ABSTRACT

A SUITABLE SOFTWARE ARCHITECTURE FOR VIDEO DISCUSSION BOARDS AS APPLIED TO THE OLE BOARD

by Benjamin R. Warman

This thesis suggests a software architecture for a video discussion board using the Online Language Environment (OLE) Board as a case study. A more suitable architecture will enable it to meet its goals. Goals for the new board are to support future developers and users. Developer-oriented goals are ease of readability, maintenance, and enhancement. User-oriented goals are ease of use, new or improved features such as data collection, and enhanced security. The old system is out of date, making planned expansion difficult. The new system uses a Model-View-Controller architecture utilizing updated frameworks. Updated protocols based on XML were introduced, and elements were added to allow for new features. Developer goals have been met with minimal drawbacks. The new board was used by 42 students at the University of Aizu and Rose-Hulman Institute of Technology. New features proved useful and security was increased, with some decrease in usability during initial testing.
A SUITABLE SOFTWARE ARCHITECTURE FOR VIDEO DISCUSSION
BOARDS AS APPLIED TO THE OLE BOARD

A Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Computer Science and Software Engineering Department

by

Benjamin R. Warman

Miami University
Oxford, OH

2012

Advisor______________________________
Dr. Douglas Troy

Reader______________________________
Dr. Scott Campbell

Reader______________________________
Dr. Paul Lyddon
# Table of Contents

1. **INTRODUCTION** ................................................................................................................. 1

2. **BACKGROUND AND RELATED WORK** .............................................................................. 1

   2.1 **THE OLE PROJECT** .................................................................................................. 1
   
   2.1.1 **THE OLE BOARD AS A TEACHING TOOL** ......................................................... 1
   
   2.1.2 **THE OLE BOARD AS A TOOL FOR RESEARCHERS** ........................................... 2
   
   2.1.3 **THE OLE BOARD AS A TOOL FOR COLLABORATION** ....................................... 3
   
   2.2 **SURVEY OF OTHER VIDEO BOARD TYPE PRODUCTS** ........................................... 3
   
   2.2.1 **ELUMINATE AND WIMBA** ................................................................................ 4
   
   2.2.2 **VIEWPOINT** ....................................................................................................... 4
   
   2.2.3 **CONVERSATIONS** ............................................................................................. 5
   
   2.2.4 **THE VIDEO DIARY** ............................................................................................ 5
   
   2.2.5 **VOICE THREAD** .................................................................................................... 5
   
   2.2.6 **YOU/TUBE (PRIVATE CHANNEL)** ..................................................................... 5
   
   2.2.7 **VIDEO ANT** ......................................................................................................... 5
   
   2.2.8 **WEBSWAMI** ........................................................................................................ 5
   
   2.2.9 **ADOBEB CONNECT** ............................................................................................ 6

3. **ORIGINAL OLE BOARD** ..................................................................................................... 6

   3.1 **THESIS TOPIC** .......................................................................................................... 6
   
   3.2 **ORIGINAL ARCHITECTURE OF THE OLE BOARD** ................................................... 6
   
   3.2.1 **LOGICAL ARCHITECTURE** ................................................................................ 6
   
   3.2.2 **PHYSICAL ARCHITECTURE** ............................................................................... 8
   
   3.2.3 **WEAKNESSES** ...................................................................................................... 9
   
   3.3 **DEVELOPMENT GOALS** .......................................................................................... 10
   
   3.3.1 **GOALS** ................................................................................................................ 10
   
   3.3.1.1 **DEVELOPER-ORIENTED GOALS** .................................................................. 11
   
   3.3.1.2 **USER-ORIENTED GOALS** ............................................................................. 12

4. **RE-ARCHITECTED OLE** .................................................................................................... 13

   4.1 **OVERALL ARCHITECTURE** ....................................................................................... 13
   
   4.2 **CLIENT** ..................................................................................................................... 14
   
   4.2.1 **EVENT HANDLING AND FLOW** ......................................................................... 15
   
   4.2.2 **VIDEO** .................................................................................................................. 17
   
   4.2.3 **ADMINISTRATOR INTERFACE** ............................................................................ 18
   
   4.3 **CLIENT PROTOCOL WITH COLDFUSION** ................................................................. 19
   
   4.4 **NEW FEATURES AND FRAMEWORKS** .................................................................... 21
   
   4.4.1 **COLDFUSION APPLICATION FRAMEWORK** ....................................................... 21
   
   4.4.2 **FLEX APPLICATION FRAMEWORK** .................................................................... 22
   
   4.4.3 **DATA COLLECTION** .............................................................................................. 23
   
   4.4.4 **OTHER ADDED FEATURES AND IMPROVEMENTS TO THE USER INTERFACE** ... 25
   
   4.5 **ADDITIONAL CHANGES TO THE DATABASE** ............................................................ 26

5. **ASSESSMENT** ....................................................................................................................... 28

   5.1 **ASSESSMENT METHODOLOGY** .................................................................................. 28
   
   5.2 **ASSESSMENT OF DEVELOPER-ORIENTED GOALS** .............................................. 28
   
   5.2.1 **READABILITY AND MAINTAINABILITY** ............................................................... 28
List of Tables

Table 1. Survey of features for ten potential competitors of OLE................................. 4
Table 2. Summary of system preferences ........................................................................... 37
Table 3. Summary of preferences, excluding those who answered "neither." ................. 37
Table 4. Comments from Aizu students .............................................................................. 37
### List of Figures

Figure 1. Old logical layout of the OLE Board. ....................................................... 6
Figure 2. Diagram of old OLE database for textual data. ........................................ 7
Figure 3. Old physical layout of the OLE Board. ....................................................... 9
Figure 4. Implemented architecture ........................................................................ 13
Figure 5. A typical network exchange ....................................................................... 16
Figure 6. Video recording interface for topic creation .............................................. 18
Figure 7. Administrator interface ............................................................................. 19
Figure 8. Example XML ........................................................................................... 21
Figure 9. The videos and activity_log tables. .............................................................. 24
Figure 10. The upload_transactions table allows secure video uploading. ............... 25
Figure 11. Other tables in the database ..................................................................... 27
Figure 12. FTree_Frame4: retrieving children for a message branch in the old system. 29
Figure 13. Retrieving children for a message branch in the new system. ................. 30
Figure 14. Processing XML in old system................................................................. 36
1. Introduction

A video discussion board is conceptually similar to an online bulletin board, also known as a message board, an Internet forum, or an online forum. A traditional message board allows registered users to create textual posts that are organized into threads, sometimes allowing the addition of styling, graphics, and hyperlinks. Sites using message boards often have multiple boards categorized by intended subject of discussion. Such a board may have a hierarchical structure that can often be laterally traversed via hyperlinks.

Video message boards employ this same conceptual hierarchy, but shift the medium of communication. The traditional forum post relies heavily on text and possibly static graphics for communication, whereas a video discussion board post relies more on video and allows the use of text to become peripheral.

A message board contains a collection of threads usually pertaining to a single subject—sometimes very broad—or to a particular trait shared by a group of users [1] [2]. A thread is a collection of posts that relate to a specific topic of discussion, and a post is a single contribution by a particular user to a particular topic. Posts are often accompanied by details of the posting user as well as the date and time of the post. Users may be organized into groups to determine their permissions for the forum [3]. For example, one group of users may be restricted from posting at all (usually unregistered guests) while another can not only post in existing threads but also create new threads (usually registered users of the forum) [4].

This thesis will suggest an effective software architecture for the design of a video discussion board to support the new goals of the OLE Board. The new board aims to aid in pedagogical research and project collaboration. Developer-oriented goals are ease of readability, maintenance, and enhancement. User-oriented goals are ease of use, new or improved features such as data collection, and enhanced security. The OLE Board is used as a case study to assess the suggested architecture.

2. Background and Related work

2.1 The OLE Project

The current research builds upon an existing, early video message board called Online Language Environment (OLE), which was originally developed at the University of Arizona with the initial goal of preserving the oral histories and languages of local indigenous North American peoples. Later the OLE Board was applied to the classroom, enabling students to post messages to one another as well as to native speakers of their target language. It has been used as an asynchronous to quasi-synchronous collaboration tool for language learning since about 2005 [5]. In addition to providing another method of communication among language students, its video capabilities have provided a way for students to record and review themselves during the learning process.
The University of Arizona discontinued its support of OLE, but the system was moved to a server at the University of Aizu in Fukushima, Japan during the summer of 2010 [6]. According to Dr. Paul Lyddon, one of the major proponents of the board’s continued use and overseer of its future development, “the OLE System has to date been used in more than 200 classrooms by over 5000 learners of a wide variety of languages, including Arabic, French, Italian, Japanese, Russian, Spanish, Swahili, and Turkish, as well as English as a Second Language” [7].

Although use in Aizu ceased in February 2012 when Dr. Lyddon accepted a position at another university, it is his intent to reestablish the board in his new locale. His plans for OLE are to expand its focus from language instruction to general collaboration, and its scope will cover its use as a tool for three intended purposes: teaching, research, and general collaboration. It will retain the features that support its current functionality as a teaching tool and expand upon them. Additional tools for data collection and aggregation will be added to aid researchers in investigating the effects of OLE use on student performance. Moving toward the goal of general collaboration, there are plans to introduce an additional space in the interface for an object of collaboration, such as a design document or external video. This object, called a mediated object, will be manipulated by collaborating parties. Along with other improvements, collaboration on projects that go beyond classroom activities will be possible. It is hoped that the board itself will be used to collaborate on and manage its continued development.

The original OLE system architecture is inadequate to support the planned expansion. The tiers of the architecture are intertwined, making development of new components difficult. Aside from this, the old system is written in an obsolete version of ActionScript, and is not maintainable moving forward. A goal of this research is to discover a suitable software architecture for the board’s reimplementation. Desired features to help the board accomplish each of its intended roles are described in the next few subsections.

2.1.1 The OLE Board as a Teaching Tool
Over the summer of 2010, small improvements were made on the board within its original architecture after its move from the University of Arizona to the University of Aizu. The original vision for the OLE Board and detailed suggestions for its use in pedagogy can be found in Forger et al [5], and thus will not be discussed here.

2.1.2 The OLE Board as a Tool for Researchers
For the OLE Board to be a viable tool for researchers, significant changes to the original backend database were required. The hope is that the board will not only be useful for learning and teaching, but also as a means of data collection for educational research studies. To fulfill this role, various pieces of data need to be collected and stored by the software:

- View count for each video
- Record of who has viewed which videos, and how many times
- Record of logins and logouts
- Count of replays before a video is posted
• Count of recording attempts made before a video is posted
• Time spent actively logged in

The database design is extensible to allow for additional data to be logged in the future. Ideally, the system will also allow fully customizable data logging in the future. For example, one conference may not be part of a research project and may not need data to be collected for it while another may need only a subset of data to be collected. To save space, the system provides for activation and deactivation of usage data collection on a conference level. Security concerns in the interaction between the existing client and database have also been addressed to protect any data collected. These are addressed in section 3.2.3 Weaknesses.

2.1.3 The OLE Board as a Tool for Collaboration

The vision of the OLE project includes its eventual use as a general purpose collaboration tool. In the nearest future, the hope is for the board to be used to manage its own development, which will allow parties from both Japan and the United States (and theoretically anywhere) to contribute to the project in addition to using it for teaching or research. The software for the project has been open source since it began, and it will remain so as development progresses in order to facilitate this goal. Most major contributions to its development, particularly in the early stages after the redesign, will be made by students of its primary project coordinator, Dr. Lyddon.

2.2 Survey of Other Video Board Type Products

There are many products similar to the OLE Board that provide multimedia distance communication, but the focus of many of those projects are too broad or too narrow to fit nicely into the niche the OLE Board intends to carve, particularly where pedagogical research is concerned. Other tools could potentially be adapted to be used in place of the OLE Board, and in these cases it is important to learn what we can from the software architectures that were employed in their implementations. A survey of ten existing products revealed the following spectrum of features. Elluminate and Wimba have been discussed together since they are now managed by the same party.

