The Document Synchronizer - A Partial Solution for Real-Time Replication

by

Ben H. Shih

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Signature of Author ... [illegible] ....
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by ...... [illegible] ....
Richard C. Larson
Director, Center for Advanced Educational Services
Thesis Supervisor

Accepted by ...... [illegible] ....
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

The Document Synchronizer - A Partial Solution for Real-Time Replication

by

Ben H. Shih

Submitted to the Department of Electrical Engineering and Computer Science
on May 28, 1996, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Document Synchronizer is a tool which brings together the database sharing capabilities of Lotus Notes with the communication channels of the Intel ProShare Video Conferencing System. The Document Synchronizer explores the use of this powerful, peer-to-peer, data channel as an extension to the database replication system within Lotus Notes. This new data pathway is leveraged to maintain concurrency across replicas of databases, and this is done in a way which maximizes the usefulness for the users. As video conferencing systems and its related infrastructures become more prevalent in the future, this type of extension will become a highly useful feature in all distributed database systems. The work that has been done for the Document Synchronizer serves as an exploration and a proof-of-concept for these extensions. It is hoped that the lessons learned in building the Document Synchronizer can be generalized for the future implementation of these systems.

Thesis Supervisor: Richard C. Larson
Title: Director, Center for Advanced Educational Services

Acknowledgments

On the Lotus side, I must give great thanks to Glenn Edelson, Paul Kleppner, Eric Pesciotta, and Rob Ullmann for their guidance and assistance with this project. Each of them have contributed to this project, and their touches are visible in all parts of this system. I must also give a special thanks to Sasha Willoughby for being a very supportive co-worker and friend during this time.

On the MIT side, I must recognize our amazing course secretary, Anne Hunter, without whom this undertaking would have been impossible. Thank you, Anne, for always having all of the answers. I must also thank my thesis advisor Professor Richard Larson for his honest and insightful comments. He has helped me to significantly improve what you now hold in your hands.

On my side, I thank all of my friends for always asking me about my work. They were the ones who kept me on my toes and they were the ones who pushed me to start and to finish this project. I also thank my family for their continual and continuous support.

The one person who has stood with me throughout this process is my girlfriend Eileen Chen, and I just thank her for always being understanding and encouraging, and also for the sacrifices that she had to make due to this undertaking.

Above all, I thank God. I thank Him for the wonderful creations that He has made for us to examine and to study. I thank Him for the mercy and grace that He has bestowed upon us. Without Him, there would be no meaning in any of this.

1 Introduction

The power of team work and cooperation has always been apparent in human society. What one person cannot do alone can be done by a team, and the whole is greater than the sum of the parts. Indeed, it is said that:

Two are better than one, because they have a good return for their work:
If one falls down, his friend can help him up.
But pity the man who falls and has not one to help him up!
Also, if two lie down together, they will keep warm. But how can one keep warm alone?
Though one may be overpowered, two can defend themselves.
A cord of three strands is not quickly broken. (Ecclesiastes 4:9-12)

Team work is becoming a bigger issue as the problems that we face today become more complex and require more expert knowledge. One person simply cannot adequately deal with all of the issues. In particular, one context for this is the business organizations of today. As these organizations globalize, their people become more geographically separated from one another, and their teams are finding it more difficult to work together. It is in response to these changes that Groupware has emerged.

Groupware is the body of software which empowers the computer to mediate cooperation and increase the productivity of teams. Another way to think of it is as software which is specifically designed to make computer systems more useful to teams of people rather than single individuals. These types of applications typically operate on one or more of the following three fronts: communication, collaboration, and coordination. The desire for communication is obvious. In order for teams to work effectively, they must be able to communicate information with each other. Collaboration describes the sharing of information. A natural part of the team process is for people to share their information as they work together to complete their project. Coordination is the control of complex tasks and processes. As multiple users are given the capability to act on shared information, some control must be erected to impose some order. Otherwise, the result would be chaos.

One class of applications which attacks all of these fronts is the shared distributed database system. These databases are a natural place for collaboration, allowing multiple users to simultaneously access shared information. In terms of coordination, most of these database systems already have built-in protocols or processes which manage concurrent accesses. These shared databases can also be naturally extended as communication mediums, allowing users to send messages to one another. Distributed database systems are further desired because their architecture matches the needs of the globalizing organization. These systems, being geographically scalable, enable users to have contact with each other and to share information despite spatial separation.

As organizations continue to grow, limits to these shared database systems have begun to emerge. Ideally, as one user modifies some shared information in a shared database, these same changes would be made immediately visible for all users. While the propagation of this information in a small and tightly connected system can be accomplished without a significant delay, as the users become separated by great distances, these delays become material. From a theoretical point of view, a solution to this problem is not immediately obvious, because the physical laws of time and space constrains the speed of this information propagation process. This problem seems to be unsolvable, and the ideal seems to be unachievable.

However, from the perspective of usage, this problem can be constrained. It is not necessary for every user to see all of the changes at all times. It is sufficient to give these users access to only the information that they need to use. Also, they only need this information when they are actually using it. In other words, it is not necessary for the users to see the changes immediately if the information is not relevant to his or her task at hand. Constraining the problem in this way leads to ideas of partial solutions to the real-time replication problem. The Document Synchronizer is one such solution. The Document Synchronizer seeks assistance from the users in determining which pieces of information is most useful to them and then leverages a temporary but powerful data channel to deliver and update the desired information.

2 Background

The Document Synchronizer is built on top of two existing applications. Lotus Notes is the shared distributed database system and the Intel ProShare Video Conferencing System provides the data channel being used to transfer the information. Before a discussion on the Document Synchronizer, it is helpful to understand some of the fundamentals of these systems. It should also be mentioned that this project took form within the Real-Time Notes Group at Lotus, which was responsible for creating applications that tie together Lotus Notes and ProShare. The author became respon-

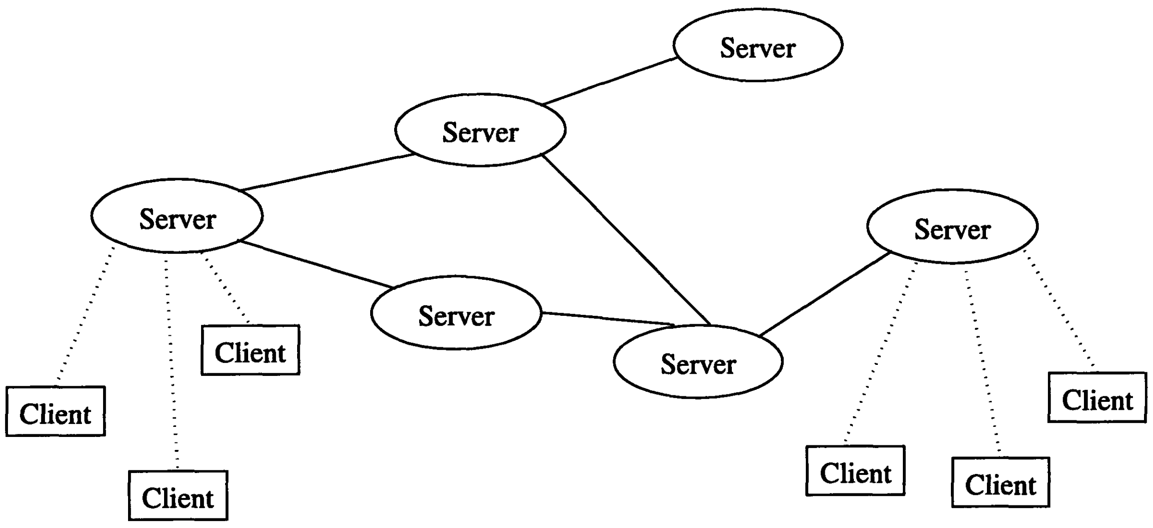

Figure 1: Notes Client/Server Architecture

sible for the Document Synchronizer project as an intern at the Real-Time Notes Group.

