Smart Briefcases

Tiago Ferreira Nogueira Leite

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.

Keywords

Synchronization, optimistic replication, consistency, conflict resolution

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, 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 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.

It is also important to note that Smart Briefcases must be an efficient and a user-friendly system. If conflicts do not occur a user does not even realize that the system exists, as the system does not slow down or interrupts the user without need. 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 vi) the system must be efficient and user-friendly.

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 [1] and Live Mesh [2] 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 [3] 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 [4] and SyncToy [5, 6] 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.

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

These are the main factors that differentiate Smart Briefcases from the solutions already available.

1.4 Paper Structure

This paper is organized as follows: In section 2 the architecture of Smart Briefcases is presented. In section 3 the actual implementation of the solution is detailed. The obtained results of the evaluations performed, to the implemented solution, are presented throughout section 4. Finally, section 5 describes the related work while section 6 presents the conclusion of the developed work.

2. ARCHITECTURE

This section presents an overview of the architectural approach used in Smart Briefcases and explains how the system works. The chosen solution was designed having in mind all the goals presented in the previous section. The solution employed, as explained in the following sections, allows a fine grain control that helps minimize conflicts and gives more control over the data being modified by a user.

The system architecture of Smart Briefcases is portrayed in Figure 1.

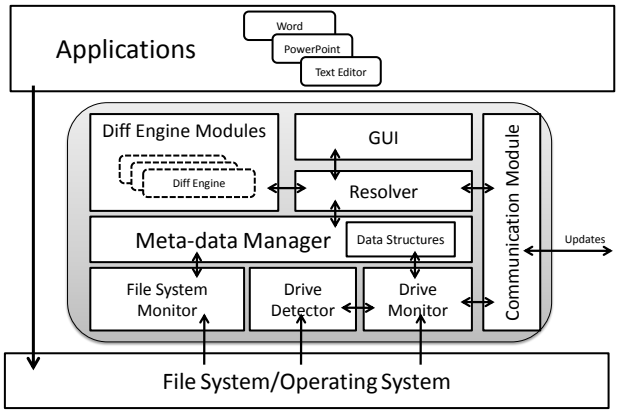

Figure 1: Smart Briefcases Architecture.

2.1 Overview of Smart Briefcases

Smart Briefcases is an application that allows a user to keep data replicated and consistent throughout all the computers he owns. 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.

With Smart Briefcases, a user can create virtual directories that are similar in interface and usage to the ones offered by the user's currently used file system. These directories, called briefcase folders, manage the files stored within them and allow the applications to transparently access and modify them. This is achieved by monitoring the user's action and intercepting each application request to the file system. This allows Smart Briefcases to know when a file or folder is modified, renamed, created or deleted. The information regarding these modifications is kept as metadata. When convenient, the user can ask the system to synchronize the modified files and folders with older versions stored in another computer.

To allow the user to independently modify each isolated replica, an optimistic replication approach is employed. Using the collected metadata, Smart Briefcases, is able to detect modifications between two different replicas and perform the required operations to synchronize them. In the end both replicas are left in the same state. However, if conflicts occur, Smart Briefcases informs the user and provides relevant information to help him resolve the conflicts and achieve a consistent state.

2.2 Monitoring Briefcase Folders and Storing Metadata

A briefcase folder is the name of any folder created through Smart Briefcases' user interface. From the moment a user creates one of these folders, the application instantiates a File System Monitor, a module that is responsible for monitoring changes performed inside a briefcase folder. Every time a file or folder, inside the briefcase is created, modified, deleted or renamed an event is triggered inside the briefcase's File System Monitor. When this happens, the File System Monitor collects all the relevant information regarding the modification performed.

The Metadata Manager is the module that contains the structures where data concerning each briefcase is stored. When a user creates a new briefcase, a tree-like structure is created in order to store information regarding each file and folder created within said briefcase. When, for example, a user creates a file inside the briefcase, an event is triggered inside the File System Monitor. The module then sends the collected information to the Metadata Manager. Sent Information consists of the name and path of the file modified, the date and time at which the modification occurred and the type of the modification performed. With this information the Metadata Manager creates a node inside the Tree structure to represent the newly created file.