1. Video communication (asynchronous vs. synchronous): allows for video to be posted or used directly in live or recorded communication
2. Interactive (through messages, comments, video recording, and/or sharable content): allows users to author and/or respond to content in either text or video form, and may allow user interactions to be shared externally
3. Mediated object(s): allows an object such as an image, document, or even an external video to be embedded in a conversation, thread, etc.
4. Audiovisual transcription: allows users to view audiovisual content and create a transcription for it
5. Multiple-attempt content creation: allows users to create and review multiple concurrent versions of created content before submission
6. Statistic extraction for researchers: aggregates and allows access to statistical usage data.
7. Indication of previously viewed content
8. Email notification: allows optional emails to be sent on certain events (e.g., addition of new content)

Table 1. Survey of features for ten potential competitors of OLE

<table>
<thead>
<tr>
<th></th>
<th>Video</th>
<th>Interactive</th>
<th>Additional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asynchronous</td>
<td>Synchronous</td>
<td>Mediated object(s)</td>
</tr>
<tr>
<td>Elluminate†</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Viewpoint</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Conversations</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Video Diary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YouTube (private)</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Voice Thread</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Ant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wimba†</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WebSwami</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adobe Connect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Board</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Planned for OLE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Highlighting indicates insufficient public access to determine possession of those features
* YouTube keeps track of this content, but may not always report it to users
** These objects are limited by the system; free creation is restricted
† There is collaboration between these products

2.2.1 Elluminate and Wimba

Elluminate is a virtual classroom environment intended to conduct classes through the Internet [8]. It focuses on synchronous interaction for live teaching over the Internet, but allows asynchronous communication through text and graphics. It is essentially web conferencing focused on education and the classroom and allowing publication of captured interactions (video or otherwise). Elluminate has recently become affiliated with Wimba, which also focuses on synchronous interaction for live teaching over the Internet [9].

2.2.2 Viewpoint

Viewpoint is a tool from Michigan State University that allows uploading and subsequent sharing of existing video as well as recording new video and sharing it [10]. There is no interaction or any mediated object; multiple saved attempts at recording for comparison before submission are not allowed.
2.2.3 Conversations
Conversations is another tool from Michigan State University [11]. It allows teachers to pose questions in video form to students and receive video responses. OLE aims for more interaction between students. A second version of Conversations has recently been released and warrants further investigation to determine what new features it offers.

2.2.4 The Video Diary
The Video Diary has been discontinued. It was intended for a single user to create and share or create and save video logs (vlogs). Not much can be learned from this product as its focus and implementation were entirely different from any aspect of OLE beyond simply recording video and allowing it to be posted for viewing.

2.2.5 Voice Thread
Voice Thread is very similar to the OLE Board but with a different structure [12]. It takes a comment-centric approach, placing most attention on the “mediated object” instead of the conversation, which could be useful in general collaboration situations. A similar interface could eventually be incorporated into the OLE Board, perhaps through a different view within the user interface.

2.2.6 YouTube (Private Channel)
YouTube [13] is another possible alternative to OLE, though the OLE system will need access to data at a level that YouTube will be unable to provide. OLE will likely offer more control over video and playback statistics and record keeping since all statistics would be stored on YouTube’s servers. The main drawback here is the inability to record multiple attempts and compare them before upload, much like Viewpoint. The video comment feature of YouTube is also very similar to Voice Thread in that each posted video focuses mainly on that video, with responses being secondary. This may not always be best if a true conversation is desired.

2.2.7 Video Ant
Video Ant [14] allows users to annotate Flash video, particularly and most easily video from YouTube. This is a desired feature of the OLE Board, but it is the entire focus of Video Ant. OLE would also like to see some features that Video Ant lacks, such as a transcription-friendly looping function that allows repetition of a small portion of the video. Video Ant could possibly be incorporated into the OLE Board, though it may be decided that a deeper integration of this functionality directly into the board itself is desired.

2.2.8 WebSwami
WebSwami [15] is a good tool for teaching a language. It allows users to record questions and to receive video responses from students. It also provides a number of predefined activities utilizing video, text, etc. However, it does not allow for free creation of activities, and it is geared entirely toward an academic context, providing automated grading and the like but restricting its use in the process.
2.2.9 Adobe Connect
Adobe Connect is a synchronous video conferencing tool [16]. It may be a good example to follow for support of synchronicity in the future, but its focus is entirely live interaction or remote presentation. There are no tools for researchers or educators.

3. Original OLE Board

3.1 Thesis Topic
This thesis will suggest an effective software architecture for the design of a video discussion board to support the new goals of the OLE Board. The OLE Board will be used as a case study to assess the chosen architecture.

3.2 Original Architecture of the OLE Board
Framing the architecture in modern terms, the old incarnation of the OLE Board operates in what is best described as a loose and incomplete Model-View-Controller (MVC) architecture with a split in the third tier (controller) between the view and the model [17]. Thus, it has a 3-tier architecture in which the interface (view), the data (model) and the functionality (controller) are sometimes not completely separated, and sometimes completely muddled. These tiers are split between two virtual physical tiers. Diagrams of the logical and physical architectures follow.

3.2.1 Logical Architecture

![Old OLE Board – Logical Architecture](image)

**Figure 1.** Old logical layout of the OLE Board.

The logical architecture closely reflects a 3-tier Model-View-Controller architecture in all but video management. There is an administrator and a conference interface available.
These interfaces were developed in separate environments that produced the same type of deliverable, a Flash object (swf). The conference view provides access to the board and various user settings. It is developed in Flash. The administrative view was developed as still movie frames but exported to a format which is no longer editable in the newest version of Flash (which was CS5 at the writing of this document). The model consists of textual data and video data. The model code written in ColdFusion creates a clear definition for a controlling component for all data that is not video, though most control code is nonexistent or tightly coupled with the view code. Non-video data encompasses user information (including login information, profile data, enrollment data, and a record of read messages), permissions, the message boards, and their accompanying message text. It is stored in a MySQL database. Flash Media Server manages video data, but controller logic for video data is embedded in the Flash user interface. (See Figure 1).

The text database has the structure depicted in Figure 2. It consists of eleven tables, all of which except for the admin table have one or more relationships established with other tables. The admin table stores the administrator password. The administrator interface does not ask for a username; it simply asks for the password and checks it against the password stored in the admin table (which is stored in plain text). All tables except read_messages were present in the system before it was moved to Aizu. The read_messages table was added to keep track of messages that each user had viewed, and mark them appropriately. There may be a more scalable approach to accomplish this.

![Diagram of old OLE database for textual data](image)

**Figure 2.** Diagram of old OLE database for textual data.
Most aspects of the text database are self-explanatory. There are only a few that may need further explanation. The hasstream field of the messages table is currently the only data kept in the text database that references the videos stored in the video database. In the new system, more data is stored about videos, which are referenced differently as well, for reasons discussed in section 3.2.3 Weaknesses. Another potentially confusing aspect of the database is the oid field in the messages in the database. Its use is not immediately apparent, but it functions as the mechanism to allow reordering of topics while maintaining a static reference. Reordering is currently only accomplished through the mid of a message. To change the order, the mid is changed, but the system needs a static ID to continue to properly reference the message elsewhere in the system (e.g., to retrieve videos associated with the message).

The term topic is polysemic in the OLE Board. Messages within a conference are currently displayed to users as a tree. Users refer to messages at the first level of this tree as topics. They are marked in the interface with a blue circle while messages at lower levels (i.e., children of topics, or children of further messages) are marked with a green box. However, there is no concept of a topic defined this way within the system. The system simply marks any message whose parent is 0 (indicating no parent) as a topic. The term topic within the database refers to the subject of a message, i.e., the title of the message that the user sees before clicking on the message to read its body text or view an accompanying video. Thus, the topic field of the messages table stores what a user would likely refer to as the subject of a topic or message. This confusing terminology has been eliminated in the new system.

3.2.2 Physical Architecture

From July 2010 until March 2012, the OLE Board system was hosted on two virtual machines (VMs) running on a single physical machine. All servers, both virtual and physical, ran CentOS. One of the virtual machines had ColdFusion 9, a web server (Apache), and MySQL installed. The web server used came bundled with ColdFusion, and operated out of port 8500. This first VM was responsible for most user interaction, including all interactions with the message board. The other VM ran Flash Media Server and a web server (Apache). Server-side ActionScript on this VM managed the video, including its delivery.

The physical machine had a little less than 1GB of memory and 5GB of hard disk space available to it. The rest was split between the VMs, giving each virtual server 1.5GB of RAM and 60GB of hard disk space (see Figure 3).

The virtual machines were accessible via the VMware Server 2.0.2 WebAccess interface. This hosting setup for the OLE Board was discontinued with the relocation of Dr. Lyddon in March of 2012.
3.2.3 Weaknesses

The old implementation of the OLE Board is inadequate for extensive expansion and long term-maintenance. The technologies originally used to create it are becoming obsolete and need to be updated. On the security front, database queries are not sanitized and user passwords are stored in plaintext. In terms of other weaknesses, there is nothing in place to collect any relevant data on the use of the board or its videos, making the job of the researcher difficult. Representation of videos within the system impedes the storage of these data and the reuse of videos. Terminology within the system is sometimes ambiguous between developers and users. While the user experience is not bad in the old system, some minor tweaks could significantly improve that experience.

The entire system was developed in Flash using an old version of ActionScript (2.0) that, while object oriented, has distinct disadvantages in Rich Internet Application (RIA) development to the newer ActionScript 3.0. In addition, Adobe is now focusing its efforts on ActionScript 3.0, and any older code will not be maintainable moving forward. Even the Flash application itself (e.g., Flash CS5) is not an optimal development environment for a Rich Internet Application such as the OLE Board. Since initial development of OLE began, Adobe has developed the Flex framework and a tool called Flash Builder specifically for RIA development using Flash technology [18]. The Flex framework used in the new implementation of OLE makes use the newer ActionScript 3.0. For further
discussion of the advantages of these newer Flash technologies, see section 5.2
Assessment of Developer-Oriented Goals.

The backend does not utilize locks or the newer framework used by modern ColdFusion
applications. ColdFusion developers will be familiar with this newer structure, which
gives a standardized start-up and access environment for the application code that
automatically implements the required locks for a multi-user environment.

All communication between the client and backend was done using proprietary
communication protocols designed for Adobe technologies. While appropriate for the
chosen tools at the time, any features developed with tools that cannot use these protocols
would be unable to communicate with the system. This essentially limits development to
include only technologies developed or otherwise directly supported by Adobe.
Communication via a non-proprietary technology such as XML is more desirable as it
allows a wider range of tools to be used while maintaining a standardized and extensible
communication protocol among them.

There is no data stored about videos or the use of the board in general. Videos are
identified solely by the message and conference they are associated with. This robs them
of independence within the system, which is important to cleanly store the desired
information about them for researchers. Lengths of videos are not stored, nor are statistics
such as a play count, number of replays before posting, or and other indicators that would
be useful to educators and researchers using the board. The only piece of information
stored about videos in the database is the fact that a message has an accompanying video.
Thus, the only source of information for videos is their associated text messages. In a
system focused on collaboration in language education that revolves around the use of
videos, this is a significant deficiency. With more information available independent of
the general user interface, the board is a more potent tool for researchers and educators.

Some terminology used by the system is out of sync with terminology familiar to users,
such as the term topic described in section 3.2.1 Logical Architecture. This detracts from
ease of maintenance since bugs described by users may use terms that are referred to
differently within the system, causing confusion and possibly introducing new bugs
instead of fixing old ones. In addition, due to the way video is managed and the way
messages are reordered in the interface, there is some unneeded complexity in the
database. This is described in more detail in section 3.2.1 Logical Architecture.

3.3 Development Goals

3.3.1 Goals
Goals for the new board can be divided into two categories based on who they help:
developer-oriented goals and user-oriented goals. Goals to help developers are increased
ease of readability, maintenance, and enhancement. Goals to help users include creation
of a number of new features as well as improvement of existing ones. New features
include video uploading and elements in the database to allow for data collection by
researchers. Improved features include video playback and recording, message
management, security, and ease of use. All design decisions take these goals into consideration.

3.3.1.1 Developer-oriented Goals

The new design aims to improve readability and maintainability by changing and reworking the software architecture at key points. This is accomplished primarily by replacing the user interface with a more solid object-oriented implementation using ActionScript3.0, implementing selected protocols using XML, and using updated ColdFusion and Flex frameworks where possible.

The code for the client in the old system is difficult to understand and navigate and, therefore, difficult to maintain. Written in old ActionScript using the Flash development environment, it was technically object-oriented, but did not make extensive use of object-oriented practices. It is difficult to find target sections of code because there is little separation of functionality in the code and because the entire development environment is alien to anyone trained in development with most other procedural programming languages. The old Flash paradigm involves a stage on which the programmer (or more appropriately, the animator) can place objects and then define their positions, look and feel, and their actions through the interface or by using ActionScript. Using different frames, a developer can create what are essentially different states of a program and move between them based upon user actions or other events in the program. Interface design was not a goal in the development of frames, which were originally designed to be key frames between movement in an animation.