2.1 Lotus Notes

Lotus Notes stands out as the most prominent example of Groupware. With a steadily growing user base of more than two million, it is recognized as the industry standard. As such, it is being used by large organizations that are geographically dispersed. These attributes make it an appropriate context for the Document Synchronizer to be created in. It is indeed the case that organizations which utilize Lotus Notes in their work have offices and and personnel that are geographically dispersed yet require to cooperate to perform their duties.

One reason that Lotus Notes has become tremendously popular is its scalability and flexibility. Lotus Notes is based on a dynamic client/server architecture. (See Figure 1) Notes Databases are the basic building blocks in Lotus Notes. It is within these Notes Databases that shared information is stored. Each of these databases can be customized and used to manipulate the contained data in interesting ways. These Notes Databases reside on Notes Servers and are accessible by the set of Notes Clients that are connected to them. Additionally, the Notes Servers can connect to other servers as well. One reason for this is to create or to maintain database replicas on other servers. Thus, replicas of one database might exist on multiple servers and be accessible from different sets of clients. These communications can take place in a variety of mediums. For example, clients typically communicate with servers that are on their LAN, or they can also call into appropriately enabled servers using modems. Servers also have the ability to use modems to communicate with each other as well. This flexible architecture scales up with the organization. As more users are added, additional servers can be easily incorporated to handle the extra load. Furthermore, because servers can maintain contact with each other through regular telephone lines, it is both easy and inexpensive for the system to grow geographically with the organization.

One of the most important components in this system is the replication engine. The replication engine is responsible for ensuring that database replicas are synchronized with respect to each other. The paradigm that Lotus Notes has adopted is one of lazy updates, which means that no special effort is being made to ensure that replicas are always, or as much as possible, synchronized with each other. Replication is scheduled to occur between servers automatically. Typically, replication between servers is scheduled to occur several times in the course of a day. The idea is that even if there is a window of time in which there is a discrepancy, as long as it is small it would not affect the work that needs to be done. This works well for clients that are only a few server hops away from each other. However, if there are several servers involved in a connection, then updates might only propagate after a full day or even a couple of days. For example, a user in Los Angeles might have to wait for a long time to see the changes made by a user in New York because there might be many servers involved in the replication process between these users. This is where the Document Synchronizer fits into the picture. While the replication engine is based on an existing communication network, there seem to be situations in which this network can be augmented by temporary connections. For example, perhaps the user in Los Angeles can communicate directly with the user in New York and receive the appropriate updates through this connection, bypassing the replication process.

2.2 Intel ProShare Video Conferencing System

Intel's ProShare Video Conferencing System is a state-of-the-art video conferencing system. It provides many useful features such as multi-point conferencing, application and document sharing, a shared white-board, and a file transfer utility. As a stand alone product, ProShare already addresses some of the issues of working in groups. It is certainly an example of Groupware, enhancing the communication between members of a team. An obvious example is that ProShare is able to achieve a level of communication which almost match actual, face-to-face, meetings through its high-bandwidth video conferences. The ability to share applications and a white-board allows the users to collaborate in real-time on anything from a spreadsheet to a proposal. Additionally, the capability for file transfer allows any data that can be serialized into files to be transmitted and shared.

The combination of Lotus Notes and ProShare makes things even more interesting. Beyond the built-in applications that came with the system, ProShare also provides an Application Programming Interface called Application Services which enable the user to build customized applications. One of the main components of this extension is a general data channel which can be used to send data to and from participants involved in a conference. Lotus Notes, in turn, also has a powerful Application Programming Interface which allows the user to create applications which can access and manipulate Notes Databases and Notes capabilities. Putting these two tools together, an application can be built which leverages the ProShare data channel to transmit Notes information. This means that Lotus Notes users who are in a ProShare conference with each other can achieve a new level of data sharing. This is the premise of the Document Synchronizer, which is a tool that allows users who are in a conference to share Notes data with each other. The goal of the Document Synchronizer is to simulate the process of replication. Thus, participants in a ProShare conference will be able to update each others' Notes Database replicas through the ProShare connection.

3 Design

The design of the Document Synchronizer is geared toward leveraging all of the different features that have been exposed by Lotus Notes and ProShare. The design is based on several important assumptions which govern the workings of the Document Synchronizer. The goals of the design is expressed through the description of a typical Document Synchronizer session. The basic architecture is organized into four modules: the User Interface, the Document Table, the Callback Unit, and the Notes Engine. These modules divide up the responsibilities of the Document Synchronizer. There is also an external architecture which governs the workings of the Document Synchronizer session, consisting of a communication framework and a set of protocols.

3.1 Underlying Assumptions

The assumptions which had to be made in building the Document Synchronizer stem from two basic causes. The assumptions which limit and contain the problem of replication form one class. The second set of assumptions stems from the limit of the tools involved. That is, since the ProShare and Lotus toolkits have not been designed with this combined usage in mind, they lack certain capabilities that might be provided by more appropriate toolkits. These two classes differ in what they imply. In the first case, these are assumptions that are necessary for containing the real-time replication problem and cannot be avoided. In the second, these are assumptions which can be revised or eliminated through better utilities or tools. In a sense, the first are permanent assumptions and the second are temporary assumptions.

There are three specific areas where assumptions have been made to contain the problem of real-time replication. First, it is assumed that users have sufficient knowledge to know which documents are needed. This leads to the notion that it is sufficient to replicate documents that are specified by the user. This seems to be reasonable because it should be clear what information is needed to attack a problem and to create and analyze a solution. Second, it is assumed that this limited amount of replication will meet all of the user's needs. In other words, the guess is that it is enough to provide a limited set of documents. Again, this is an arguably reasonable assumption. In most cases, accomplishing work should not require looking at so much information. It is thirdly assumed that the speed at which these transfers and updates occur is sufficient for the user's needs. To clarify, the assumption is that the difference between true real-time and the small delay that is still experienced in using this system is small enough that it does not have an adverse effect from the perspective of the user.

Because of what is lacking in the ProShare and Notes combination, other assumptions are necessary to address this problem. The first assumption is that the users are always in a cooperative state, that there are not malicious users. This is required because the Document Synchronizer pathway, as a brand new pathway that is not controlled by existing protocols, bypasses many levels of security. Unfortunately, there was not a sufficient exposure of these security features from the Notes API for it to be included. Indeed, neither was there a component in ProShare which could adequately handle the implementation of Notes security protocols. The second assumption is that the users of this system have access to certain high bandwidth communication mediums, in this case, ISDN phone lines. The communication which happens on ProShare takes place for the most part on these lines, which guarantee a certain level of performance. However, given the growing popularity and the decreasing prices of these lines, it seems to be clear that they will become much more commonplace in the near future.