Also, when a file or folder has been renamed or deleted, the node that represents that file or folder is updated with that information. The representation of one of the nodes that represent a folder inside a briefcase folder can be seen in Figure 2.

In the end, by storing information that represents each file and folder created inside the briefcase, the Tree Structure becomes a faithful representation of the directory tree constituted by those same files and folders. This information is invaluable to correctly perform the synchronization process and detect possible conflicts that might occur. An example of a representation of a Directory Tree is shown in Figure 3

2.3 Detecting the creation of Synchronization Pairs and Communication between Replicas

After creating a briefcase, the user must copy that briefcase to another location in order to create a synchronization pair. After establishing a synchronization pair each briefcase becomes an independent replica that can be modified by the

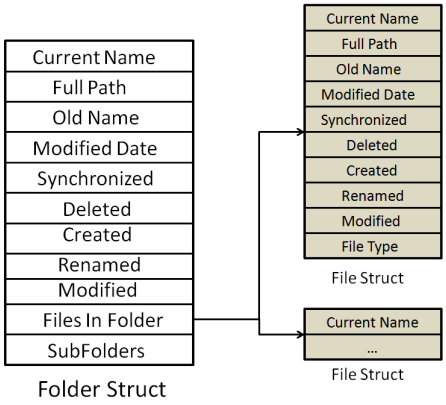

Figure 2: The image represents the FolderStruct and the FileStructs stored within. The FolderStruct is a structure that contains important information concerning a certain folder. A FileStruct on the other hand contains information concerning a file. This information is used by the Resolver during the synchronization process.

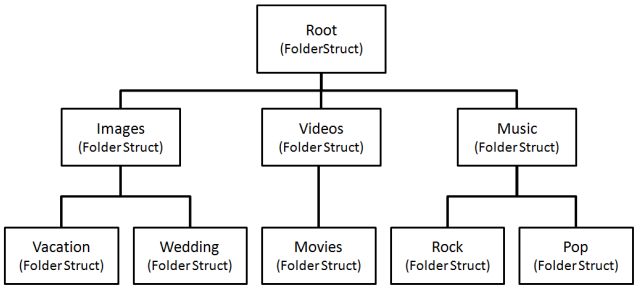

Figure 3: The conceptual representation of a DirectoryTree stored by the Metadata Manager for each Briefcase. In this image the Root folder represents a Briefcase. The Root's FolderStruct has a list that contains three other FolderStructs, each for a different of its sub-folders (Images, Videos and Music). Similarly, each of these sub-folders also contains a list that stores FolderStructs representing its sub-folders.

user anytime. When possible, the user may request Smart Briefcases to synchronize the two folders.

A synchronization pair is established every time Smart Briefcases detects that an already existing briefcase is copied to another location. A synchronization pair can be classified as "Local" if the user copied the briefcase to a different location inside the same computer or "Remote" if a user copies a briefcase to another computer using a USB flash drive, propagates it through a local network or sends it by email. From the moment a synchronization pair is successfully formed, modifications performed to each briefcase can be propagated to one another.

When Smart Briefcases is started by the user, it instantiates a Drive Monitor for each drive detected in the user's machine, including USB flash drives. The Drive Monitor monitors modifications performed to folders inside each drive. This is performed in order to detect when a certain briefcase has been created inside a drive. Every time this happens an event is triggered inside the Drive Monitor.

However, the Drive Monitor also receives events when regular folders are created. In order to differentiate the creation of regular folders and briefcases, each briefcase contains a hidden file inside, called Settings.ini. This file contains information detailing the path of the briefcase folder, the IP and port used by the running instance of Improved Briefcase and a global unique identifier that is composed by the IP, port and number of the briefcase folder.

When a drive monitor detects that a folder has been created inside the drive it is monitoring, it checks if the Setting.ini file exists inside the folder. In case it does not exist the created folder is ignored by the Drive Monitor as it is identified as a regular folder. In case the Settings.ini file exists, the Drive Monitor checks if it is a new briefcase being created or if it is a pair being formed. If the briefcase has been copied from another device it communicates with the other machine in order to establish the synchronization pair using the IP and port provided by the Settings.ini file. Communication between two devices is only possible if they are located in the same local network.

