Ad-hoc Collaborative Document Annotation on a Tablet PC

Albert Huang
Department of Computer Science
Brown University
ashuang@cs.brown.edu

Thomas W. Doeppner and Ugur Cetintemel, Readers
{*twd,ugur*}*@cs.brown.edu*

May 2, 2003

1 Abstract

The use of technology as an effective educational tool has been an elusive goal in the past. Specifically, previous attempts at using small personal computers in the classroom to aid students as collaborative and note-taking tools have been met with lukewarm responses. Many of these past attempts were hampered by inferior hardware and the lack of an efficient and user-friendly interface. With the recent introduction of Tablet PC products on the market, however, the limitations imposed on software developers for mobile computing systems have been dramatically lowered. We present a collaborative annotation system that allows students equipped with tablet computers to work cooperatively in a either an ad-hoc or a structured wireless classroom setting.

2 Introduction

The Electronic Student Notebook project aims to investigate the potential of an electronic notebook as an educational tool, with the intention of eventually supplanting the paper notebook in the classroom. Currently, paper notebooks have the advantage of a virtually negligible cost combined with widespread availability and adoption that makes them the default choice when taking down notes or annotations. They also, however, are often difficult to organize and cumbersome. Maintaining separate notebooks for each class a student participates in quickly makes organization for a single term difficult, and it is typically the case that notebooks are discarded soon after the course ends.

The electronic notebook has the potential to address the organizational difficulties of paper notebooks by providing an intuitive and user-friendly interface for managing documents. Notes taken during class, along with lecture slides or documents distributed by the professor, can be saved and easily accessed later on by the student with the click of a button.

In addition to being able to do all this, however, the electronic notebook provides us with the ability to do something we could not achieve with a typical paper notebook – collaboratively annotate documents and lecture slides. It is often the case that students learn better through the explanations and viewpoints of their peers, who are usually more similar to the student in terms of educational background than the pro-

fessor. Students may often find it easier to understand a difficult concept once they have seen notes taken by other students in the same class.

In this paper, we present a robust system for collaboratively annotating documents in a classroom setting. Due to the diverse nature of documents and file formats, our system aims to support as many different document formats as possible, providing an intuitive annotation interface that is not unlike the paper notebook.

3 Design Rationale

Like most systems, the design must be based on the needs and requirements of the end user. In this section, we consider a few of the various scenarios that we hope to address and outline some guiding principles that aid us in designing our system.

3.1 Usage Scenario

At the beginning of class, the professor distributes lecture slides for that day. The slides are immediately multicast to students who are present in the classroom, and again on demand as latecomers trickle in. As the professor conducts the lecture, students begin taking notes. The professor draws on her electronic slides much the same way she would draw on transparencies, and her notes are multicasted to everyone present.

Certain students find the professors' notes a little easier to understand if viewed in a certain way, and make private annotations on the same set of slides. Some students have agreed to work closely together in the course, and choose to share their notes with each other, so that as they annotate the slides, they each see exactly what notes the others are taking. When the slides become too cluttered with notes, they are able to choose whose notes to show and whose to hide.

Later on at night, a student decides to review the material presented during class. Since all the documents and annotations have been cached locally, all that needs to be done is choose which ones to review.

Since their is no need for students to frantically scribble down everything the professor has written, students are able to focus more attention the materials being presented. Notes can be taken not only on the initially distributed slides, but also on the professors' own notes made in real-time.

3.2 Requirement Specification

Clients should be able to:

- create and annotate documents at any point in time, even when isolated from a structured network.
- annotate a variety of difference document formats (e.g. PDF, Word, Powerpoint, HTML, etc.)
- select sharing policies for annotations on documents. This could range from not sharing any notes at all to sharing with a select number of clients to sharing indiscriminately with all active clients. Policies should be set on a document wide basis, so for any given document either all the notes are shared, or none at all.
- distribute shared documents and annotations to other clients in a secure and efficient manner. Given our wireless network environment, a reliable multicast system should serve this purpose well.
- retrieve previously shared and annotated documents for review and further annotation.

In addition to these requirements, we would like to be able to apply minor changes and revisions to documents after we have begun annotating them, and reconcile the old annotations with the updated document. This is a difficult problem that has not seen a satisfactory solution to date, and we do not attempt to fully address it in this paper. We do, however, provide an interface in the API for applying changes and revisions in documents, and leave it to specific implementations to optionally reconcile updated documents with old annotations.

4 Architecture

4.1 Document structure

In order to support as many different document formats as possible, we define a document as a collection of individual files. The motivation to diverge from the traditional notion of a document as a single file comes primarily from the desire to support formats like HTML, where a document is often split across multiple files, especially when the document contains images that we also want to preserve.

No attempt is made to interpret what kind of files are actually part of the document. We leave this task to the implementation, and treat all files in a document as a simple collection of data.

