A Browser-Based Collaborative Multimedia Messaging System

Susan Gayle Gentner

Follow this and additional works at: https://scholarworks.gsu.edu/cs_theses

Part of the Computer Sciences Commons

Recommended Citation
ABSTRACT

Making a communication tool easier for people to operate can have profound and positive effects on its popularity and on the users themselves. This thesis is about making it easier for people to publish web-based documents that have sound, video and text. Readily available software and hardware are employed in an attempt to achieve the goal of providing a software service that enables users to compose audio-video documents with text.

INDEX WORDS: Applet, Communication tools, Internet, Java, Software as a service, Multimedia, Web Cam, Web Service
A BROWSER-BASED COLLABORATIVE MULTIMEDIA MESSAGING SYSTEM

by

SUSAN GENTNER

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2009
A BROWSER-BASED COLLABORATIVE MULTIMEDIA MESSAGING SYSTEM

by

SUSAN GENTNER

Committee Chair:    Xiaolin Hu

Committee:         Anu Bourgeois
                   Michael Weeks

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
May 2009
DEDICATION

This goes out to my aunts Barbara and Martha, whose warm and steady love gives me strength even when I am lonely in my search to build and learn.
ACKNOWLEDGEMENTS

My husband Matt gave me guidance and love that has gotten me to this point in my career. He is always there to lend a helping hand, to give me encouragement when I need it, and to look out for me even when I am stubborn.

Next, my advisor and committee, Dr. Hu, Dr. Bourgeois, and Dr. Weeks. It goes without saying they taught me much but they also inspired me to learn and to seek what I was interested in.

Last, but certainly not least, are my best friends and fellow academics Stefanie Markham and Mary Hudacheck-Buswell. They help me in numerous simple ways like going to lunch and commiserating with me about various topics. They have and always will hold a special place in my life.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS..................................................................................................................v

LIST OF FIGURES...........................................................................................................................ix

1. INTRODUCTION...........................................................................................................................1
   1.1 Background..................................................................................................................................1
      1.1.1 Participation Grows...........................................................................................................1
      1.1.2 An Aesthetic Mosaic.........................................................................................................1
   1.2 Problem Definition....................................................................................................................4
      1.2.1 Awkward Composition Process........................................................................................5
      1.2.2 Clogged Transport Medium...........................................................................................6
      1.2.3 Configuration Mismanagement.......................................................................................6
   1.3 Alternative Methods ................................................................................................................8
      1.3.1 Publishing Beats Sending...............................................................................................8
      1.3.2 No System is Perfect.......................................................................................................9

2. APPROACH AND CHALLENGES...............................................................................................11
   2.1 Approach..................................................................................................................................11
   2.2 Challenges................................................................................................................................13
   2.3 Concepts..................................................................................................................................16
   2.4 Related Technologies...............................................................................................................17
      2.4.1 Webmail............................................................................................................................17
      2.4.2 Transport Layer Security.................................................................................................18
5.1 Future Work........................................................................................................41
5.2 Development Environment...............................................................................42
BIBLIOGRAPHY.......................................................................................................48
APPENDIX: SOURCE CODE LISTING....................................................................52
LIST OF FIGURES

