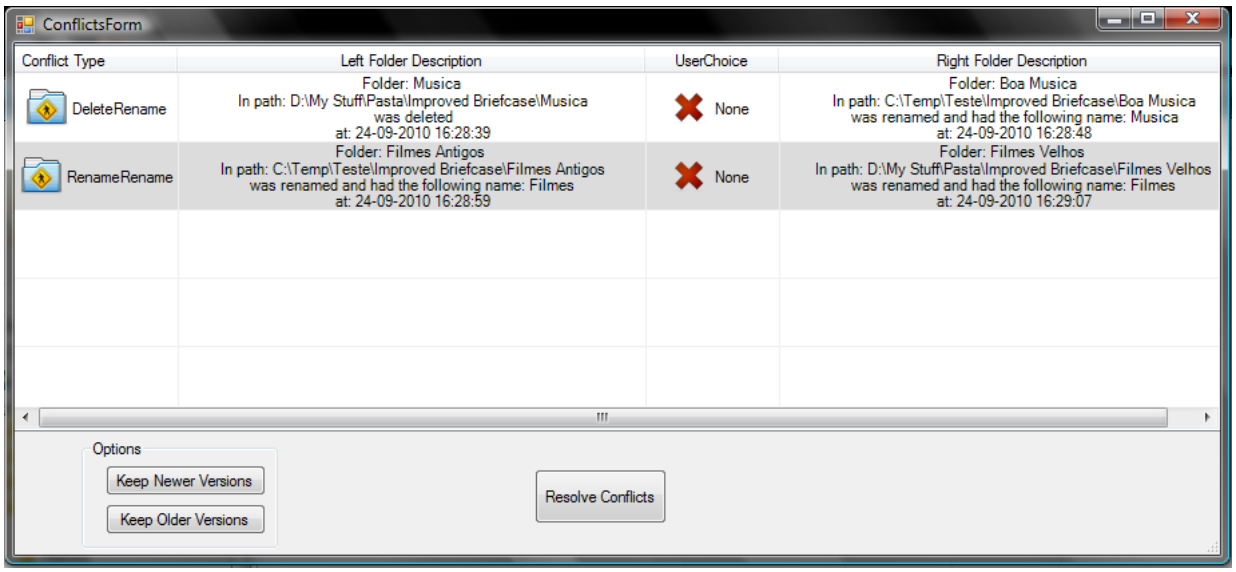

Figure 3.9: The form shown to the user when conflicts related to folders are detected. The figure displays a rename-rename conflict and a delete-rename conflict.

To do this the user may simply press the right mouse button on each conflict and choose to keep the right folder, the left folder or none of them. However, a conflict marked with none as the user's choice will not be resolved. Until the user decides on an option to resolve each folder conflict the Resolver will not start the file resolution phase.

The user also has the option of keeping all the oldest modifications or the newest modifications. When the user selects one of these options the Resolver automatically makes the decisions based on the timestamp kept for each modification.

After making all the decisions and pressing the "Resolve Conflicts" button the Resolver iterates through the list of conflicts and depending on the type of conflict, performs a different actions. Each of these actions will be explained in the following subsections:

3.7.4.1 Resolve rename-rename Conflicts

To resolve a rename-rename conflict, the Resolver takes similar actions to the ones taken during the renaming resolution. The Resolver finds what was the user's decision for each of this conflicts and ascertains the path of the folder that needs to be renamed. If the path exists locally the Resolver is only required to rename the local folder. However, if the folder is located in a remote computer the Resolver requests through the network that the other computer performs the renaming.

Also, the folders' FolderStructs are updated in each replica with the current folders' name and are marked as synchronized. In the end both folders in the two computers will have the same name.

3.7.4.2 Resolve delete-renames Conflicts

First, the Resolver checks whether the user decided to keep the renamed folder or the deleted folder. In case the user chose to delete the folder, the Resolver discovers the path of the folder that needs to be deleted.

If the folder is stored locally the Resolver deletes the folder. Otherwise, if the folder is stored in a remote replica the Resolver requests the folder deletion through the network. Also, the FolderStruct representing this folder must also be deleted from the Directory Trees of both replicas.

In case the user chose to keep the renamed folder the Resolver discovers the path to where the renamed folder must be copied. After this is done, it copies the renamed folder along with its contents to the other replica or from the other replica depending on the location of the renamed folder. The method used to copy the folder from one computer to the other is similar to the one used during the folder creation resolution.

3.7.5 Folder Conflicts Resolution Concludes and the Synchronization of Files Begins

After all the conflicts related to folders have been resolved, there are still some folders that have not been resolved. Namely the sub-folders of folders which were previously in conflict. These folders will now be resolved. To accomplish this, the Resolver simply performs the synchronization of deleted, renamed and created folders again. The folders that have already been resolved are ignored in this step because they were previously marked as synchronized.

When all folders' modifications have been synchronized and no more conflicts remain, the Resolver can start the synchronization of files. The synchronization of files is similar to the synchronization of folders. The main differences are that files are synchronized in four steps instead of three. First, the Resolver synchronizes deleted files, followed by renamed files, then the created files are synchronized and finally the Resolver synchronizes modified files. Through all these steps the Resolver collects and stores all the conflicts detected in order to display them to the user.

Each of the four steps will now be described with some detail.

3.7.5.1 Synchronization of Deleted Files

As with folders, the deletions of files are the first type of modification to be handled. The reason to this is that deletions must be handled before the renames or modifications in order to discover all the deleted-renamed conflicts and deleted-modification conflicts. This is important since these conflicts must be filtered and should only be resolved during the conflict resolution phase.

The Resolver first analyzes both Directory Trees and collects the FileStructs of all deleted files in each computer. Next, the Resolver compares the collected FileStructs and filters the cases in which the same file was deleted by the user in both computers. When this case is detected the Resolver simply deletes the FileStructs of these files from the Directory Trees. This is enough since the files no longer exist and there are no more references of them in the Trees.

Next, the Resolver iterates through all of the deleted files and finds the path where they are stored in each computer and the FileStructs that represent them. By accessing the FileStruct of the file which was not deleted and still exists in the other computer, the Resolver can verify the file's state. If the file is set as modified the two files are in a deleted-modified conflict. Likewise, if the file is set as renamed the two files are in a deleted-renamed conflict. In either one of these cases the conflicts are stored in a list of Conflicts along with important information to help the user resolve the detected conflict. The information will be displayed to the user during the conflict resolution phase. After the conflict is stored the Resolver can simply start resolving the next file in queue.

If no conflicts have been detected the Resolver only needs to delete the file in the computer where the file still exists and remove the files' FileStructs from the Directory Trees of both computers. With these actions concluded the next deleted file in the queue will be resolved.

When there are no more deleted files to be resolved the synchronization of deleted files is over and the Resolver starts synchronizing renamed files.

3.7.5.2 Synchronization of Renamed Files

Handling a file renaming modification is very similar to how the synchronization of renamed folders is performed. The Resolver must discover the current name of the renamed file in one replica and rename the unchanged file in the other. However, the Resolver must first verify if there are any conflicts or if the synchronization is really required.

First, the Resolver searches the Directory Tree from both briefcase for all renamed files' FileStructs. Next, with the FileStructs collected, the Resolver verifies if two correspondent files have been renamed in both computers. If they were renamed with the same name, both files are set as synchronized and ignored. However, if the files were renamed with different names they are in a renamed-rename conflict. The FileStructs of these files are stored and will be resolved eventually.

After the files in conflict are filtered, the Resolver goes through each renamed file, finds their paths and their FileStructs. The FileStructs of the files that were not renamed are analyzed to check if they are set as created. If this is the case the file was created in the other computer with the same name as the renamed file and needs to be resolved during the synchronization of created files. Therefore, the Resolver ignores these files for now.

Otherwise, the Resolver renames the file or requests the other replica to do it depending on the file's location. After the file is renamed the Resolver starts the process again for the other remaining renamed files. When no more files remain to be handled the resolution of renamed files ends and the synchronization of created files begins.

3.7.5.3 Synchronization of Created Files

To synchronize created files, the Resolver is only required to copy the created file to the computer where it does not yet exist. As with the synchronization of other types of modifications the Resolver iterates through the Directory Trees to collect all the files that are set as created.

Next the FileStructs of created files are compared to check if two files were created with the same name in the same path in each computer. In this case the files are in a create-create conflict and information is stored to be resolved later.

Otherwise, if no conflicts were found the created files are simply copied to the computer where they do not yet exist. To send a file through the network to another machine the Resolver reads the stream of bits from the file and sends it to the other computer. Although it is not currently implemented, the fact that files are sent as streams of bytes enables the Resolver to send only the bytes that differ from the two replicas. This would reduce the quantity of data sent when synchronizing file creations. This feature is marked for future work.

After this is performed for all created files the creation resolution is complete.

3.7.5.4 Synchronization of Modified Files

The synchronizing of modified files is handled exactly in the same way as the synchronization of created files. The Resolver finds all the modified files in the Directory Tree and if no conflicts have arisen the modified file is copied to the other replica.

If a modification conflict was detected the correspondent object is created in order to help the user decide which modifications he wants to keep.

3.7.6 File Conflicts Resolution

The resolution of files' conflicts is identical to the one performed in the case of folders. A windows form is created using all the information collected during the previous phases. This form displays all the previously detected conflicts, their cause and a description of the modification performed in each computer.

3.7.6.1 Resolve rename-rename Conflicts

When a rename-rename conflict is detected the user is shown the two different names given to each replica and must decide which one he wants to keep. The resolution process is exactly the same as the one used in the case of folders which is explained in section 3.7.4.1. Therefore, the process will not be explained here.

3.7.6.2 Resolve Delete-Renames Conflicts and Delete-Modifications Resolutions

In these cases the user must decide if he wants to keep the renamed/modified file or if the file is unnecessary and must be deleted. Again, the resolution process is exactly the same as the one used in the case of delete-renamed conflicts for folders which is explained in section 3.7.4.2. The process will not be explained here.

3.7.6.3 Resolve Creation Conflicts and Modification Conflicts

When a file with the same name is created in each machine the user must decide which of the files he wants to keep. This is also true in the case the same file is modified in both machines. In both of these cases when a user decides which file he wants to keep, that file is simply copied to the other computer while overwriting the other version.

However, to better help the user decide which version of the file he wants to keep, Smart Briefcases, has several integrated diff engines. These, allow the user to compare the contents of files and detect differences between them. This is one of the main features that differentiates Smart Briefcases from other available solutions. The Diff Engine Modules containing the diff engines are explained in section 3.8.

The user also has access to external applications that help him see the differences between files and in some cases allows him to reach a better resolution by merging files' contents. Currently, the only external application used by Smart Briefcases is WinMerge which allows viewing and merging only plain text files. However, other applications are very easy to include.

3.7.7 Conclusion

After all modifications have been resolved the Resolver iterates through the DirectoryTrees from each briefcase and marks all structures that represent folders and files as synchronized. Next, all the files and folders which are in a conflict and have not been resolved are set as not synchronized. This way, the next time the synchronization process is started the conflicts are detected again and shown to the user who may resolve them.

Finally, if one of the briefcases is stored in another computer the Resolver requests that the remote computer marks its Directory Tree as synchronized. With these actions performed, the synchronization process is concluded and the synchronized briefcase are identical.

3.8 Diff Engine Modules

When a conflict is detected between two files, to help the user decide which file he wants to keep, he has the option to view the content of each file side by side. When this option is selected a window opens displaying each of the files next to each other. Colors highlight the differences between the lines where the files differ.

To detect the differences between files, a difference engine is used. This engine compares two text files and returns an ArrayList containing all the differences between them. This information can be used to display the modifications to a user and help him resolve them faster.

The difference engine was downloaded from the Code Project website [33]. This algorithm was used for several reasons: It is written in C#, which helps the integration with Smart Briefcases, it is generic and

reusable, it gives correct results even for large data sets and it has lower memory requirements compared to other existing solutions.

This engine works very well for plain text files. However, one of the goals of Smart Briefcases is that it should be able to provide this kind of information to users for several file types, namely office documents and presentations. Also, it should be extensible so that it is relatively simple to add other engines to compare new file types. How these objectives are accomplished is explained in the following sections.

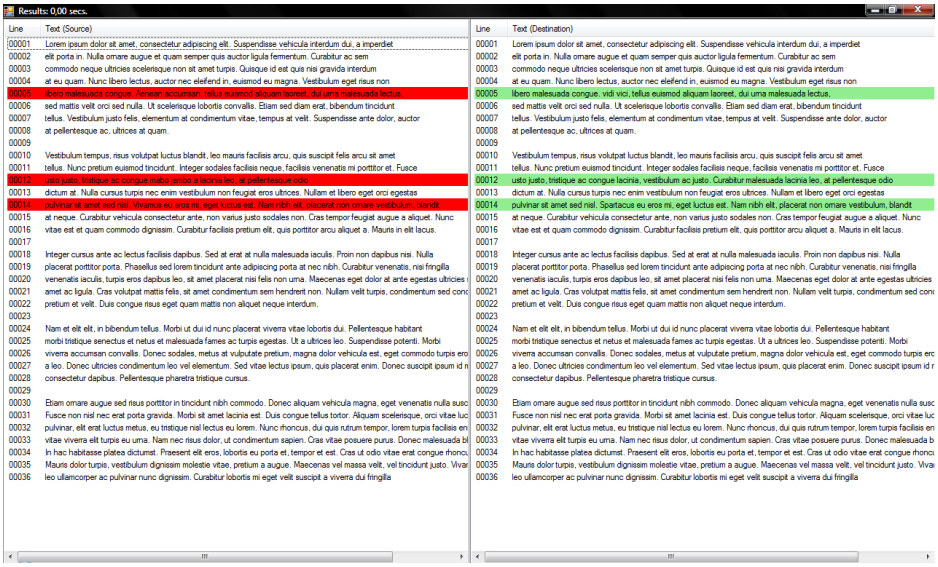

Figure 3.10: The difference form shows the comparison between two plain text Files. The colors show that line 5, line 12 and line 14 are different in each replica.

3.8.1 How to get information from other file types

Files that are not composed by plain text cannot be simply read in order to be compared and display the differences to the user. In order to accomplish this, two things are required: the use of an API that is able to read content from the file and a difference engine that is able to compare the files. Also, inside some binary files there can be images, videos, tables and other content that is not easily interpreted by difference engines.

In Smart Briefcases the diff process was implemented for Microsoft Word and Power Point files. However, only the text is compared. Although it would be possible to compare the formatting of the files, or other objects inside, it would require a lot of time in order to implement an efficient difference engine that would always return correct results.

In order to perform a diff visualization for Microsoft Word and Power Point files, Smart Briefcases fetches the text from files and sends it to the difference engine. Next, the engine marks the differences between the replicas. Finally, Smart Briefcases creates the form in which the differences between the files are shown.

3.8.1.1 Read Content from Microsoft Office files

Microsoft Word and Power Point files are not plain text files. Therefore, the process of reading their contents is not so simple as creating a stream and get data from it. Typical Differencing software is not able to read the content from these files since docx and pptx files are, in fact, zip files containing xml files that describe the structure and content of each file. Fortunately, Microsoft has an API available that allows developers to read, and modify Office files.

Therefore, Smart Briefcases has separate modules to read the contents of files with different file types. Currently there are only two modules: a module that reads the text from a Microsoft Word file; and a

module that gets the text from each slide in a Power Point presentation.

Both modules use the Open XML SDK which contains functions that allow a program to access the structure and content from a Microsoft Office document. Using these functions, Smart Briefcases is able to detect the xml file inside a docx or pptx that contains the text and fetch it. The text from each file is written in a temporary plain text file which can be read by the difference engine.

To help the user easily detect the location of differences in files, when the text is written in plain text files the module separates the content based on the semantic properties of the file type. For example a Word file is divided in paragraphs while a Power Point file is divided in slides. The result is shown in Figure 3.11.