If communication between replicas is successful the instances of Smart Briefcases running on each machine share the details of each briefcase and store in structures the information required to perform future communications. When this process finishes the two briefcase folders are considered a pair but will only communicate again when the user requests Smart Briefcases to perform synchronization between these briefcases. The devices are able to communicate between them using the information collected previously from the Settings.ini file.

A Drive Monitor also intercepts events every time a file or folder is deleted or renamed in the drive it is monitoring. This is used to allow a user to rename briefcases as he sees fit or to delete briefcases that are no longer required.

If a briefcase folder is renamed, the Drive Monitor receives the event and updates all necessary structures with the new name. If the renamed briefcase is part of a remote synchronization pair, Smart Briefcases sends a request through the network in order to inform other replicas of the modification.

In case a Briefcase is deleted, the Drive Monitor is responsible for eliminating all the information regarding the briefcase and breaks the synchronization pair if the briefcase was part of one. If the deleted briefcase was part of a remote synchronization pair, Smart Briefcases sends a request through the network in order to inform other replicas that they should also delete all information regarding the deleted replica.

This allows users to have total control over the created briefcases. They are able to create new briefcases, rename them and delete them with no limitation whatsoever.

2.4 The Synchronization Process

The Synchronization Process is initialized by the user's request simply by pressing the Synchronization button located in the Smart Briefcases 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 was initialized is where all the comparisons between briefcases, conflict detection and conflict resolution are performed. All the information that is needed to perform the process are the two DirectoryTrees, which contain all the information of modifications performed to each briefcase in each replica. This is the main reason why only one replica is required to perform all the computational work.

Therefore, when synchronizing all pairs at once, the replica where the process was initiated iterates through each of the existing pairs of unsynchronized briefcase folders, gets the DirectoryTrees from each of the folders of each pair and performs all the required actions to synchronize them. The replica where the process was initiated requests the DirectoryTree of the briefcase located in the remote replica. The remote replica propagates the DirectoryTree and from this moment on, the DirectoryTrees from the synchronization pair can start being compared and synchronized.

The fact that the processing work is only performed by one replica is important to minimize the number of communications between the replicas. One replica makes all the work and discovers what actions are needed so that each briefcase reaches an identical state. The only information sent to the other replica 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 DirectoryTrees 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 synchronized. In the second phase only modifications performed to files are synchronized.

It was chosen to divide the process in these two phases for three reasons. The first reason was simply to ease the implementation of the Resolver as it was easier to differentiate the synchronization of folders from files.

The second reason was to handle the case in which files have been modified inside folders which have been renamed. By synchronizing folders first, by the time the Resolver starts synchronizing 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 synchronization. The problem would become even more complex if all the folders that compose the path where the file is stored 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 conflicts have been completely resolved can the resolver start resolving file conflicts. 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 the synchronization of folder deletions, folder renames and folder creations by that specific order. Then, if there were conflicts detected, they are displayed to the user who must give his input in order to resolve them. Only if all conflicts are resolved will the resolver start resolving file's modifications.

In the second phase the resolver proceeds through similar actions than in the previous phase. It resolves file's deletions, renames, creation and modifications by this specific order and in the end displays all conflicts that were found. If the user has resolved 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 folder in each replica and their respective DirectoryTrees are identical.

2.5 Conflict Resolution

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 an informed decision when resolving the conflict later.

An object of the type conflict stores: the structures of both 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 the choice of the user, which represents the modification he wants to keep.

2.5.1 Types of Conflicts

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 replica while the replica of the same file or folder has been renamed in the other machine.

3) **Creation:** A replica of a file has been created in one replica while in the other machine 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 machine. This conflict only occurs with files since modifications performed to folders are ignored by the File System Monitor.

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 machine.

2.5.2 *User's Choices*

The user choice field is important 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. There are three options presented to the user:

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 in which 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.

2.5.3 *The Conflict Form shown to the user*

After the deletions, renames, and creations of folders have been resolved, the Resolver verifies if conflicts were detected. If this is the case, 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. The same happens after the deletions, renames, creations and modifications have been resolved.