Six of these frames constituted the old OLE Board, with a separation for global and local actions written in ActionScript. All global actions are kept together regardless of what they operate on, be it user information, videos, messages, or some other conceptual object. Local actions are tied to a particular frame, and many of these are intermediate functions responding to events that redirected to a function in the global code. Thus, fixing a problem meant determining what frame it might be originating from and digging through a large body of code with mixed functionality in search of the offending function or line. Further discussion of this problem and its alleviation is in section 5.2.1 Readability and Maintainability.

Communication protocols between the client and backend are proprietary and developed for use with only Adobe products. The protocol used allows remote function calls between Flash and ColdFusion, which means a set of functions must be decided upon in advance, and changes to the set need to be uniformly accepted and implemented properly. Choosing a more open and extensible communication tool, such as XML, provides standardization in communication independent of available functions and should improve maintainability. It also provides flexibility for future enhancements by allowing the incorporation of technologies not developed by Adobe. In addition to function-independent standardization, the hope is that a non-proprietary protocol will give developers more freedom in future design decisions and expansion.
Another problem with maintainability is that the old system is not using any current ColdFusion framework or the improved Flex framework. The application framework in ColdFusion allows developers to use predefined hooks at certain points in the application lifecycle to perform tasks related to session management, cleanup and other common actions with the proper locks in place for a multi-user environment. For example, session variables can be changed during the start of a session with no fear of race conditions or other concurrency issues.

The Flex framework produces object-oriented ActionScript 3.0 code. It is also a newer technology that Adobe will be focusing its attention on for future application development with Flash technology. This means not only better enterprise support via security patches and classes but also community support in troubleshooting programming problems. Flex’s development environment is intended to make development quicker and easier for data-centric applications, which many Rich Internet Applications are, including OLE [19]. Since extensive development by students is planned for OLE in the future, incoming students can be taught the most up-to-date technology, and no time will need to be wasted learning how to program in a language that has been replaced. There are also many pre-programmed components available in Flex that accomplish many common tasks that used to require additional programming from the developer, including video playback.

3.3.1.2 User-oriented Goals

The main user-oriented goals are user-friendliness and new features or improvements to increase utility, ease of use, and security. The ability to upload video and collect data for research purposes are two such new features. Improvements on general user experience and security include video playback and recording, message management, secure password storage, and small interface tweaks such as allowing a user to press enter instead of clicking a button to log in. With tables in the database to store metrics of the board’s use, utility of the board to researchers has significantly increased, and following more common security practices has provided a good base of security that the old system lacked.

Components in the Flex framework have been improved or added to the ActionScript libraries since the original development of the old OLE Board using Flash ceased. This means a better-tested and possibly more user-friendly interface.

The old OLE Board had no tools for researchers beyond a cryptic catalog of videos organized by conference and message numbers as referenced in the text database and a logically connected database of message and conference text. New treatment of videos within the system will allow more data to be stored on them as well as increase the flexibility of their use throughout the system.
4. Re-Architected OLE

4.1 Overall Architecture
The Model-View-Controller architecture is preserved and strengthened where possible. The largest change to the architecture was the introduction of a controller tier. Coupling between the view and model code was decreased with this change, but the only independent controlling component introduced between the existing view and model tiers was the application framework provided by ColdFusion. This framework was used where it proved useful, but many of its methods were not implemented. All textual communication is done via XML, and elements were added to the database to allow for new features, including video uploading and data collection by researchers. The final architecture defined a controller tier for the board. The presence of this tier will ease the process of enhancement. For example, with a standard set of intermediate functions, the introduction of new interfaces will be much simpler as they will be independent of the access and manipulation of data within the board. While the control tier was perfectly modeled by the ColdFusion components in the original architecture, it was essentially missing; software objects in the interface provide that missing tier. The new architecture is illustrated in Figure 4.

![Figure 4. Implemented architecture](image)

A modular design was considered for the system, but simple object orientation was ultimately preferred. The benefits of modularization, including the reduction of resource demands by switching off rarely used features, add-on development by the community,
and independence among elements of the system, were either inconsistent with other
goals of the project or accomplished by other means. Any expansion that occurs, and all
existing features, will be required or otherwise beneficial to all parties using the board,
and there are no plans for community development at this time. Independence among
elements was aided by object orientation. Therefore, modularity was deemed
unnecessary.

Object orientation and updated frameworks were the main vehicles for the achievement
of design goals. As one of the industry’s predominant design paradigms, not only can
object orientation offer a certain level of independence among elements, but the level of
familiarity with the approach among developers increases maintainability simply by
being more accessible to a significant portion of those with programming training. Object
orientation has also significantly increased readability of the code by consolidating it into
a few files that are more easily navigable than the scattered, and at times vague,
organization of the previous code base. Updating frameworks also increases
maintainability; moving forward, up-to-date support will become unavailable to older
alternatives.

The object concept has been extended beyond the software itself to establish guidelines
for data collection. Data is collected by logging actions on objects throughout the system.
There is a set of conceptual objects that the system can operate on, though the use of
these objects and their associated actions is not strictly enforced. The established
conceptual objects, which are based on the implemented software objects, include topics,
messages, videos, users, roles, permissions, conferences, and passwords. A standard label
for general information is also available.

While under development, it became apparent that direct communication between the
client and video management was desirable. Adding another tier between storage and
playback would have only served to overcomplicate the system and increase latency
without offering significant benefits. Treating video storage as a separate entity on the
same tier as the textual database simplified development and allowed the leverage of
communication libraries for Flash Media Server available in Flex. The standardized
interface for video recording and playback was also abandoned as it became apparent that
the interface needs for the two tasks were too dissimilar.

4.2 Client

The user interface, and the client code in general, underwent a complete redesign with the
updated and more strictly object-oriented ActionScript 3.0 and Flex framework. The
interface itself is written in MXML, which is a convenience afforded by the Flex
framework. It allows pre-programmed components to be inserted as tags and provides
hooks to insert functions that operate on those components, mostly responding to events.
All tags are translated into ActionScript 3.0 when they are compiled, and it is possible to
have the compiler keep that generated code, if desired.

The code to control interaction with the backend is written directly in ActionScript 3.0
classes that were designed to match the organization of ColdFusion components on the
backend as well as to consolidate code with similar functionality and encapsulate actions and data where appropriate. The organization is as follows. The distinction between interface and control code is convenient to more clearly define the view and controller tiers of a Model-View-Controller architecture, though they are still coupled and run together on client machines.

**MXML Components (Interface/limited control code)**
- oleboard.mxml
- oleadmin.mxml

**ActionScript 3.0 Classes (Client/control code)**
- Events
  - ResultsProcessedEvent.as
    - CONF_PROCESSED
    - USER_PROCESSED
    - CHILDREN_PROCESSED
    - REFRESH_TOPICS
    - TOPICS_REFRESHED
    - REFRESH_PARENT
    - MESSAGE_DELETED
  - UserEvent.as
    - USER_AUTHENTICATED
    - AUTHENTICATION_FAILED
- Functionality
  - Conference.as
  - Message.as
  - Role.as
  - Topic.as
  - User.as
  - OleVideo.as
  - OlePermissions.as
  - Global.as

User authentication logic is encapsulated in the *User* class, which also used for management of user account information, including the conferences they belong to as well as the roles they have in those conferences. Authentication only happens through the *userLogin* component. This component makes use of the authentication events in the *UserEvent* class. It is used as any other MXML component in the interface, making use of a *userAuth* event hook that allows the interface to respond to a successful authentication. It was written as a separate component to allow it to be used in both the conference interface as well as the administrators’ user interface, and any other interfaces that are designed in the future, e.g., a researchers’ interface to view and manipulate data collected by the board.

**4.2.1 Event Handling and Flow**
Event hook functions for interface components are coded in ActionScript inside a *script* tag within the MXML. Some do nothing more than flow into the control code. Other functions in these *script* tags perform state transitions, initialization for the client and translation between data formats for viewing and communication. Comments throughout the code divide these utility functions based on the objects on which they are intended to act.
There is an `HTTPService` object in the ActionScript code corresponding to each ColdFusion component on the backend, separating calls to different services to help organize the code. All network communication calls in ActionScript 3.0 are asynchronous. Thus, the `ResultsProcessedEvent` class is an important element that is triggered when the calls are completed to inform components of the interface when the necessary changes are ready to be made, given the results of a call. A typical network transaction, depicted in Figure 5, starts with a retrieval of the displayed data from the interface components, and a translation into the object that is expected by the control code. The objects are then translated fully into the XML used to communicate between the client and the backend, an event listener for the return of the network call is started if necessary, and the asynchronous call is started by the control code. A result handling function is triggered when the call returns and the results are translated for the client if necessary; some interface components, such as those displaying messages and topics, use tree-like structures to display data and can use the returned XML directly. Finally, the appropriate `ResultsProcessedEvent` is fired and the appropriate data structures and interface component is updated, or the user is informed of an error. Functions that handle `ResultsProcessedEvents` are in the `script` tag of the MXML interface to allow the appropriate level of access to update display components.

Figure 5. A typical network exchange

Following events through the system is somewhat complicated by the transparent nature of some event triggering, but it is often unnecessary to follow an event to its lowest level in the data-centric environment provided by the Flex framework. As far as the programmer can tell, or ever need know, event handling for calls to the server is encapsulated as much within the appropriate class as possible. For example, to refresh topics, a call is made to the `getTopics` function in the conference class. This function makes the asynchronous call to the corresponding conference component on the backend. When results are returned, a result handler, also in the `Conference` class, handles the results and updates the data structure that maintains the topics for the current conference. This data structure is tied to the appropriate interface components through the `dataSource` field. When the structure is updated, an internal event is fired which is handled by those components. Similar event flows occur with the modification of topics and messages.

Care must be taken when creating new `Topic` and `Message` objects that the proper event listeners are added. Enough listeners are added by default so the object can communicate
with the backend, but access restrictions among objects prevent tying new objects to the proper data structures to update the interface during construction.

Efforts were made to simplify event handling as much as possible while preserving encapsulation and limiting the use of global data structures and methods. To that end, event chains were reused where possible. When a parent message or topic is refreshed after a message is edited, for example, a `ResultsProcessedEvent.REFRESH_PARENT` event is fired and handled by a listener in the topic class. Rather than begin a distinct event flow, this listener will invoke either the `getChildren` method in the `Message` class or `getTopics`, depending on what kind of object has been updated.

A global user data structure is used to keep current user data as well as the user’s current conference data. Allowing this break in encapsulation greatly simplified the communication between MXML components and the control code, which is accomplished via events, as described above. It is this small set of global structures that are tied to interface components and updated when network calls are made. Error codes, timeouts, and utility functions for searching, sorting and time conversion are also available globally. Though many fell into disuse during development, they were preserved in case they prove useful in the future. All other elements are encapsulated as much as possible.

### 4.2.2 Video

Video playback, recording and upload functionality has been programmed into the main MXML interface. The `OleVideo` class manages the connection with the Flash Media Server, handling the mechanics of recording video. It also creates and updates database records for videos. Video playback is handled by a Flex video player component. Upon login, the connection with the FMS is established. Any errors during connection are reported to the user. Because this connection is paramount to the video functionality of the board, if the user tries to post a message before the connection is established, the system informs them that it will not be possible to capture video and audio until the connection is established. Text can still be posted if there is a problem with the connection to the FMS.

When a user chooses to post or edit a message, a panel appears containing the appropriate interface for the chosen action. Posting a topic will yield the most powerful interface, as a window for email notifications (which are not fully implemented yet), in addition to a subject and potentially a message body and video, needs to be supplied during creation. This interface is illustrated in Figure 6. At the top of this interface are two video displays: one for recording so the user can see themselves during the creation of a video, and another for playback. The video display for playback is simply another instance of the Flex video player component. The video display for recording is accompanied by buttons to start and stop the recording as well as specify whether video, audio, both, or neither should be captured. There is also a small input box for uploading video should the user choose to upload instead of record a message through the interface. Though it has a simple design, this is one of the most complicated components in the interface.
There are a number of use cases that had to be taken into account when allowing both video recording and uploading. Perhaps the user only wants audio to be captured. Maybe they want both audio and video captured. Maybe they want neither to be captured, but instead opt to upload a video. Each of these cases and others requires a set of actions to be performed by the control code in order to function properly. The system maintains an instance of OleVideo to keep track of the status of the current video being recorded or uploaded. After a message is posted, the instance is cleaned up and prepared for the next post. Because of the complexity in the management of this variable, the video recording elements break encapsulation to an extent, making the network call for the creation of a new video record, and handling its results, in the conference interface rather than the OleVideo class.