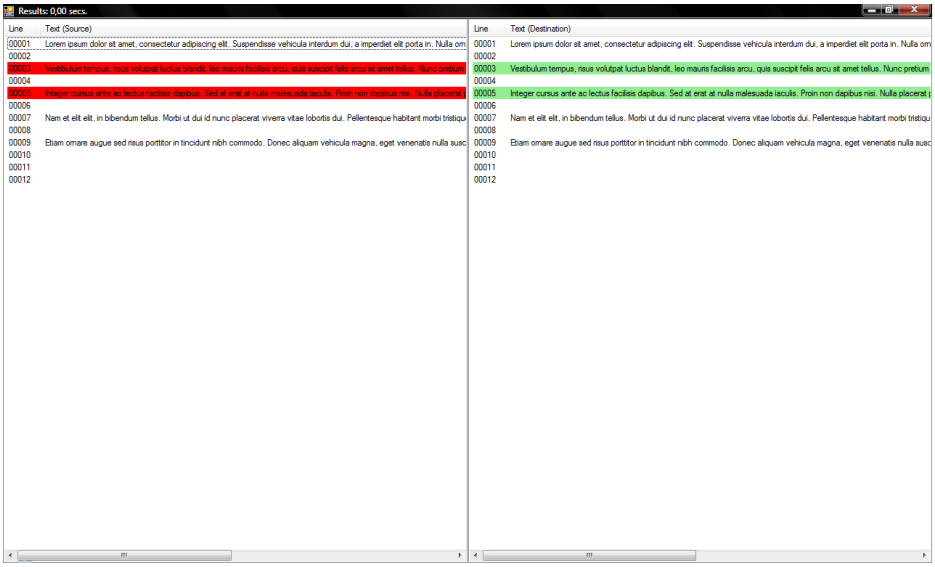

Figure 3.11: The difference form shows the comparison between two Word Files. The colors show that the second and third paragraphs differ in their content.

3.8.2 Extensibility: How to support other file types

To allow Smart Briefcases to be extensible to other file types it is easy to add other difference engines to the system since each one is completely independent from the rest of the code. When a user requests a comparison between two versions of the same file, Smart Briefcases detects the file's type and depending on the type, it calls a different module to perform the comparison. Currently, by default the plain text module is called if the file type is unknown to Smart Briefcases.

With the creation and integration of one additional module it is easy to allow Smart Briefcases to support the comparison between new file types. However, depending on the complexity of the type of files a developer wants to compare, there are different ways of providing this additional functionality.

3.8.2.1 Extensibility: Add a module to compare files based on text

If the file type to be added is based on text a developer just needs to create a new class that is able to read the text from the binary file and provide the text to the difference engine already built into Smart Briefcases.

For example, imagine a file that is encoded in a rich text format (.rtf). A developer is interested in adding to Smart Briefcases the functionality to provide the differences in text between two rtf files. He simply needs to create a function that receives a rtf file and creates a temp plain text file with the text read from the rtf file. Then, the developer provides the Smart Briefcases' difference engine with the plain text files created from both versions of each rtf file.

With only this information Smart Briefcases is able to detect the differences in both files and present them to the user the same way it presents the differences between two plain text files.

3.8.2.2 Extensibility: Add a module to compare complex binary files

In this case the developer needs to create the module for the specific file type and add a new difference engine that is able to compare content more complex than text e.g. images, video, etc. He also needs to add some controls to the interface in order to present the differences to the users. If the developer is able to implement the difference engine and the interface controls he simply has to integrate these two components with the File's conflict form, in order to invoke his difference engine whenever conflicts happen between the newly supported file type.

3.8.3 External Differencing Tools

Besides the difference engine present in Smart Briefcases, it is possible to integrate other applications that compare files from different replicas. This option is provided by Smart Briefcases in order to better help the user resolve conflicts by providing more functionality found in more mature applications. One example of functionality that, currently, is only offered by using an external tool is the possibility to merge files. This allows a user to achieve identical versions of a file without losing any information.

Currently, the only application that can be called through Smart Briefcases' interface is WinMerge. WinMerge, is an Open Source differencing and merging tool for Windows. It is able to compare files, presenting differences in a visual text format that is easy to understand and handle. Also, it can be used to merge files and achieve an identical version. Unfortunately, WinMerge only works for plain text files.

Several other programs were considered as possible options to be integrated with Smart Briefcases. However, no other programs were found that were free, could be instantiated with arguments through the command line and provided comparisons between files that were not plain text files. However, if an application with this requirements is found, it can be easily added to Smart Briefcases by adding the option to the conflict resolution interface and by implementing a method that executes the application.

3.9 Communication Module

Each instance of Smart Briefcases in each computer can invoke and be invoked by other Smart Briefcases running in different machines. This is used to share information between replicas and ultimately achieve data synchronization.

Before a synchronization pair can be formed between two replicas, each instance of Smart Briefcases in each computer must send their IP address and Port to each other. With this information each instance of Smart Briefcases is able to freely communicate with its pair. During the synchronization process a replica simply computes the modifications that need to be performed and invokes the required functions in its pair to "make" the data identical.

The communication module contains the interface and the implementations of the functions that will be remotely invoked from other replicas. Also, it is responsible for opening the communication channel from which messages from other replicas are received. The communication can only be performed between replicas that are connected to the same local area network.

3.10 Graphical User Interface

The graphical user interface of Smart Briefcases is a file synchronizer that allows a user to synchronize his shared briefcases and resolve conflicts in case they occur. During the normal execution of Smart Briefcases

the graphical user interface is simply a tray icon. This was done to ensure that the interface would not be distracting to the user.

A menu can be invoked by right clicking the tray icon. This menu, that is displayed in Figure 3.12 presents several options to the user. The first option allows a user to create a new briefcase in a chosen location. After the folder is created it starts being monitored by Smart Briefcases. The next option allows a user to synchronize shared folders by selecting one of two options. The first option summons a list of all the briefcases located in the current machine and the user can select an individual pair to be synchronized. The other option goes through all of the synchronization pairs and synchronizes them all.

The third option is used to call a form that was used during the implementation of Smart Briefcases. The final option exits the application. When this option is selected all the data and structures used by the application are saved to files and the program terminates.

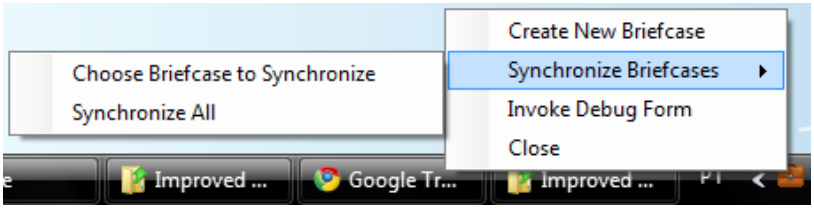

Figure 3.12: The menu that is presented when a user right clicks Smart Briefcases' tray icon.

During the synchronization process the tray icon also provides a visual representation of its progress. It informs the user of the task that is being performed and the percentage that has already been performed of the entirety of the process. This visual aid is provided by a balloon tip that comes out of the tray icon. An example is shown in Figure 3.13. The goal is to keep the interface hidden from the user while informing him of what is being performed in the background.

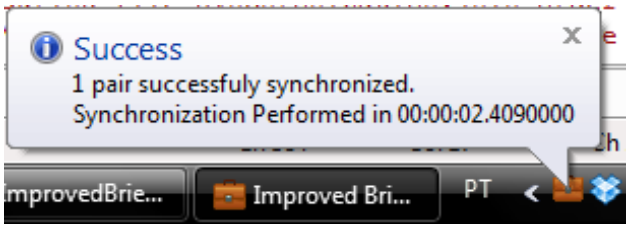

Figure 3.13: The dialog that informs the user of what is being done during the synchronization process without interrupting other tasks.

A conflict resolution window is displayed to the user only in case conflicts between two replicas are found. The user can still choose to close the window without resolving anything. The next time he tries to synchronize the same pair of briefcases the same window will be shown displaying the unresolved conflicts. The conflict resolution window displays information required by the user and allows him to decide which modification he wants to keep in order to solve the conflicts. A conflict resolution window is displayed in Figure 3.14

3.11 Advantages and Disadvantages of the technologies used

3.11.1 Advantages and Disadvantages of using a File System Watcher

The use of the File System Watcher component has several advantages but it also brings some overhead and creates some delays during the execution of Smart Briefcases. The reasons why this component was used and some of its advantages are explained below.

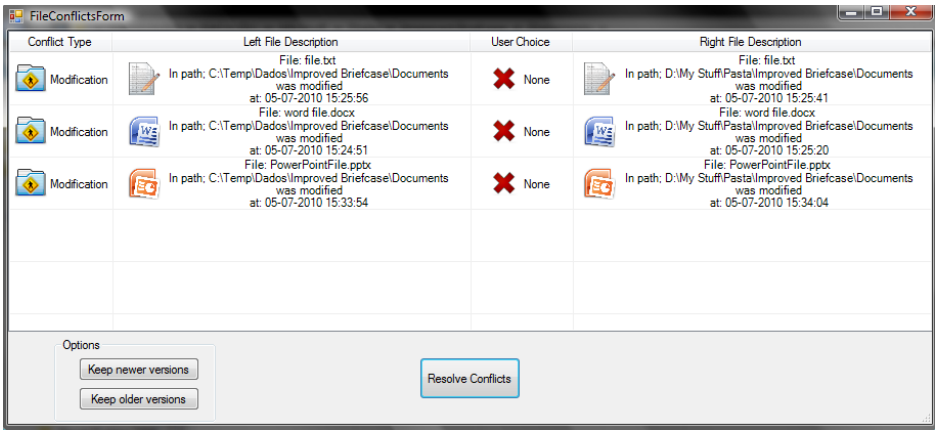

Figure 3.14: The conflict resolution window.

3.11.1.1 Advantages

Since the file system watcher is part of the .Net Framework it is easily integrated with an existing project. The developer needs only to use the FileSystemWatcher Class and create handles to several events that will be called every time a file or folder is modified/created/deleted. This handles are very easy to use and provide all the information needed for each event risen.

Furthermore, the component is efficient since it avoids the need for the system to constantly poll the watched directory for changes, and there is no time lapse between scans of the directory.

The alternative systems researched were difficult to use and integrate with a Visual C# project and had several limitations that far outweighed the gain. Some of these solutions have to be integrated using extensive unmanaged code and others do not have documentation on how to use them.

Therefore, the FileSystemWatcher control seemed the best and easiest solution to use.

3.11.1.2 Disadvantages

Although the FileSystemWatcher control works relatively well in most cases it has several limitations:

1) The Created event fires very early when a file is getting copied into the watched directory [1]. Therefore, we get an error "The process cannot access the file because it is being used by another process." if we try to open the file from the Created event handler. This makes it impossible, for example, to create an hash to newly created files or files that were copied to the folder.

2) The FileSystemWatcher stores the unhandled events in an Internal Buffer. The internal buffer's default size is 8192 bytes (8 kilobytes). The events are stored in this buffer until they are passed to the Win32 APIs to be handled. However, if there are too many concurrent modifications to the watched folders the buffer fills up and eventually overflows [2]. This can cause the FileSystemWatcher to lose some of the modifications performed. It is vital that this does not happen during the synchronization process or when a user is modifying his folders.

Two actions were taken to prevent the FileSystemWatcher from overflowing. The first action was to create a delay of 10 milliseconds every time a file or folder is created and 40ms each time a rename is performed during the synchronization process. With these delays the file system watcher has time to view the modification and update the Directory tree without new events being triggered. The times chosen were the values for which all tests performed were successful.

[1] see http://msdn2.microsoft.com/en-us/library/system.io.filesystemwatcher.created.aspx
[2] see http://msdn2.microsoft.com/en-us/library/system.io.filesystemwatcher.internalbuffersize.aspx

It is important to note that this delay is only applied when the pair of briefcases is stored in the same machine. When the synchronization is performed in remote replicas the delay created by the communications is enough to prevent the buffer to fill up.

The other action was to increase the size of the internal buffer. This size can be established by the developer. However, the documentation mentions that increasing the size of the buffer is expensive, as it comes from non paged memory that cannot be swapped out to disk. Still, the FileSystemWatchers used by each File System Monitor are used with the Internal Buffer with size 12 kilobytes. This specific value was chosen taking into account what is written in the documentation. It is said that the size of the buffer should be a multiple of 4 kilobytes for better performance. 12 kilobytes is the next multiple of 8kilobytes and was the minimum possible value that would adhere to these rule.

These two actions increase the time the synchronization process takes to finish and have a certain cost to the system. However, this cost is required in order for the system to work correctly.

3) The Changed event fires multiple times when either files or folders are modified. This creates pointless events that fill the internal buffer needlessly. Fortunately in Smart Briefcases all modification events performed to folders can be ignored which minimizes the problem. However, modifications to files cannot be ignored and in most cases each modification triggers two changed events in the FileSystemWatcher. Unfortunately, there is no known solution to resolve this problem but it does not cause problems or much delay during execution.

Chapter 4

Implementation and Evaluation

This chapter describes some relevant aspects of the development process of Smart Briefcases followed by a presentation of the results obtained during the evaluation of the submitted solution. Section 4.1 focuses on the technologies used to implement each module that constitutes Smart Briefcases, why they were chosen and what they provide to the solution. Section 4.2 provides a quantitative and qualitative evaluation performed in order to verify the efficiency and ease of usage of the Smart Briefcases system.

4.1 Implementation

Smart Briefcases was implemented using Microsoft Visual Studio 2005, C# 2.0 and the 2.0 .Net Framework. The submitted solution was tested under Windows XP, Windows Vista and Windows 7.

The File System Monitor and the Drive Monitor modules use a .Net control named File System Watcher. This module is capable of monitoring a certain folder and triggering events whenever a modification is performed in said folder. Using this functionality, Smart Briefcases is able to store information regarding modifications performed in briefcases. Figure 4.1 in page 61 shows how the file system monitor receives information regarding modifications performed to files and requests that the information be stored. Likewise figure 4.2 in page 62, shows how the Drive monitor receives information regarding the creation of a new briefcase.

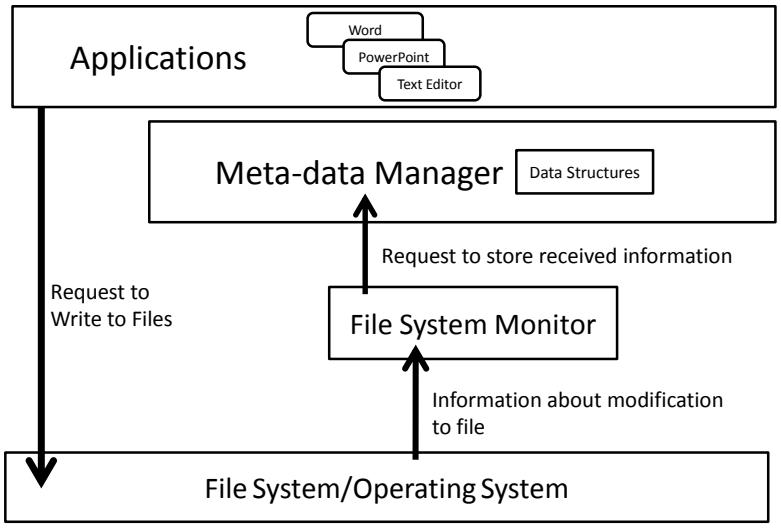

Figure 4.1: The figure shows how the file system monitor receives information regarding modifications performed to files and requests that the information be stored.