Unfortunately, the tools included in the 2.0 .NET framework are unnecessarily difficult to use and do not provide enough functionality to display the collected information, regarding conflicts, to users in an efficient and attractive way. In order to create the Windows form displaying conflicts a component named ObjectListView [7] was used. ObjectListView is a C# wrapper around a .NET ListView component.

When shown the conflict's form the user must decide, based on the information displayed, which replica he wants to keep. For example, in the form presented in Figure 4, 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.

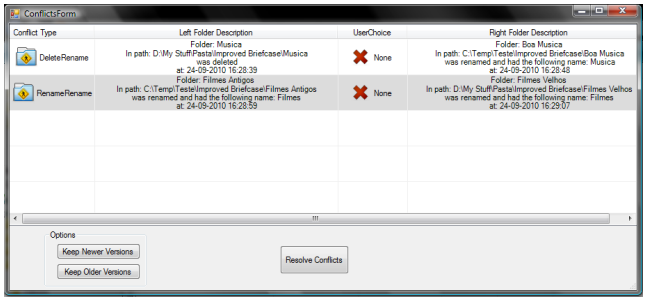

Figure 4: The form shown to the user when conflicts related to folders are detected. The figure displays a renamed-renamed conflict and a delete-rename conflict.

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 the user makes all decisions and presses the "Resolve Conflicts" button the resolver iterates through the list of conflicts and depending on the type of conflict and the option chosen by the user, performs a different action that creates a consistent state.

In case conflicts are detected when resolving files a similar form is displayed to the user. The main difference between the folders' conflict form and this one is the information displayed for each conflict. Also, an icon is displayed that shows the type of the folder in conflict. For example, in figure 5 the conflicts form shows three conflicts. One conflict happened between two replicas of a plain text file, other was between the replicas of a Microsoft Word File and the last was between the replicas of a Microsoft Power Point file.

However, this feature is currently only available for file types that are supported by Smart Briefcases' difference engines.

Extending this functionality for other file types is very easy.

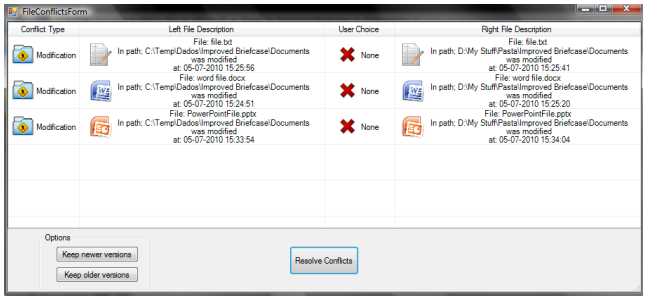

Figure 5: The figure shows a conflict resolution window. It is displayed to the user when conflicts are detected during the synchronization. In this specific image three conflicts were detected where both versions of each file were modified in both replicas. To help the user the files' icons change depending on the file type of the conflicting files. Information is also displayed describing what has changed in each file.

2.6 Difference Engines

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 [8]. 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.

The form that displays the differences between two plain text files to a user is shown in figure 6.

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.

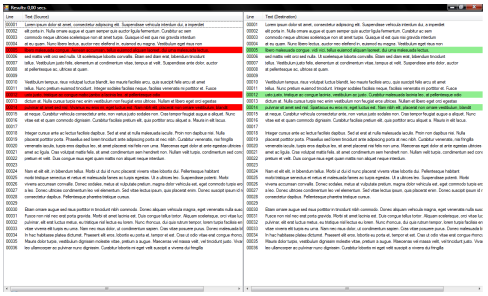

Figure 6: The difference form shows the comparison between two plain text Files. The colors show that line 5, line 12 and line 14 are different in each replica.

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.

2.6.1 External Differencing Tools

Besides the difference engine present in Improved Briefcase, it is possible to integrate other applications that compare files from different replicas. This option is provided though 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. IMPLEMENTATION

Smart Briefcases was implemented using Microsoft Visual Studio 2005, C# 2.0 and the 2.0 .Net Framework. The sub-

mitted 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 briefcase folders.

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 folder. 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.