Figure 1: Firefox peeks from beneath the text-only Lynx web browser........2
Figure 2: A graph of early web usage data from [1]..................................3
Figure 3: Something is missing on Bob's end.........................................7
Figure 4: Viewer in my MomentShare desktop application from 2006........13
Figure 5: Browser-based moment viewer with better video in 2009.........14
Figure 6: A browser warning about an untrusted certificate signature......19
Figure 7: PushToShow has a three-tier architecture...............................27
Figure 8: Class Diagram of the Client Side.........................................29
Figure 9: Web Service Classes.........................................................30
Figure 10: Record moment sequence diagram.....................................30
Figure 11: Send moment sequence diagram........................................31
Figure 12: The ShareMoment web page.............................................32
Figure 13: The ListMoments web page ..............................................34
Figure 14: The TakeMoment web page ..............................................35
Figure 15: A PushToShow workstation...............................................37
Figure 16: Some cameras did not perform as advertised.........................39
Figure 17: PushToShow achieves high video quality..............................40
Figure 18: The pgAdmin3 database management tool.........................43
Figure 19: The NetBeans IDE..........................................................44
Figure 20: Applet class diagram.....................................................52
Figure 21: CapturePanel class.................................................................54
Figure 22: FileUploader class...............................................................58
Figure 23: Moment class.......................................................................60
Figure 24: MomentUploader class........................................................63
Figure 25: MomentUploadSvc class......................................................65
Figure 26: VideoUploadSvc class..........................................................68
Figure 27: WebCam class......................................................................71
1. INTRODUCTION

This thesis is about making it easier for people to publish web-based documents that have sound, video and text.

1.1 Background

The main motivation for this work is that making a communication tool easier for people to operate can have profound and positive effects on its popularity and on the users themselves. A more elegant tool that requires fewer steps to use is easier to learn and will return some emphasis from limits of technology to the vastness of users' creativity.

1.1.1 Participation Grows

Sometimes a little difference can mean a lot. When web pages could be viewed with stylized text flowing around colorful in-line images, Internet content began to grow exponentially and the number of web pages doubled every three months. And the same functional power of publishing and linking live documents from all over the Earth that was still there.

1.1.2 An Aesthetic Mosaic

To users in 1993, the Mosaic graphical web browser made it much easier on their eyes to read web pages than the text-based Lynx web browser, created just one year earlier. Mosaic was received like a dazzling new theater in
towns and cities across the globe. Its HTML audience grew wildly, and, as
more was given to users, they frequently got the itch to give back. Figure 1
illustrates the visual differences between a text-only and graphical web
browsers. And figure 2 shows the increase in HTTP traffic of web pages after
the introduction of the graphical Mosaic web browser in 1993 [1].
In retrospect, the move from text-only to graphical web browsers seems like an obvious change that might also have been an easy change. For a dozen years prior, Tim Berners-Lee and countless others had spent millions of hours (and billions of dollars) to do the “heavy lifting” while building the world wide web. In American history, it is the railroads' and interstate highways' completions that we still celebrate today, more than specific models of locomotives and automobiles.

*Figure 2: A graph of early web usage data from [1].*
In figure 2, I have graphed some per-protocol traffic measurements slightly before and after the debut of Mosaic. It is probably for the simple reason that Mosaic could graphically display the multimedia of web pages that the audience for hypertext transfer protocol (HTTP) began to grow, usurping the file transfer protocol (FTP) in early 1995 as the most-used category of Internet traffic. Users really appreciate elegance in their tools, and Mosaic changed the way that most people worked on the Internet.

The graphical Mosaic browser was built by two guys in just six months; just two! [2] That does not seem to be enough time to create a single folk song like those sung by the teams that were working on the railroad. And yet their achievement is stunning nonetheless. When Marc Andreessen and Eric Bina created Mosaic, they brought a tremendous amount of new technology to bear fruit from a simple vision. The small dare between Marc and Eric produced an elegant tool that made such a big and positive difference. It would be thrilling for me to achieve one millionth of the public service that Marc Andreessen and Eric Bina brought to us, by creating Mosaic.

1.2 Problem Definition

Part of the challenge here is seeking an “ideal” elegance, similar to reducing a complex and verbose mathematical proof. Often, the first impression that a user gets with document creation software is a lengthy installation process
that consumes part of their computing device. And after a customary
reboot, the user braces herself to descend into a new “confusopoly” [3] that
she has just committed to, and begin the complex and esoteric process of
creating her first multimedia document. The logistics involved in sharing her
art with its intended recipients are another unsavory challenge, but she
keeps her mind clear to focus on what should be simple: creating a message
that has some video and text.