4.2.3 Administrator Interface

The administrator interface has been redesigned as well. It was the first element of the board to be rewritten, and was written while learning ActionScript 3.0 and the Flex framework. Due to this and other factors, its design differs from the conference interface in some respects. The largest difference is in event handling. Rather than encapsulating event handlers in their corresponding classes, all result handling functions were grouped into a single file, adminResultHandlers.as. Calls to administrator functions are coded directly into the administrator interface using Flex CallResponders.

The administrator interface, pictured in Figure 7, does make use of many of the same classes as the conference interface for interaction with the backend. Some techniques also remained the same between the two interfaces, such as tying a data structure to an
interface component and updating the shared data structure rather than directly manipulating the components of the interface. However, separating administrator functions from general user functions seemed wise, so administrator functionality is programmed directly into the interface rather than into the corresponding ActionScript classes. With this separation, there is less of a chance that users could hack the conference interface to perform administrator actions. They are only left with attacking the server if they wish to elevate their privileges. Due to the desire to keep administrator actions separated from general user actions, as well as time constraints, the administrator interface was not reworked after better event handling organization was developed.

![Administrator interface](image)

**Figure 7.** Administrator interface

The enrollment list for each conference now appears between the list of users and conferences, with buttons to allow easy movement of users to and from conferences. Specific user information appears below the user list when a user is selected, and this same form can be used to enter information for a new user. Individual conferences are added using a form under the list of conferences. Simple sorting is allowed to facilitate searching for a particular user or conference as the number of users gets larger. Some other small but useful new features to aid in user and conference management are discussed in section 4.4.4 Other Added Features and Improvements to the User Interface. The goal of these design changes was to improve the flow, coherency, and utility of the interface.

### 4.3 Client Protocol with ColdFusion

The use of the Flash Remoting protocol between ColdFusion and the client was replaced with XML [20]. For a discussion of the benefits and drawbacks of using XML, see section 5.2.2 Communication Protocols. This protocol switch required significant changes to the backend in addition to the interface. All ColdFusion code was redone to
accommodate communication over XML. The protocol was kept as simple and logical as possible without complicating other facets of development. Most conceptual objects in the system have a corresponding tag.

Some components of the interface, namely the component for browsing topics and messages, can use the tree structure and information provided by XML to populate their displays. In these cases, it is necessary to keep the information “flat” so as not to display branches where none logically exists. Because of this, most information about topics, messages and videos is communicated via attributes within XML tags. In all other cases, where the tree structure is irrelevant to the display of the data, only identification information is sent as attributes while tags are preferred for all other information.

The administrator interface communicates solely in XML to facilitate batch operations on users and conferences. In the conference interface, XML is mostly used in one direction—from the server to the client—unless additional information is required by the server to complete the transaction. For example, when a user creates a message, the contents of that message need to be communicated; in cases like this, the client constructs XML that describes the new object and sends it to the server. In cases where less information is required, such as message deletion, where only the message ID is required, plain HTTP POST variables are used.

Figure 8 is a commented example of valid XML, excluding the XML header. The example is not a valid document per se; it illustrates all valid tags, which are used as needed by the system. Tags that are unneeded by a particular function are ignored. To drop a user or conference, _uid_ or _cid_ must be supplied as an attribute. No other information is required. Dropping topics, messages and videos only requires a POST variable supplying the appropriate ID as these are dropped through the conference interface. Dates are currently not handled properly by the database, thus any attributes supplying dates will be ineffectual at this time.

```xml
<ole>
<!-- Add or edit a user. Only username and passhash are strictly required -->
  <user>
    <username>newuser</username>
    <passhash>5e884898da28047151d0e56f8dc6292773603d0d6aabbd62a11ef721d1542d8</passhash>
    <given_name>Dude</given_name>
    <family_name>Sweet</family_name>
    <email>dude@sweet.com</email>
    <creation_ip>10.33.7.219</creation_ip>
  </user>

<!-- Add a conference. Both inner tags are required -->
  <conference>
    <name>newconference</name>
    <data_collection>1</data_collection>
  </conference>
</ole>
```
<!-- Enrollment. All IDs are required -->
<enroll cid="304">
  <role rid="1">
    <user uid="1" />  
  </role>
  <role rid="3">
    <user uid="10"/>  
  </role>
</enroll>

<!-- Unenrollment. All IDs are required -->
<unenroll cid="303">
  <user uid="8" />  
</unenroll>

<!-- Add a topic, message and video. XML to edit a topic, message or video drops all tags but the item of interest -->
<topic title="topic title" cid=123 uid=42 email_start="1/1/1970" email_end="1/19/2038">
  <msg uid="42" cid="123" parent="-1" author="usr" subject="subj" text="text">
    <video id="123" />
  </msg>
</topic>

<!-- Full video tag. Length is in seconds -->
<video id="123" mid="123" length="120" views="3" attempts="5" replays="3" lastViewed="1/1/1970" lastEdited="1/1/1970" />

Figure 8. Example XML.

4.4 New Features and Frameworks

4.4.1 ColdFusion Application Framework

Though there are many third-party frameworks available for use with ColdFusion, the application framework is available from Adobe, and requires no additional configuration to use. The ColdFusion application framework is driven by a ColdFusion component called Application.cfc, and it provides a controller component to manage access to the model tier from the view. At various points throughout the application’s lifecycle, methods in this component are invoked. These methods may be overridden, giving the developer convenient hooks at points where certain actions would be most appropriate, from start-up code to session management and cleanup. The general life cycle is as follows [21]:

- A request is received and accepted from a client
- Starting from that page’s location, ColdFusion traverses directories to find Application.cfc and instantiate it.
- The Application component starts the application and establishes the association between it and the requested page.
- Application events are handled as they occur
- Control passes to the application code itself.
The application that ColdFusion provides is actually a devoted memory space that is shared by all components within an application. This memory space and the relationship between it and the application components are established and maintained by Application.cfc. The running application is not intrinsically associated with the application code; the association between the two must be re-established at each page request. These and many other details and subtleties are discussed by Ben Nadal in his article and presentation on mastering the application framework [21].

Within this framework, the developer has control over application behavior at each page request. Throughout the life cycle of an application, the following Application events will occur one or more times [22]:

- Application start: a page request is made for an application that is not running
- Application end: an application times out or the server shuts down
- Session start: a page request is made outside of an existing session
- Session end: a session has timed out
- Request start: a request is received via HTTP, the event gateway, SOAP, or Flash Remoting
- Request: immediately after a request start is finished; used for content filtering
- Request end: all pages and ColdFusion components have been processed for this request
- Exceptions: an exception occurs that is not handled with a try/catch block

Locks are implicit on the affected memory space (e.g., application, session, etc.), throughout each of these events. Methods to respond to all of these events have been included in OLE’s Application component, but it currently only uses the methods for Application start, Session start and Session end. When the application starts, a number of global constants are declared for system objects and actions, XML tags, error codes, system messages, and an object to access global utility functions. These must be re-declared whenever the application restarts. When a session starts, variables to store user information are established, along with a flag to determine whether data should be collected and logged. This must be repeated whenever a new session is established. The end of a user’s session is logged as a LOGOUT event when the session ends. The time that the application spends running in memory is configurable, as is the length of a session.

**4.4.2 Flex Application Framework**

OLE utilizes the open source Flex framework, which was developed for Rich Internet Application development and gives developers trained in other programming technologies an environment that is more familiar and intuitive than standard Flash development [23]. It focuses on event-driven, data-centric application design. Application interfaces are written primarily in MXML, a tag-based language that allows access to a library of pre-programmed components. These tags are translated into ActionScript 3.0 by the compiler, which is also used to program all actions in the application.
Since it is built on ActionScript, development in the Flex framework shares many of its common design patterns. Flex components can take advantage of data binding, which allows them to respond to changes in variables. All network communication is asynchronous, and accomplished through the use of an AsyncToken. These are given to CallResponders, which will wait for the results of the call associated with the token and run the desired code when the call completes. Flex components also depend heavily on the use of events for communication, using them implicitly in data binding and allowing developers to explicitly dispatch events and set event handlers for various component events. More information on the Flex framework may be found at [24].

Adobe offers an Eclipse plugin called Flash Builder to make development using the Flex framework more developer-friendly. In addition to the standard syntax highlighting, project and code navigation, and etc., it provides an interface to establish communication with web services or other functionality available over the network, as well as an environment pre-configured to compile, run and debug Flex projects. Though Flex can be used without Flash Builder, the new implementation of OLE was developed using this environment.

4.4.3 Data Collection

The desire for data collection was the largest influence on new elements and concepts within the system. Changes were made across the entire system to accommodate data collection. Videos are now conceptualized as independent objects within the system, and have been given their own table in the database to store the desired information about their use. Thus, there is now also a Video ColdFusion component that controls access to, and manipulation of, this data. The OleVideo class was added to the client to manage the creation and viewing of videos, as discussed above. Videos may be associated with any one message. Communication of all new information is fully integrated into the new XML protocol, including a tag to enable or disable data collection.

The adaptation of the system for data collection had the largest effect on the database. Not only was a table added specifically for videos, but another table was added to log actions in the system, including video views, replays, and recording attempts, in addition to user and administrator actions. Figure 9 below describes these tables. Where possible, the ColdFusion application framework has been leveraged to log these actions (e.g., when a user logs in or a session ends). However, most actions are logged as they are executed in the corresponding ColdFusion component.

CREATE TABLE `activity_log` (
  `aid` int(11) NOT NULL auto_increment,
  `cid` int(11) NOT NULL default '0',
  `uid` int(11) NOT NULL default '0',
  `user_ip` varchar(40) NOT NULL default '',
  `time` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `action` varchar(15) NOT NULL default '',
  `object` varchar(15) NOT NULL default '',
  `oid` int(11) NOT NULL default '0',
  `description` varchar(100) default NULL,
  PRIMARY KEY (`aid`)
CREATE TABLE `videos` (
`vid` int(11) NOT NULL auto_increment,
`mid` int(11) NOT NULL default '0',
`vid_length_sec` int(11) default '0',
`vid_last_viewed_time` timestamp NOT NULL default CURRENT_TIMESTAMP
on update CURRENT_TIMESTAMP,
`vid_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',
`vid_last_edited` timestamp NOT NULL default '0000-00-00 00:00:00',
PRIMARY KEY (`vid`)
);

Figure 9. The videos and activity_log tables.

The activity_log table records which conference each activity originated from, as well as the ID of the user performing the action, and the date and time. If the activity affects an existing object within the system, such as a message, user, conference, etc., then the ID of that object is also recorded. An optional description may also be included with the activity. This is most useful for miscellaneous messages and administrator activities. During creation of a conference, data collection can be enabled or disabled. There is currently no mechanism to toggle data collection after a conference has been created. Administrator activities are always logged for all conferences, regardless of whether data collection is enabled.

A number of activities of interest and their associated objects are recorded in the activity_log table. For convenience and the sake of completeness, all administrator actions are logged in the activity log as well. Available system actions (defined as constants in Application.cfc) include logging in, logging out, creation, editing, deletion, views, replays, recording attempts, and enrollments. These actions may be applied to one or more conceptual objects, as appropriate. Objects that may have actions associated with them include messages, topics, videos, users, and conferences. Miscellaneous messages with no associated object may be logged as INFO. PASSWORD is also included as a distinct object because it provides an easy mechanism to determine if users have changed their passwords, which is often desired if an administrator creates many users at once with systematically generated, or otherwise weak, passwords. It is possible for developers to customize what objects and actions are entered into the activity log, but standard sets of objects and actions were established as server-side constants to give a level of standardization to logging and simplify the retrieval of statistics.

