The Educational On-Line System

Nick Williams

Imperial College
University of London
UK

njw@doc.ic.ac.uk

William Cattey
Robert French
Bruce Lewis

Massachusetts Institute of Technology
USA

wdc@athena.mit.edu
rfrench@athena.mit.edu
brlewis@athena.mit.edu

ABSTRACT

The Educational On-line System (EOS) is a suite of programs which enables students and lecturers to exchange course materials within the Project Athena environment at the Massachusetts Institute of Technology in Cambridge, MA. Early versions of this software used a command-line interface. This paper outlines the implementation of a user interface to the file exchange system, using the Andrew Toolkit (ATK) to produce an X Window System application.

1. Introduction

For the past two years, the MIT Writing Program has held some classes in the "electronic classroom", where twenty workstations are connected to the campus network. In a typical session the instructor might present a letter as an exercise for the students to take, rewrite and turn in. The professor can then hold a review session at his workstation, giving students immediate feedback. In a traditional classroom setting, students would have to go home, type and photocopy their letter, and shuffle papers around for the first ten minutes of the next class session.

Electronic means of file exchange have been available for a long time. When Project Athena, MIT's experiment in integrating computers into education, still used timesharing machines, there was a *turnin* program that was no more than a shell script that piped *tar* output into *rsh*. This meant that the instructor had to be UNIX literate to do anything with the files.

Another interface was introduced in the 1987–88 school year that used the Sun Network File System. UNIX file protection modes determined who could access which files. The user interface, when taking inventory of files turned in, had to do the equivalent of a *find* over the remote file system. As the number of files grew, this operation got more and more expensive.

This NFS-based system was more sophisticated than than the original turnin. It allowed the submission, editing, and pickup of files. It was designed to support a classroom paradigm for files, used by students and teachers who were not necessarily computer literate. Files were edited with GNU Emacs and exchanged either by a simple textual-command oriented interface, or from within the editor itself.

With the introduction of the Andrew Toolkit [Pal88a], which contains multi-media editing facilities combined with an Emacs-like command set, it was decided to create an application that was more user friendly. A design was produced by Hagan Heller [Hel89a, Bar87a], from which the new system was implemented.

This paper describes the latest version of the Educational On-line system, which aims to bring both the interface and the file exchange mechanism closer to the requirements of its users. [Env86a].

1.1. Hierarchy of Papers

The UNIX concept of a *file* is not designed to deal with the different states a paper evolves through in a classroom situation. The file exchange system provides a classroom–oriented paradigm for organizing submitted files into divisions of courses and types of paper. It is common to refer to files stored in the file exchange server as *papers*. The types of paper are:

TURNEDIN
: This is a paper which has been turned in by a student, but has not yet been graded.

TAKEN
: The paper is currently being graded.

GRADED
: The instructor has finished grading the paper and it is now available for the student to collect.

PICKEDUP
: The student has retrieved the graded version of the paper.

HANDOUT
: The paper has been submitted by an instructor for general use by the class.

EXCHANGE
: This type of paper may be submitted by any user and is accessible to the entire class. This enables groups of users to exchange papers among themselves, informally, for review.

These types determine who is allowed to read and/or write the various files. Graders have access to everything. Students are not allowed to write handouts or read other students' non-exchange files. The next section describes the flow of papers through these different types.

1.2. The Life Cycle of an Assignment

The student creates a paper in one of two ways: he or she produces one from scratch or, edits a copy of one of the available *HANDOUT* papers.

While editing, the student may submit the paper several times, using the *EXCHANGE,* facility, to allow other students in the same course to comment and discuss their work as it evolves.

After the student has finished editing his or her assignment, the paper is turned in for grading. The paper would have a status of *TURNEDIN.*

When the instructor finally chooses to grade the paper, he or she selects the paper from the list and the paper changes its status to *TAKEN.* After making annotations, the instructor returns the edited paper to its author, and the paper would become *GRADED.*

Finally, the student, who originally submitted the paper, would request to see if any papers had been returned from grading. The student would see this paper in the list and, after collection, the paper would change its status to *PICKEDUP.*

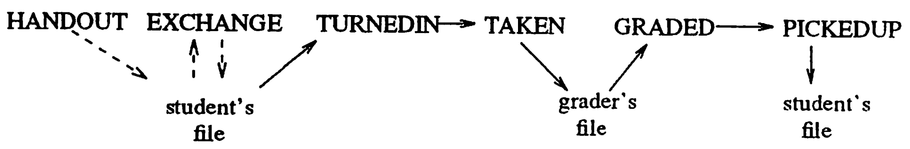


Figure 1: *The path and states of a paper in the system*

1.3. Overview

The interface can present two different views depending on the user of the program. The student can run the command *eos,* which provides functions for turning in papers to be graded and retrieving marked papers from the instructor. The instructors can type the *grade* command, which provides functions for reviewing which papers are available for grading, editing and annotating submitted papers, and then returning papers to their authors.