3.2 A Document Synchronizer Session

A Document Synchronizer session is based on two or more individuals joined in a ProShare conference and accessing information from their Notes databases. The assumption is that this is a pre-scheduled conference, so the users know in advance what the subject of discussion is and who the other participants are. At the scheduled time, the conference is established between two users, and the other users are joined into the conference as they arrive. When all of the participants (or enough of them, as deemed by those present) have arrived, everyone initializes their copy of the Document Synchronizer. The participants select the Notes Database that contain the information that is needed for this meeting, and they are presented with a list of documents in this Notes Database. This is the list of documents that they will be able to work with during this Document Synchronizer session.

At this point, participants can begin adding these documents to the shared list. Once they are added to the shared list, the Document Synchronizer performs its first task, which is to check whether or not the documents are synchronized across all participants. If not, the users are informed of this. At this stage, people can begin discussing material from the documents that are identical, knowing that everyone sees the same material. Later on during the meeting, if it becomes desirable to discuss a Notes document that is not identical across all participants, the second main function of the Document Synchronizer can be used. First, the participants choose a particular user whom they believe holds the best version of the Notes document. This user can then exercise the Document Synchronizer's send document function to send his own version of the Notes document to everyone else. Through these two functions, the Document Synchronizer increases the progress that can be made during the conference.

3.3 Internal Architecture

The architecture of the Document Synchronizer is organized into several modular components with separate functions. (See Figure 2) Everything centers around a specialized data structure called the Document Table, which is responsible for maintaining the list of documents that are being processed by the Document Synchronizer and also their statuses. The User Interface component receives inputs from the user and also displays appropriate results as well. The Callback Unit is the module which interfaces with the ProShare data channel. It is used to both transmit information and to respond to unexpected ProShare events, such as when a user gets disconnected. Finally, there is also a set of Notes functions that can be considered the Notes Engine. This component is responsible for accessing information from Lotus Notes.

3.3.1 Document Table

The Document Table keeps track of all of the different information that is known about every document that has been processed. Though it is basically a a list of documents, it builds on this structure significantly. A document is added to the list

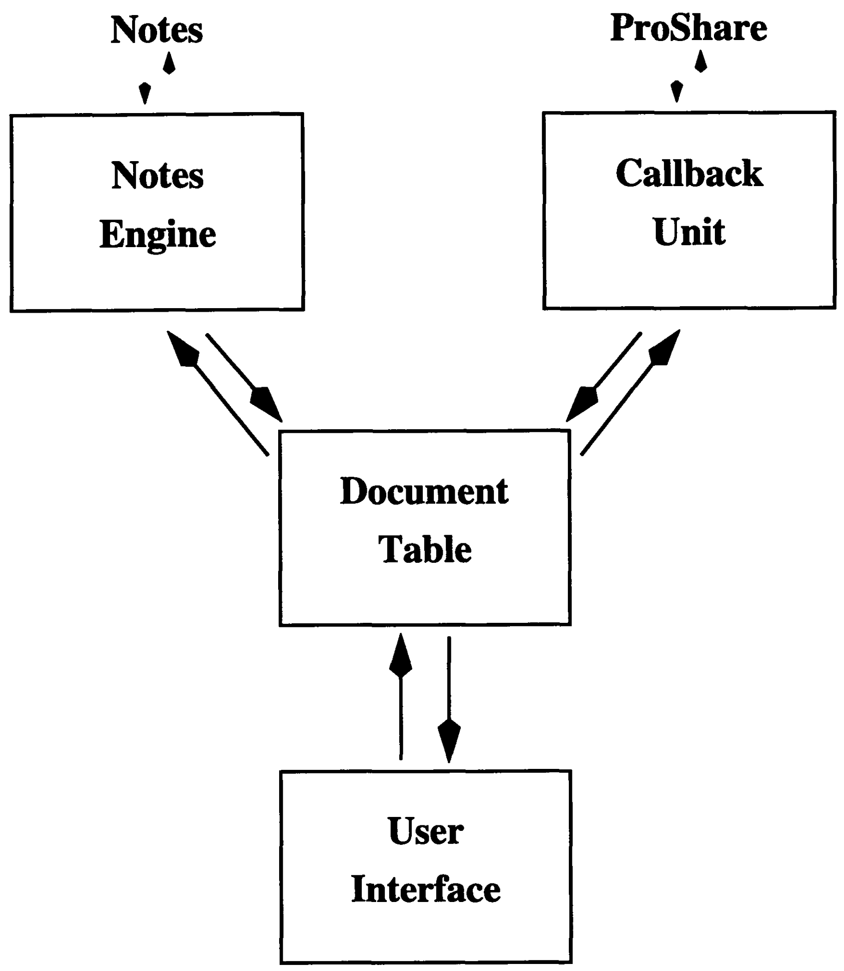

Figure 2: Document Synchronizer Internal Architecture

when any of the participants decide that they would like to use it. As soon as it is added, the Document Table directs the program to figure out several attributes about this document. First, the Document Table determines the NoteID of the document. The NoteID is a unique identifier that is assigned to each Notes Document by Notes. It is possible to use the NoteID to determine whether or not Notes Documents are replicas of each other and whether or not they are synchronized with each other. The Document Table stores the NoteID because of these features. It also uses these NoteIDs to identify the document. Now, using the information generated from the NoteID, it requests the other Document Synchronizers to search for their own versions of the document and to return the corresponding NoteIDs. For each participant that this request message was sent to, the participant's SentTo flag, which keeps track of whether or not the program has sent its NoteID to the participant, is set to True. The Document Table also sets the Status of the document to be Unknown, because it is not known whether or not this document is synchronized for all of the participants. (See Figure 3 for an illustration of this structure.)

Because of the request that was sent out, the other Document Synchronizers sends back their own NoteIDs, which is stored by within the Document Table. After the Document Table has received the NoteIDs from all of the other participants, it is able to determine whether or not everyone has the same version of the document. If so, the document's Status is set to be Synched. Otherwise, the Status is determined to be UnSynched. At the end of this process, the Document Table will have been populated with a list of NoteIDs corresponding to each of the participants. This is how the completed entries in the Document Table are created. The information

NoteID	**Status**	**Document Title**	**Node List**
NoteID	**Status**	**Document Title**	**Node List**
NoteID	**Status**	**Document Title**	**Node List**
⋮	⋮	⋮	⋮

User ID	**Document NoteID**	**SentTo Flag**
User ID	**Document NoteID**	**SentTo Flag**
User ID	**Document NoteID**	**SentTo Flag**

Figure 3: The Document Table

stored in these entries become useful for other parts of the application.

The Document Table also updates itself in response to changes in the Notes Documents that have its information stored in the Document Table. This can basically happen in two ways. First, during a Document Synchronizer session, it is possible for a document to be transferred from one participant to another. In this case, the recipient's document NoteID will change, because the document has been altered. Thus, the recipient of this transfer would broadcast the new NoteID, and all of the Document Synchronizers would receive these messages and update the information that is stored in the Document Table. The Document Table also checks its list of NoteIDs for the particular document to see if the Status of the document has changed.. If so, it will make the appropriate updates. For example, through updates, documents that were not synchronized could become synchronized, in which case, the Status will be changed to Synched. Secondly, it is possible for a Notes Document on the list of shared documents to get updated by Notes, either because the user has made changes independently or because replication takes place. Both of these possibilities will change the NoteID of the The Document. The Document Table is aware of these events. When this occurs, the new NoteID is received from Notes and stored in the Document Table. As before, the Document Table also checks the Status of the document. Thus, the Document Table attempts to maintain a consistent and updated view of the world. Also, though there are no direct communication between them, every participants' Document Tables will hold identical sets of information.