A document, while able to have multiple files, must have a single file designated as its entry point. This allows us to determine for the UI which file to actually load when displaying the document. For single-file documents such as PDF and MS Word, the sole file is always the entry point. For multiple-file documents like HTML, the entry point is the first page to display (quite often `index.html` or a similarly named file).

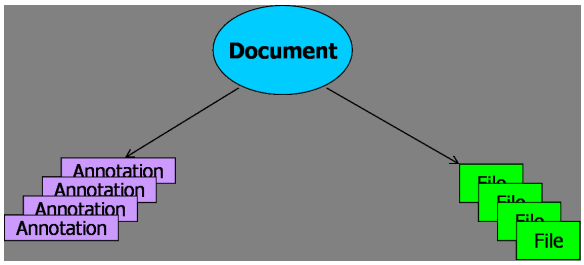

Figure 1: representation of a document.

4.2 Annotations

In order to support as many different document formats and methods of annotation as possible, we adopt a fairly abstract and high level concept of an annotation. An annotation is defined by three properties

User The identity of the user who created the annotation.

TimeStamp The date and time that the annotation was received on the local client. For all locally originated annotations, this will be the creation time. For all other annotations, it will be the time the annotation was received.

Data The meat of the annotation, this is also the most abstract. We leave it to the implementation to define the structure of the annotation data, requiring only that it be comparable to other objects (for caching purposes) and serializable, as defined by the Microsoft .NET CLR.

In our current implementation, annotations are further described as strokes of ink interpretable by the Microsoft Tablet PC SDK. In previous implementations, they have been interpreted as Microsoft Office Comment objects interpretable by the Office Automation libraries. Our framework, however, makes no distinction between these different types and will accept any serializable data structure as a valid annotation.

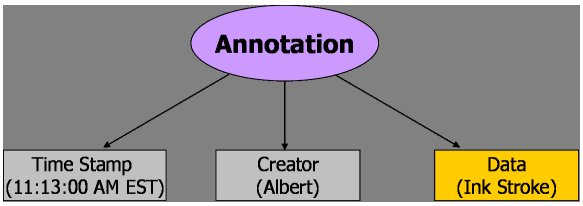

Figure 2: representation of an annotation

Each document, along with the actual data files, also has a set of annotations associated with it and an access specifier that allows the user to set sharing policies on a document wide basis. The access specifier operates in three modes.

private The default mode, this is analagous to the traditional model of note taking where no notes are shared with any other participants.

public All notes taken by the client on the document will be multicasted to all participants. If every participant operates in public mode, then the result is almost identical to the shared whiteboard model of annotation.

custom The user can specify an access control list, indicating exactly who to share annotations with. Annotations will be securely unicasted to the selected clients.

4.3 Document Cache

The document cache serves as the repository where all documents and annotations are stored. When a user imports a document into the system to be annoated, a copy of the document is made and imported into the cache, leaving the original files untouched. The cached document is made available for viewing and sharing, and is central to both real-time collaborative annotation as well as offline review of past collaboration sessions.

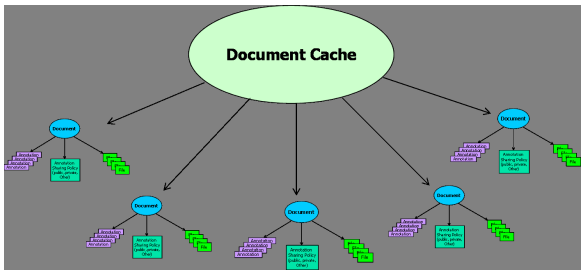

Figure 3: representation of the document cache

The document cache maintains two notions of a file – a local file, and a cache file. The distinction is made to separate the actual system level implementation of storing and locating cached files from the interface we want to present to users accessing files in the cache. For example, there is no reason other modules in the system need to know about where on disk a cached file resides.

4.3.1 Local File

The local file is little more than a wrapper around a file on disk. It provides integrity services such as computing an MD5 hash of a file, and read and write operations on the file, but little more.

4.3.2 Cache File

This is what the document cache exposes to all other modules. A cache file is a wrapper around a local file and presents read and write operations, but does not allow direct access to the local file. A cache file also provides an interface to iteratively write downloaded portions of a file to disk, indicate when the local copy is complete, and whether or not the local copy is intact (by comparing computed MD5 checksums with checksums advertised by the sender).

4.4 Session

The session is the heart of the collaboration system. Conceptually, a session is the logical representation of a group, class, or project meeting. Each session contains its own document cache, which it uses to maintain persistent local copies so that the user can return later on and review the results of a collaborative session.