1.2.1 Awkward Composition Process
When Alice bought her laptop, she was thinking ahead and she picked a
model that included a built-in web cam. Her friend Bob was stationed in Iraq
then, and now Bob is fighting in Afghanistan. Since the camera was already
attached and staring back at her, Alice assumed that there would be some
easy software included that would help her to share a moment with Bob that
was comprised of some sound, video, and text. She wanted to send
something personal just for Bob, that would remind him about the great
home he had to return to someday.

For her Microsoft Windows-based laptop, Alice found a free Microsoft
download for something called MovieMaker that helped her to make a fun
15-minute home movie. The movie was ready and Alice was proud of it, so
she attached it to an e-mail message and sent it off to Bob. Instead of any
thanks from Bob on the next day, Alice found a cryptic error message that indicated her home-movie e-mail message to Bob had been rejected because it “exceeded his incoming e-mail message size limit(s).”

1.2.2 Clogged Transport Medium

Luckily for Alice, the MovieMaker tool had video editing features that she learned to use to pare down her home movie to just the juicy highlights. The movie for Bob was a small fraction of its original length and size, and Alice used zip compression to shrink the movie another 12% before attempting to re-send it as an e-mail attachment to Bob.

This time Bob received the message from Alice. He called Alice on the phone while her message, bigger than most, was being retrieved from the e-mail server. Bob apologized that he would need to delete the message from his inbox after it was safely on his Mac, because he needed inbox space to receive other e-mail that might come.

1.2.3 Configuration Mismanagement

Bob was an Army man, good at following instructions. So he quickly and carefully followed the instructions in Alice's e-mail to:

1. save the attached movie,
2. note its location on his file system,
3. unzip the compressed file to get at the real movie that Alice had sent,

4. and delete the zipped clutter to save some disk space.

Now Bob was starting to feel some suspense because he had been growing to like Alice more during his lonely missions overseas. It made Bob feel great that Alice had sent this special treat.

Bob was curious and uncertain as to what he would see, but he was positively certain that he would enjoy the home movie from Alice and be watching it frequently until she sent the next movie. When the file was done being decompressed, Bob used Finder on his Mac PowerBook to Open the movie file. Yet instead of Alice's smiling face, a terse error message appeared that read:

![QuickTime error message](image)

*Figure 3: Something is missing on Bob's end*
1.3 Alternative Methods

Soon Alice was back to square #1 with a different approach. She would make a third movie, this time less personal. And she would revise the written note to be less personal too. Then Alice would upload the movie with the message to YouTube and e-mail the URL to Bob.

Alice and Bob were grateful that this YouTube approach worked pretty well. Alice did not get everything she originally wanted in her movie to Bob, and it was still awkward and time-consuming to capture the video, edit a movie and upload her final cut. But YouTube had helped to solve the codec and transport problems they had had at first, so Alice and Bob could feel better connected despite being apart.

1.3.1 Publishing Beats Sending

The publishing and distribution method seemed cohesive, and the codec mismatch problem was solved in one fell swoop. And, assuming the sender already had a video capture tool installed, there was no required software except for a Flash-enabled web browser. All Alice needed to do was to register for a YouTube account and use her web browser to upload her video moment, complete with sound and with text. This software as a service (SaaS) freed Alice and Bob from some obstacles that are side-effects of proprietary software. But, was this a trade-off into other forms of lock-in?
Although the original intended audience was just Bob, Alice needed to edit her home video because YouTube does not have fine-grained security and most video is more or less public. Once Alice uploads her home movie, it becomes transcoded into a proprietary Adobe Flash codec and that version is held as content owned by YouTube. It is likely that keyword matching is done by the parent company Google to arrange advertisements in and around Alice's home movie. So Bob sees a few sidebar gift ideas for Alice along with the moment that she originally intended to share.