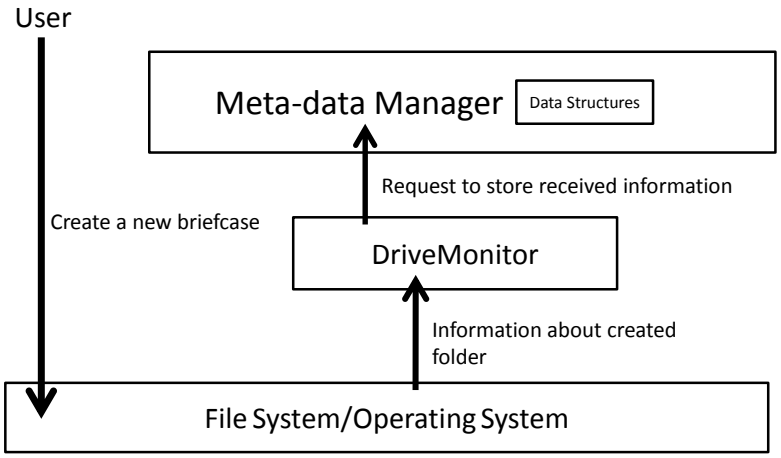

Figure 4.2: The figure shows how the Drive monitor receives information regarding the creation of a new briefcase.

The main difference between the File System Monitor and the Drive Monitor is that in the former the File System Watcher has the responsibility of watching a certain briefcase. On the other hand, the file system watcher inside each Drive Monitor, observes modifications performed on each drive currently mounted in the user's machine. For more details on how these modules work see Sections 3.3 and 3.5.

The drive detector module uses native code that was downloaded from the Code Project website. [1] The drive detector receives signals from the windows operating system that concern the mounting and unmounting processes of USB flash drives. The signals in question are:

- DBT_DEVICEARRIVAL - sent after a device or piece of media has been inserted.
- DBT_DEVICEQUERYREMOVE - sent when the system requests permission to remove a device or piece of media. Any application can deny this request and cancel the removal.
- DBT_DEVICEREMOVECOMPLETE - sent after a device has been removed.

When one of these signals is received an event is triggered which allows Smart Briefcases to start or stop monitoring a USB flash drive, for example. A detailed explanation of the usage of the Drive Monitor is found in section 3.6.

The graphical user interface was implemented using Windows Form controls. The exception is the conflict resolution form which uses a custom control called ObjectListView [31]. This control is a C# wrapper around a .NET ListView. The ObjectListView was chosen since it is much easier to use than any control found inside the .Net framework. Additionally, it gives much more options of customization, more functionality and allows the conflict information to be displayed in an easy to understand way. For example, when a certain file is in conflict an icon is displayed to help the user identify the type of the file. By using a simple List view, implementing this functionality would take a lot of time and work.

The Communication Module was implemented using .Net Remoting, which provides several mechanisms of remote method invocation found in the .Net Framework. This module allows the transfer of files or updates between remote replicas.

To implement the difference engine modules that fetch the text from .docx and .pptx files the Open XML SDK for Microsoft Office was used. This SDK provides an API that allows a developer to create and edit Microsoft Office files programmatically. With this functionality Smart Briefcases is able to get the text from Office files and display a comparison between two distinct files to the user.

Figure 4.3 in page 63 details the technologies used in the implementation of Smart Briefcases.

[1] http://www.codeproject.com/KB/system/DriveDetector.aspx

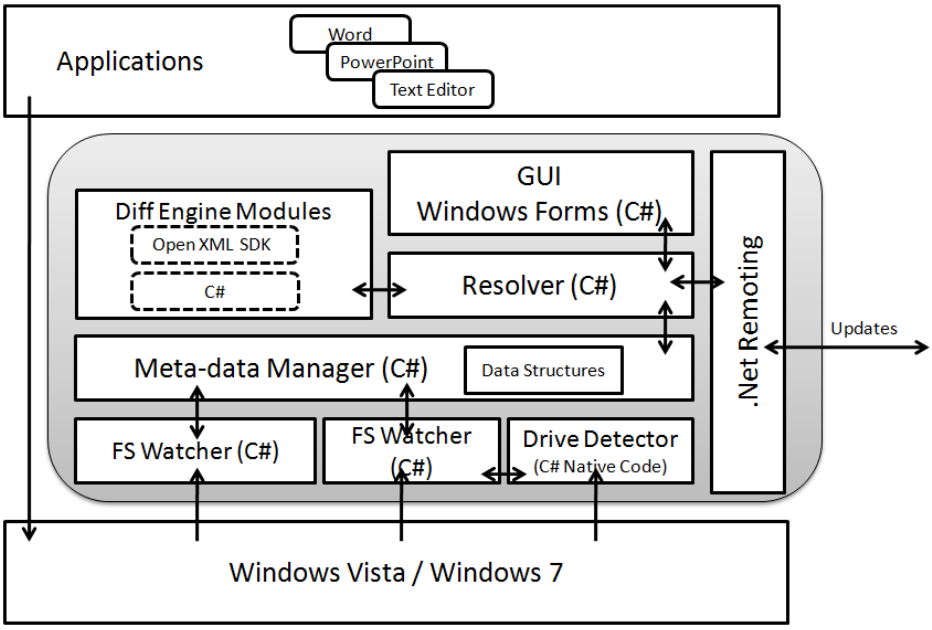

Figure 4.3: The modules that compose Smart Briefcases and the technologies that are used.

4.1.1 Why was C# chosen as the main programming language?

There were several reasons why C# was chosen as the main programming language. These reasons are detailed in the list below:

- Smart Briefcases is a system built for the Windows operating systems. During its execution the system must perform several file system operations. For example, files and Folders must be renamed, deleted, created or moved. The .Net Framework provides an extensive library that supports the previously mentioned functionality.
- Some functionality requires the use of unmanaged code and C++ functions. .Net Framework allows unmanaged code or Windows native functions to be used transparently with C# code. Without the possibility of including native code some functionality would be very difficult to implement if not impossible.
- When conflicts occur between Word or PowerPoint files, Smart Briefcases is able to display the differences in content between said files. Microsoft provides the OpenXML Office SDK to allow developers to read and modify the contents from Office files. However, currently this SDK is only available in C# which prevents the integration of the SDK with projects that are not built using the .Net Framework.

For these reasons the use of C# as the main programming language seemed logical as it greatly speeds up and eases the development process.

4.1.2 Implementation Problems

During the implementation of Smart Briefcases there were some functionalities that were not implemented. This section details what these functionalities were, why they were not implemented and what solutions were used instead.

4.1.2.1 Creation of briefcases through Windows context menus

Since the start of Smart Briefcases' implementation it was decided that a user should be able to create a briefcase in any folder, by pressing the right mouse button, accessing the context menu and selecting

an option that would allow the user to create a new briefcase. Figure 4.4 in page 64 shows how this works for Microsoft's Briefcase. The described functionality is similar to how Microsoft's Briefcase works. Unfortunately, in Smart Briefcases this functionality was not implemented.

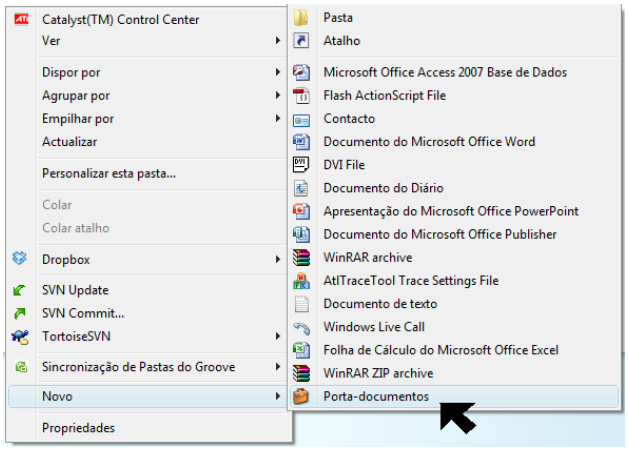

Figure 4.4: The figure shows how a Microsoft's briefcase can be created through the context menus of Windows.

In order to create a new context menu entry it is required either the use of unmanaged code or the use of the recently released Microsoft .Net Framework 4.0. The implementation of context menu entries using unmanaged code is extremely complex, requires the modification of the device's registry and the creation of a great number of C++ classes which code is not sufficiently well explained in the literature.

The best approach would be to use the .Net Framework 4.0 which allows to safely create this functionality using C#. However, this framework has only been released very recently and requires new tools such as Microsoft Visual Studio 2010. This fact was discovered when Smart Briefcases was almost finished. Due to this reason this feature was assigned as future work.

However, an alternative was implemented in order to let users create briefcases where they require. An option was added to the menu that pops up when a user right clicks Smart Briefcases' tray icon which allows a user to select the path in which a new briefcase is to be created. It works well and is a viable solution until the context menu entry is added.

4.1.2.2 Adding icons to files and folders inside a briefcase to reflect their current state

Some file synchronizers, such as Dropbox [10], modify the icons of shared folders and files in order to reflect their state. They could be for example, synchronized or not synchronized.

This functionality was implemented in Smart Briefcases using unmanaged code. Using this code the icon of a folder or file is changed to an icon of our choice. This is the same code that is used to add the briefcase icon to briefcases.

However, when a briefcase is copied to another location in order to create a synchronization pair and an icon of a folder or file inside the copied briefcase changed, instead of getting the chosen icon, the icon of the file or folder would become the default icon that it had before. Several approaches were used to resolve this problem and none worked.

It seems that when a briefcase is copied to another location, Microsoft Windows does not update something that has to do with the new location of files and folders. Therefore, when the icons change they revert to the original icon. No solution was found to solve this problem. Due to this, the functionality was marked as future work.

4.2 Evaluation

In this section we present the results collected from the evaluation of Smart Briefcases. The final build of Smart Briefcases has been evaluated in four different categories:

- **Memory Usage:** As explained in section 3.4 in Smart Briefcases data structures store meta-data that represents each file and folder. The information is required to perform the synchronization process. It is important that the memory occupied by Smart Briefcases is kept to a minimum.
- **Performance:** The synchronization process must be fast and efficient. This means that it should be able to synchronize two replicas in the least amount of time possible.
- **Bandwidth Usage:** The amount of data propagated when synchronizing two replicas must also be kept to a minimum since a user will probably be interested in replicating large batches of modifications throughout his devices.
- **Ease of Use:** Smart Briefcases must be simple to use and provide visual aid detailing, for example, in which step of the synchronization process it is on or what files are in conflict. Also, conflict information must be presented in a way that does not confuse the user and helps him understand exactly how to resolve each conflict.

These are the categories in which Smart Briefcases was evaluated. The results are presented in the following sections.

4.2.1 Testing Environment

The evaluation of Smart Briefcases' efficiency, speed and ease of use, was performed in two different machines connected through a wireless network. The specifications of these machines are detailed in Figure 4.5.

All tests were performed using the same build of the Smart Briefcases application. In order to create this build a setup file was generated using Microsoft Visual Studio. Afterwards, the setup was used to install Smart Briefcases in each testing machine. The general behavior of the application in all machines was identical with the exception of some visual elements in the graphical interface that change depending on the operating system.

4.2.2 Memory Usage

One of the evaluation parameters of Smart Briefcases that was focused on, was the amount of memory that the application uses during execution. It is important for any application to have a minimum memory footprint.

Smart Briefcases keeps meta-data for each file and folder that is stored in each briefcase. This information is crucial in order to store modifications performed, synchronize replicas and detect conflicts. Since each new file or folder is represented by a new node, that is inserted in the Directory Tree that stores meta-data, it is important that each of these nodes occupies a small amount of memory (see Chapter 3 for a detailed description of this process). This is important since it is expected that a user might store thousands of files inside a single briefcase.

Additionally, the amount of memory used by Smart Briefcases is also related to the number of File System Monitors and Drive Monitors that are waiting for events. Each File System Monitor and Drive Monitor keep their own structures. The number of File System Monitors increases with each briefcase created. The number of Drive Monitors increases with each hard drive that is mounted in the machine where the application is running.

In order to evaluate the amount of memory that is being used by Smart Briefcases, the system was executed in the two testing machines. Then, a synchronization pair was formed between two briefcases

Machine 1 – ASUS notebook
Processor: Intel® Pentium Core 2 DUO P8600 2.40 Ghz
RAM: 3070 MB DDR2-800Mhz
Hard Drive: 500 GB 5400 RPM
Graphic Card: ATI Mobility Radeon HD 3470 1GB GDD2
Network Card: Wireless LAN 802.11n
OS: Windows® Vista Home Premium 32-bit
Machine 2 – Desktop
Processor: AMD Athlon 64 3200+, 2.0 GHz
RAM: 1024 MB DDR2-400Mhz
Hard Drive: 200 GB 7200 RPM
Graphic Card: NVIDIA Geforce 6200 GT
Network Card: DWL-G510 Wireless LAN card 802.11n
OS: Windows® 7 Home Premium 32-bit
Wireless Router
ADSL 2/2+ Wireless Gateway Asus WL-600G

Figure 4.5: The specification of the two machines used during the evaluation process.

stored in both machines. Finally, an increasing number of files and folders were created inside each briefcase while synchronizing the pair.

Creating files and folders inside a briefcase are the only operations that directly affect the memory used by Smart Briefcases. When a file or folder are created a new object representing the file or folder is created and stored inside the respective Directory Tree. When performing other modifications such as renames, deletes or modifications the object of the file or folder is updated which does not increase the memory used by the system. Every time a certain number of files and folders had been created in each briefcase the value of occupied memory by Smart Briefcases was observed and recorded.

In order to measure the amount of memory that is being used at a certain time, a Microsoft program called Process Explorer [2] was used. Process Explorer allows a user to check the runtime status of a specific application in real time. For example, it allows one to analyze CPU usage, Memory usage, threads that are running, TCP ports that are in use, I/O, etc.

Therefore, a large number of files were added to the briefcases in each replica and values were measured and collected by Process Explorer and Windows Task Manager. The collected information is displayed in Figure 4.6 in page 67.

As can be observed, there is a noticeable increase in used memory as the number of files inside the briefcase increases. However, even while storing 2048 folders and 16384 files, the memory threshold is still at reasonable values. Especially, when compared with other file synchronizers studied in chapter two.

Also, since each briefcase has its own Directory Tree structure and File System Monitor, another similar test was performed. However, this time the evaluation was performing with two pairs of briefcases (two briefcases in each computer). The briefcases contain the same number of files and folders within. This allows one to analyze the cost of having several different briefcases in the same machine. The results are shown in Figure 4.7.

The increase in memory used by Smart Briefcases when two distinct briefcases are stored in each computer is noticeable but is minor. When storing approximately 16384 files and 4096 folders the difference between storing one briefcase and two briefcases is simply 1.8MBs in machine 1 and 1.2MBs in machine 2. The difference is negligible. Therefore, a user is able to store several briefcases without being greatly

[2] http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

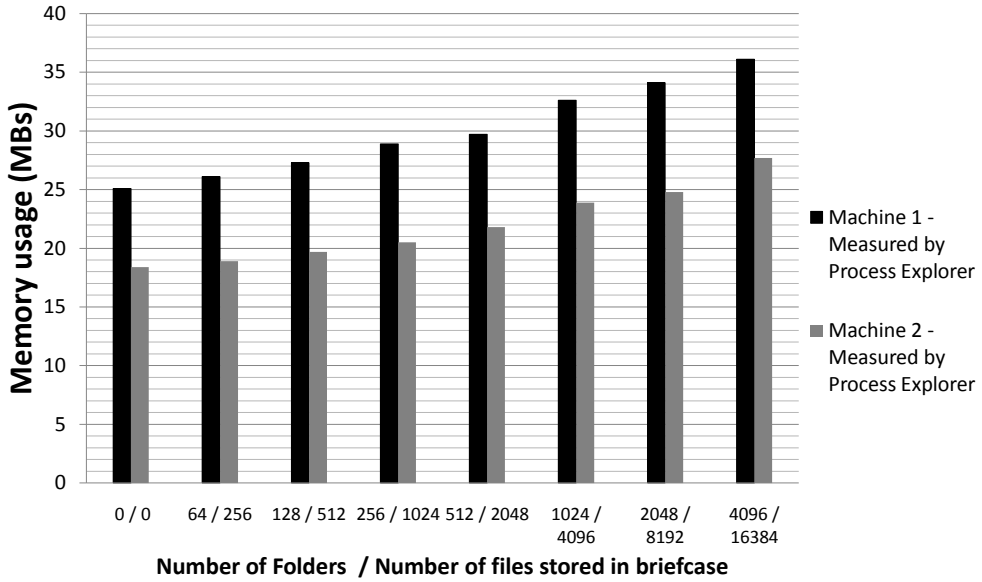

Figure 4.6: The relation between the increase in memory used by the application and the number of files and folders stored within a briefcase.