The drive monitor module uses native code that was downloaded from the Code Project website [2]. This code receives signals from the windows operating system that concern the mounting and unmounting processes of USB flash drives. When one of these signals is received an event is triggered which allows Improved Briefcase to start or stop monitoring a USB flash drive, for example.

The graphical user interface was implemented using Windows Form controls. The exception is the conflict resolution form which uses a custom control called ObjectListView. 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 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.

4. EVALUATION

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 collected results are shown in figure 7.

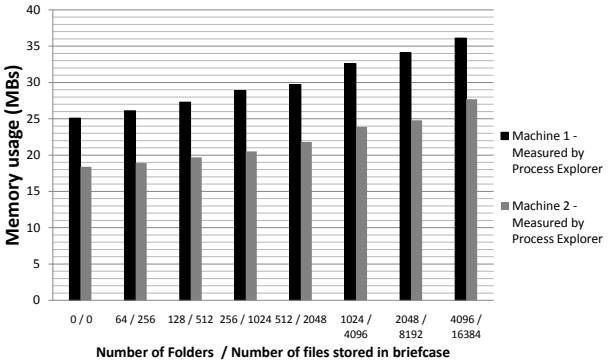

Figure 7: The graphic shows the relation between the increase in memory used by the application and the number of files and folders stored within a briefcase folder.

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 (figure 8). The propagation of deletions and renames throughout replicas takes no more than some seconds (see figures 9 and 10). 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.

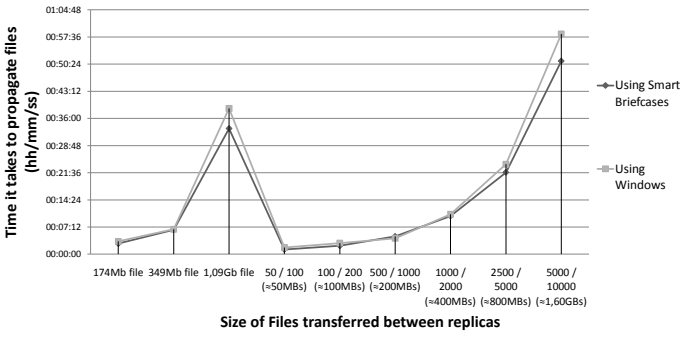

Figure 8: The graph shows the comparison between the time it takes to transfer files through the network using Windows and using Smart Briefcases.

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

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

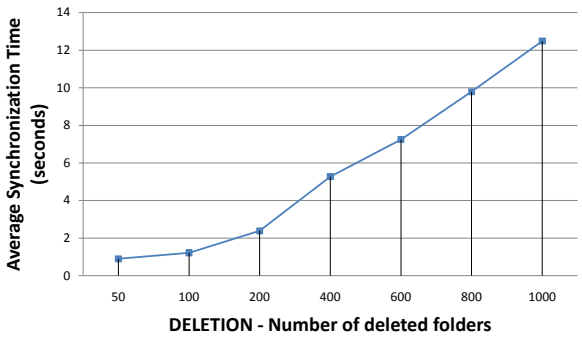

Figure 9: The graphic shows the time it takes to synchronize two briefcase folders stored in two different machines. In this case an increasing number of folders with files stored within were deleted. The two replicas were then synchronized.

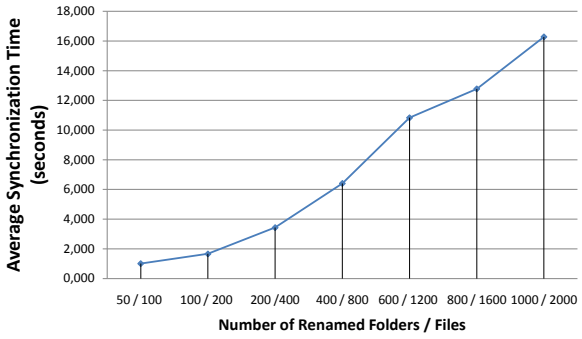

Figure 10: The graphic shows the time it takes to synchronize two briefcase folders stored in two different machines. In this case an increasing number of folders and files were renamed. The two replicas were then synchronized.

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 (see figure 11).