Some statistics can only be retrieved through calculation using logged activities. Attributes of videos that are not cumulative, i.e. identification, length, and dates for creation, last view and last edit, are stored in the videos table. Some of these dates are also available in the activity log, but the number of views, replays and recording attempts must all be aggregated from the activity log. While the number of replays and recording attempts are always collected where possible, views are currently not logged. See section 6.1 Limitations to Data Collection for further discussion of this limitation. Time spent logged in must also be calculated from LOGIN and LOGOUT actions.
4.4.4 Other Added Features and Improvements to the User Interface

New and improved features were added to both the administrator and conference interfaces. Video uploading was an important feature, and it was the first to be added to the conference view after the reimplementation was complete; message management and user account management were improved. In the administrator interface, batch functions were added to facilitate user and conference management. Some simple sorting functionality was included to help organize users as the population grows.

Video uploading requires access to both the textual database and video storage. Thus, the proper features must be modified or enabled on both servers. For the simplest setup, the upload script was put on the video server. With FMS, this requires modification of the crossdomain.xml file to allow users to reach it. Naturally, CGI must be enabled for the script to run. For more security, the upload process involves an “upload token” that is stored in a database table shown below in Figure 10. This token is nothing more than a database record with a timestamp. It is created when a user chooses to upload rather than record a video. When the upload is initialized, the token is used for authentication. If it has expired or no token for the user’s IP address can be found, the upload is denied. To determine if a user is allowed to request an upload, the upload_vid permission was added to the database.

```
CREATE TABLE `upload_transactions` ( 
  `id` varbinary(16) NOT NULL default '\0',
  `time` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `ip` varchar(40) default NULL,
  PRIMARY KEY  (`id`)
);  
```

Figure 10. The upload_transactions table allows secure video uploading.

Improvements to message and account management have been made. Users are now allowed to edit and delete their messages. Changes to attached videos cannot currently be made without recreating the message, but the user can edit the subject and body of any message created by them. They can delete their message as long as it has no messages beneath it in the hierarchy. If that is the case, a marker is put in the message’s place so no responses to the message are orphaned or lost. Account information is more tightly grouped with other functionality in the interface in the hope that it is more accessible.

Batch functions are now available through the administrator interface for enrollment, dropping users from conferences, and the creation and deletion of users and conferences. Changes to the communication protocol aided the implementation of this change and increased its utility. XML is relatively easy for humans to read and write. Sets of users or conferences can be described in XML and uploaded to the server to speed up the administration process. The same tags used to describe these objects throughout the rest of the system are used for these administrative purposes. Comma separated value (CSV) files may also be used, allowing lists of users or conferences to be imported from spreadsheet programs, for example. These CSV files are translated into the appropriate XML before they are uploaded. Example CSV files are in Appendix 2.
4.5 Additional Changes to the Database

A number of changes were made to the database to solve the terminology issues described in section 3.2.1 Logical Architecture. The topics table was added, changes were made to how videos are associated with messages, and the allow_management permission was repurposed. Other changes were made to the database to consolidate information or accommodate new features. Login information and user information are now stored in a single users table and the admin table has been eliminated. Changes to accommodate new features are discussed in earlier sections.

Information stored about users has been augmented as well as consolidated. The date when the user was created and the IP address from which it was created are now stored, along with the last date and IP address the user logged in on. The nonce and verified fields were added for open registration via confirmation emails, in case such a system is desired in the future. The superuser field now identifies administrators, replacing the admin table that stored a single password and thus allowed no more than a single administrator, unless the password was shared.

Topics are now stored in a table separate from messages, which helps to solidify system terminology and allows finer grained control over topics themselves. Date fields were added to specify windows for notifications of changes to topics via email. The last time a message was posted in a topic is also stored to help distinguish read and unread messages.

The old hasstream field of the messages table has been replaced with a foreign key in videos referring to its parent message. See Figure 11. This allows a single video to be associated with a single message, which was the desired behavior. It also separates the relationship between messages and videos from the file organization used in video storage. Instead of only flagging messages that have videos, it points directly to the video it is associated with, giving enough information to retrieve the video without encoding more information in the organization of video files.

CREATE TABLE `messages` (  `mid` int(11) NOT NULL auto_increment,  `tid` int(11) NOT NULL default '0',  `msg_text` text,  `msg_subject` varchar(50) default NULL,  `uid` int(11) default '-1',  `root` int(11) default '-1',  `parent` int(11) default '-1',  `msg_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',  `msg_last_edited` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  `msg_flag` tinyint(1) default NULL,  `author` varchar(20) NOT NULL default '',  `cid` int(11) NOT NULL default '0',  PRIMARY KEY (`mid`));

CREATE TABLE `topics` (  `tid` int(11) NOT NULL auto_increment,  `cid` int(11) NOT NULL default '0',  `topic_title` varchar(50) default NULL,  `topic_last_msg_time` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  `topic_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',  PRIMARY KEY (`tid`));
A few permissions changes were made. The *allow_management* permission is now used to determine if a user is allowed to manage permissions whereas it used to allow users to reorder messages. The permission *upload_vid* was added to determine if a user may upload a video; this is separate from the requirement of a valid upload token in the *upload_transactions* table. If the *upload_vid* permission is false, a user may not upload a video at all, regardless of whether there is a valid token in the table. All other permissions have the same purpose as they had in the old system, and these purposes are evident in the permission names.

All other tables were kept as they were in the old system; relationships between tables are largely unchanged, with the exception of the new relationships established with new tables. A diagram of the final database organization, along with all of the SQL used to create the database, is available in Appendix 1 – Database Organization.
5. Assessment

5.1 Assessment Methodology
User goals are assessed with the help of Dr. Lyddon. Developer goals are assessed by the author, who developed and executed the reimplementation. The redesigned and completed implementation serves to an extent as proof of the chosen architecture’s effectiveness. The efficacy and usability of the major changes to the architecture are analyzed, including readability and maintainability, communication protocols, new frameworks and new or improved features. Assessment of usability is made by the author, who not only programmed the new board, but has used the old board for classroom activities in the past. Additional assessment of usability is made through data collected at the University of Aizu while Dr. Lyddon was using the board [25]. The author had no part in the collection of data from users of the board, and it was delivered with no identifying information.

Dr. Lyddon used the new implementation of the OLE Board in his classroom from October 2011 until February of 2012. The board was used weekly for about eight weeks for exchanges in Japanese and English between 27 students at the University of Aizu and 15 students at Rose-Hulman Institute of Technology. After this semester of use, all 42 students who used the board were asked the question “Which technology did you like using better, YouTube or the OLE Board?” Summaries of this data, as well as feedback from Dr. Lyddon, were used in the assessment of usability of the system and its new features [25].

5.2 Assessment of Developer-Oriented Goals

5.2.1 Readability and Maintainability
The use of the object-oriented paradigm was strengthened in OLE, leading to a more solid, readable and maintainable MVC architecture through increased encapsulation, decreased coupling between the view and model code, and increased cohesion among control code in the client. As stated by Perepletchikov et al., “high quality software should exhibit low coupling… and high cohesion” [26]. Flash was originally developed as a “web drawing and animation package,” and “was not intended for traditional application development on or off the web” [27]. This is immediately evident when existing code must be changed in the old system. As discussed in section 3.3.1.1 Developer-oriented Goals, the old Flash development environment targets animators, not programmers. It encourages organization that reduces readability, mixes event handling and control code, and otherwise highly couples the interface with model and control code, decreasing maintainability [26].

In the old client, it is not uncommon to see access to a backend function near or within a function that handles translation of data into a viewable form, increasing coupling between the view code in the client and the model code on the server. The only user-defined “objects” used in the old code are tied directly to the concept of frames, only separating local and global actions. These frames do group functionality to an extent, but if the interface layout were not available, that functionality would not be immediately
apparent. Most names for these groups of code follow the pattern 
[Scope]_Actions_Frame[number], e.g., Local_Actions_Frame3. The set of code that 
manages message viewing and retrieval is stored in the tree component that is used to 
display messages, and it is called FTree_Frame4.

These uninformative names are not a large detriment to readability while developing in 
the Flash environment, but that is because code is inextricably linked to the interface and 
its layout; the only way to access the code is through the interface design. Not only is the 
layout of the interface tightly coupled with its code, but view and model code are also 
tightly coupled between the client and server. Separation of the view from the model 
code is desirable to allow easier development for multiple interfaces. This separation has 
become even more important with the increase in the quantity and type of devices that 
can access Internet applications. For example, Figure 12 gives an excerpt of code from 
FTree_Frame4 that retrieves the children for a message branch.

```
treeListener.nodeOpen = function(eventObject){
    openNodes['' + eventObject.node.attributes.data.mid] = 
        eventObject.node.attributes.data.mid;
    populateTree(eventObject.node.attributes.data.mid);
}
...[code unrelated to children retrieval]...

function populateTree(parentID) {
    messageService.getChildren(confInfo.cid, parentID);
}
...[other interface code excluded]...

getChildren_Result = function(rs) {
...[declarations and initializations excluded]...
for (i = 1; i <= rs.COUNT; i++) {
    var nodeData = new Array();
    nodeData['author'] = rs.DATA[i]['author'];
    nodeData['mid'] = rs.DATA[i]['mid'];
    nodeData['topic'] = rs.DATA[i]['topic'];
    nodeData['created'] = rs.DATA[i]['created'];
    nodeData['hasChildren'] = false;
    nodeData['isTopic'] = false;
...[other processing code excluded for space]...
    if (nodeData.hasChildren){
        treeNodes[0].setIsBranch(treeNodes[nodeData.mid], true);
        if (openNodes['' + nodeData.mid] !== null) {
            treeNodes[0].setIsOpen(treeNodes[nodeData.mid], true);
            populateTree(nodeData.mid);
        }
    }
}
...[code unrelated to children retrieval]...
```

Figure 12. FTree_Frame4: retrieving children for a message branch in the old system.

This excerpt shows the local mixing of model and view code that is common throughout 
the old system. This code is local to the FTree object, meaning that either this component
will have to be reused, which is not possible because it is no longer supported in this form, or the functions for processing children will have to be rewritten for any additional interfaces. The old system makes direct calls to the backend and processes the raw data within its view code.

The new system makes greater use of software objects, increasing encapsulation and reusability. Shown below in Figure 13, the view methods for the retrieval of children in the new system make use of intermediate software objects as source data structures; this is an example of the data-centric design favored by the Flex framework. Since these methods are encapsulated in software objects, they may be imported and reused by additional user interfaces in the future. It is these objects that handle communication with the server, which allows the view code in the client to remain independent of the model code on the server.

```javascript
private function retrieveChildren(event:AdvancedDataGridEvent):void{
    var info:XML = new XML(event.item.toString());
    try{
        var t:Topic;
        var msg:Message;
        for each(var item:XML in info.msg){
            t = usr.conf.topics[String(item.@tid)];
            if(...) {
                msg = t.messages[String(item.@id)];
                msg.getChildren();
            }
        }
    } catch(err:Error){
        ...[alert user of error]...
    }
}
...[other interface methods and components excluded]...
```

Figure 13. Retrieving children for a message branch in the new system.

Cohesion has also been improved with the increased use of software objects without sacrificing the separation of view and control. For example, in the old system’s interface, the message retrieval code is separated from the code for deleting messages, even though both actions manipulate message data. Retrieval of messages through the interface occurs in the $treeFrame4$ component, while deletion occurs in $localActionsFrame5$. This is because the control and view code have not been separated; therefore, message retrieval needs greater access to the $tree$ component to be displayed. Though they are in
separate files, the manipulations implied by both of these methods are intertwined with code to display the data.

In the new system, as is the case with all methods that manipulate message data, the method for message retrieval is part of the Message class, as is the method for message deletion. They are both used through the appropriate event handling methods in the interface code. Display of messages is a view task, and is therefore handled solely within the interface code. Methods that handle movement of data from the model to the view are all located in the interface code, with the same level of access to all interface components. They use the methods in the Message class for data manipulation, rather than directly accessing model code on the server.

Whenever possible, cohesion is preserved in software classes in the client. In some cases, it is preserved even when it increases coupling. For example, a number of global variables are maintained by the various ActionScript classes. The interface code initializes some of these variables across all classes, increasing coupling within the client. However, the variables that the interface uses define the addresses of the services on the backend that are associated with the objects in the client. Therefore, to decrease coupling between the interface code and server-side control code, lower coupling between elements of the client was sacrificed in favor of higher cohesion.

The current approach to video recording is an exception to the preference for higher cohesion. There is no Flex component for video recording, forcing an attempt to design a component from a set of individual components. Its design incorporates a decreased level of encapsulation compared to other components. Its complexity reduces both readability and maintainability. It is imbedded in the main interface, losing the configurable independence of the polished, pre-programmed Flex components. This cobbling together of interdependent components is the same approach used in the old board, and its adoption in this element of the new board may have replicated or introduced the bugs described in section 5.3.1 Video Features. The component became very complicated as development progressed. See section 7.1 Video Recording for further discussion and recommendations.