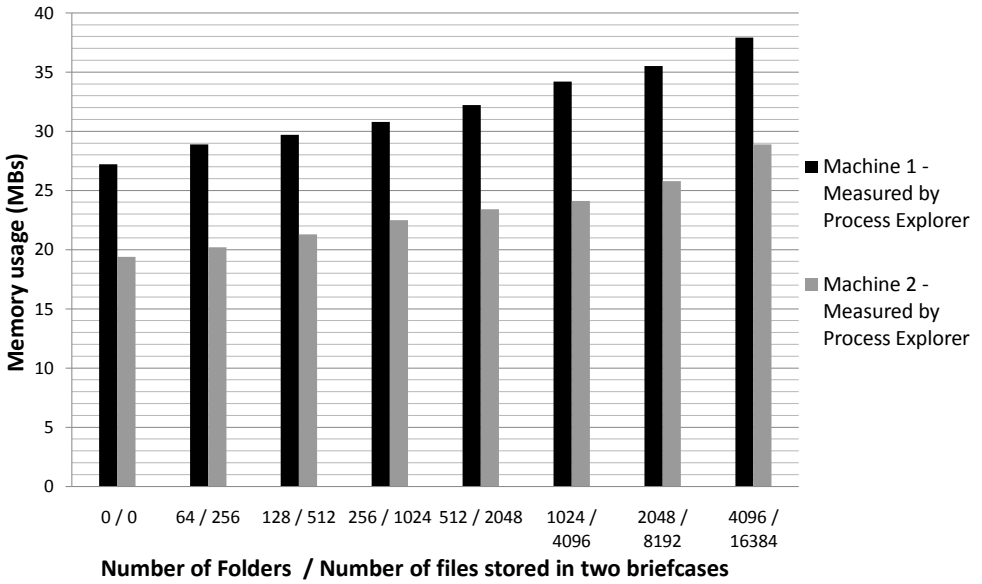

Figure 4.7: The relation between the increase in memory used by the application and the number of files and folders stored within briefcases.

penalized in memory usage.

4.2.3 Performance

The experiments described in this section evaluate the speed of the synchronization process. It is important for this process to be the fastest possible since a user may need to synchronize large batches of modifications at a time. Therefore, the time the system takes to synchronize newly created files, deleted files and renamed files was measured separately.

In Smart Briefcases, synchronizing modified files is the same, w.r.t. performance, as synchronizing created files. As with newly created files, when a file is modified and needs to be synchronized, the file is propagated as a whole. This is a drawback when comparing Smart Briefcases to other solutions that only

propagate the contents that were modified, which greatly decreases the time of propagation and the amount of data transferred between replicas. However, this feature can be integrated in Smart Briefcases and is currently marked as future work.

In order to evaluate the speed it takes to synchronize replicas, it was necessary to form a pair between two briefcases, both locally and remotely. Next, several modifications are performed to one or both of the replicas. Finally, the synchronization process is performed while measuring the time it takes to complete the process. This process is repeated ten times for each test while recording the time the process takes to complete. With the collected results an average was calculated. The average was used to created the graphs displayed in the sections below.

Also, it was decided to measure the cost of employing file system operations, such as file and folder creation, when Smart Briefcases is monitoring a folder. Since Smart Briefcases instantiates a File System Monitor that constantly watches a folder and stores the modifications performed within, it is possible that there is a slight delay when performing some of these operations. Therefore, the delay of creating files and folders inside a briefcase was compared to performing the same action in a normal Windows directory.

4.2.3.1 Speed of the Synchronization Process - File and Folder Creations (Empty Files)

The first evaluation performed, tests the speed of synchronizing files and folders created within a briefcase. This means that since the pair of briefcases was last synchronized new files and folders have been created inside one of the briefcases. Therefore, the newly created files and folders need to be propagated to the other computer. The size the files occupy directly influences the speed of propagation, and the bigger a file is more information needs to be sent from one computer to another, the first batch of tests are performed using only empty files. This way, it is possible to test the time of propagation without the delay of transferring the content of files through the network. This test shows the lower limit of the time taken to synchronize a pair of briefcases in which new files and folder were created.

Figure 4.8, shows the time taken to synchronize the creation of an increasing number of files and folders. The synchronization is performed between two remote briefcases located in different computers. The synchronization of created empty files through the network is reasonably fast since creating 1000 folders and 2000 files takes only 14.485 seconds in average. When synchronizing files that have content within, it is expected that all the time that exceeds this 14,485 seconds is spent on propagating the contents within the files.

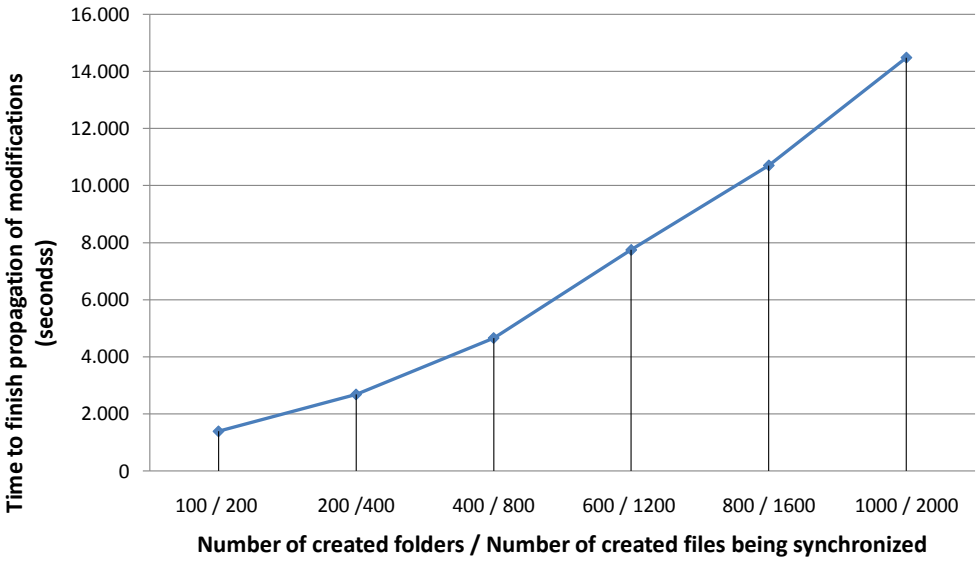

Figure 4.8: The graph shows the time it takes to synchronize an increasing number of folders and files.

It is important to compare Smart Briefcases to other available solutions. For this reason the same test was performed using Microsoft's Briefcases. In this case an increasing number of folders and empty files is created in the shared folder and is synchronized with its respective briefcase stored in the other computer.

Figure 4.9 shows the comparison of the time taken to synchronize the folders and empty files using Smart Briefcases and Microsoft's Briefcase. As can be observed in the figure, Smart Briefcases takes a lot less time when synchronizing empty files. When synchronizing 1000 folders and 2000 empty files Smart Briefcases takes 14.485 seconds while Microsoft's Briefcases takes 104.90 seconds.

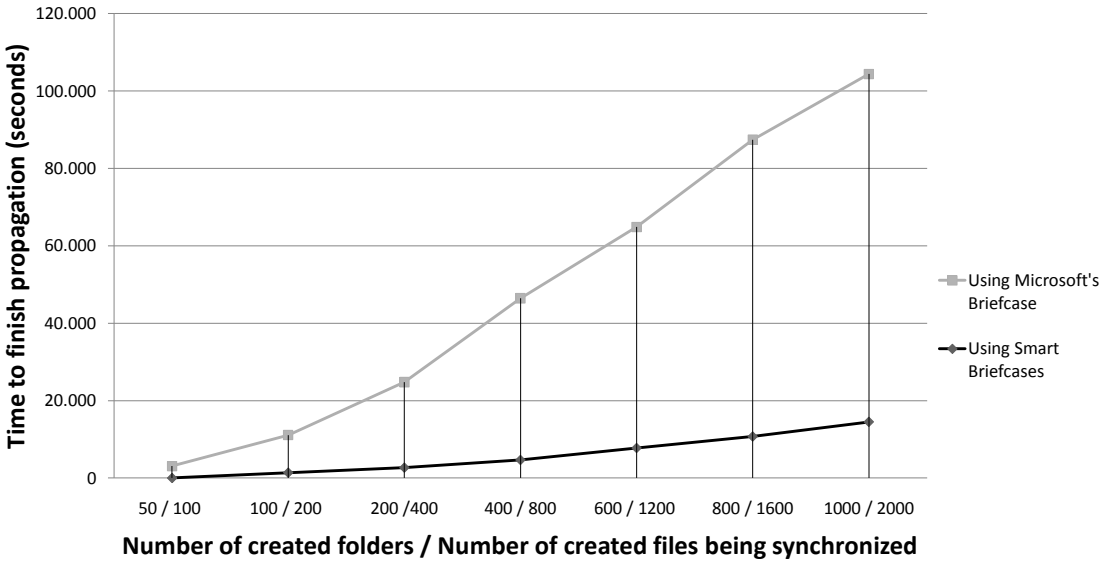

Figure 4.9: The graph shows the comparison between the time it takes to transfer files with 0 bytes through the network using Microsoft's Briefcase and using Smart Briefcases.

The same test is performed comparing Smart Briefcases and using a shared windows folder to propagate the empty files to the other computer. In this case a shared folder was created between the two computers. This shared folder allows a user to propagate files from one computer to the other. Using this folder an increasing number of folders and empty files are created in the shared folder and are propagated through the network to the other computer.

Figure 4.10 shows the comparison of the time taken to synchronize the folders and empty files using Smart Briefcases and propagating the same number of folders and files using a Windows shared folder. Again, Smart Briefcases takes a lot less time than Microsoft's Windows. When synchronizing 1000 folders and 2000 empty files Smart Briefcases takes 14.485 seconds while Microsoft's Briefcases takes 74.640 seconds.

4.2.3.2 Speed of the Synchronization Process - File and Folder Creations (Files with content)

The previous tests show that Smart Briefcases synchronizes newly created empty files relatively fast when compared to other solutions. However, in most cases users will synchronize only files that have content within them. To test this new scenario instead of creating empty files, files with 619208 bytes of size were created. This value was chosen based on the average of the size of files within the author's personal folder. This folder contains 23094 files which occupy 14.3GB. Two tests were performed using these files.

The first test measures the time is takes to synchronize newly created files with 619208 bytes between two briefcases connected by a local network. In this case contents need to be propagated through the network which greatly delays the synchronization process. Figure 4.11 details the results obtained from the tests. As can be observed, it takes 23 minutes and 28 seconds to synchronize one thousand folders and two thousand files.

In order to evaluate how the size of the synchronized files affected the speed of synchronization several tests were performed in which files with an increasing size were synchronized between two remote computers.

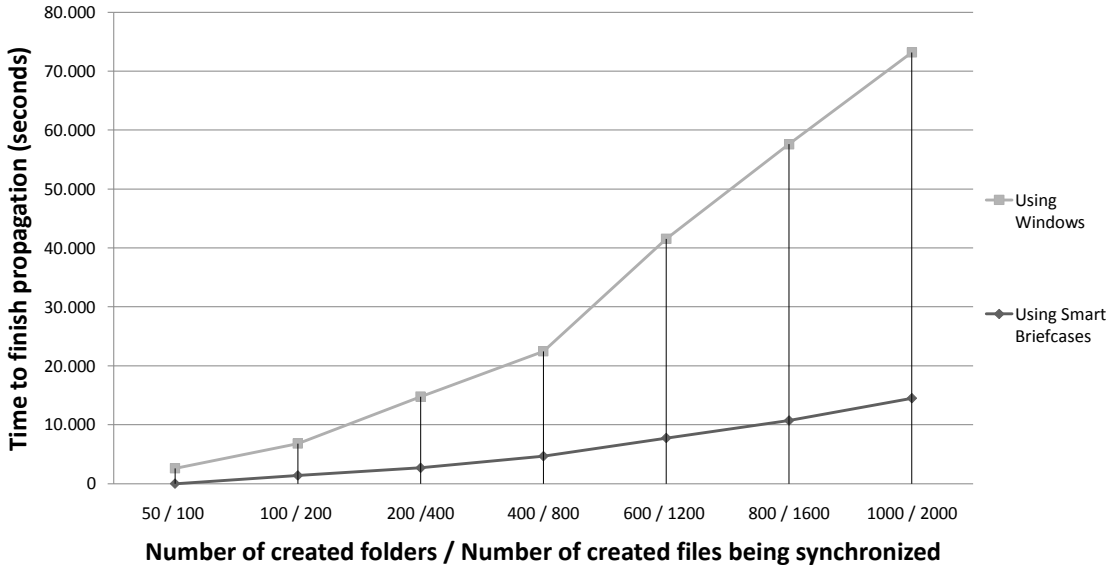

Figure 4.10: The graph shows the comparison between the time it takes to transfer files with 0 bytes through the network using Windows Shared Folders and using Smart Briefcases.

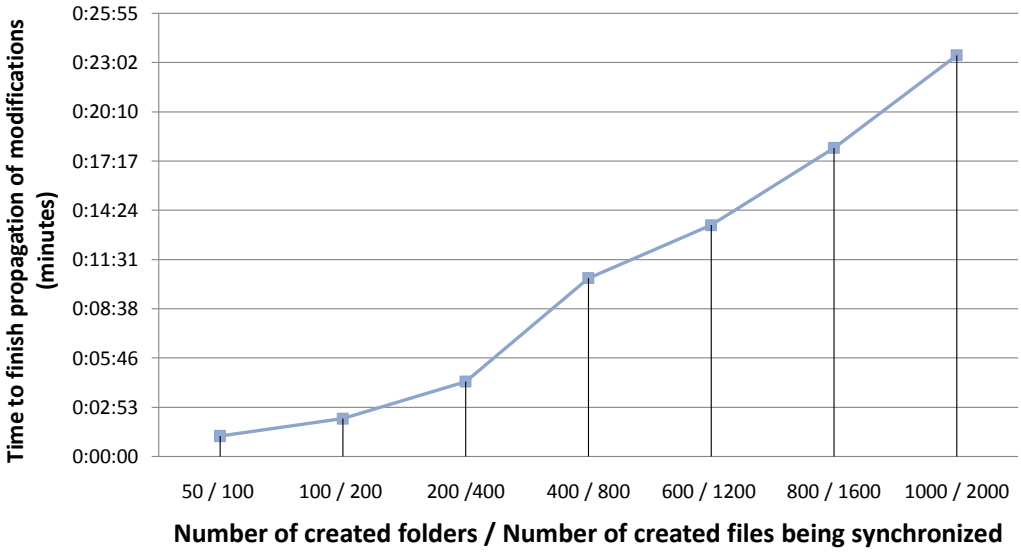

Figure 4.11: The graph shows the time it takes to synchronize an increasing number of folders and files that were created in only one replica.

Figure 4.12 displays the results of these tests. First, the tests were preformed with files with 128kbs. In this case the synchronization of 1000 folders and 2000 files takes 10 minutes and 56 seconds. Next, the synchronization of files with 512kbs was performed taking 21 minutes and 6 seconds to synchronize the same number of files and folders. Files with 1000kbs take 1 hour, 23minutes and 48 seconds. Finally files with 2000kbs take 2 hours, 56 minutes and 42 seconds.