3.3.2 User Interface

The User Interface receives the input of the user and displays any appropriate changes and updates. This happens mainly through two dialog boxes. The first is the DatabaseDialog, which is where the user selects the particular Notes Database that is desired. The second is the MainDialog. All of the other user interaction occurs through the MainDialog, including the selection of documents and the displaying of document Statuses.

The DatabaseDialog is enabled as soon as a conference is started. When it is opened, the DatabaseDialog uses the Notes Engine to create a list of local Notes Databases that are accessible by the user. (See Figure 4) It then parses the titles of these database and presents them to the user. The user can then choose the database that he or she wants to use for the duration of this conference. When a database is selected, the DatabaseDialog then uses the Notes Engine to scan through it to build a list of Notes Documents that are in this Notes Database. The names and the NoteIDs of these Notes Documents are returned, and this information is passed on to

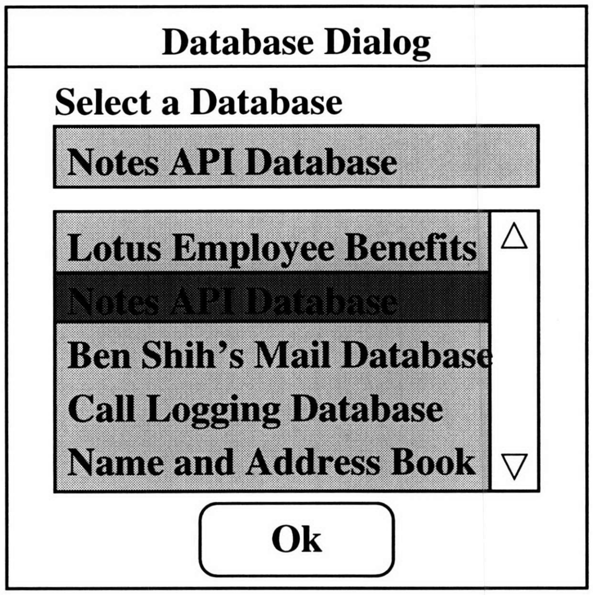

Figure 4: Schematic of the DatabaseDialog

the MainDialog. This action also closes the DatabaseDialog, as it has accomplished its task, and opens the MainDialog.

The MainDialog is opened by the DatabaseDialog, and it is immediately fed a list of NoteIDs and names of Notes Documents. This is used to populate the MainDialog's list of local Notes Documents. (See Figure 5) The names of these documents are displayed for the user, while the NoteIDs are internally stored for identification purposes. Now the user can select from this list of local Notes Documents and choose the ones that he or she would like to process with the Document Synchronizer. The selected Notes Documents are deposited into the DocumentTable, initiating a set of activities in that module, as previously specified. These documents are also displayed in the list of shared documents. If the DocumentTable determines the Status of a shared document to be UnSynched at any time after this, the MainDialog signals this to the user by appending a "`(+)`" after the document name. The user can access one

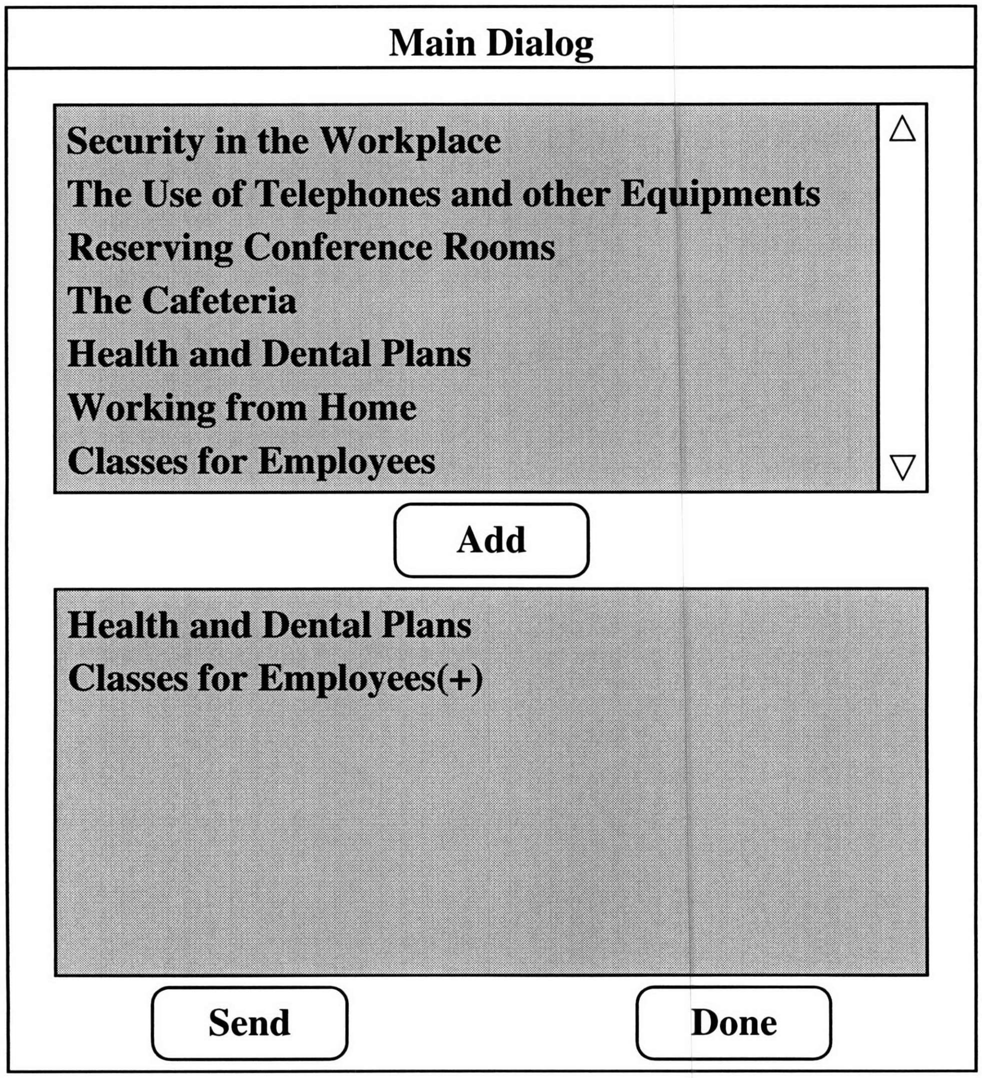

Figure 5: Schematic of the MainDialog

more feature through the documents that are on the shared documents list. The user can select any of these documents to be sent over the ProShare connection. Thus, what would happen is that the users negotiate to determine who has the newest or best version of the document, and this user then uses this feature to send this Notes Document to the others. Finally, since the DocumentTable continues to be updated by the other participants' programs, the MainDialog needs to ensure that all of the information is accurately updated. To do this, it checks through each of the documents on the shared list whenever it is idle. For each document, it queries the DocumentTable and the Notes Engine to see if there have been any changes.