Though cohesion may not contribute to maintainability of the code [28], it does contribute to readability by giving the developer a clue where they may start looking for target sections of code. The grouping of like functionality is a start toward this increased readability. A simpler, yet possibly more significant, contribution is really only a side effect of this cohesion through software objects, i.e., the descriptive names of many of the source files. These give developers much greater guidance during debugging and maintenance than interface layouts or source files with names like Local_Actions_Frame3.

The introduction of intermediary classes and the use of MXML for interface code also contribute to readability by further separating view and control code. If a developer needs to manipulate the display of data, they know that a good place to start would be in the MXML code of the interface; if they need to make changes to data movement, they know
that a good place to start would be the ActionScript classes. The exception to this rule is the administrator interface, which was developed first and groups all of its result handlers in a single file, but does not provide classes to encapsulate them. The reason for this is described in section 4.2.3 Administrator Interface.

Though intermediary classes constitute part of the solidified controller tier, there is still no controller module that is both fully utilized and completely independent. The majority of the control tier and the view tier have been separated in the interface as described, but they are still both in the interface, and both written ultimately in ActionScript. The ColdFusion application framework, discussed in section 5.2.3 ColdFusion Application Framework, provides an independent controller tier, but is not fully utilized. Therefore, the new OLE is still not completely within the MVC architecture; the tiers have just been more clearly defined.

Architectural and organizational elements aside, ActionScript 3.0 itself offers distinct advantages over its predecessors for application development. Based on the ECMAScript standard, it strengthens object orientation and encourages greater use of the paradigm. It incorporates type safety, improved performance, and a convenient and standardized syntax for XML creation and manipulation. New runtime exceptions make applications easier to debug, and classes in ActionScript 3.0 can now be sealed, which means that methods and fields cannot be dynamically added after an object is created [29].

5.2.2 Communication Protocols

XML (or eXtensible Markup Language) [30] was designed to be naturally extensible, and is not tied to Adobe products, which will allow for a more adaptive communication protocol as changes are made to the system. XML also provides a level of standardization independent of the available function set in the client and backend, which increases maintainability by reducing the amount of code that needs to be reworked as the communication protocol is updated or extended in the future. In addition to function-independent standardization, the hope is that a non-proprietary protocol will give developers more freedom in future design decisions and expansion. XML schema documents or Document Type Definitions can add even more standardization by strictly requiring elements or even data types for the data, which can improve maintainability by making missing tags and other errors in communication more obvious. However, such documents were not drafted for the new implementation of the board.

The natural tree structure coupled with the use of descriptive tags means it is not only readable for a computer, but also readable by humans without any tools more advanced than a simple web browser or even a text editor, which means data moved between the client and the server is readily understandable; it only needs to be intercepted by a tool such as Firebug to be immediately useful to developers.

A downside to XML is its additional overhead. With its headers and repeated syntactical elements such as opening and closing tags, there is more data is sent over the network for each exchange than would be necessary in other protocols. This is why requests requiring less information, like deletions, are done in plain POST variables rather than creating a new XML document only to communicate a single number. An alternative is JavaScript
Object Notation (JSON), which conveys information in name-value pairs [31]. It is also easy for humans to read and write, and is much lighter weight than XML.

JSON was considered for OLE, but XML was chosen for a few reasons. Flex does not provide native libraries to read and produce JSON, while it does provide libraries for XML, along with convenient syntax built into ActionScript 3.0. Tags were kept as small as possible in most common exchanges, utilizing abbreviations for tag names and encoding information in attributes rather than nested tags, decreasing the amount of text needed to relay the required information. Since the OLE Board makes heavy use of video, which takes significantly more bandwidth than text, any additional overhead from XML should be a marginal concern. If it should become a larger concern, XML can be easily compressed.

5.2.3 ColdFusion Application Framework
The ColdFusion application framework provides a standard skeleton and an independent controller tier for MVC applications that will be familiar to many developers. It has been leveraged wherever possible. Many of the available hooks are left unused, but in practically all cases, this is because they are not immediately useful for the purposes of the board. In some cases however, the hooks could potentially simplify the application if they were fully implemented. The lack of complete utilization of the framework is not a large concern because the application framework is built into the ColdFusion application life cycle. Failing to utilize any hook will have no detrimental effect on the application per se since the application will move through all points in the life cycle regardless of whether hooks are implemented for them.

The main point at which the framework could have been more effectively leveraged is in the onCFCRequest method, which allows code to execute before a ColdFusion component runs. Currently, authentication checks for functions requiring elevated privileges are done at the beginning of those functions. This resulted in a lot of repeated calls in code where the single hook provided by onCFCRequest could have allowed a single written call to suffice for all cases. However, this hook was discovered later in the development process, and a reliable method for discovering which ColdFusion component—and more importantly, which method—was being called was not found, so it was left unimplemented.

5.2.4 Flex Framework
The Flex framework provides pre-programmed interface components and a set of development paradigms that focus on data within the application, but the framework is a hefty 1MB for clients to load initially. However, the benefits of the Flex framework may outweigh its large initial download size. Flex lends itself well to the MVC architecture; it can now be used to develop for mobile devices, which is an increasingly important aspect of online applications [32]; it utilizes ActionScript 3.0, which is compliant with the international ECMAScript standard [29]; and it provides many pre-programmed components that manage many lower-level tasks, speeding development.

MXML is a markup language, based on XML, for use within the Flex framework [33]. It is used for the interface components comprising the view, while ActionScript is used for
controlling logic. MXML code and ActionScript code have significantly different syntaxes, making them visually very different, which strengthens the conceptual break between the code tiers on the front end, and contributes to readability as described above. Though using ActionScript objects for control logic does not provide a controller tier independent of the client, the encapsulation provided by the objects allows them to be reused in other interfaces programmed using the Flex framework.

As of late 2011, Adobe has declared its intention to move their support from Flash Player to Adobe AIR for mobile applications [34]. Adobe AIR allows standalone deployment of applications written in web languages like Flex, ActionScript, HTML, and JavaScript [35]. They will also be contributing to the continued growth of HTML 5, but Flex will remain a viable alternative for application development, be it mobile or otherwise.

HTML 5 may overtake Flex in Rich Internet Application development, but it was still under heavy development when the redesign of OLE began. There were multiple issues with the video tag, causing even sites like YouTube and others to delay its adoption. Issues immediately affecting OLE were a lack of microphone and camera access, and an inability to seek in previously recorded video. Other issues that could have affected future development include the inability to view video in full screen and a lack of live streaming [36]. It is likely that many of these issues have been or will be fixed, but for the infant development of a system that depends so much on video, the Flex framework was favored for its more mature libraries for video management.

Choosing Flex also meant fewer cross-browser issues. There are standards set for many web development tools, but they do not always behave the same across all browsers. Flex has significantly fewer issues in this regard since it is developed solely by Adobe, which provides plugins to their specifications for browsers on all supported platforms. For time considerations, opting for tools that would not introduce complications unrelated to the core application seemed a prudent choice. Other than the large download size, going with Flex did not significantly restrict future development possibilities. Flex is compatible with many popular development techniques and tools, including AJAX [37].

Flex even offers an extensive library of interface components, comparable to many offered by HTML and CSS. These components are all pre-programmed, and can even take care of lower-level tasks, freeing the developer to focus on the application rather than its individual elements. The video player component is a key example of this, but it is not the only one. The AdvancedDataGrid component used to display and browse topics, messages and videos, is another great example of the power of Flex components.

The AdvancedDataGrid, following the data-centric philosophy of Flex, provided attributes and tags to bind itself to a data structure managed in ActionScript. The grid could then react to changes in that data structure, populating and repopulating itself as necessary. The developer need only worry about getting the data into the proper structure, and the component manages its display. Since XML is used in data communication, the data can be immediately moved to the proper variable with no further manipulation. The translation of the XML by the grid is accomplished by utilizing the tree structure of
XML. Given an XML document and MXML to describe how the XML should be displayed, the grid will display the information and adapt to changes in the document.