Two functions are common to both interfaces. These are represented by the buttons **Help** and **Guide.** The **Help** button invokes the Andrew help system in a fresh window starting at the help page relevant to the interface being used, and the **Guide** button creates a new window from which users can view the "Style Guide". A hypertext document containing advice on how to create a well-written document.

These interfaces rest upon the file exchange client library, a collection of functions to facilitate interaction with the file exchange servers, which in turn control access to course-related files by maintaining a database of courses, grades, students and files [Mar88a].

1.4. The Andrew Multi-Media Toolkit

The Andrew Toolkit solved a central problem in exchanging and annotating papers. It did so in a way that no other application, or toolkit could match. The problem is: How can a grader, looking at a complex formatted document, make annotations that are easily related to the paper which the author composed? The best way is to integrate the annotator, the formatter, and the editor. The grader sees the formatted document, and places notes in it where they are relevant. The author views the annotations in the formatted display, but receives a revisable document. The annotations themselves can easily be deleted, and their ideas incorporated into later revisions.

The Andrew Toolkit offers not only a multi-font text object and a text editor built on it, but also other media types such as line drawings, spreadsheets, and equations. It also offers the ability to add new media types by just compiling a new media module and putting it into the library of dynamically loaded modules. This was the method used to create a new media type of *note* which is used to annotate papers. Student authors can produce either simple text, richly formatted text documents, or complex documents containing text, equations and line drawings.

Having an annotator integrated with a WYSIWYG editor and then integrating that with a file exchange mechanism provides a synergy of functionality.

2. The User Interface

Normally, users would be editing text, but the facilities are provided for the user to edit raster images, animations and any other Andrew Toolkit object, all of which can be sent through the file exchange system.

The editing system is very similar to that of GNU Emacs, having the same type of buffer system in which multiple files can be stored and edited simultaneously. It also uses many of the same keyboard commands.

2.1. Exchange Papers

From the definition, an exchange paper has to be accessible from both the grading interface and the student interface, so that all groups can access the papers. To this end, both interfaces contain a button, labelled **Exchange,** that allows access to these papers. When this button is pressed, a new window is created that contains a list of the current exchange papers and also a set of buttons with which the user can submit papers of their own and take copies of other papers. Authors are permitted to delete their own papers by selecting a menu command that attempts to remove all the selected papers.

The list of papers describes each paper by giving the author and the title, both of which are defined when a user submits a paper for the first time. When a user wishes to edit or examine another paper, he or she first selects a range of papers using the mouse, and then clicks on the required function, listed below:

- Print, which sends a copy of the paper directly to the printer,
- Edit, which copies the paper into a file belonging to the user and immediately prepares the editor for editing the file.
- Save, to take a private copy, but not to do anything with the saved copy just yet.

The same method of showing a list of papers and providing buttons for performing actions on those papers is used when looking at handouts and when an instructor is selecting which papers to grade.

2.2. The Style Guide

This is a document accessible from both interfaces, that contains information and hints on how to write a good paper. The document is designed in such a way that it is internally cross-referenced by use of the hyperlinks provided in the Andrew Toolkit. These allow a user to click on a labelled button that resides in the text and cause a new window to appear containing another document.

2.3. The Students' Interface

The student's window contains a row of six buttons and an editing region (see figure 2). The major functions provided in this interface are for submitting papers to be graded and for retrieving marked papers.