3.3.3 Callback Unit

The Callback Unit is the interface between the ProShare communication channel and the Document Synchronizer. There are really two separate responsibilities for the Callback Unit. That is, it has to handle two separate classes of messages. The first is the new set of messages which has been defined for the Document Synchronizer. These messages are responsible for communicating events from one Document Synchronizer to another. For example, when a new document is added by a user, the Notes information of this document is delivered to the other programs through this set of messages. This class of messages is also responsible for the delivery of Notes Documents. If and when the user chooses to send a Notes Document, it is through these new messages that the appropriate Notes information is broadcasted to the other Document Synchronizers.

The second class of messages is the pre-defined ProShare events signals. These are messages which describe a status change in the conference. For example, these messages signal when a new user has joined the conference or when a user is disconnected. The Document Synchronizer also needs to monitor these types of messages and respond to them appropriately. For example, in the situations mentioned above, the Document Synchronizer needs to update the Document Table to reflect the new conference situation. If a participant has been disconnected, then he or she should be removed from the Document Table. If a new participant has joined an ongoing conference, then the Document Synchronizer has to take steps to ensure that the new participant receives all of the information that has been transmitted before his or her arrival. This way, his or her Document Table will be accurately updated.

3.3.4 Notes Engine

The Notes Engine is responsible for sending and receiving pertinent information to and from Lotus Notes. It is basically carrying out the commands that the other modules want Notes to dispatch. In the DatabaseDialog, the Notes Engine is used to scan through the appropriate Notes directories and generate the list of local Notes Databases. Later on, when the MainDialog needs to produce a list of Notes Documents, the Notes Engine is employed to search through the Notes Database to produce the names and NoteIDs of all of the contained documents. This information is then presented and displayed to the user by the MainDialog.

Another set of interactions occur when the Document Synchronizer receives a request to add a Notes Document to the shared documents list. Using the received NoteID as a key, the Notes Engine accesses Notes to figure out whether or not this particular participant also has the same document or a different version of it. The Notes Engine also plays a major part in the maintenance of the Document Table as well. As was mentioned, when the program is idle, the Document Table is scanned. For every document listed in the Document Table, the Notes Engine checks with Notes once again to ensure that the information has not changed.

3.4 External Architecture

It is also worthwhile to examine the external architecture of this system, that is, the structure of a Document Synchronizer session. This can be broken into two separate parts. First, the communication framework describes how the Document Synchronizer has been organized for performing its duties. Secondly, a discussion of the message protocol of the Document Synchronizers reveals how the interaction between Document Synchronizers occur at a lower level. It also proves, in a sense, that the algorithms which perform these updates work correctly.

3.4.1 Communication Framework

The communication framework is built based on two main ideas working in conjunction. One is the aggressive client, and the other is the broadcasted message. The aggressive client is a client which strives to be self-sufficient. It tries to gather all of the information that it needs by itself and only interacts with the other clients as peers. There is no server in the sense that there is no superior party which is better informed. The broadcasted message goes along as a complement of the aggressive client. The idea is that if messages are broadcasted, all of the clients can potentially receive the appropriate information.

The aggressive client interacts with the other clients as peers in order to obtain information that is needed. There is basically two things that the clients do in order to accomplish this. First, it listens to all incoming messages in the hope that it will receive information that is useful to it. After parsing the message, it may well turn out that it does not provide any new information. However, the idea is to always err on the safe side. The second aspect of this aggressive client is that it also tries to do its best for the other clients. Because there is a understanding that the other clients are also trying to act in a self-sufficient manner, each client individually tries to make sure that their peers are well-informed. Thus, each client also actively tries to maintain a understanding of the states of their peers. When it detects a situation in which its own information will be helpful, it will send the message to the client that needs it. These aggressive clients thus work together as peers in order to update each other and make progress.

The broadcasted message is employed for almost every type of communication between the Document Synchronizer clients. Because the ProShare multi-point conferences are established through specialized Multi-point Control Units (MCUs) that have the ability to broadcast messages, this type of communication is only marginally more expensive than the single recipient messages. Thus, with the perspective that the broadcasted message is really transferring more information in the sense that the message is getting through to more programs, the broadcasted message became the favored message type.

The broadcasted message fits very well with the idea of the aggressive client.

When a message is being sent in response to a request, chances favor that one or more of the other clients also need this information, because all of the clients are trying to maintain identical states. Because this message is broadcasted rather than transferred point-to-point, the other clients which happen to be listening receive this message without having to make another separate request. Thus, these aggressive clients become well-informed with relatively high efficiency. The only non-broadcasted messages were the requests, because these do not need to be received by more than on person, if it is expected that the party queried will be able to provide the necessary information.

3.4.2 Message Protocol

The communication protocol of the Document Synchronizer centers around four different tasks. These protocols ensure that the Document Synchronizer will work correctly in all of these situations. The first task is the addition of new Notes documents into the list of shared documents. The second is dealing with users who join into a conference after documents have already been added to the shared list. The third is dealing with a user leaving (unexpectedly) from the conference. Finally, the fourth case takes care of the transmitting of a Notes document from one Document Synchronizer to another.

The Addition of a Document The most basic operation of the Document Synchronizer is the addition of documents into the list of shared documents. This action is started when a user selects and adds a particular document from his or her User

Interface, but completing the process requires the participating Document Synchronizers to interact and communicate with each other at another level. This begins from the Document Synchronizer on which the user has added the document.

This particular client, equipped with a list of participants in the conference, broadcasts to them the title and the NoteID of this Notes document so that they would know to add it to their Document Tables. It also creates a new entry in its own Document Table. As was mentioned, this entry would contain the NoteID and the title of the document, along with a list of document information for all of the users that it has sent a message to. Each of these structures contain a place for a NoteID and also the SentTo flag, as mentioned. Since the Document Synchronizer has broadcasted its information to all of the current users, this flag will be set to True for all of the clients.

Because the Document Synchronizer now anticipates that the other users will eventually broadcast their own Notes information also, it simply waits for these messages. When it receives this information, it will store it into the appropriate structure. When it has heard from every user on its list, it will be able to determine the Status of the document. If all of the NoteIDs are identical, the Status is Synched, otherwise the Status is UnSynched.

Another set of actions happen on the the recipient side. When a request to add a particular document is received, the Document Synchronizer first checks to see if it already has the document in its Document Table. If so, it simply adds the new information into the appropriate node because it is able to identify the sender of the message. If the document is new, it uses the Notes Engine to get its own information and create a new entry, filling in the appropriate node. It then also broadcasts its own request message, which would in effect inform all of the other participants of its own NoteID. This process continues until all of the clients have broadcasted their information, which means everyone becomes fully informed. (See Figure 6 for an illustration of this process)

The Joining User Because the ProShare system allows new users to join a conference at any time after the conference has started, the Document Synchronizer has to deal with this situation. One notable consequence of this is that a new user can join a conference in which the other users have already begun to create a substantial list of shared documents, which means that this new user has to receive a substantial amount of information in order to catch up to the other users.