For example, when retrieving the children of a specific message, all the programmer needs to do is call the `getChildren` method, and through event chains, the XML structure containing the set of topics and messages for the current conference will be updated, which will then trigger the `AdvancedDataGrid` to refresh itself. Figure 13 illustrates this. The binding of the grid to the XML structure is done through the `dataProvider` tag in MXML. For comparison, Figure 14 illustrates how translation from data to display had to be handled in the old system. Many lower-level functions for handling XML had to be invoked explicitly, complicating the code and taking focus away from the bigger picture.

```javascript
getChildren_Result = function(rs) {
  hideWaitCursor();
  if (rs.CODE == 0) {
    if (rs.PARENTID == 0)
      topicNodes = new Array();
    
    treeNodes[rs.PARENTID].removeAll();
    if (rs.COUNT == 0)
      treeNodes[0].setIsBranch(treeNodes[rs.PARENTID], false);
    else
      treeNodes[0].setIsBranch(treeNodes[rs.PARENTID], true);
  }

  for (i = 1; i <= rs.COUNT; i++) {
    var nodeData = new Array();
    ... [initialization of nodeData fields excluded]...
    if (rs.DATA[i]["hasChildren"] == 1)
      nodeData["hasChildren"] = true;
    if (rs.PARENTID == 0) {
      nodeData.isTopic = true;
      topicNodes.push(nodeData);
    }
    ... [initialization of node text excluded]...
    treeNodes[nodeData.mid] =
    treeNodes[rs.PARENTID].addTreeNode(nodeText, nodeData);
    if (nodeData.mid == selectedNode.attributes.data.mid) {
      treeNodes[nodeData.mid];
      treeNodes[0].selectedNode = treeNodes[nodeData.mid];
      selectedNode = treeNodes[0].selectedNode;
    }
    treeNodes[0].setIcon(treeNodes[nodeData.mid], "topic_icon");
    if (!nodeData.isTopic)
      treeNodes[0].setIcon(
      treeNodes[nodeData.mid],"message_icon");
  }
  
  if (nodeData.hasChildren) {
treeNodes[0].setIsBranch(treeNodes[nodeData.mid], true);
if (openNodes["" + nodeData.mid] != null) {
treeNode = treeNodes[nodeData.mid], true);
    populateTree(nodeData.mid);
}
}
treeListener.change(null);

Figure 14. Processing XML in old system

5.3 Assessment of User-Oriented Goals

5.3.1 Video Features

Major changes to video playback and recording were made in the new system. Video uploading was also added. There was an improvement over the old board in many of these aspects. In the case of video uploading, the improvement was simply the ability to upload a video. Some slight improvements can be made to the uploading mechanism, including the addition of a progress bar, but it behaves as expected, and most importantly, successfully uploads videos. Improvements in video playback and recording were made for a better quality of experience, though user feedback indicates this may not have been as successful as was hoped.

The interface for video playback is much more unified now. In the old board, the buttons were individual components that were loosely tied together with ActionScript. Occasionally, the play and stop buttons would not perform as expected, and playback would start and stop at seemingly random points in the video. The new board makes use of the Flex video player component, which the author can attest has significantly decreased bugs due to the playback mechanism itself. It incorporates video, playback controls, volume controls, and seeking into one component that can be configured to behave as the developer desires. Such a component was not found for video recording, and this may have played a part in the stability and reliability of that element.

Due to suspected bugs in video recording, the playback experience is still not as smooth as desired, and it is most likely these bugs that led to some of the results reported from user feedback. Videos will occasionally have a large period of time, typically a minute or two, where they will appear not to be playing. It was discovered that they are indeed playing, but the desired video content is at the very end of the video. In these cases, the video will be reported as much longer than it actually is. It can also be frustrating for users, who may believe the video file is broken. The cause has not been determined, but, from testing, it appears that this “dead time” in a video correlates to the time between posting videos. This may point to a bug in video management during recording.

5.3.2 User Feedback on General Usability

User feedback on the board was mixed. Given the choice of OLE, YouTube, or neither, half of all students preferred OLE. The majority of that half was at the University of Aizu, where OLE was preferred by two thirds of students. The ratio was reversed at
Rose-Hulman, where about 60% of students preferred YouTube, which was the tool they had previously used for the same classroom activities. Excluding those who preferred neither OLE nor YouTube, over half of all students preferred OLE, at almost 57%. A summary of these results appears in Table 2 and Table 3.

Table 2. Summary of system preferences

<table>
<thead>
<tr>
<th></th>
<th>Neither</th>
<th>OLE</th>
<th>YouTube</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aizu</td>
<td>7.41%</td>
<td>66.67%</td>
<td>25.93%</td>
<td>100%</td>
</tr>
<tr>
<td>Rose-Hulman</td>
<td>20.00%</td>
<td>20.00%</td>
<td>60.00%</td>
<td>100%</td>
</tr>
<tr>
<td>Total</td>
<td>11.90%</td>
<td>50.00%</td>
<td>38.10%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 3. Summary of preferences, excluding those who answered "neither."

<table>
<thead>
<tr>
<th></th>
<th>OLE</th>
<th>YouTube</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aizu</td>
<td>72.00%</td>
<td>28.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Rose-Hulman</td>
<td>25.00%</td>
<td>75.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Grand Total</td>
<td>56.76%</td>
<td>43.24%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Comments about the board had noticeable trends. Privacy of content was a popular positive aspect of the OLE Board, since it is a closed system. The clear organizational structure of the threads was popular among both instructors and students. Several students noted that they liked the interface, ranging from the general organization to the simplicity of a particular task, such as video recording. Both sides consistently commented that bugs detracted from the use of the new OLE Board. Data was collected during the board’s first use outside of a development environment, so a number of these bugs were expected, and many of them were fixed.

Table 4. Comments from Aizu students

| Comment                        | Number |
|                                |       |
| OLE gives privacy              | 6     |
| OLE is easy to use/has a good interface | 10   |
| OLE is difficult to use/has many bugs | 7    |
| YouTube is easier/better      | 4     |
| YouTube is hard to use/not suitable | 6    |

It is not known which bugs were most frustrating to students, or if it was more a matter of quantity; however, as speculated above, problems with video recording and playback may have spurred many of the comments about bugs and errors. Another likely candidate is the speed, or lack thereof, that students at Rose-Hulman may have experienced. With the server in Japan, it is likely that videos took a while to load at times. Though students from both Aizu and Rose-Hulman were displeased with the bugs, many mentioned that the board would be useful as it matures and the amount of bugs decreases. A breakdown of the comments from students at Aizu is in Table 4. The total does not equal the number of students at Aizu because some comments fell into more than one category.

37
While individual comments were available for students at Aizu, only summaries of comments were available from Rose-Hulman [25]. Overall, the students at Rose-Hulman thought they improved their language skills, which was the purpose of using the board. Of the 15 people who used the board at Rose-Hulman, 14 experienced technical issues. The one who did not vote in favor of the OLE Board; many of the others noted that they might prefer OLE if more of the bugs are fixed. Some students also mentioned that they liked the layout and organization of the OLE Board.

5.3.3 Data Collection

Most of the desired data can now be accurately collected by the board. The changes to the conceptual layout of the system, as well as augmentations to the database, made this possible and will allow for expansion and tweaking of data collecting capabilities. Some difficulties were encountered in the collection of data. Videos uploaded to the board do not go through the recording apparatus, which prevents accurate measuring of recording attempts and replays before posts for those videos. The lengths of uploaded videos are not recorded. Other difficulties encountered in accurately recording data are discussed in section 6.1 Limitations to Data Collection.

As one would expect, when a lot of data is collected, space can be used quickly. This is true in OLE. As discussed above, the activity_log table collects data based on individual events in the system. This allows the freedom to choose how the data will be aggregated, but can also cause a rapid growth in consumed disk space, depending on the frequency of use and the number of conferences that data is collected for. Periodic backups and purges of this table will help manage disk space on the server.

Data collection through the board could prove useful in the future, but the data collected during testing and initial deployment at the University of Aizu was lost when Dr. Lyddon left the university. In addition, there is currently no user interface to view or aggregate collected data outside of the database environment. Therefore, use of the data would be hindered without basic training in MySQL, or until it could be exported from the database.

As discussed, numerous features were added to the database to support data collection, not the least of which was the introduction of videos as a separate conceptual object. Videos now exist in the textual database as objects independent of messages, giving them a more solid presence in the system than was afforded by the hasstream reference in the old system. This was a key factor in the ability to collect the desired data about use of the system since video is a crucial element of a video message board.

Videos are not completely independent, however. The relationship between messages and videos is one-to-one. That is, a video should not exist within the system without a single corresponding message. It may not be associated with any other messages. This is a detriment to reuse of videos in the system, requiring a video to be stored twice, or a symbolic link to be established in video storage, if the video is attached to more than one message. This design choice was made because most use of the board requires new
videos to be created each time a message is posted; however, if an instructor wanted to reuse a video for instance, it would also have to be created anew.

5.3.4 Additional Improved Features

Improvements to the user interface, as well as some enhancements to security, were also made. The conference interface was almost entirely redesigned for a more intuitive user experience. The administrator interface is now more powerful, but was only used by Dr. Lyddon and the author during testing. Security measures were added to protect the database. These are transparent to users, but their feedback indicated that privacy was a significant concern, and some of the added security implicitly protects that privacy.

The conference interface was well received by users of the board. They did not use the old board for comparison, but user feedback indicated that the component for browsing messages was well liked, and even preferred over YouTube’s interface. The administrator interface was redesigned in the hope that it would be more coherent and easy to follow, but some functions were not immediately apparent at times. Some guidance was necessary to fully utilize its functionality. It now offers more powerful features, including the ability to enroll multiple users at once, and batch functions to add multiple users and conferences at once. These functions are designed so limited technical knowledge is needed to use them. For the latter two, an XML document or CSV file can be uploaded with the click of a button. The CSV file will likely prove most useful when user or conference lists are created by a human hand. Example CSV files are in Appendix 2.

In the new system, passwords are stored as hashes so intruders do not have easy access to all users’ passwords and will be forced to perform brute force attacks if they wish to recover them. Password hashes are not salted. Therefore, if an adversary manages to crack a password that is commonly used in the system, they will have access to all user accounts that use that password. SQL queries with user input were not sanitized in the old system. Simple calls are now made to query preparation functions in ColdFusion and PHP to help prevent SQL injection attacks in the new system.

6. Limitations and Challenges

6.1 Limitations to Data Collection

View counts are currently not logged because an accurate and uncomplicated method for counting views was not found. It is possible that a user could start a video and switch to another very quickly. This logically would not be a view, and thus should not be counted, so counting views on the opening of a video is inaccurate. It is also possible to skip through most of the video, or to repeat portions several times before the video finishes playing all the way through. There is a limited amount of access allowed to the Flex video player component, and all attempted methods for dealing with these possibilities accurately without significantly complicating the code were unsuccessful, and thus simply removed.

All aspects of a video are stored in an instance of OleVideo as it is being created. Since uploaded videos bypass this structure, no information about their creation or length can
be easily managed. Uploaded videos only exist as an ID in the textual database and a file in video storage. It may be possible to learn the length of a video after it is uploaded, but that will require post-processing of uploads, which is not currently implemented.

6.2 Dropped features

Several features of the old board were dropped before development on the new design began. Many were intended to be added again after development was complete, but time constraints prevented their reintroduction. Dropped features include:

- read messages
- garbage collection
- email notifications
- live updates to topics, messages, and changes in the message tree
- conference backups

The tracking of read messages was dropped because the chosen method was not readily scalable. It involved an additional table in the database that tracked each message read by each user. Since it is expected that each user in a given conference will read at least half, if not all of the messages in that conference, this table could grow to be quite large. Alongside the activity_log table for the collection of usage data, this could deplete disk space on a lower-end server fairly quickly.

An unrealized goal of the new system was to design it in such a way as to eliminate the need for garbage collection. As development progressed, it became apparent that garbage collection was implemented in the old system for a good reason. Complexities in video management before posting can lead to orphaned videos, and tokens in the upload_transactions table can expire without being removed. As noted in section 5.3.3 Data Collection, data collection can lead to rapid consumption of disk space through the logging of events in the activity_log table. Accounting for all possible use cases in which videos, messages or upload transactions could be left unused in the system would be a complicated endeavor. Performing periodic cleanups is a much simpler prospect.

The remaining features were dropped due to changes in structure and technologies in the system. Email notifications were dropped with the intention to move them to a more suitable ColdFusion component, decreasing coupling between components. Live updates, which propagated changes in topics and messages as they were made to all connected clients, were dropped because they relied on functions in the old client and communication over the Flash Remoting protocol that was used by the old system. When communication between the client and ColdFusion was changed to an XML-based protocol, those old functions became obsolete. Conference backups were dropped because a method would need to be devised for backing up videos attached to posts; only textual data was backed up in the old system. Since a clear method for video handling was not decided at the beginning of development, conference backups were dropped with the intention to re-implement them accounting for the new video organization.
7. Future Work and Recommendations

7.1 Video Recording

A separate, independent component would be ideal for video recording through the interface. Synchronous network calls would be a viable alternative to an independent component and would allow the video recording elements to remain imbedded in the interface while reducing complexity of the code. If either of these options were implemented, it may allow requests for video IDs to be encapsulated—like other data manipulation calls in the system—without increasing the complexity of communication with other elements of the system. A separated component would also bring the design further in line with the Flex framework, allowing it to communicate with the main interface by supplying hooks to event chains through MXML attributes, similar to the AdvancedDataGrid or the userLogin component. Focusing on cleaning up this component has the potential to reduce many of the more frustrating bugs experienced in video recording and playback.

A reworking of the approach to video recording will be necessary to accomplish this. The current component is imbedded in the main interface, and became complicated, making it difficult to maintain. An instance of OleVideo communicates with the textual database and video storage while videos are created. Managing this variable with the current approach requires many use cases to be covered in a small set of code. The general approach to the design of this component was to centralize the management of these use cases as much as possible. To this end, the checkVideo method was written, growing more complex as development progressed. The main purpose of this function is to clean up the instance of OleVideo after it is used so it is ready for the next posting. Further complicating the component is the ability to include or exclude video, audio, or both.

This complexity led to a decrease in the typical encapsulation of requests to the server, as full encapsulation would even further complicate the management of videos. Encapsulation of a request for a new video ID, for example, would mean an event chain contained completely within the OleVideo class. The method that made the call would have to block until the asynchronous call completed; no technique for accomplishing a block like this was found. Another option would be to create another event chain that could be invoked after the encapsulated call was complete, but in the current video recording process, this seemed like an additional complication with little benefit. Time constraints prevented the implementation of a completely separate component for video recording.

An alternative to an independent component is to implement synchronous network calls for video management. While this is not a standard Flex design paradigm, it is possible [38]. Synchronous network calls would allow video management to be completely encapsulated while leaving video recording imbedded in the main interface. Such calls would also avoid overcomplicated event chains that would be necessary if encapsulation was preserved and the recording element was left imbedded. Because the video management functions would block until a response was received from the server, elements of the interface would not have to block and wait for an event to be triggered.
telling them it is safe to proceed. The results of the call could also be directly returned; returning data from network calls via events may require a custom event to be designed in order to carry the desired data.

7.2 Future Improvements

Future improvements should initially focus on addressing bugs encountered during development that detracted from user experience. Implementing one of the suggestions from the previous subsection should contribute to smoother development and debugging. A method for garbage collection can then be implemented to keep the system running efficiently. After the system is more stable and reliable, dropped features can be reintroduced, followed by improvements to features, efficiency, and security. Development of new features to support data collection and other goals of the board can then continue.

Aside from improvements to video recording, two elements of the system will require attention. The permissions tab in the conference interface does not communicate properly with the backend, necessitating permissions changes to be made through the database. The deletion of topics in ColdFusion should be reworked to delete not only the messages within the topics, but the topics themselves. Currently, empty topics are not removed after their messages are deleted.

A number of changes to the designs of dropped features will be required to reintroduce them into the system. Conference backups will have to be redesigned to accommodate the new structure of topics and messages, as well as to incorporate the video files in addition to textual data. The method for tracking previously viewed content may need to be redesigned to increase scalability. The old design may suffice if viewed content is accounted for in the garbage collection system. Implementing email notifications should be a fairly simple process; however, handling of dates will need to be reconciled between the client and database in order for notification windows to be properly scheduled. Live updates to content can be implemented through Flash Media Server, but functions to respond to these updates will need to be rewritten in the client. It may also be necessary to reintroduce the Flash Remoting protocol for communication of live updates between Flash Media Server and the client.

The system would benefit from a method of garbage collection. There are a couple of possible solutions that would keep the system running efficiently. The first is to implement a garbage collection system utilizing the ColdFusion application framework. Hooks at the beginning of a request could be used to check the database and video storage for orphaned videos, expired upload transactions, or a bloated activity_log table. If any of them grow beyond a specified threshold, they could be removed. If the first approach is determined to be unsuitable for the needs of the board, scripts at the operating system level could be written and scheduled to periodically clean the system.

A number of new features could support data collection and improve user experience. An additional interface could be developed to allow researchers to access and manipulate collected data. This would reduce the amount of technical expertise required to extract
useful data from the database. A mechanism to toggle data collection for conferences after they are created would allow greater control of data. Though it would not support data collection, a progress bar for video uploads is a small addition that could significantly impact user experience.

8. Conclusion

The Model-View-Controller architecture remains the best model to describe the architecture of the OLE Board. With the introduction of the Flex framework, this architecture is strengthened with further separation of the view and controller elements through the use of MXML for display and ActionScript for control logic. The object orientation of ActionScript 3.0 affords the board an updated and more common design paradigm that is more suitable for RIA development than the older and animation-oriented Flash environment. The application framework available through ColdFusion introduces an independent controller tier, and provides organization and some convenient hooks in the ColdFusion application life cycle.

Decreased coupling between the view and model tiers increased maintainability and expandability. With increased coherency also afforded by object orientation, the code is easier to follow, which is important in the system’s event-driven environment. XML is used for most textual communication, giving a communication protocol independent of the function set available in the system and independent of the technologies used to develop elements of the system. Changes to the database allowed for the addition of new features like data collection and video uploading.

Users appreciated the clarity and simplicity of the interface, but were discouraged by bugs encountered during the development process. Though many bugs were fixed, only about half of the students preferred OLE over YouTube. However, many students noted that the board was useful and could be even more useful as development progresses. Though data collection was implemented, it is not currently accessible without basic knowledge of SQL.

During development, some limitations to the full implementation of data collection were discovered. It was also necessary to cut some features that existed in the old implementation. Several bugs were found, and many were corrected, but some remain. Future efforts will initially focus on addressing bugs and the reimplemention of dropped features. Dr. Lyddon’s students will take over in development, and use of the board will eventually expand to a regional and national scope in Japan before the ultimate goal of an international audience.

With further work to reduce the number of bugs remaining in the system, as well as the reintroduction of the dropped features and addition of new features, user feedback indicates that the board will be a useful tool as it matures.
References


[32] "Mobile Development Using Flex 4.5 SDK and FlashBuilder 4.5 | Adobe Developer
http://www.adobe.com/devnet/flex/articles/mobile-development-flex-

[33] Trilemetry, "Introducing the MXML and ActionScript Languages," Adobe, 22
March 2010. [Online]. Available:
15 August 2012].

[34] D. Winokur, "Flash to Focus on PC Browsing and Mobile Apps; Adobe to More
Aggressively Contribute to HTML5," Adobe, 9 November 2011. [Online].
[Accessed 6 August 2012].


30 June 2010. [Online]. Available:
http://www.wired.com/business/2010/06/youtube-html5-video-is-no-match-for-
flash/. [Accessed 6 August 2012].

August 2012].

Appendices

Appendix 1 – Database Organization

New OLE database organization. Some tables have been preserved from the old board, but a number of new tables and the necessary relationships have been established.

-- MySQL dump 10.11
-- Host: localhost  Database: ole
-- -------------------------------------------------------
-- Server version 5.0.77

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
DROP TABLE IF EXISTS `activity_log`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `activity_log` (  
`aid` int(11) NOT NULL auto_increment,  
`cid` int(11) NOT NULL default '0',  
`uid` int(11) NOT NULL default '0',  
`user_ip` varchar(40) NOT NULL default '',  
`time` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
`action` varchar(15) NOT NULL default '',  
`object` varchar(15) NOT NULL default '',  
`oid` int(11) NOT NULL default '0',  
`description` varchar(100) default NULL,  
PRIMARY KEY  (`aid`) ) ENGINE=InnoDB AUTO_INCREMENT=2359 DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;

DROP TABLE IF EXISTS `conf_role_perm`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `conf_role_perm` (  
`cid` int(11) NOT NULL default '0',  
`rid` int(11) NOT NULL default '0',  
`add_topic` tinyint(1) NOT NULL default '0',  
`rem_topic` tinyint(1) NOT NULL default '0',  
`upd_topic` tinyint(1) NOT NULL default '0',  
`allow_text` tinyint(1) NOT NULL default '0',  
`allow_audio` tinyint(1) NOT NULL default '0',  
`allow_video` tinyint(1) NOT NULL default '0',  
`allow_management` tinyint(1) NOT NULL default '0',

48
`rem_message` tinyint(1) NOT NULL default '0',
`msg_size` int(11) NOT NULL default '0',
`msg_length` int(11) NOT NULL default '0',
`reply_self` tinyint(1) NOT NULL default '0',
`reply_topics` tinyint(1) NOT NULL default '0',
`reply_all` tinyint(1) NOT NULL default '0',
`upload_vid` tinyint(1) NOT NULL default '0',
PRIMARY KEY (`cid`, `rid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

SET character_set_client = @saved_cs_client;

-- Table structure for table `conferences`

DROP TABLE IF EXISTS `conferences`;
SET character_set_client = utf8;
CREATE TABLE `conferences` (
  `cid` int(11) NOT NULL auto_increment,
  `conf_name` varchar(100) NOT NULL default '',
  `data_collection` tinyint(1) NOT NULL default '1',
  PRIMARY KEY  (`cid`)
) ENGINE=MyISAM AUTO_INCREMENT=323 DEFAULT CHARSET=utf8;

SET character_set_client = @saved_cs_client;

-- Table structure for table `default_rolePerm`

DROP TABLE IF EXISTS `default_role_perm`;
SET character_set_client = utf8;
CREATE TABLE `default_role_perm` (
  `rid` int(11) NOT NULL default '0',
  `add_topic` tinyint(1) NOT NULL default '0',
  `rem_topic` tinyint(1) NOT NULL default '0',
  `upd_topic` tinyint(1) NOT NULL default '0',
  `allow_text` tinyint(1) NOT NULL default '0',
  `allow_audio` tinyint(1) NOT NULL default '0',
  `allow_video` tinyint(1) NOT NULL default '0',
  `allow_management` tinyint(1) NOT NULL default '0',
  `rem_message` tinyint(1) NOT NULL default '0',
  `msg_size` int(11) NOT NULL default '0',
  `msg_length` int(11) NOT NULL default '0',
  `reply_self` tinyint(1) NOT NULL default '0',
  `reply_topics` tinyint(1) NOT NULL default '0',
  PRIMARY KEY (`rid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
`reply_all` tinyint(1) NOT NULL default '0',
`upload_vid` tinyint(1) NOT NULL default '0',
PRIMARY KEY (`rid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

SET character_set_client = @saved_cs_client;

--
-- Table structure for table `enrollment`
--

DROP TABLE IF EXISTS `enrollment`;
SET @saved_cs_client = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `enrollment` (
    `UID` int(11) NOT NULL default '0',
    `CID` int(11) NOT NULL default '0',
    `RID` int(11) NOT NULL default '0',
    PRIMARY KEY (`UID`,`CID`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;

--
-- Table structure for table `messages`
--

DROP TABLE IF EXISTS `messages`;
SET @saved_cs_client = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `messages` (
    `mid` int(11) NOT NULL auto_increment,
    `tid` int(11) NOT NULL default '0',
    `msg_text` text,
    `msg_subject` varchar(50) default NULL,
    `uid` int(11) default '-1',
    `parent` int(11) default '-1',
    `msg_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',
    `msg_last_edited` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
    `msg_flag` tinyint(1) default NULL,
    `author` varchar(20) NOT NULL default '',
    `cid` int(11) NOT NULL default '0',
    PRIMARY KEY (`mid`)
) ENGINE=MyISAM AUTO_INCREMENT=154 DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;

--
-- Table structure for table `roles`