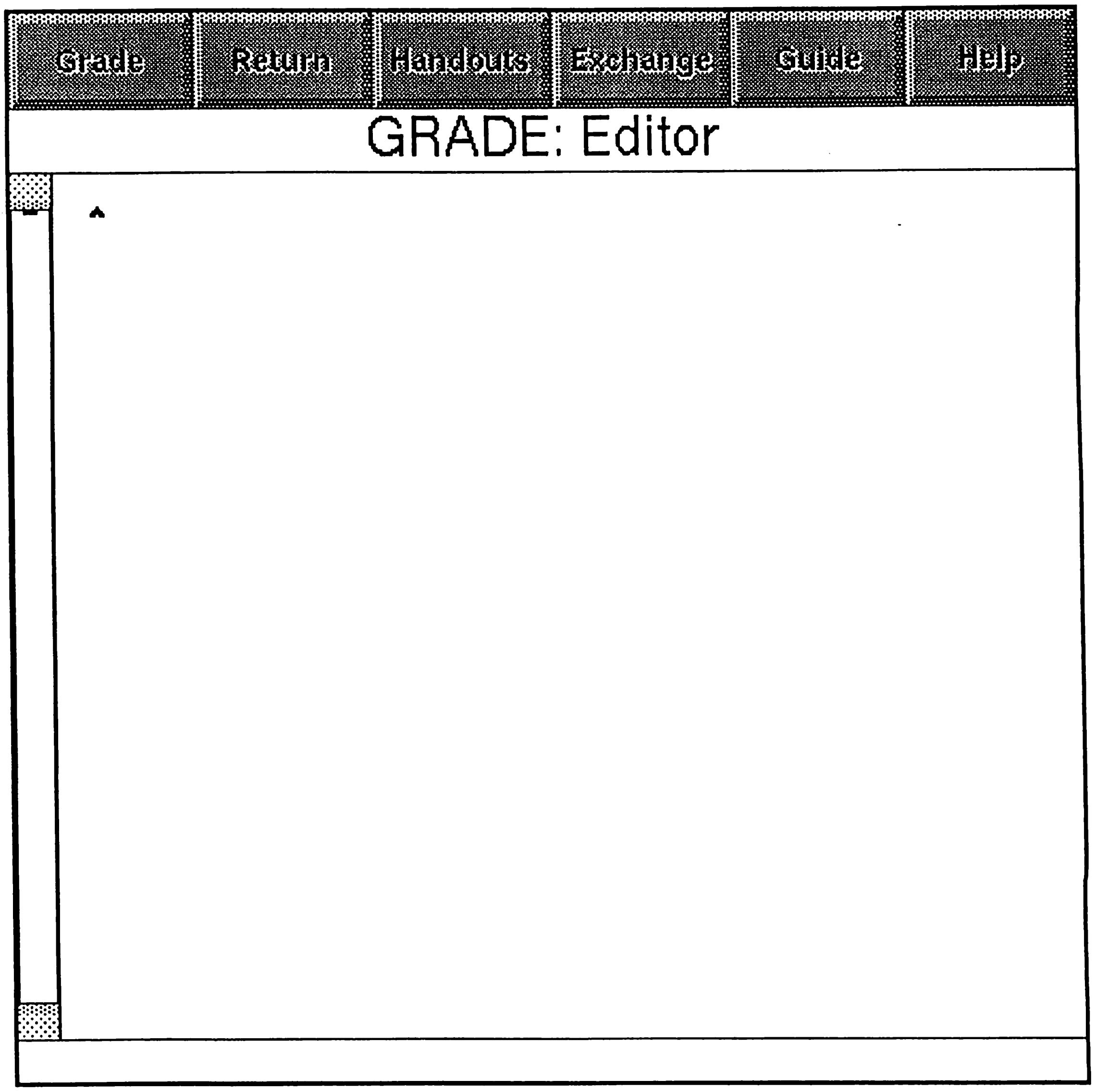


Figure 2: *The Student's Interface*

2.3.1. Turning in assignments

To turn in an assignment, the student has a button labelled Turnin which pops up a new window asking for the details of the assignment to be turned in. The user can specify the name of a file to submit or use the contents of the buffer currently being edited. The student must also supply a number for the assignment that is used to categorize the papers for the graders.

Once the user is satisfied with all the details, he or she presses an **OK** button to submit the assignment.

A **CANCEL** button is also provided which allows the user to abort the submission procedure at any time before the **OK** button is pressed.

2.3.2. Collecting Assignments

To collect assignments, the student has a **Pickup** button. When this button is selected, a new window is created with the list (if any) of papers which the student can collect. The student can then select which papers for EOS to collect. When a paper is received, as well as placing it into a file, it is also placed into a buffer so that the student can view the file immediately.