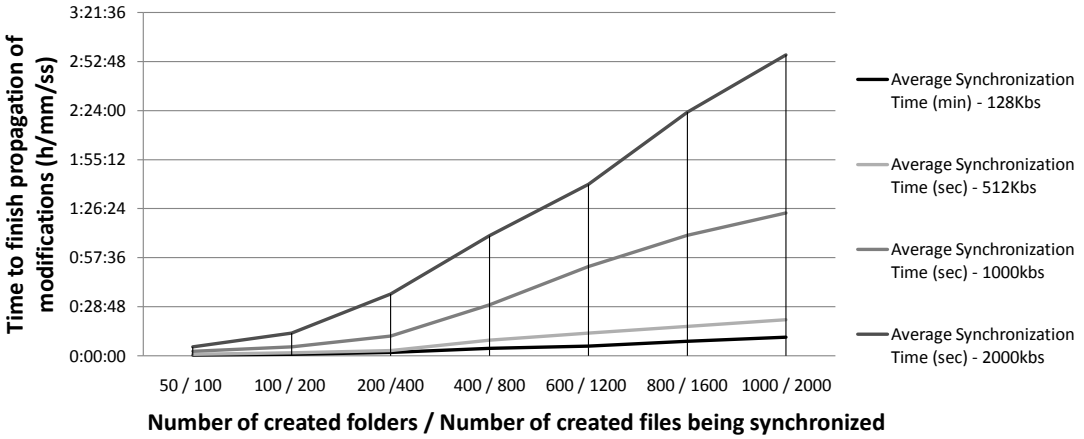

Figure 4.12: The graph shows the time it takes to synchronize an increasing number of folders and files with different sizes.

The times obtained in the previous tests seem very high. In order to understand if the obtained values were normal or if the synchronization process of Smart Briefcases took too much time, some tests were performed to compare the propagation speed of Smart Briefcases with propagating the same files through Windows.

The result of this comparison is shown in figure 4.13. In most tests, propagating files through Smart Briefcases took a little less time than sending them through windows. However, the difference is minor and we can conclude that propagating whole files through Smart Briefcases is as fast as using Windows.

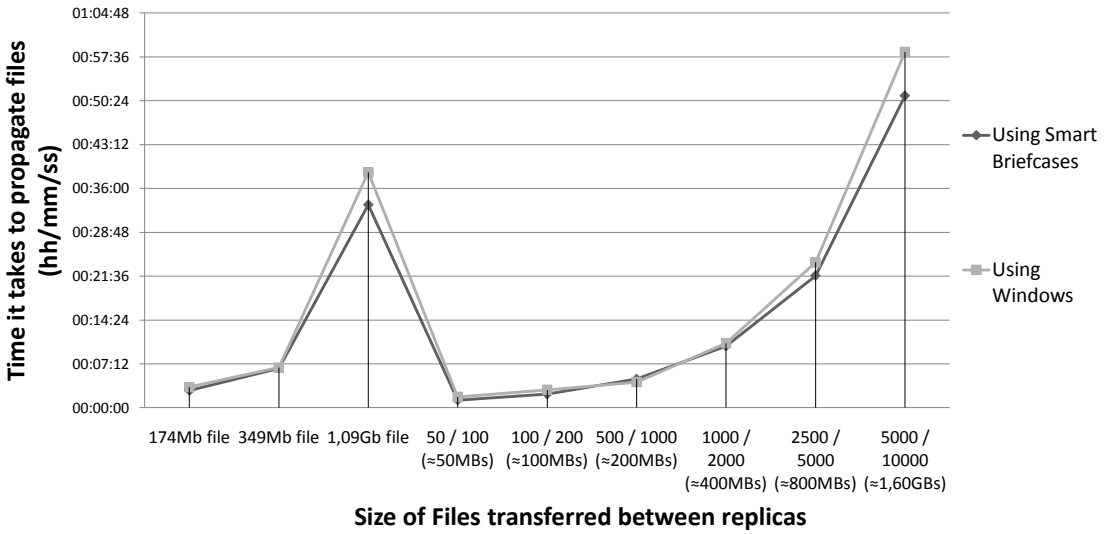

Figure 4.13: The graph shows a comparison between the time it takes to transfer files through the network using Windows and using Smart Briefcases.

The same tests were performed between Smart Briefcases and Microsoft's Briefcases. The result of this comparison is shown in figure 4.14. In most tests, propagating files through Smart Briefcases took almost the same time as propagating the same number of files and folders through Microsoft's Briefcases.

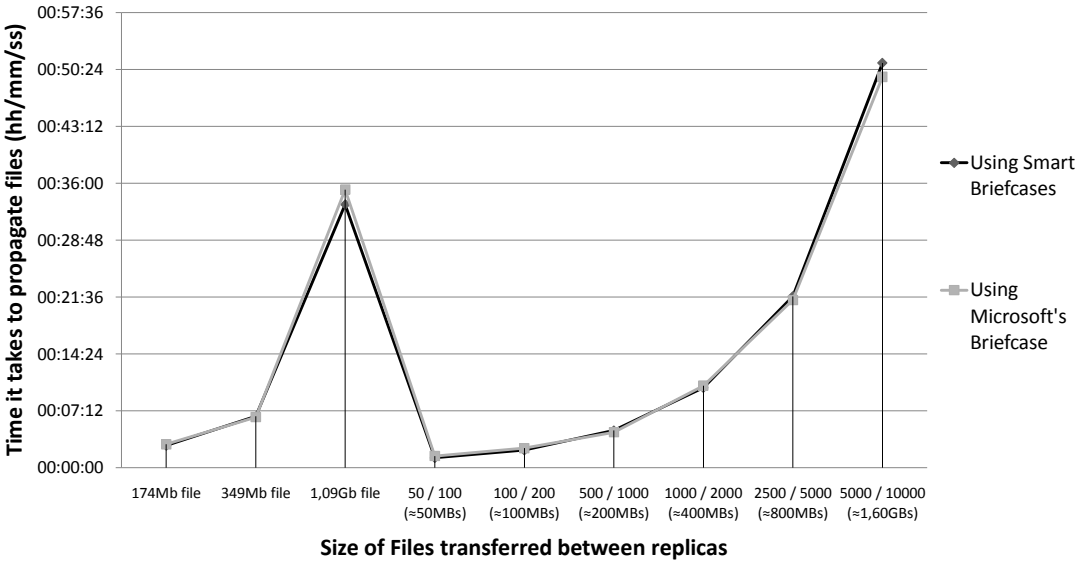

Figure 4.14: The graph shows a comparison between the time it takes to transfer files through the network using Microsoft' Briefcase and using Smart Briefcases.

4.2.3.3 Speed of the Synchronization Process - File and Folder Renaming

In order to evaluate the synchronization of folders and files that have been renamed within a briefcase, a pair of briefcases is created. Each briefcase contains the same number of files and folders within. Next, a certain number of files and folders is renamed in one of the replicas. Finally, the synchronization is performed propagating all renamed files and folders while observing and recording the time it takes to finish the process.

The results can be observed in figure 4.15 in page 73. When synchronizing a pair of briefcases located in remote computers the value obtained when synchronizing 1000 renamed folders and 2000 renamed files is only 16,281 seconds. This is a very reasonable time and can be achieved because in this case, Smart Briefcases, does not need to propagate content from files. The system is only required to send messages to the other computer informing that a certain file or folder must be renamed to a specific name. The time measured is deemed acceptable since thousands of files can be renamed in less than twenty seconds.

4.2.3.4 Speed of the Synchronization Process - File and Folder Deletions

The process used to evaluate the speed of propagating deletions of files and folders is the same used when testing creations and renames. A pair of briefcases is created and an increasing number of folders are deleted in one of the replicas. Afterwards, synchronization is performed in which the files deleted in one replica are deleted in the other. The time taken by this process is observed and recorded.

The results of the test are shown in figure 4.16 in page 73 shows the results of the tests performed. The obtained values are considered to be within reasonable values. To delete 1000 folders present in a remote replica, Smart Briefcases takes only 12,488 seconds.

4.2.3.5 The Cost of Using Smart Briefcases

During evaluation it we decided to compare the difference in speed while performing file system operations when Smart Briefcases is monitoring the device's drives and briefcases and when Smart Briefcases is not in use. To do this, a briefcase is created and an increasing number of files and folders are created inside the briefcase while measuring the time this operation takes. The same tests are performed in a normal Windows Directory. Each test is performed 10 times while measuring the time of each. Using the collected values an average is calculated. The average values are used to create the figure 4.17.

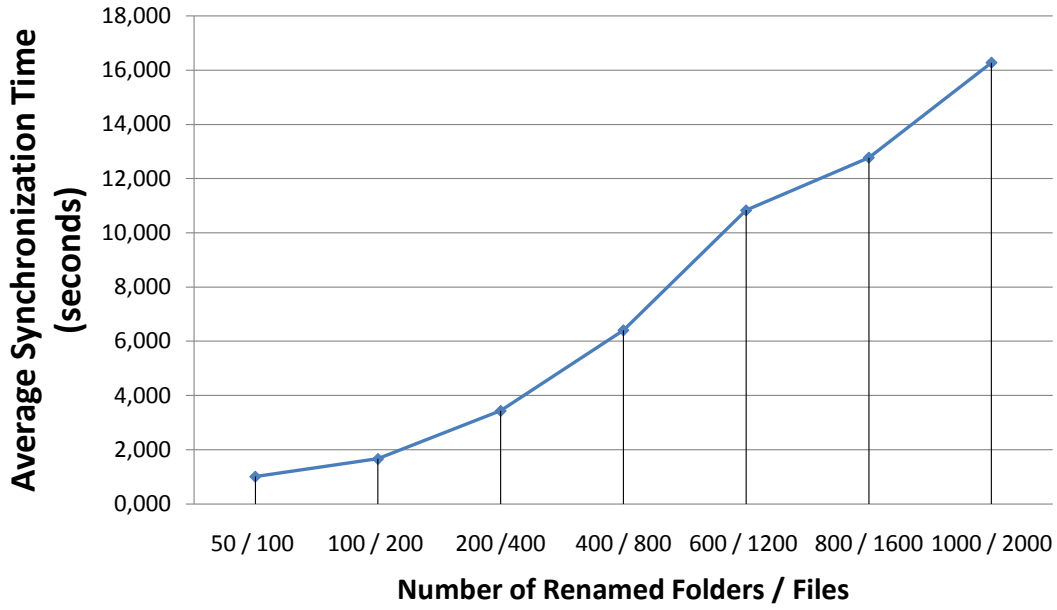

Figure 4.15: The graph shows the time it takes to synchronize two briefcases stored in two different computers.

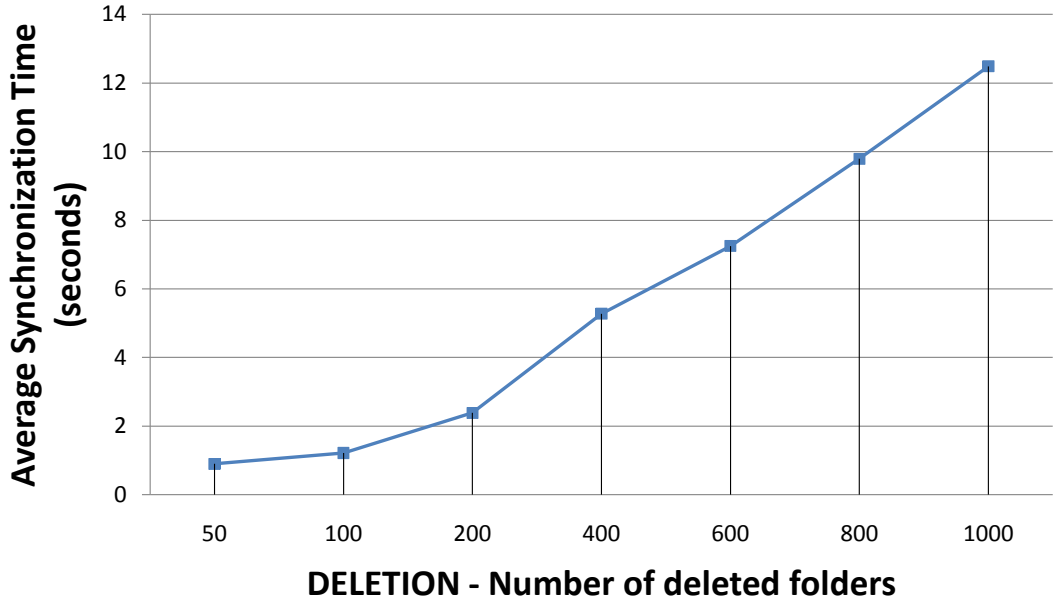

Figure 4.16: The graph shows the time it takes to synchronize two briefcases stored in two different computers. In this case an increasing number of folders with files stored within were deleted.

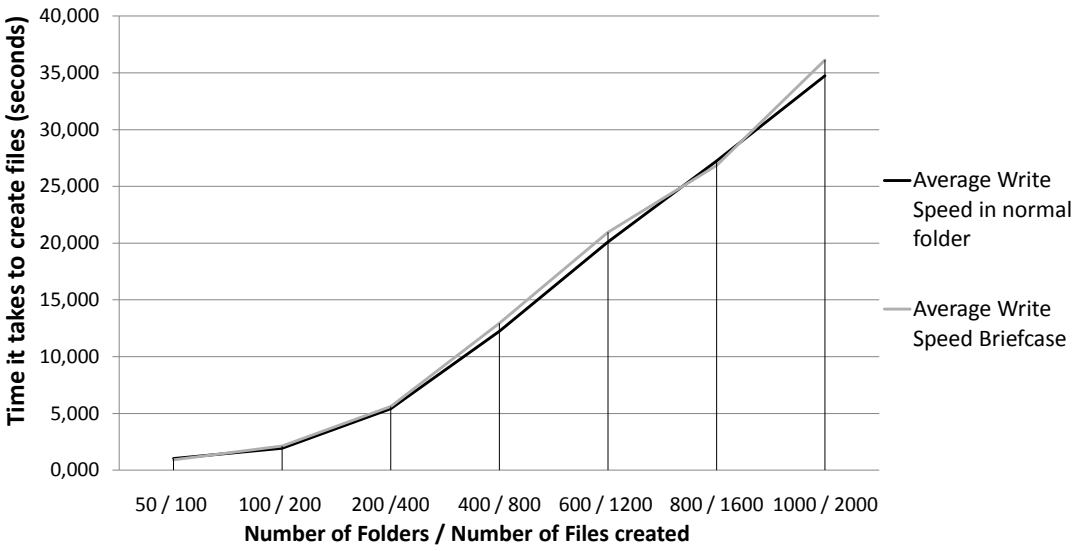

Figure 4.17: The graph shows the comparison between the time taken to create a certain number of files and folders inside a briefcase and a normal folder.

As can be observed in figure 4.17 the time taken to create files inside a briefcase is very similar to the time it takes to create the same files in a windows folder. The reason for this is that the file system monitor that is watching the briefcase does not stop or delay the file system operations. The file system monitor receives the operations being performed inside each briefcase in an asynchronous way.

We can conclude that no noticeable delay exists when a user is creating files inside a briefcase.

4.2.4 Bandwidth

It was important to evaluate the bandwidth used when synchronizing modifications between two remote replicas. The amount of data that is sent though the network must be kept to a minimum which also influences the time it takes to finish the process.

In order to measure the amount of data that is sent when synchronizing two replicas the program Wireshark[3] was used. This program allows a user to see all packets that are sent through the network. Using this functionality, it is easy to create a filter that only displays the packets that concern Smart Briefcases. It is also possible to see the total size that is occupied by the packets. Hence, getting a value that represents the total size of data that was transferred through the network during the synchronization process.

While performing all tests that measured the synchronization speed, Wireshark was receiving all the packets that were transmitted. By the end of each test it was possible to measure not only the time it took to finish the process but also the total size of traffic that had been transferred between replicas.

Figure 4.18 represents the amount of data, in kilobytes, that was transferred between replicas during the synchronization of newly created files and folders. The data propagated through the network during the tests performed is composed of: the content of the files propagated, several TCP packets and packets that contain information pertaining to the .Net Remoting protocol. In total, when synchronizing 800 newly created folders and 1600 new files, 975 287 kilobytes were transferred between replicas.

The measured size of information transferred between computers is a little larger than the total size of the files propagated. The remainder of the size is due to the transmission of other packets such as .Net Remoting messages, TCP messages and messages sent by Smart Briefcases.