This is where the special structure of the Document Table becomes very useful. When a new user joins the conference, ProShare takes the responsibility of informing all of the other participants of that event. When this has been detected by a Document Synchronizer, two things happen. The first is that the user is immediately added to the list of participants, so this new user will be enabled to receive messages from this point on.

The second thing is that the Document Table is traversed, and for every entry in the Document Table, a new node is added for the new user. Specifically, the new user is added to the list of users and Notes information which exists for every entry in the Document Table. In addition, its SentTo flag is set to False. The task is then completed when the Document Synchronizer enters an idle loop and begins to check

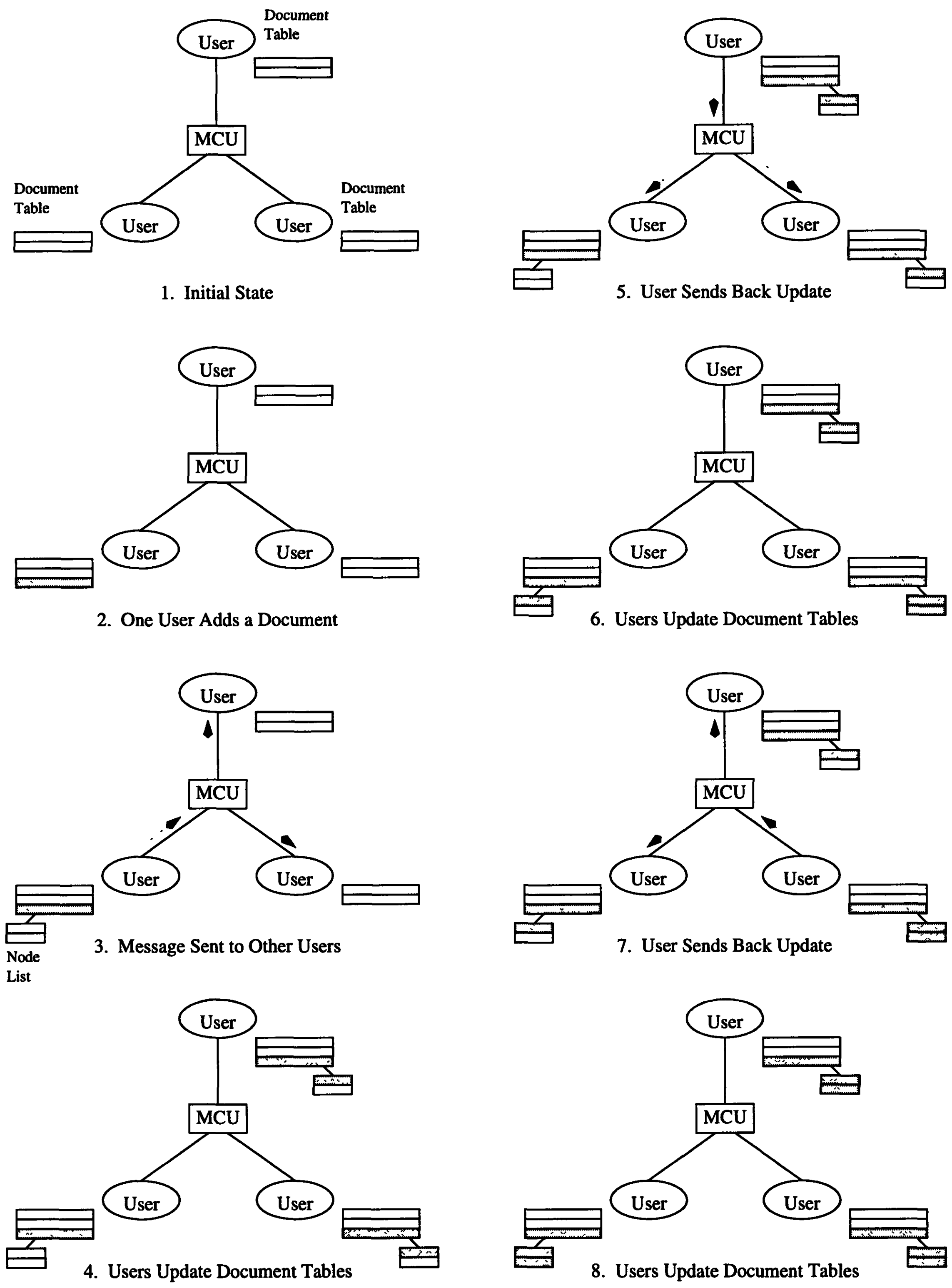
1. Initial State
User
Document Table
MCU
Document Table
User
User
Document Table
2. One User Adds a Document
3. Message Sent to Other Users
Node List
4. Users Update Document Tables
5. User Sends Back Update
6. Users Update Document Tables
7. User Sends Back Update
8. Users Update Document Tables

Figure 6: The Addition of a Document

every document on the Document Table. As it comes across these documents, it checks every node in the list of Notes information, and if there are any of them which have a SentTo flag set to false, the Document Synchronizer knows that it needs to send this participant its own information, so it goes on to do that.

From the perspective of the new user that has joined the conference, all of the other participants will know that this new user needs to hear from them. That is, since the new user does not have anyone's information, everyone needs to send the Notes information that they have on every document. The new user listens to these messages and eventually becomes fully informed. The new user is also allowed to add Notes Documents as soon as he is connected. If he or she adds a document that is new, it will done just as a normal addition, since the other users are immediately aware of him or her. If the new user tries to add a document that is already listed in the shared documents list, then his or her information will simply be added into the new node that was created when the user joined. The one drawback in this system is that new users might not immediately catch up to all of the other users, because it takes some time for all of the Document Synchronizers to individually enter into an idle loop and send to the new user the necessary messages. However, given that this user has joined the conference as a late participant, it is probably fair and appropriate that it takes some time for him to reach the stage of the other participants. (See Figure 7 for an illustration of this process)

The Leaving User A corresponding complication comes from users who can both leave a conference at will at any time or be unexpectedly disconnected from the confer-