1.3.2 No System is Perfect

Being centralized and proprietary were key business aspects to protecting the investors first and then later Google as the parent company of YouTube. This seems prudent and fair because YouTube cost hundreds of millions of dollars to launch, plus about a million dollars per day for bandwidth costs. After 18 months, Google announced that it had bought YouTube for $1.65 billion- which serves as a ballpark figure of the startup cost for the new video sharing website. But when founders of YouTube Chad Hurley, Steve Chen and Jawed Karim launched their website in February of 2005, they eliminated most of the obstacles that Alice and Bob had encountered.

Part of the motivation of this work is to explore the feasibility of restoring privacy and freedom to a medium that is similar to YouTube in convenience
to users, but less encumbered with proprietary modification and ownership restrictions. Rather than an ending, I see that there has been an exciting new opportunity for building helpful software. Instead of a “killer” application that might lock-out developers, I see a fascinating new range of ways that I might be able to serve users by helping to make it quicker and easier to publish their new creations.

The central question of this thesis is whether it is feasible to make it easier for people to record and exchange moments with current commodity personal computing hardware. To answer this, in the next chapters I describe my understanding of the technical challenges. Then I detail an approach to meet these challenges with a system design and an implementation that demonstrate success for the most difficult challenges.

My PushToShow prototype is included as an original part of this thesis work, and this prototype will prove that browser-based web pages today can “see” and “hear” their end-user's scenario. This state of the art in web pages may be easily and precisely controlled by casual users to capture and share important moments from their time and space with others.
2. APPROACH AND CHALLENGES

The approach that I took was to build a prototype that could be used to effectively demonstrate my concept of an easy tool for people to create messages with sound, video and text.

2.1 Approach

This thesis has an emphasis on development because Murphy's Law seems to be the *only* law when it comes to building a browser-based multimedia creation and exchange prototype. Without trying some techniques by building and testing bits of software together, I anticipated that my planning and research would seem to have little meaning because, as Murphy states, “Whatever can go wrong WILL go wrong.” So I set to work to build a working prototype with four main attributes:

1. ease of use
2. browser-based
3. great video quality
4. use free or cheap components

Ideally, I hope that my prototype will grow to a stage where it may be adopted for incubation into an open source project that students and volunteers will enjoy benefiting-from and contributing-to. So I have tried to use free or cheap hardware and software components. The ease of use and
browser-based goals were intended to greatly help me in demonstrating the prototype, which itself is meant to promote the feasibility of concepts for improvement in this thesis. And the browser-based goal also brings a tacit advantage in that the prototype will be a web application; that there will be no lengthy installation for end users, and that it may be extensible with standard off-the-shelf security enhancements.

My skepticism for planning too far out (without a prototype) comes from a term project that I built three years ago during a course called “Digital Signal Processing” taught by Dr. Michael Weeks. In a way, that term project was a prototype for this prototype (figures 4 and 5 respectively); at that time called “MomentShare.” It succeeded in showing that it is feasible to build a Java desktop application that will capture from a web cam and help a user to send the “moment” with text over an e-mail transport. And it showed that there were two big areas of improvement:

1. browser-based instead of desktop installation
2. web publishing instead of message sending

In the time since I built my MomentShare prototype, an unrelated website for sharing family photos [4] has been established on the Internet. So although I still refer to my new message type as a “moment,” I refer to my new browser-based prototype as PushToShow.
Challenges

The goal of great video quality comes from the typical “how fast can it go?” curiosity that I share with other software developers. But it also stems from a probable future requirement that users will want their moments to look good, even when the video portion of each moment is presented in full screen mode. I wanted the moments to have a fast frame rate that keeps recorded motions fluid and with a high resolution that approaches conventional NTSC analog TV (640x480@25hz) to keep image shapes crisp and clear. So during a summertime of trial and error I was able to find a configuration that captured 15 frames per second at VGA resolution.

Figure 4: Viewer in my MomentShare desktop application from 2006

2.2 Challenges
and still keep CPU utilization down to 25% and affordable compressed bit-rate bandwidth of barely 1 megabit per second.