[3]http://www.wireshark.org/

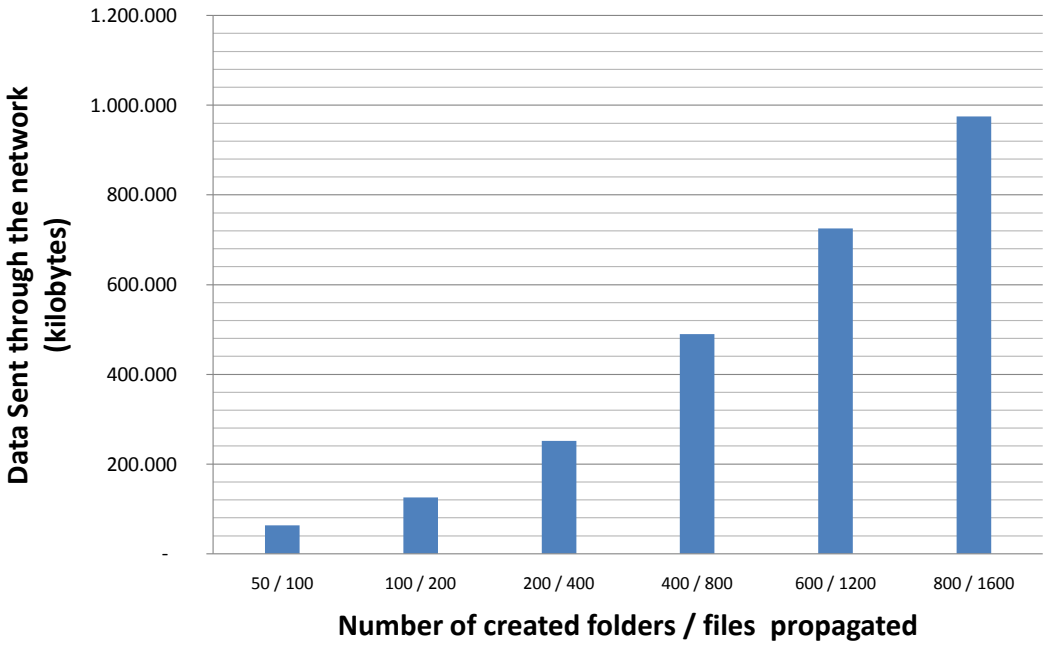

Figure 4.18: The graphic shows the size of the content sent through the network when propagating files and folders that were created in one of the replicas since the last synchronization.

Figure 4.19 represents the amount of data, in bytes, that was transferred between replicas during the synchronization of renamed files and folders. The size of data transferred is much smaller than the previous test since the contents of files do no need to be propagated. In order to rename 800 folders and 1600 files 2.553.017 bytes were propagated.

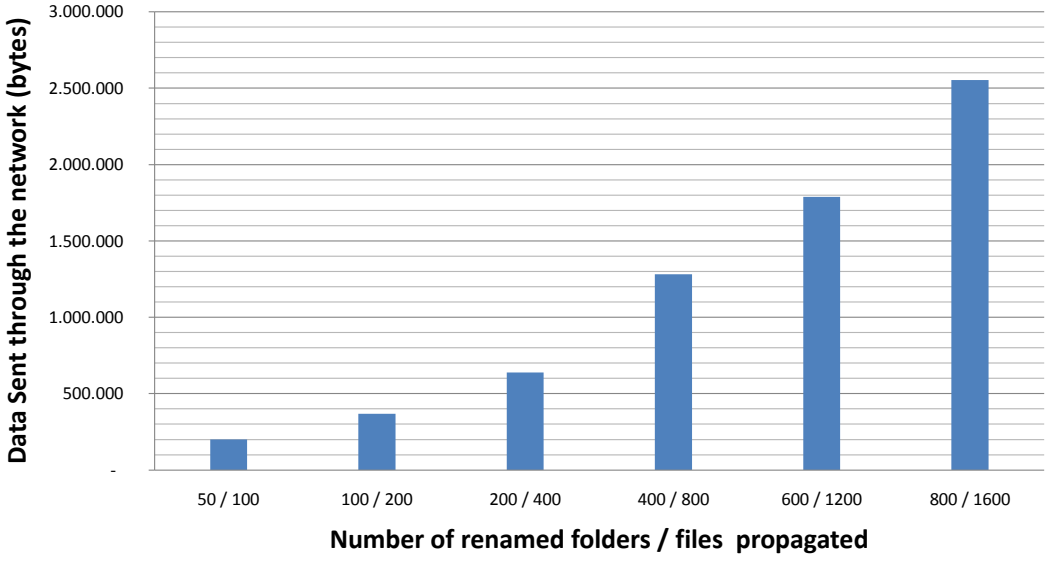

Figure 4.19: The graphic shows the bandwidth spent when propagating the new names of files and folders that have been renamed since the last synchronization.

Finally, figure 4.20 shows the size of data sent when synchronizing deleted folders. Again, the amount of data propagated is within reasonable ranges. In order to delete 800 folders and 1600 files 1.976.637 bytes were transferred.

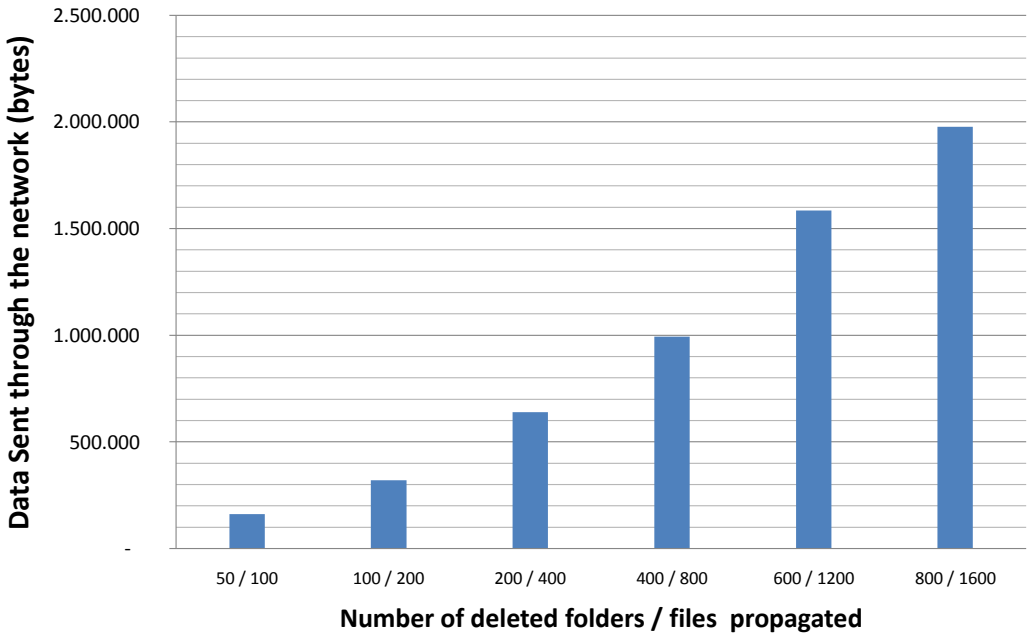

Figure 4.20: The graphic shows the bandwidth spent when propagating deletions of files and folders between replicas.

4.2.5 Ease of Use

It is very important that, when using Smart Briefcases, a user does not feel overwhelmed or lost. Also, when displaying information regarding conflict resolution, the user must not have doubts on what has caused the conflict and what to do in order to successfully resolve it without losing information.

In order to better understand what could be improved in the user interface and what users required to successfully use Smart Briefcases, it was given the opportunity for some users to test the application. Smart Briefcases was tested by about 15 users. Most are students of Instituto Superior Técnico. During these tests users were observed and their opinions were recorded. Through the users' testimonials it was possible to add some elements that improve the user experience. It also enabled the improvement of other elements that had already been implemented.

This section highlights, through screenshots, several components of the graphical user interface that were included in order to help the user understanding what steps he has to perform in order to create a synchronization pair, what tasks are being performed during the synchronization process or what conflicts have occurred.

4.2.5.1 Ease of Use - Helping the User Create a Synchronization Pair

When a user first initiates the Smart Briefcases application, if no briefcases are detected inside the replica a balloon tip is shown explaining how the user can create one of these folders (figure 4.21 in page 77). Then, when the user creates the briefcase a new balloon tip is shown explaining the steps the user needs to perform in order to copy the newly created briefcase and create a synchronization pair (figure 4.22 in page 77). Finally, the moment the user creates a synchronization pair, either in the same replica or in two remote replicas, a new balloon tip pops up explaining what the user has achieved and how he can synchronize the two folders (figure 4.23 in page 77).

This visual aid is important to help the user understand what steps he needs to perform in order to successfully use Smart Briefcases. During the first experiments in which users tried Smart Briefcases, the balloon tips were not yet implemented. It was noticed that most users could not create briefcases or synchronization pairs without first receiving an explanation of how the program worked. Since during

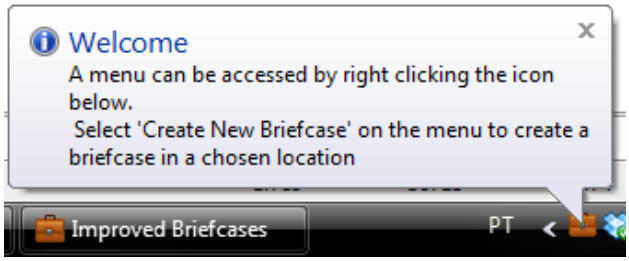

Figure 4.21: The balloon tip that pops up whenever the application is initiated and no briefcases are detected.

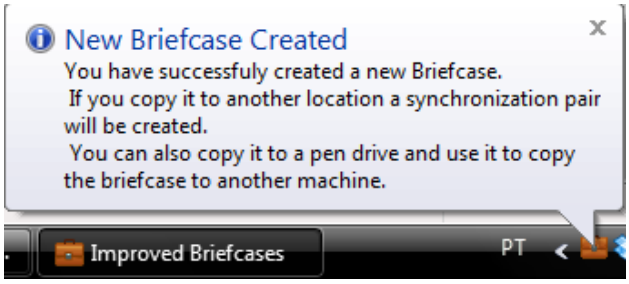

Figure 4.22: The balloon tip that pops up when a user creates a new briefcase.

normal usage there is no one to explain to a user what steps he needs to perform, the balloon tips were included to explain step-by-step the program's usage.

After the balloon tips were included, most users successfully created pairs and synchronized briefcases without further assistance. This proves that including this element in the user interface was a success.

4.2.5.2 Ease of Use - Information Provided During the Synchronization Process

Depending on the number and type of modifications performed to each briefcase, the synchronization process can take more or less time. In some cases - for example, when large files have been copied to a briefcase - this process can take more than one hour. For this reason it is important that there is some element within the user interface that informs the user that the process is still being performed and what task is being performed at the moment.

To accomplish this, balloon tips are used. During synchronization, balloon tips constantly pop up informing the user which step is being performed. Some examples of balloon tips are shown in figure 4.24 in page 78 The reason for presenting balloon tips, instead of presenting for example a window with a progress bar or other controls is the fact that balloon tips do not require any interaction from the user and do not stop any tasks that the user is currently performing. The user is visually informed without being interrupted.

Balloon tips are also shown when conflicts are detected. They not only inform the user that conflicts

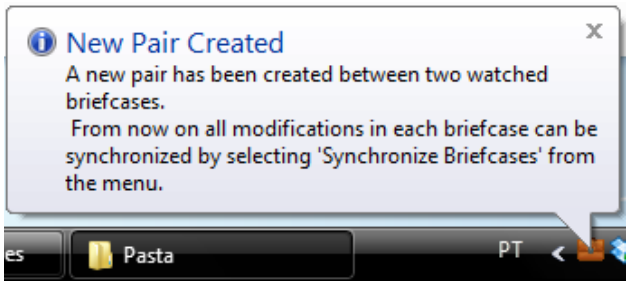

Figure 4.23: The balloon tip that pops up when a user successfully creates a synchronization pair.

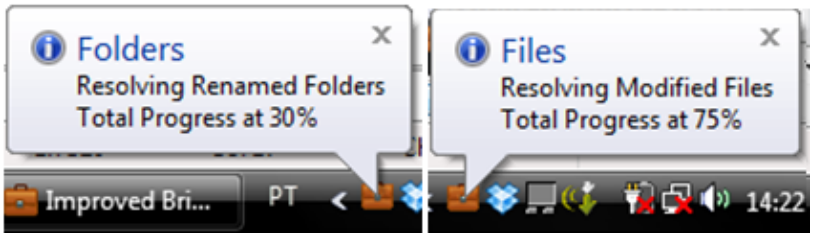

Figure 4.24: A balloon tip informs the user of the progress of the synchronization process. The message is updated throughout the process

were detected but also explain how he should use the conflict form in order to successfully resolve the conflicts. These balloon tips are displayed in figure 4.25 in page 78.

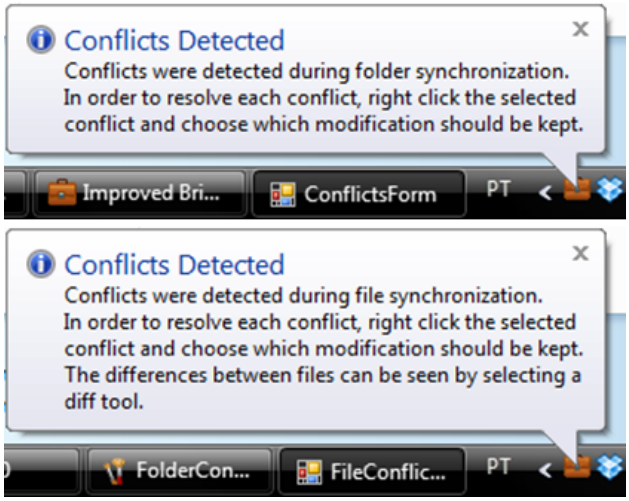

Figure 4.25: These are the balloon tips that pop up whenever conflicts are detected.

However, a balloon tip disappears after sometime. To allow a user to still see the progress of the synchronization process even after a balloon tip has disappeared the context information of Smart Briefcases' icon is updated. Therefore, a user is able to see the progress by hovering the mouse on top of the tray icon. This behavior is shown in figure 4.26 in page 78.

Figure 4.26: The user can still see that the synchronization process is still being performed by hovering the mouse over the tray icon. The message is updated throughout the process.

4.2.5.3 Ease of Use - Conflict Resolution

When conflicts are detected it is crucial to provide all the required information to users so that no files are lost in the process and the intended resolution is achieved. However, the information must be shown in a certain way that does not overwhelm the user and actually helps him resolving conflicts. Also, all information that is not relevant should be filtered. These observations were taken into account when designing the Windows forms that present the detected conflicts to the user.

Therefore, when conflicts occur a form is shown. Through the examination of the form the user can immediately determine: what type of conflict has occurred; what files or folders caused the conflict; what

modifications were performed to each file or folder; at what time each modification happened and where they are located in each machine. This information was deemed as invaluable for a user to be able to decide which modification he wants to keep.

Figure 3.9 in page 50 provides an example of a conflict form displaying two conflicts that happened between folders. The form informs the user that the first conflict was caused due to a folder being deleted in one replica while in the other replica the same folder was renamed. The second conflict was caused by the same folder being renamed to different names in both replicas.

To resolve the conflict the user simply needs to right click over the pair of folders he wants to resolve and a menu with several options appears (see figure 4.27 in page 79). The user can select "Keep Left" which deletes both folders, "Keep Right" which renames both folders with the same name or "None" which will ignore this conflict and will not resolve it. When the conflicts were caused by files the user has not only these options but he can also use diff tools in order to see what was modified within each file's contents.