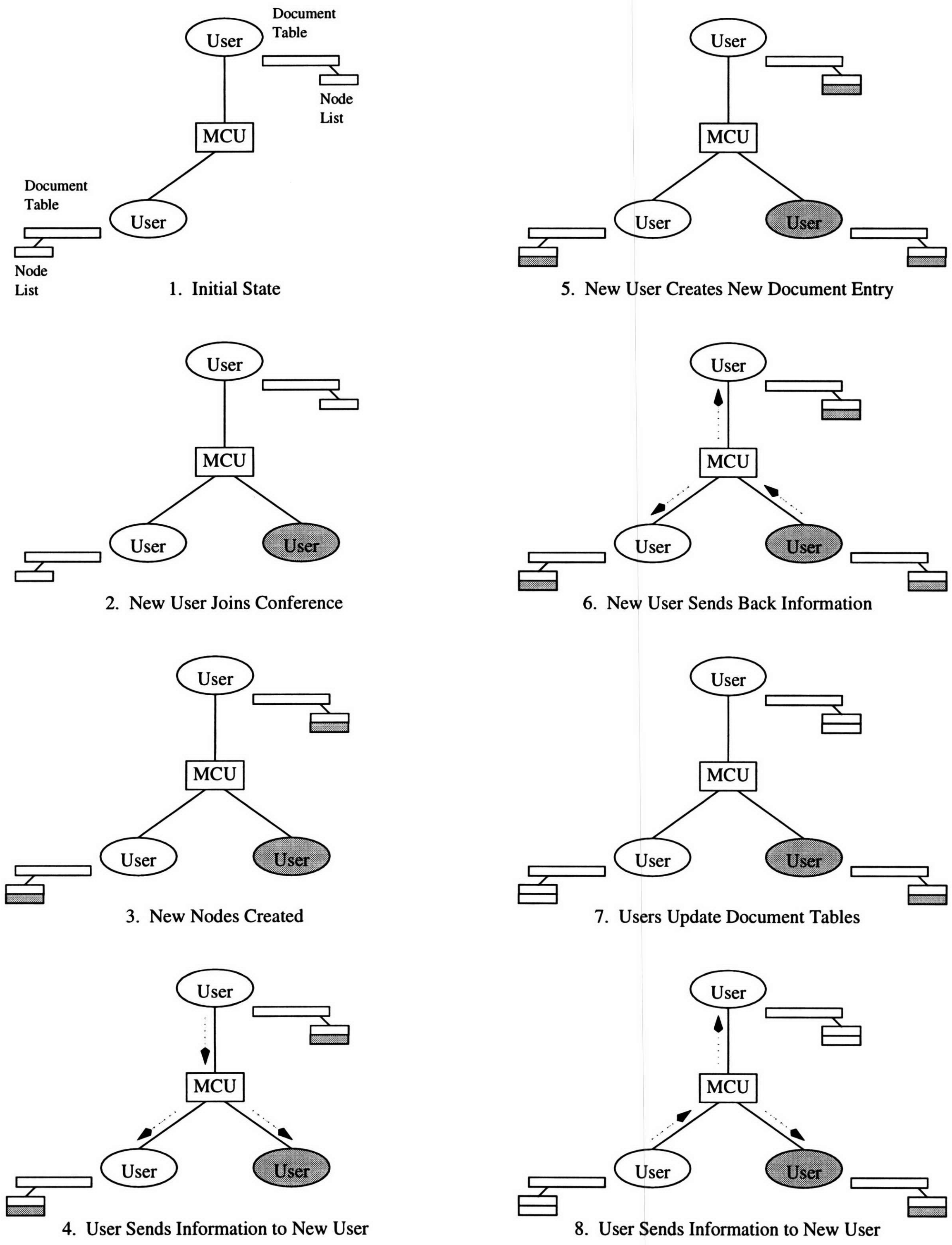
User
Document Table
Node List
MCU
1. Initial State
2. New User Joins Conference
3. New Nodes Created
4. User Sends Information to New User
5. New User Creates New Document Entry
6. New User Sends Back Information
7. Users Update Document Tables
8. User Sends Information to New User

Figure 7: The Joining User

ence. The Document Synchronizer also needs to deal with this situation adequately. Once again, ProShare sends out a signal to all of the Document Synchronizers that are still in the conference to inform them about the user that has been disconnected. The first response is that the Document Synchronizers immediately remove this disconnected user from its list of users, so that there will no longer be attempts to send out messages to this participant in the future.

Secondly, the Document Synchronizer also traverses through the Document Table and processes the user list for every document. The program removes the the node which describes the disconnected user's Notes information, so that if the Document Synchronizer had been waiting for a response from this particular user, it can now give up. Because the removal of this user can change the Status of the document (for example, if this user held the only copy of the document that was different from everyone else's copy), the Status of the document is re-checked. Because all of the still connected Document Synchronizers individually makes these repairs by themselves, the conference situation will be reflected as a whole and the Document Synchronizer session can continue without failure.

The Transfer of a Document The final action that the user can perform with the Document Synchronizer is the actual transfer of a Notes Document from one client to another. A set of messages is specially designed for this task. Before this can be discussed, however, a basic explanation of a Notes document must be given. A typical Notes document is composed of a list of different types of data. These different types consisted of numbers, text, time/date, author names, and also a flexible open type called Rich Text, which encompasses multimedia elements such as graphics, audio clips, and video clips. Because these types are defined differently, a separate message was written to transmit each of them. Actually, because the Rich Text type turned out to be quite complicated, there is no support for this data type in the current Document Synchronizer.

Equipped with these messages, a document is transmitted in a series of steps. A user starts the process when he selects a particular document on the list of shared documents and presses the Send Button on the MainDialog. At this point, the information is broadcasted out to all of the other users. Thus there are two implicit assumptions. The first is that verbal negotiation has occurred between the users in the conference and an agreement has been reached that this particular user owns the best version of the document and he or she should send it out to everyone else. The second assumption is that every other user desires to have this new version of the document. That is, all users are willing to have their versions of the document replaced. The pressing of this Send Button assumes that all of these assumptions have already been satisfied, and the program proceeds to send out the Notes Document.

The Document Synchronizer now basically retrieves all of the Notes fields of the document and then methodically sends it out to the other clients. The process begins with the Notes Engine, which is directed to go through the selected document to produce a list of the names and the types of the fields that are in the document. The Document Synchronizer then sends an initialize message to signal that a document transfer is about to begin. Then it proceeds through the list of fields and sends a specialized message for each field in the list. This message contains the field name, type, and value, which is sufficient information for the recipients to either create or update the field in its version of the Notes Document. When the Document Synchronizer has finished processing the whole list, it sends a final shutdown message to let the other Document Synchronizers know that the transfer has completed.

On the recipient side, when the Document Synchronizer receives the initialize message, it also accesses the Notes engine to open the appropriate document, preparing it for modification. As the subsequent messages come in, it is able to parse the contents and make appropriate updates in the document. Finally, when the shutdown message is received, it saves and closes the document in its new, updated state. This final save is done in a special way such that it simulates the replication engine. This basically makes it look as if it was replication which updated this document.

4 Implementation

The Document Synchronizer was implemented using two critical tools within an environment that was somewhat volatile. These two tools were the ProShare Application Services API and the Notes API. The volatile environment was due to a combination of several factors and necessitated the application of several programming techniques.

4.1 The Tools

As an application which integrates Notes and ProShare, the Document Synchronizer depended heavily on two tools: the ProShare Application Services API and the Notes API. The Application Services API provided a way for the Document Synchronizer to communicate with ProShare. The Notes API allowed the application to interact with Notes and to access and modify Notes Documents.

4.1.1 ProShare Application Services API

The ProShare Application Services API exposed a variety of calls for controlling the many different elements within a ProShare conference and for sensing the different events which may occur during a conference. In fact, the ProShare Application Services API provided much of the framework for the Callback Unit. In building the Callback Unit, the API provided the commands to access the ProShare data channel and to monitor the participants in the ProShare conference. Event handling was also controlled through this API. In particular, the events were directed to the Callback Unit, which allowed the application to respond to the different event signals, as described earlier. For example, it was through this API that the Callback Unit received signals when users left and joined the conference.