Our concept of a session loosely follows the client/server model in that a single user must first host a session before any other users may participate. Once clients have joined a session, however, the role of the host is diminshed to the point where clients may continue to operate even after the host disconnects (e.g. the professor leaves class early while some students are still engaged in a discussion) Prior to joining a session, a client must first be authenticated and authorized by the host before it is allowed to participate. All communication after the initial joining process is encrypted via a symmetric key encryption scheme, where participants are given the session-wide key upon being authenticated by the host.

Lastly, it is possible to operate a session in what we call saved mode. In saved mode, a client is not connected to any network, but is able to view the contents of a session's document cache and annotations, as well as any other data accumulated throughout the session.

4.5 Reliable Multicast

Most of the messages that will be passed around on the network will be targeted at multiple listeners. This subsystem provides anonymous reliable multicast services built on top of UDP that provides the same delivery guarantees as a typical reliable transport protocol such as TCP. This system was provided by Roberto Almanza as a part of his Masters Project, and is critical for scalability of our system to large classrooms.

4.6 Document distribution

Document distribution is done in six stages. We require the fairly complex exchange of messages to reduce the likelihood of unnecessary network traffic in a session with a large number of participants. Consider the following scenario: Ten students are participating in a collaboration session, looking over three documents. If an eleventh student joins late, she must obtain those documents somehow. A naive approach would be to query for available documents, and for clients to immediately send all the documents they have. In our scenario, however, this could potentially result in the same three documents being sent ten times each – clearly a waste of network resources. To solve this problem, we look to typical human group interaction. If a person asks a question to which multiple people may know the answer, people will usually wait a random amount of time and respond if no one else has responded. In our case, there is no need to tell a new participant about a document you have if someone else has already advertised it.

4.6.1 Document Query

When a client first joins an ongoing session, it will multicast a query for available documents.

4.6.2 Advertisement

Upon hearing a query, a client will multicast an advertisement for all the documents it has cached. [1] A document advertisement will also be multicasted when a client imports a new document into its cache. The advertisement describes the individual files that make up the document.

[1] This is actually what we're trying to avoid, because in a session with n participants each with the same m documents, a single document query will generate $n * m$ advertisements instead of the desired m. This will be fixed in the future.

4.6.3 File Query

When a client detects an advertisement for a file it does not yet have, it sets a timer to request the file. If the timer expires before the client detects a request for the same file, a message is multicasted requesting the file.

4.6.4 Offer

When a client receives a file query for a file it has cached, it sets a timer to respond with a file offer. If the timer expires before the client detects another offer for the same document, a message is multicasted offering the file.

4.6.5 File Request

When a client receives an offer for a file, and the client sent out a file query, it unicasts a file request to the first client that makes an offer.

4.6.6 Transfer

If a client receives a unicasted file request, then it will immediately begin multicasting the file.

5 Implementation

5.1 Environment

Previous versions of the Electronic Student Notebook project, of which this is a continuation, were implemented with C++ and the Microsoft Foundation Classes library. This time, however, we chose to switch to C# and the Microsoft .NET Framework for a number of reasons.

First and foremost, the .NET Framework and C# language provide an abstract, high level API in which to create user level applications. As mentioned earlier, there is built in support for the concept of serializable objects. Instead of needing to worry over packet-level protocols and the order in which we pack the bytes in, we are able to focus are attention on the content of the messages being passed. Although this will result in relatively decreased performance, we make the reasonable assumption that the platforms we are targeting have sufficient processor, memory, and network resources to support our demands. The systems that we have been testing on all have 1GHz Transmeta processors with 20 GB disks and 256 MB RAM, comparable to a standard desktop computer today.

Secondly, the speed of development that we achieve using the Microsoft .NET toolset could not be achieved in any other development environment. The performance losses that we suffer as a result of unoptimized message passing and a bytecode language are heavily offset by the rate at which we are able to progress.

5.2 API

We separate the core components of our framework, which are described above in section 4, into a shared library that we expose to specific implementations. Functionality for creating and participating in sessions, importing and annotating documents, and loading saved sessions are all built into a DLL.

5.3 User interface

The actual interface exposed to the user specifies what document formats it is able to annotate, as well as defines the exact structure of the annotations and how to display them alongside the document. Currently, we have built a GUI based on the Tablet PC SDK that interprets pen strokes as annotations. Supported document formats are exported into Windows Metafile (WMF) images and used as an image underlay on which users are able to write and draw notes. A print spooler is currently being implemented that would allow applications

to print directly to WMF images that could be imported and annotated.

In this specific implementation, we have opted to ignore much of the document specific data such as hyperlinks and text and treat entire documents as collections of images. This hampers our ability to revise and update documents, but also provides a more intuitive interface for taking annotations. We are also developing a GUI linking against the same core library that embeds Microsoft Office application windows in our UI and automates them to insert annotations. By working directly with the Office file formats, we are able to preserve all of the original document data, as well as efficiently and effectively apply minor updates and revisions to documents while still preserving the accuracy of our notes.