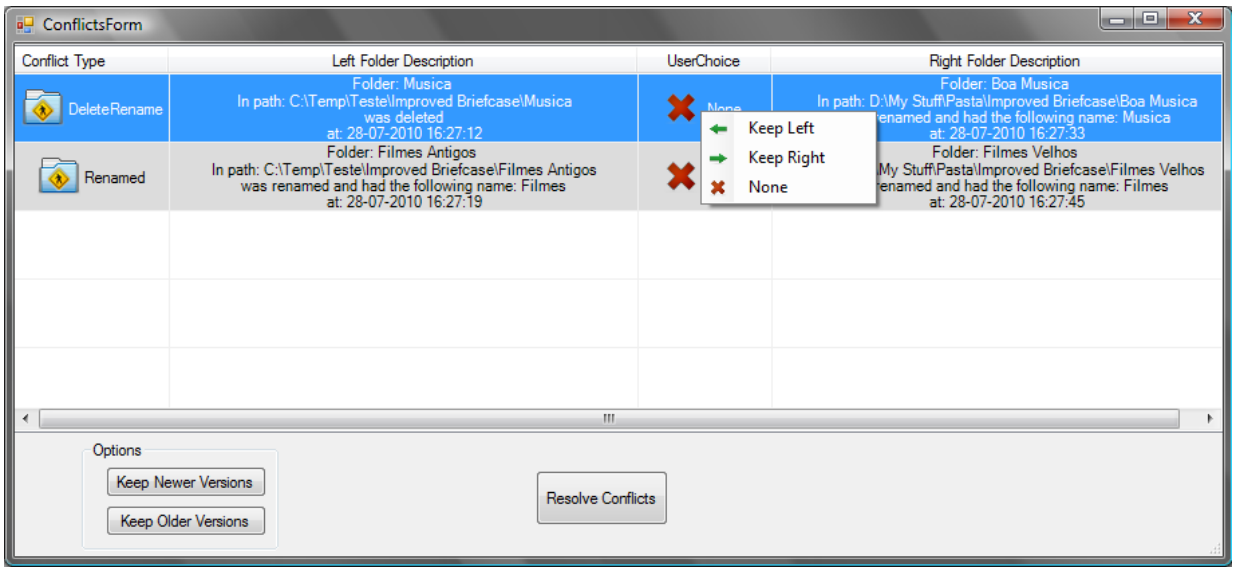

Figure 4.27: The menu offers the user several options to resolve the selected conflict.

The user can also choose to keep all newer or older modifications. This prevents the user from unnecessarily going through the list of conflicts one by one when he already knows that the older or newer modifications are the ones he wants to keep.

Another functionality that was added in order to better inform the user is the addition of icons that represent the file type of the files that are in conflict and the type of modification performed to said files. Currently only the file types that are supported by the differencing tools included in Smart Briefcases (Plain text, Word and PowerPoint files) have icons associated. An example of the use of icons is displayed in figure 3.14 in page 58.

4.3 Summary and Conclusions

Smart Briefcases was subjected to several tests with the purpose of obtaining an accurate estimate of its effectiveness and efficiency when used in scenarios that are close to real world usage and user expectations.

The obtained results show that the memory footprint of Smart Briefcases, even when storing 4096 folders and 16384 files inside a briefcase, is within very reasonable values (Windows Vista - 36,10 MBs / Windows 7 - 27,7MBs). This is especially true for machines with Windows 7 installed where the values measured in some cases were more than 10MBs lower than the values measured in the same conditions in machines with Windows Vista. The reason for this is unknown.

The time it takes to complete the synchronization process was also deemed to be within reasonable values. The synchronization of newly created files, where the file's contents need to be propagated through the network, takes about the same time as transferring the same files through the network using Windows or synchronizing two briefcases using Windows' Briefcases. The propagation of deletions and renames throughout replicas takes no more than some seconds. Even in the tested situation in which 1000 folders and 2000 files were renamed in one of the replicas, the propagation of these modifications took only 16,281 seconds. This result is considered to be very fast and within user's expectations.

The exception is the propagation of files that were modified. No matter how small the modification performed is, the file is still propagated in its entirety. This can be improved by adding an algorithm that identifies the particular bits of a file that were modified and propagates only these bits. This feature is currently marked as future work.

The values measured when evaluating the bandwidth required to synchronize two replicas are also considered to be within reasonable values. When propagating created files there is a noticeable overhead of data propagated but this is also found when transferring files through the network using Windows.

In this chapter it was also shown what actions were performed and what elements were added to Smart Briefcases in order to improve the usability and the user's experience when using the application. Although these are two very subjective areas, it was the global opinion of users who tested the application that the elements added improved the ease of usage and helped them when operating Smart Briefcases.

Chapter 5

Conclusions and Future Work

Nowadays, due to the increasingly low cost of computational devices and the need for people to keep working anywhere anytime, a user is expected to own several of these devices. As a result, it is expected that files are replicated throughout several of the user's machines. This may happen, for example, due to the need for a user to access and modify certain files wherever he is. However, manually selecting the files to replicate, deciding what files were changed and what files need to be updated can be a very time consuming and monotonous task. This brings forth the need for file synchronization solutions that help the user with this task.

Creating a file synchronizer is relatively simple. However, creating one that really helps the user replicating files without creating duplicates or erasing needed files, provides information when conflicts occur and acts as the user expects is not as simple. There are already several commercially distributed solutions that provide file synchronization functionality. Some even come already installed in newly purchased machines. However, most of these solutions have at least one of the previously mentioned problems.

As detailed in Chapter 2, conflict resolution is an area that most existing solutions fail to successfully handle. Some solutions simply resolve the conflicts by performing a default action that in most cases is not what the user requires (Synctoy, ActiveSync), others simply inform the user that there is a conflict in a certain file and leave the user to find where the conflict occurred and to perform the resolution manually (Live Mesh, Dropbox). Some solutions do not even accept some operations performed to files and folders (Microsoft's Briefcase).

This dissertation proposes a file synchronizer that helps a single user who owns several computational devices maintain all replicated files consistent by applying mechanisms that identify conflicts, help the user resolving said conflicts and propagate updates between devices without modifying applications already used by the user. The file synchronizer is called Smart Briefcases.

As described in Chapter 3, where Smart Briefcases' architecture is analyzed, Smart Briefcases monitors the user's work while he creates, deletes, renames and modifies files and folders. All modifications are stored as meta-data. During the synchronization process structures from two replicas are compared in order to understand what changed since the last synchronization, what needs to be updated and to detect conflicts. When conflicts are detected, Smart Briefcases is able to present the relevant information to the user which helps him perform the action that resolves the conflict.

While most conflicts can be resolved through the Smart Briefcases' graphical interface alone, the conflicts where the same file was modified in different replicas or conflicts where the user needs to merge the files' contents in order to prevent the loss of information can only be resolved by the user's manual intervention. The user is required to access the conflicting files, search for what differs and resolve the conflict. However, in some supported files (plain text, Microsoft Word and Microsoft PowerPoint) the user is able to use a built-in differencing tool in order to see the exact place where the files were modified in each replica. This prevents the user from having to manually compare the two files, which can take a lot of time depending on the file's size.

When the user wants to merge two conflicting files with file types that are not supported by Smart

Briefcases he has to perform the merging manually. He can still choose to keep one of the versions over the other but this may cause him to lose information. However, in order to attenuate this problem Smart Briefcases was built in a way that allows other differencing engines to be added programmatically which allows other file types to be supported in the future.

In Chapter 4 Smart Briefcases was evaluated in order to test the effectiveness and efficiency of the solution when used in scenarios that are close to real world usage and user expectations. During the tests Smart Briefcases' memory footprint and bandwidth usage did not deviate from reasonable values. The synchronization process was also deemed to be reasonably fast in most cases with the exception of the propagation of file's modifications since files are retransmitted as a whole. It was also detailed the efforts performed during implementation in order to increase the ease of use and the user's experience when using Smart Briefcases.

As a conclusion, Smart Briefcases successfully helps users maintain files replicated and consistent throughout replicas while providing valuable help in case of conflicts. This is achieved without the need to modify the applications the user already uses and without adding significant overheads to the user's machines.

5.1 Future Work

As is usual with a project of this scope there are always some aspects that can be improved, added or completed in order to improve the delivered solution. This aspects are presented in this sections divided in major and minor.

5.1.1 Major Aspects

- **Allow synchronization of a briefcase between more than two computers**

  Currently, Smart Briefcases is only able to create synchronization pairs between two computers. It would be interesting if the solution could share the same briefcase between several computers and successfully perform pair-wise synchronization between them while maintaining correct results.

- **Add more differencing engines to Smart Briefcases**

  One of the innovations that Smart Briefcases introduces is the detailed information it provides to users when conflicts occur. By adding more differencing engines users would be able to easily identify differences when conflicts occur between modified files. File types that would bring added value to Smart Briefcases include, images, databases, Microsoft Excel files, etc.

- **Improve data propagation efficiency**

  Implementing an algorithm that identifies what content of a certain file was modified and propagates only that content is required in order to reduce the amount of data propagated and the time it takes to synchronize file modifications.

- **Add security and privacy mechanisms to data propagation**

  Currently data is sent through the network in plain view. This brings certain security and privacy risks especially if Smart Briefcases is used in public or open networks. In order to be able to assume realistic trust models in real world usage the privacy of the information sent should be ensured.

5.1.2 Minor Aspects

- **Add mechanisms to create a briefcase through context menus**

  As explained in section 4.1.2 Smart Briefcases should present an interface better integrated with Windows. This means that a user should be able to right click in a certain folder and create a new briefcase. For reasons detailed in section 4.1.2 this was not possible. However, with the recently released Microsoft .NET Framework 4.0 it is possible to create this functionality without resorting to unmanaged code.

- **Reflect the state of files and folders inside a briefcase through icons**

  As was explained in section 4.1.2 it would be interesting to change the icon of a file or folder depending if they are synchronized, have been modified or were created since the last synchronization. This functionality is used in some file synchronizers such as Dropbox [10]. However, due to how Smart Briefcases works all implementation attempts failed. Additional mechanisms should be studied in order to successfully implement this functionality.

Bibliography

[1] S. Balasubramaniam and B. Pierce. What is a file synchronizer? In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 98–108. ACM New York, NY, USA, 1998.

[2] S. Balasubramaniam and B. C. Pierce. What is a file synchronizer? In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 98–108, New York, NY, USA, 1998. ACM.

[3] J. Barreto. Haddock-FS, A Distributed File System for Mobile Ad-hoc Networks, 2004.

[4] J. Barreto. *Optimistic Replication in Weakly Connected Resource-Constrained Environments*. PhD thesis, 2008.

[5] J. Barreto and P. Ferreira. A replicated file system for resource constrained mobile devices. In *Proceedings of IADIS International Conference on Applied Computing*, 2004.

[6] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. *W3C recommendation*, 6, 2000.

[7] P. Cederqvist, R. Pesch, et al. Version management with CVS, 2002.

[8] Y. Chou. Get into the groove: Solutions for secure and dynamic collaboration - white paper. `http://technet.microsoft.com/en-us/magazine/2006.10.intothegroove.aspx`.

[9] J. Dolinay. Detecting usb drive removal in a c# program. `http://www.codeproject.com/script/Articles/Article.aspx?aid=18062`.

[10] Dropbox. Secure backup, sync and sharing made easy. `https://www.dropbox.com/`.

[11] Git. Git - user manual. `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html`.

[12] Git. Git, the fast version control. `http://git-scm.com/`.

[13] R. Guy, G. Popek, and T. Page Jr. Consistency algorithms for optimistic replication. In *Proceedings of the First International Conference on Network Protocols*. Citeseer, 1993.

[14] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. In *Advances in database technologies: ER'98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management, Singapore, November 19-20, 1998: proceedings*, page 254. Springer Verlag, 1999.

[15] J. Haartsen. Bluetooth-The universal radio interface for ad hoc, wireless connectivity. *Ericsson review*, 3(1):110–117, 1998.

[16] P. Keleher and U. Cetintemel. Consistency management in Deno. *Mobile Networks and Applications*, 5(4):299–309, 2000.

[17] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 210–218, New York, NY, USA, 2001. ACM.

[18] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):3–25, 1992.

[19] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)*, 22(4):321–374, 1990.

[20] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[21] Microsoft. Explore the features: Sync center. `http://www.microsoft.com/windows/windows-vista/features/sync-center.aspx` accessed on 04/01/2010.

[22] Microsoft. How To Use the Briefcase Feature in Windows XP. `http://support.microsoft.com/kb/307885` accessed on 22/11/2009.

[23] Microsoft. Live Mesh. `https://www.mesh.com` accessed on 25/11/2009.

[24] Microsoft. Microsoft sync developer center. `http://msdn.microsoft.com/pt-pt/sync/default(en-us).aspx`.

[25] Microsoft. Synching your mobile phone and pc using activesync. `http://www.microsoft.com/windowsmobile/en-us/downloads/microsoft/activesync-download.mspx` accessed on 24/11/2009.

[26] Microsoft. SyncToy 2.1. `http://www.microsoft.com/DownLoads/details.aspx?familyid=C26EFA36-98E0-4EE9-A7C5-98D0592D8C52&displaylang=en` accessed on 21/11/2009.

[27] Microsoft. When would I use Briefcase instead of Sync Center? `http://windows.microsoft.com/en-US/windows-vista/When-would-I-use-Briefcase-instead-of-Sync-Center` accessed on 25/11/2009.

[28] Microsoft. Synchronizing Images and Files in Windows Using Microsoft SyncToy (Whitepaper), 2008. Downloaded from `http://www.microsoft.com/downloads/details.aspx?FamilyID=50fa5932-0685-4fe3-9605-536f39bd6c86&DisplayLang=en` accessed in 23/11/2009.

[29] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 174–187. ACM New York, NY, USA, 2001.

[30] B. Pierce and J. Vouillon. What's in Unison? A formal specification and reference implementation of a file synchronizer. *Technical Repor MS-CIS-03-36, Department of Computer and Information Science University of Pennsylvania*, 2004.

[31] P. Piper. Objectlistview - how i learned to stop worrying and love .net listview. `http://objectlistview.sourceforge.net/cs/index.html`.

[32] G. Popek, R. Guy, T. Page, and J. Heidemann. Replication in Ficus distributed file systems. In *Proc. of the Workshop on the Management of Replicated Data*, pages 5–10. Citeseer, 1990.

[33] M. Potter. A generic, reusable diff algorithm in c#. `http://www.codeproject.com/KB/recipes/diffengine.aspx`.

[34] D. Ratner. *Roam: A scalable replication system for mobile and distributed computing.* PhD thesis, Citeseer, 1998.

[35] D. Ratner, P. Reiher, and G. Popek. Roam: a scalable replication system for mobility. *Mobile Networks and Applications*, 9(5):537–544, 2004.

[36] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys (CSUR)*, 37(1):42–81, 2005.

[37] M. Satyanarayanan. The evolution of Coda. *ACM Transactions on Computer Systems (TOCS)*, 20(2):85–124, 2002.

[38] Several. `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html`.

[39] M. Stonebraker. Concurrency control and consistency of multiple copies of data in distributed INGRES. *IEEE Transactions on Software Engineering*, pages 188–194, 1979.

[40] SyncML. SyncML Whitepaper - Building an Industry-Wide Mobile Data Synchronization Protocol. Downloaded from `http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html` in 21-11-09.

[41] A. Tridgell and P. Mackerras. The rsync algorithm. 2004.

[42] L. Veiga and P. Ferreira. Semantic-Chunks a middleware for ubiquitous cooperative work. In *Proceedings of the 4th workshop on Reflective and adaptive middleware systems*, page 6. ACM, 2005.

[43] A.-I. A. Wang, P. Reiher, and G. Kuenning. Introducing permuted states for analyzing conflict rates in optimistic replication. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 376–377, New York, NY, USA, 2005. ACM.

[44] L. Wood, V. Apparao, L. Cable, M. Champion, M. Davis, J. Kesselman, T. Pixley, J. Robie, P. Sharpe, and C. Wilson. Document object model (dom) level 1 specification. *W3C Recommendation*, pages 1–19981001, 1998.

[45] G. Wuu and A. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 233–242. ACM New York, NY, USA, 1984.