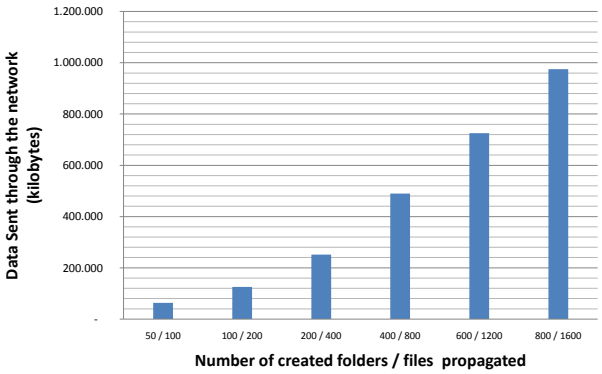

Figure 11: The graphic shows the size of the content sent when propagating files and folders that were created in one of the replicas since the last synchronization.

5. RELATED WORK

The issue of replicating content throughout several devices is already addressed by several solutions, most of which are available online.

However, most solutions either do not possess mechanisms to detect conflicts or if conflicts can be detected they are not capable of presenting information that helps the user understand what caused the conflict and how he could to resolve it. Some of these solutions are addressed in this section.

Semantic Chunks [9] 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. 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. Also, it employs some mechanisms that reduce network and memory usage and allows for a fine-grain control over updates to files.

IceCube [10] 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. It uses the collected information to simulate several possible resolutions to conflicts in order to achieve the best consistent state.

Microsoft's Briefcase [4] was created as part of Windows 95 and is still distributed with recent versions of Windows

[11]. It is a simple file synchronizer that works similarly to Smart Briefcases. However, the application has several limitations: it prevents the synchronization of renamed or moved files and when conflicts occur it does not provide any information that helps the user make an informed decision on how to reach a consistent state.

SyncToy [5, 6], is a file synchronizer built by Microsoft for Windows XP and Vista. Its goal is to synchronize large volumes of files and folders even when some of them have been renamed or deleted. Synctoy takes snapshots from shared folders. These snapshots provide enough information to detect what changes have been performed and correctly handle renames, deletions and modifications of files. Even in cases of conflict. A similar approach is used in Smart Briefcases to correctly detect delete-rename, rename-rename and delete-modification conflicts.

ActiveSync [3] is a data synchronization tool developed by Microsoft. It is available for Windows and uses Infrared [3] accessed in 06/01/2010, Bluetooth [12] or USB [4] accessed in 06/01/2010 to connect devices. It is mainly used to synchronize or backup content from mobile devices with devices with Windows installed.

Dropbox [1] and Live Mesh [2] are online storage utilities that allow a user to backup and access his files from anywhere where a computer with an Internet connection is available. Files are also automatically synchronized whenever a user is connected to the Internet. Unfortunately, as with most studied solutions, neither Dropbox or Live Mesh are able to inform the user in case of conflicts and neither solution provides the mechanisms to automatically resolve said conflicts.

6. CONCLUSION

This paper presents the architecture, implementation and evaluation of Smart Briefcases, a file synchronizer built with the objective of helping a user, who owns several computational devices, maintain all his replicated content consistent.

The proposed solution is designed in a way that monitors the user's behavior while he is accessing the shared content without slowing down the system or halting the user's work. By observing the user the system is able to collect information that is crucial to detect differences between replicas and identify conflicts if they exist. In this case the system informs the user of what caused the conflict while providing relevant information that allows the user to quickly resolve the conflict and achieve a consistent state. All this is achieved without modifying applications already used by the user.

Experimental values obtained from tests that evaluate Smart Briefcases' performance in scenarios of real world usage show that the proposed solution is able to synchronize two replicas relatively fast while maintaining a memory footprint and bandwidth usage that do not deviate from reasonable values.

As future work it is intended to extend Smart Briefcases to be able to perform several pairwise synchronization between more than two machines. Moreover, it is intended to optimize the propagation of modified files, add more difference engines and perform some improvements to the application's graphical user interface.

[3] http://science.hq.nasa.gov/kids/imagers/ems/infrared.html
[4] http://www.usb.org/home

7. REFERENCES

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

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

[3] 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.

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

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

[6] 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.

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

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

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

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

[11] 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.

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