5.4 Supported Document Formats

Currently, our system supports annotating Microsoft Word and Powerpoint documents, with support for PDF expected in the near future. We do not yet have a satisfactory model for annotating HTML, largely because our user interface discards all notions of hyperlinks and text. Being able to annotate without losing document data was a major consideration in our design, but for the first implementation we felt the intuitive pen interface gave us more in annotation versatility than what we lost in document information. When the print spooler is finished, we will also be able to annotate virtually every type of document format using the same interface.

Because document formats are so diverse, it would be infeasible to preesent a single unified annotation interface that could annotate more than one or two different types of documents without discarding some document information. We have made progress in creating annotation interfaces that annotate without losing data, as exemplified by our Word and Powerpoint automation interface, but this requires either the availability of automation interfaces, or for us to implement our own interface that can parse and understand the given document format. Document types that satisfy the former requirement are rare and far between, and the latter is beyond our current technical abilities.

6 Further Research

In the fall of 2003, our collaborative annotation system will be deployed in an actual classroom environment to analyze its effectiveness as an educational tool. With a recent donation of Tablet PCs from Microsoft, we are able to equip a number of students with Tablet PCs to use for the class. If successful, we would expand the study to include additional classes and subjects beyond computer science.

We have not currently integrated the security and authentication subsystem, and it would be interesting to see its effects once it has been incorporated into the project.

7 Conclusion

With the introduction of powerful and affordable tablet PCs on the general market, more and more students will be using them in their daily lives. We introduce a system for investigating their potential as educational tools in the specific area of collaborative document annotation. This system is both robust and adaptive, allowing us to work with a variety of different document formats and operate in an unstructured wireless environment where internet access is not available.

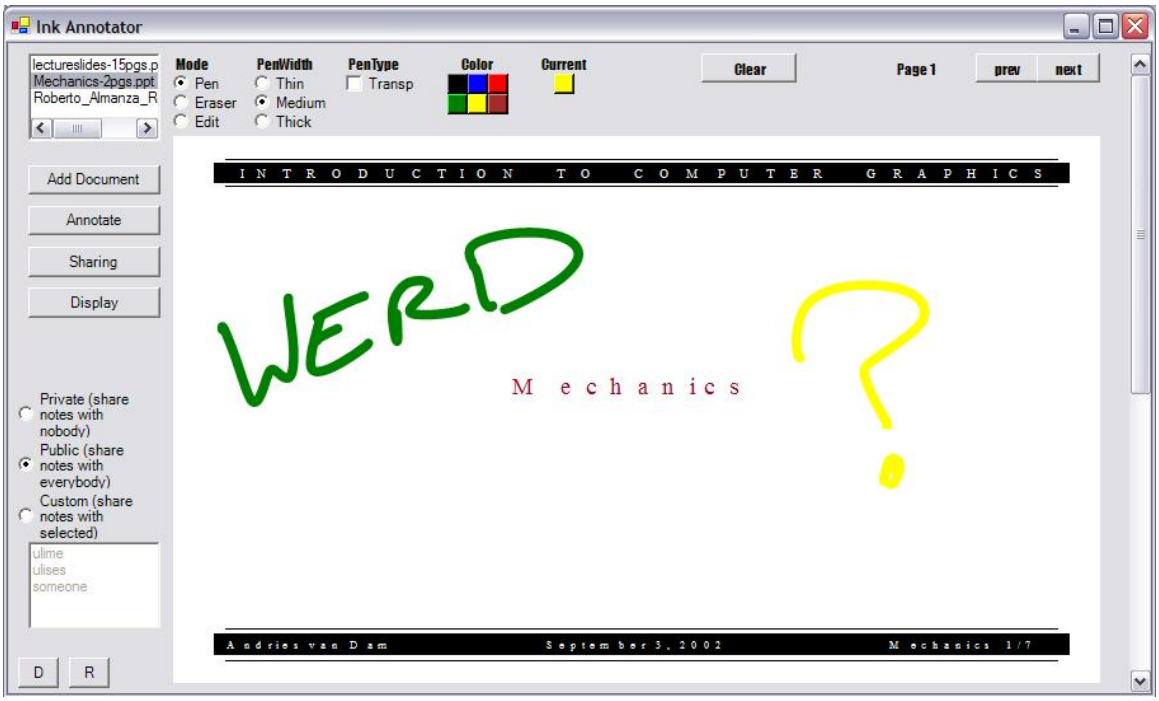

Figure 4: A screen shot of the current user interface. Documents available for annotation appear in the top left. The user can select sharing modes in the bottom left, and drawing on the current document with the stylus will result in annotations.

References

[GRI] Griess, Peter. *A Distributed, Revision-Controlled Document Storage System for the Electronic Student Notebook*, Senior Honors Thesis

[DOE] Doeppner, Thomas W. *An Electronic Notebook*, NSF Project Proposal