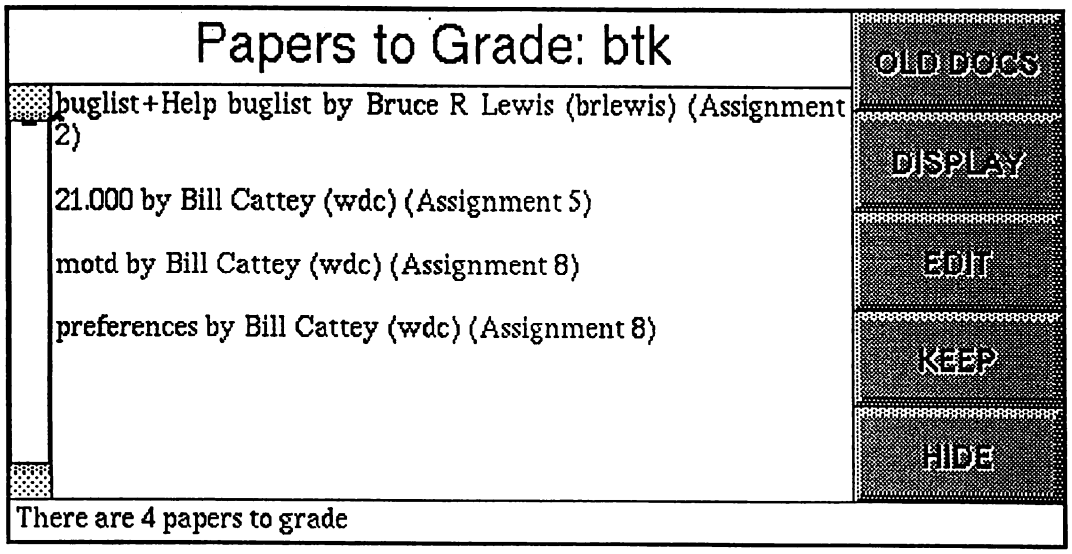
2.3.3. Picking up Handouts

If the user should want to collect any of the course handouts, then he or she would use the **Handouts** button, which creates a window of the same style as that used by the Exchange papers, although not providing the student with the facility for submitting papers. The student can then select any papers he or she wishes, and collect them for printing or editing.

2.4. The Lecturer's Interface

The lecturer's interface is very similar to the student's interface. As with the student interface there are buttons for Help and the Style Guide. The Exchange operation is exactly the same for both the lecturer and the student. The Handouts interface contains the additional **Submit** operation so that the lecturer can make new handouts available. This functionality is analagous to the **Submit** operation in the **Exchange** window.

In place of the **Turnin** and **Pickup** buttons, a single button **Grade** is present. The Grade window, produced on selecting this button, lists all papers that have been turned in. It allows the grader to select papers, annotate them, and return them to their authors.


Papers to Grade: btk

buglist+Help buglist by Bruce R Lewis (brlewis) (Assignment 2)

21.000 by Bill Cattey (wdc) (Assignment 5)

motd by Bill Cattey (wdc) (Assignment 8)

preferences by Bill Cattey (wdc) (Assignment 8)

There are 4 papers to grade

Figure 3: *The Grading window*

2.4.1. Annotating

Using the multi-media aspect of the Andrew ez editor, a *note* object was created. There are three menu commands appropriate to the note available within ez:

- "Add Note": To insert an annotation at the current cursor position.
- "Open Note": To display the text associated with the note icon currently selected.
- "Close Notes": To close all the annotations and show only the icons.

To annotate a paper, the grader clicks **Edit** from the grading window, and clicks the mouse to specify where in the paper the note should be placed. The grader then selects **Add Note** from the ez "Page" menu card. A small icon looking like two tiny pieces of paper, one on top of the other, appears. By clicking on the icon, the note is opened and the grader can type annotations. Clicking in the black region at the top of the note closes it.

When a student has picked up an annotated paper, he or she selects **Open Notes** from the ez "Page" menu card. Then he or she scrolls through the document reading all the notes.

3. How The File Exchange Library Works

The File Exchange Client Library (or, for short, the FX library) helps C programmers write applications which imitate the paper handling done in a classroom by providing procedures for storing and retrieving of papers.

Additional Procedures are provided to allow creation and modification of access control lists, to control who has grader permission and the ability to modify turned-in student papers.

The procedures were designed to balance three ideas: simple clients, supportable by a remote procedure call protocol, a simple server.

The "simple clients" idea means that it should be easy to write client programs that send and receive papers. This idea has been successfully captured in the FX library: a client program uses *fx_open()* to open a course, *fx_retrieve()* to get a file, *fx_send()* to send a file, and *fx_close()* to close a course. Access control lists are retrieved with *fx_acl_list()*, added to with *fx_acl_add()*, and deleted from with *fx_acl_delete()*.