Application Services also provided the data channel that was used by the Callback Unit. This was a basic but flexible data channel without any pre-defined messages. Thus, the message protocol specifically used by the Document Synchronizer had to be designed from scratch. However, because there was no pre-defined messages or syntax, there was no constraint on how these new messages were to be designed, which was a significant benefit. In addition, these data channels had certain valuable guarantees. Application Services guaranteed the delivery and the correct sequencing of messages as long as the conference participants were still connected, and even if these participants were lost, notification was immediately provided. Thus, though a message protocol had to be designed from scratch, it did not have to be very complicated because of these guarantees. Application Services also exposed the new feature of multipoint conferencing, providing the ability of establishing one-to-one, one-to-all, and one-to-select data channels. These channels allowed the Callback Unit to multicast messages to a controlled set of clients as a part of its operations.

4.1.2 Notes API

Parts of the Notes API also became very useful in implementing the Document Synchronizer. Two of these features were the powerful, all-purpose Notes Database scanning and Notes Document scanning routines. The Notes Database scanning function was the main routine used for collecting essential information about Notes Documents. It was both a fast and flexible routine that was configurable to exactly match the task. Similarly, the Notes Document scanning function allowed the program to scan through a Notes Document and sequentially process its fields. This became very important for the Notes transferring aspect of the Document Synchronizer, because the strategy was precisely to go through the Document and send the information one field at a time. It was also necessary to take advantage of some undocumented flags within the Notes API. In order to pretend to be a Notes replication engine, these flags had to be used to control and manipulate some internal Notes information. In particular, these flags allowed a Notes Document to be altered as if it had been updated through replication.

4.2 The Environment

The implementation environment of the Document Synchronizer posed some difficulties during the development phase. The main issue was that at the time, the ProShare system only existed at a beta stage. As such, it was in a unusually unstable state. First, because the ProShare application had not been optimized at the time, it was a very memory-intensive application. In building the Document Synchronizer, this became a very big problem because the Document Synchronizer required the simultaneous use of ProShare, Notes, and sometimes a debugger or other utilities. Because these other applications, particularly Notes, also had large memory footprints, memory usage was almost always pushing the limit of the computer, which was a high performance machine by every measure. It became a frequent occurrence for the memory of the system to be exhausted during a test execution session, which then required the computer and the other applications to be restarted.

The second issue was the dependability and performance of the ProShare beta system. Whenever a crash occurred during a test session, it became necessary to determine the source of the error, which were often coming from the ProShare system. Also, a crash typically required the whole computer to be restarted, again due to the instability of the system. Furthermore, this problem is compounded due to the performance of the unoptimized beta code. On the average, it required approximately twenty minutes to shut down and restart the whole system, which is a considerable amount of time. An additional issue was that revisions to the beta code continued to be made during the implementation phase of the Document Synchronizer. When these updated versions of Proshare were adopted, they also sometimes introduced new and unexpected problems for the Document Synchronizer. To get a sense of how these problems hindered the implementation process, almost *sixty* percent of the day was spent on shutting down and restarting the machines.

Within this volatile environment, it was necessary to use several techniques to increase the efficiency of the implementation process. One of the most important was prototyping. During parts of the implementation process when new functionality had to be explored, lightweight prototypes, which would have a much smaller chance of causing memory or other failures, were generally used. Then the complete system will be built based on using the prototype code as a framework. This style of development helped to avoid many of the other crashes that would have been happened. One example of this was a prototype which tested the properties of Application Services. This prototype established a connection between two machines and facilitated the exchange of simple version information. The code written for this prototype was consequently incorporated into the Document Synchronizer.

Because each crash and reboot was so expensive, it became optimal to explore different techniques of gathering information about a test run after its termination. One particular technique was to use a listener which would listen to and record the Windows message interactions which occurred during a session. This became very powerful in this case, because it gave a much greater sense of where the errors are coming from, ensuring that the maximum amount of progress is being made with every session.

5 Conclusions

The current Document Synchronizer has most of the capabilities that were originally intended for it. It is able to completely handle the addition of documents, the joining of users, and the leaving of users. It partially handles the document transfer process. What is lacking here is the sending of Rich Text fields, which turned out to be quite complicated. It has been proved that the sending of Rich Text can be done, but there was no time to implement it. Since this means that there cannot be any audio and video clips, among other things, it is a great loss that needs to be remedied. On the Notes interface side of things, there are also some other issues which need to be resolved. The current version of the Document Synchronizer has only been used with a test Notes Database and cannot yet work with Notes Databases in general. The problem is in the MainDialog where the titles of the Notes Documents are displayed. Because there is not a universal title field in all Notes Documents, what is needed is another set of interactions which will direct the Document Synchronizer to choose the correct field to display. In general, the system can also be considered somewhat fragile, although that may also be a consequence of the ProShare beta code.

In using the tool with the beta version of ProShare, performance was found to be acceptable, thus guaranteeing that the final performance level will certainly be sufficient. Unfortunately, different mediums were not tested, with the LAN connection being the only medium ever used. It is still open how the performance would be affected if ISDN lines were used as the conference medium. This question could not be answered because the beta version of ProShare only had the LAN connection system. It was also unfortunate that due to time constraints, this system was not put to use by actual users, so there is not a general sense of how much a system like this can contribute to a meeting. However, it seems clear that even if a tool like this was not used to enhance a meeting, it can still augment the replication engine of Notes by providing this alternate pathway that is selected by demand.

The next steps of development should answer some of the questions which remains open. As of now, ProShare has reached a final state and has been released to the public. This means that there is an opportunity for the Document Synchronizer to be enhanced, without the same problems of instability that were encountered. The first thing would be to add the features for the sending of Rich Text fields. Once this is added, the Document Synchronizer will become basically feature complete. Then the next step would be to test this system on different mediums, especially ISDN lines, since that is the network infrastructure that we expect to be dominant in this usage. Finally, usability tests should be conducted for the system to determine its true usefulness as a meeting assistant and its effectiveness as a replication enhancement.

After that, the next stage would be to consider how some of our assumptions about the usage of the Document Synchronizer can be made unnecessary. The biggest issue is security. Basically, because this system allows a Notes Document to be sent from one site to another, it is bypassing all of the security and authentication procedures which occur on the Notes Server. Because a user has complete control of his or her own local Notes Database, he or she can very easily forge a Notes Document and surreptitiously send it to another user. This cannot be allowed to happen in a system of this type. One idea is that this new pathway must also go through the authentication and security measures used by Notes Servers. Thus, it might become necessary to establish some kind of temporary server to broker the Document Synchronizer transactions. Perhaps there are also other strategies suggested by the implementation of the Document Synchronizer, but it is clear that the unsecured transfer of this data cannot be allowed in the final solution.

6 References

Coleman, David, and Marshak, Ronni T. Changing Your Organization with Groupware. *Fortune,* September 19, 1994.

Greif, Irene, and Sarin, Sunil. Data Sharing in Group Work. *ACM Transactions on Office Information Systems. 5*, 2 (April 1987), 187-211.

Intel Corporation. The ProShare Application Services Programming Manual. 1996.

Kawell, L., Beckhardt, S.,Halvorsen, T., Ozzie, R., and Greif, I. Replicated Document Management in a Group Communication System. *Second Conference on Computer-Supported Cooperative Work,* 1988.

Lotus Development Corporation. Groupware: Communication, Collaboration, and Coordination. *Lotus Development Corporation Strategic White Paper,* 1995.

Lotus Development Corporation. The Lotus Communications Architecture. *Lotus Development Corporation Strategic White Paper,* December 1994.

Lotus Development Corporation. The Lotus Notes API User's Guide. 1995.