```
DROP TABLE IF EXISTS `roles`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `roles` (  
  `RID` int(11) NOT NULL default '0',  
  `role_name` varchar(25) NOT NULL default '',  
  PRIMARY KEY (`RID`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;
```

--

-- Table structure for table `topics`

```
DROP TABLE IF EXISTS `topics`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `topics` (  
  `tid` int(11) NOT NULL auto_increment,  
  `cid` int(11) NOT NULL default '0',  
  `topic_title` varchar(50) default NULL,  
  `topic_last_msg_time` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  `topic_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',  
  `email_start` timestamp NOT NULL default '0000-00-00 00:00:00',  
  `email_end` timestamp NULL default NULL,  
  `uid` int(11) NOT NULL default '-1',  
  PRIMARY KEY (`tid`)  
) ENGINE=MyISAM AUTO_INCREMENT=27 DEFAULT CHARSET=utf8;
SET character_set_client = @saved_cs_client;
```

--

-- Table structure for table `upload_transactions`

```
DROP TABLE IF EXISTS `upload_transactions`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `upload_transactions` (  
  `id` varbinary(16) NOT NULL default '\0',  
  `time` timestamp NOT NULL default CURRENT_TIMESTAMP,  
  `ip` varchar(40) default NULL,
```
DROP TABLE IF EXISTS `users`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client  = utf8;
CREATE TABLE `users` (  
  `uid` int(11) NOT NULL auto_increment,  
  `username` varchar(20) NOT NULL default '',  
  `passhash` char(64) default NULL,  
  `given_name` varchar(15) default '',  
  `family_name` varchar(15) default '',  
  `email` varchar(100) default '',  
  `creation_IP` varchar(40) NOT NULL default '',  
  `creation_date` timestamp NOT NULL default '0000-00-00 00:00:00',  
  `last_access_date` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  `nonce` int(16) default NULL,  
  `verified` tinyint(1) default NULL,  
  `superuser` tinyint(1) NOT NULL default '0',  
  `last_access_IP` varchar(40) default '',  
  PRIMARY KEY (`uid`),  
  UNIQUE KEY `nonce` (`nonce`) ) ENGINE=InnoDB AUTO_INCREMENT=49 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `videos`;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
CREATE TABLE `videos` (  
  `vid` int(11) NOT NULL auto_increment,  
  `mid` int(11) NOT NULL default '0',  
  `vid_length_sec` int(11) default '0',  
  `vid_last_viewed_time` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  `vid_created_date` timestamp NOT NULL default '0000-00-00 00:00:00',
);
`vid_last_edited` timestamp NOT NULL default '0000-00-00 00:00:00',
    PRIMARY KEY (`vid`),
) ENGINE=MyISAM AUTO_INCREMENT=418 DEFAULT CHARSET=utf8;

SET character_set_client = @saved_cs_client;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2012-07-19 15:18:27
### Appendix 2 – Administrative Batch Operations File Formats

The following tables describe the format of CSV files for batch functions in the administrator interface. The first is for adding users to the system, and the second is for adding conferences. Headers are provided for informational purposes only; they should not be included in any CSV files that are used in the system. Bolded lines indicate that these entries will be rejected as invalid. Asterisks indicate required fields. All other fields are optional.

#### User CSV File

The text that is in the password field will be the corresponding user’s password. Passwords are hashed during processing.

<table>
<thead>
<tr>
<th>Username*</th>
<th>Password*</th>
<th>Given name</th>
<th>Family name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>newuser15, asdf, Dude, Sweet, <a href="mailto:dude@sweet.com">dude@sweet.com</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>newuser16, asdf, Dude, Sweet,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>newuser17, asdf, , , <a href="mailto:dude@sweet.com">dude@sweet.com</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>newuser18, , Dude, Sweet, <a href="mailto:dude@sweet.com">dude@sweet.com</a></strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>, asdf, Dude, Sweet, <a href="mailto:dude@sweet.com">dude@sweet.com</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>newuser20, asdf, , ,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Conference CSV File

Valid values for the *data collection* field are 1 or 0, TRUE or FALSE, or yes or no.

<table>
<thead>
<tr>
<th>Name*</th>
<th>Data collection*</th>
</tr>
</thead>
<tbody>
<tr>
<td>test33</td>
<td>1</td>
</tr>
<tr>
<td>test34</td>
<td>FALSE</td>
</tr>
<tr>
<td>test35</td>
<td>yes</td>
</tr>
</tbody>
</table>