Clients also need the ability to operate on lists of papers. These lists could be parameterized by author, paper name, or assignment number. A specific parameter such as **author "wdc"** can be named, or "*" for all authors can be named. The protocol defines a data structure called *paper* which holds each of these elements. The *fx_list()* operation returns lists of these structures that tell which specific papers are in the server that match the given paper parameters. The *fx_send()* and *fx_receive()* operations use the paper structure to identify which papers to copy.

The "remote procedure call protocol" enables the construction of a library that does its work on a remote host over the network. To the author of the client program, the library behaves as if everything is done locally, but it does look a little strange. Why would anyone use *fx_send()* when a simple UNIX *fwrite()* would do? The answer is that the additional complexity was added to make it a fairly simple matter to perform the operation over the network. The Athena File Exchange Service is written using the Sun Remote Procedure Call library. All data exchanged between the client programs and the server had to be translated into an external data representation that could be sent over the network. The Sun XDR (for eXternal Data Representation) generator took care of writing the necessary data translation routines.

The "simple server" idea is embodied in the file system and the database. The file system is the UNIX file system. Papers are stored in ordinary UNIX files. Lists of files belonging to particular authors and assignments are generated by the database. Since the server must provide service to several courses, and potentially hundreds of students, it must do the minimum amount of work.

3.1. Authentication

In order to reliably enforce access control, some method must be used to confirm that a user is who he or she says they are. This is done through the *Kerberos* authentication system. When a client issues the *fx_open()* call, the client side of the FX library assembles a request and calls the kerberos library to create an authenticator. This authenticator is then sent over the network to the server. When the server needs to make sure this user is who he or she says they are, it takes the authenticator and calls the kerberos library to confirm that it is a valid authenticator for that user.

3.2. Where To Find The Servers

The client programs, *eos* and *grade* need to be able to find and connect to the file exchange servers. How this information is specified affects load balancing and server cooperation. The server location is currently specified in one of two ways: an environment variable **FXPATH**, or through the Hesiod name service. For our limited work, having everyone in the course set an environment variable to lists the server hosts turns out to work. When we go to having many more users, the service location information is available with as much ease as one would ask for a host number to name translation through *bind*.

3.3. Server Co-operation

We are experimenting with cooperating servers. The previous version of the File Exchange service relied on NFS to store papers. The load was divided by putting only a few courses on any single NFS server. The problem was that there was only one NFS filesystem per course, so if a particular NFS server was down, those courses were out of luck. For the File Exchange service to be fully operational for all of MIT meant that all the relevant fileservers had to be running. This proved to be a strain on the operations staff near the end of the term. The new File Exchange service has a dedicated database to speed up list generation, so more courses can live on one server, and fewer servers can serve the same number of people. But we expect the need to arise for a degraded mode of operation where some servers are allowed to go down.

The current method of cooperation is to replicate the database of papers, but not the papers themselves. All servers that are up agree to hold a copy of the database. As long as the majority of the servers are up there is no problem. When a server realizes there is no longer a majority of servers running, it politely refuses to do any work until there is once again a quorum.

Once we are satisfied that the replication, synchronization, and quorum identification code is working, we will establish mechanisms for load sharing, graceful handoff to secondary servers, and clear explanation to lecturers of how to get papers that are on a server that is temporarily down.

Acknowledgements

Bruce Lewis and Rob French designed and implemented the file exchange libraries and servers. Nick Williams implemented the user interface from the design of Hagan Heller, and also wrote this paper with Bill Cattey, who had to endure coordinating the lot of us.

References

[Bar87a] E. Barrett, F. Bequaert, and J. Paradis, *Electronic Classroom: Specification for a user interface,* Athena Writing Project, Boston, MA, 4 June 1987.

[Env86a] The Committee on Writing Instruction and Computer Environment, *The requirements for the writing instruction computer environment,* 21 October 1986.

[Hel89a] N. Hagan Heller, *Designing a User Interface for the Educational On-line System,* Massachusetts Institute of Technology, Boston, MA, May 1989.

[Mar88a] Maria Camblor, Rob Shaw, Bruce Lewis, William Cattey, *File exchange for the educational on-line system design specification,* Athena Writing Project, Boston, MA, February 1988.

[Pal88a] Andrew. J. Palay, Wilfred J. Hansen, Mark Sherman, Maria G. Wadlow, Thomas P. Nuendorffer, Zalman Stern, Miles Bader, and Thom Peters, "The Andrew Toolkit - An Overview," *Proceedings of the USENIX Winter Conference*, pp. 9–21, USENIX Association, Berkeley, CA, February, 1988.