[46] S. Zachariadis, L. Capra, C. Mascolo, and W. Emmerich. Xmiddle: information sharing middleware for a mobile environment. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 712–712, New York, NY, USA, 2002. ACM.

Appendix A

Flowcharts presented in Chapter 3

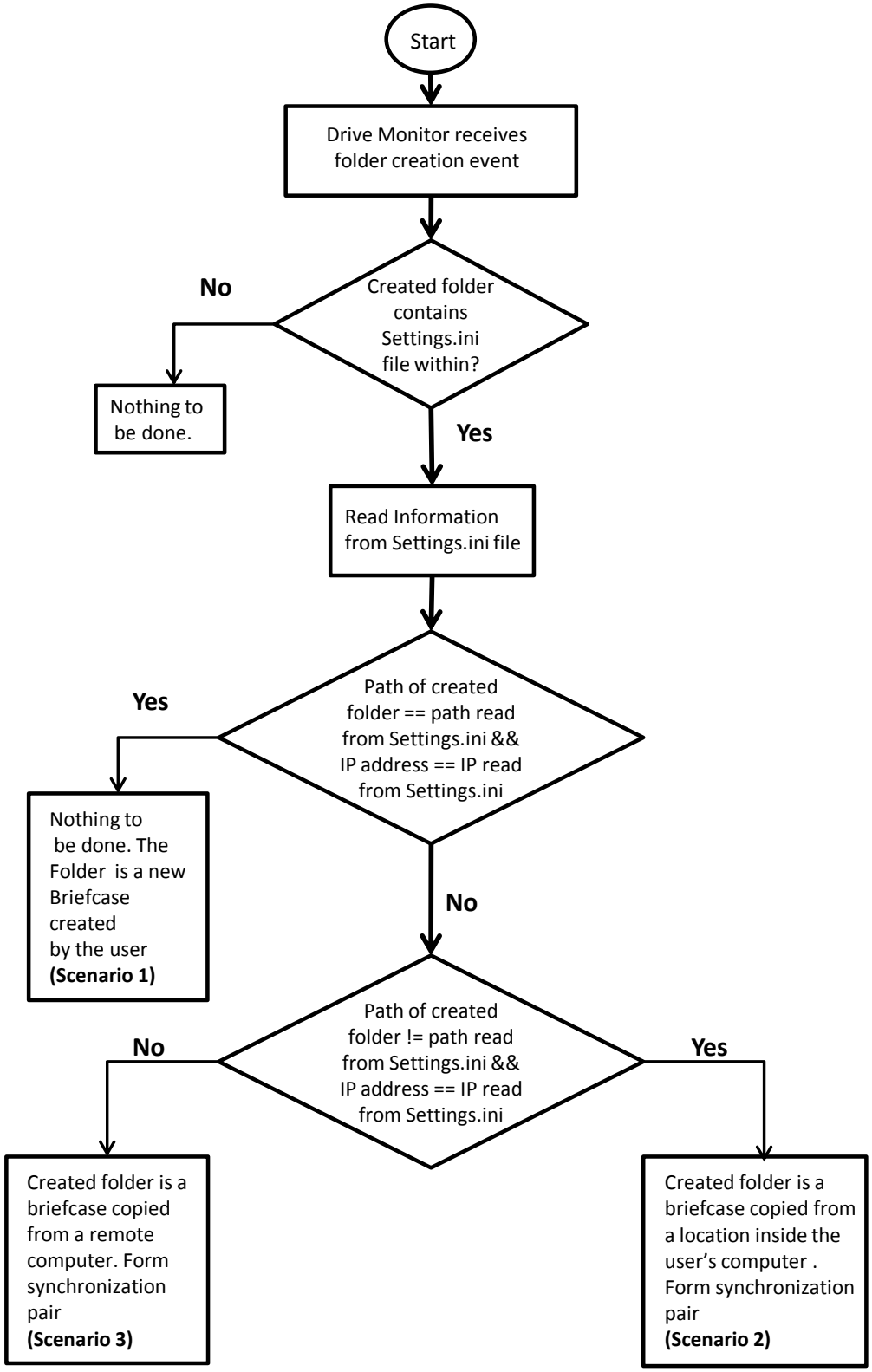

Figure A.1: The flowchart details how the Drive Monitor handles the creation of a folder inside a drive in the user's computer. The actions taken by the drive monitor are described with some detail in section 3.5.2

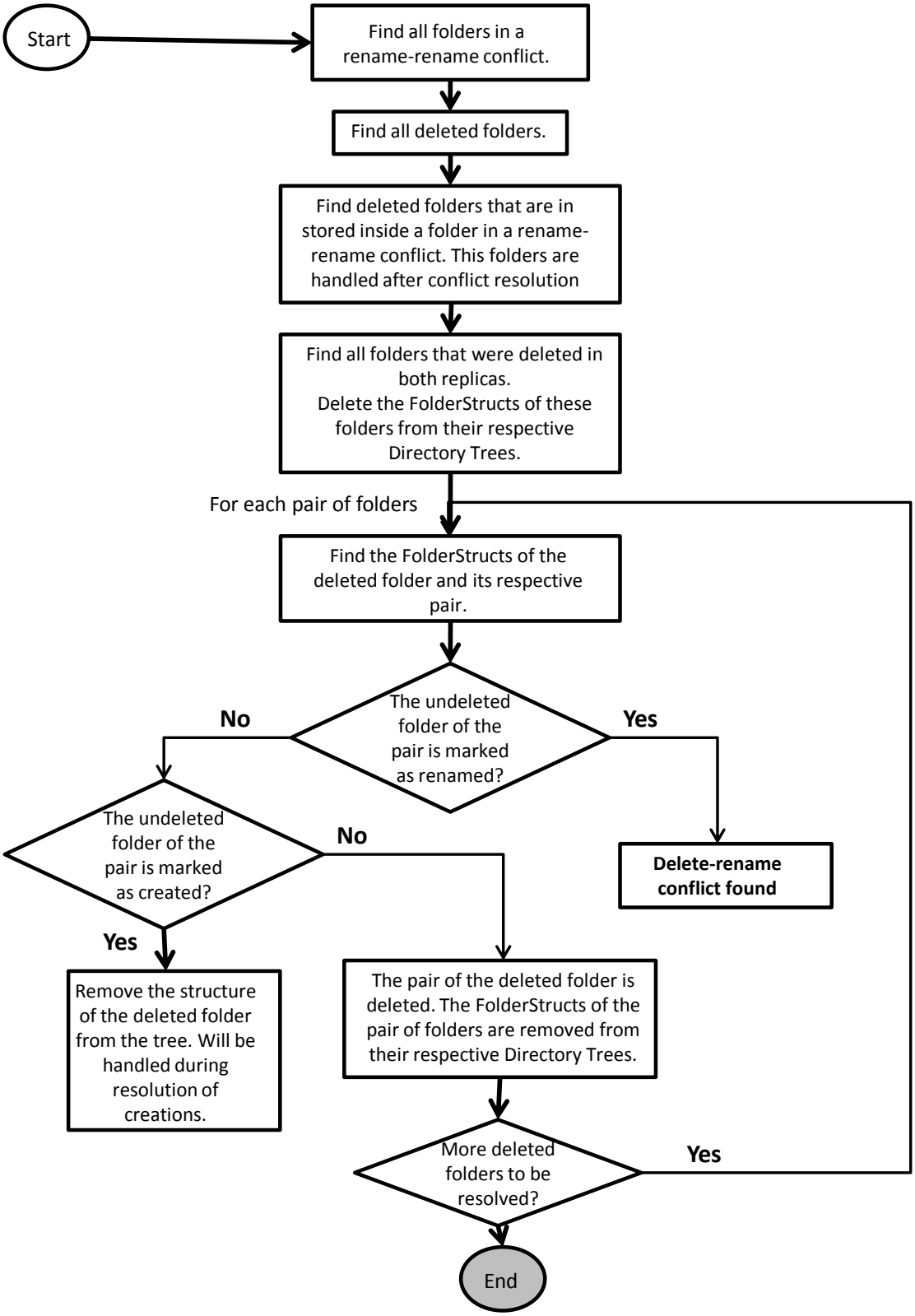

Figure A.2: The flowchart details the actions taken by the Resolver when synchronizing the deletion of folders.

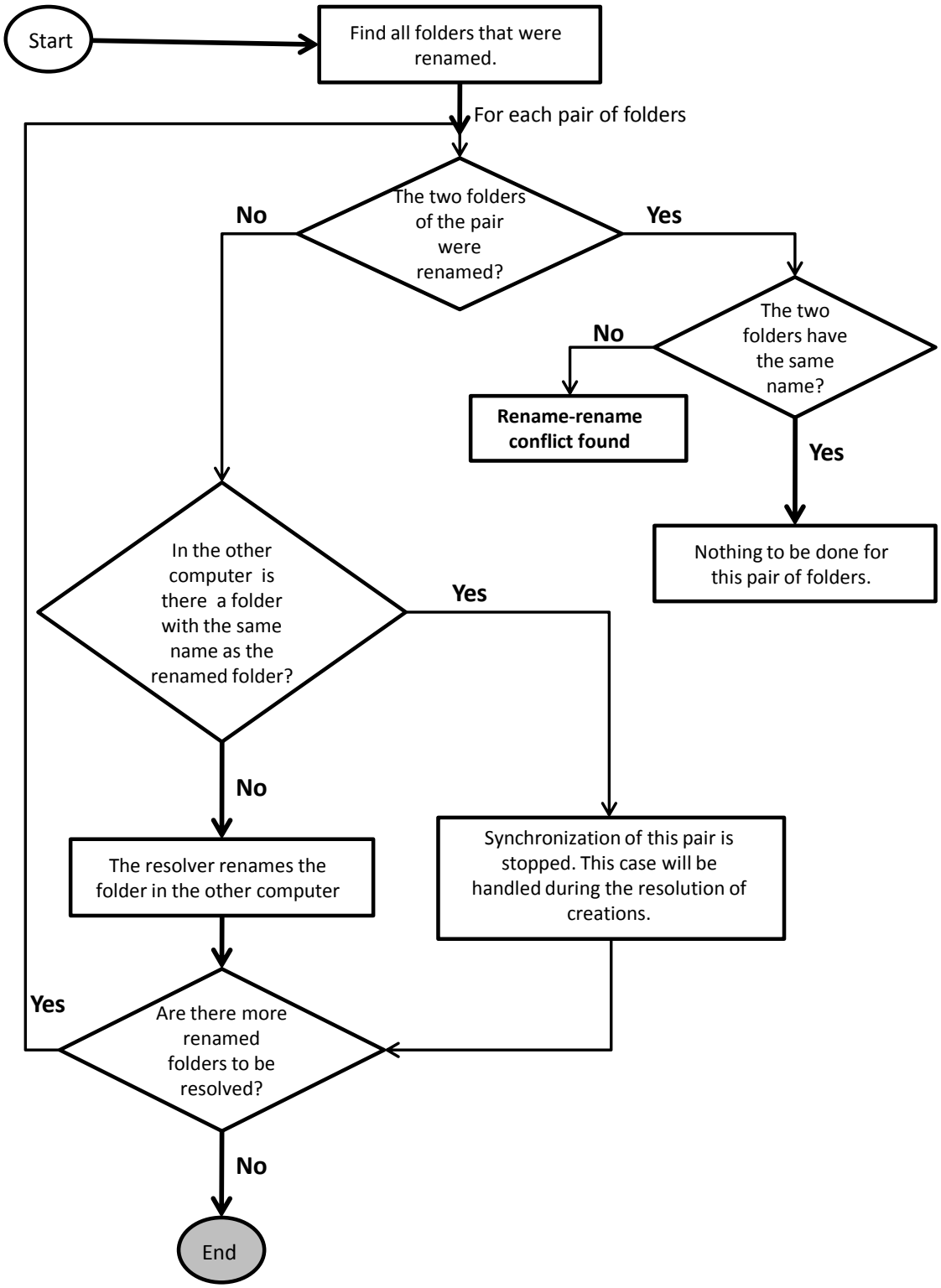

Figure A.3: The flowchart details the actions taken by the Resolver when synchronizing folders that were renamed.

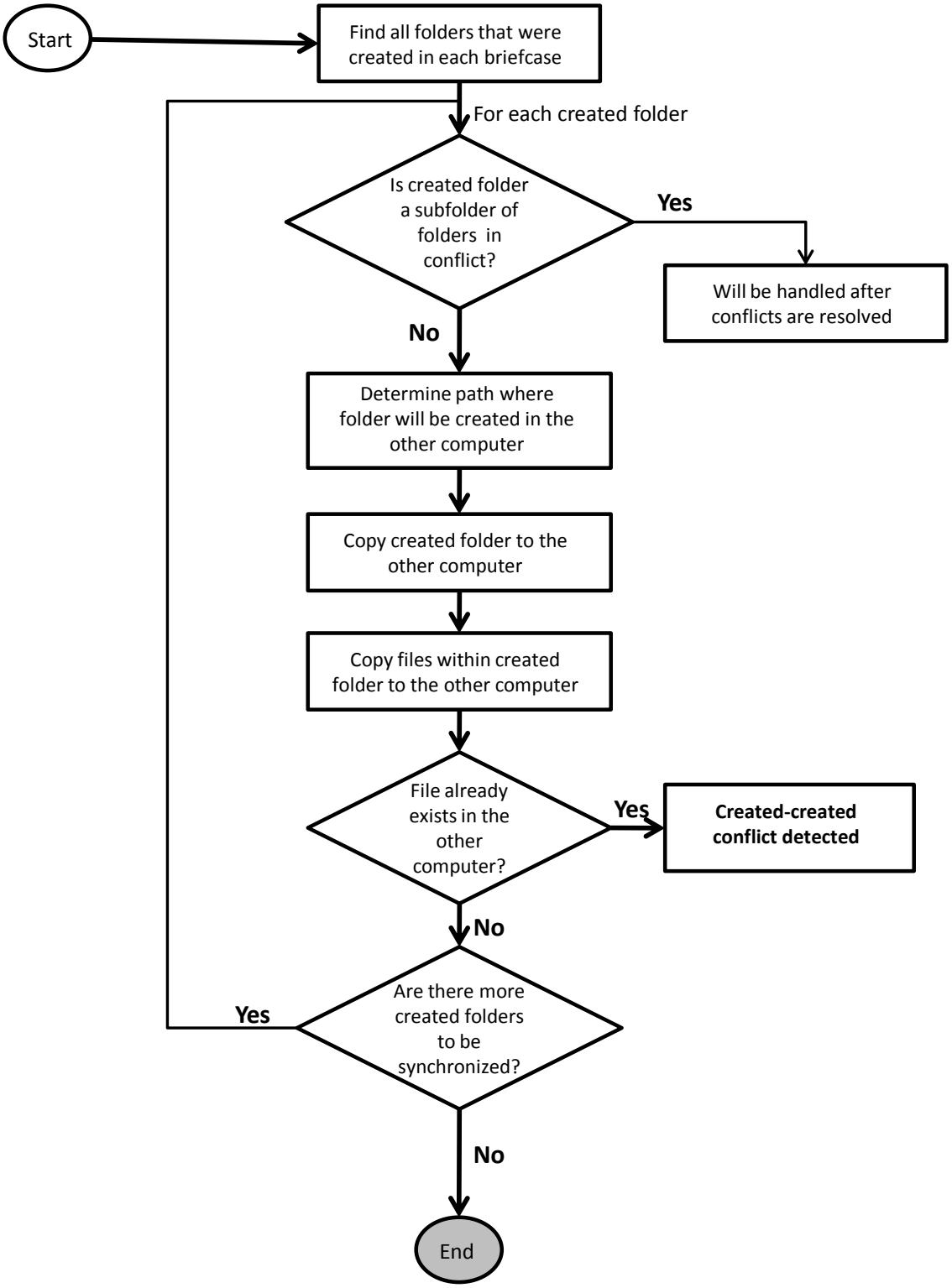

Figure A.4: The flowchart details the actions taken by the Resolver when synchronizing folders that have been created.