I found that there are copious quantities of published research in a myriad of topics that pertain to the “signs of aging” that e-mail is exhibiting [5] [6]. The main problems seem to be related to lack of security. For decades now, billions of people have learned first-hand that it is too easy for attackers to get fixed e-mail addresses and that leaving the door open to one's mailbox is not a sustainable option.
Billions of people have also been suffering unsolicited postal mail for decades as well. And although their postal addresses are fixed, there seem to be far fewer concerns about security of people's postal mailboxes.

If we try to compare an e-mail inbox to a U.S. Postal mailbox, then senders need to pay postage for each postal message and federal law prohibits anyone other than an official mail carrier to put things into someone's mailbox. Although recipients often disdain the deluge junk mail received, their suffering would be much worse if it were free to send junk mail. Here, another small difference means a lot: a small 20-cent bulk mail rate IS a cost, and this cost makes senders more eloquent. If the public were allowed to physically deposit whatever we liked into people's mailboxes for free, then sadly the term “junk mail” would take on greater scale and a whole new meaning. But that is the case with e-mail. After senders get an e-mail address, they may attempt to send whatever they like; directly to the recipient and without any postage paid.

To attach video files to this antiquated medium would resemble the “pig in a python” metaphor and make a bad situation even worse. Even with compression, video files are much bigger than today's typical e-mail message, by two or three orders of magnitude. Here web-mail and RSS technologies have helped to show that, at a minimum, the user must be given a chance to see the sender's identification, the subject line, and
message date prior to requesting the rest of the message. And, that when it comes to giving the user some features to inspect attachments prior to downloading them: the more, the merrier.

2.3 Concepts

Today we have so many new multimedia tools! There are person to person aids that range from instant-messaging clients to VoIP and video streaming tools including AIM, Skype, and Ekiga. Audio-enabled remote desktop sharing tools such as Microsoft Remote Desktop (RDP) Virtual Network Computing (VNC) and LiveMeeting. And also we have browser-based collaborative tools such as YouTube and Wiki knowledge bases.

If we pick one or more of these tools for substitution, then some valid questions might take the form “why not use X instead of PushToShow?” So here are some attempts to answer those permutations with some hasty generalizations. Instant messengers are interpersonal with an emphasis on text-based exchange, and media other-than text are handled as attachments. VoIP is interpersonal and session-based, with an emphasis on audio exchange. The RDP and VNC tools work well to interact with a remote GUI desktop but the video quality during capture and/or playback is low in terms of resolution, color depth and frame rate. YouTube and Wikis are very close, in that they allow authors to easily publish multimedia documents.
which include video. But they tend to be role-based rather than interpersonal and do little to help capture video for a shared moment, the way webmail helps writers to compose an e-mail in an integrated editor. So (to me) the current technology that is most related seems to be webmail.

2.4 Related Technologies

Webmail helps to retract the transmission cost from the recipient's terminal workstation to somewhere closer to the recipient's e-mail server. This is subtle but evident in that the recipient can peruse her inbox, mark her spam for deletion, view and respond to her good mail— all without saving any residue to the workstation she is using. Rather than dumping each message on each recipient's workstation before a message preview is available, the recipient may get her work done at arm's length and may opt to download mail pieces of particular interest after quick and careful scrutiny of their sizes and types. With webmail, she gets the meta data first to help her decide whether she wants to get her hands dirty with the complete message's details. Still, candy from strangers might be flung into her bag, but with webmail she can run a series of taste-tests without toxic risk.

2.4.1 Webmail

Some specific examples webmail that have great features and security are: Google Gmail, Yahoo! Mail, and the open-source SquirrelMail project.
In terms of security, webmail often shines with the following features:

1. identity management
2. transport layer security
3. collaborative spam control
4. freedom from software installation
5. automatic vulnerability patching

The first two items are configurable and standards-based and they are optional features that come along with any browser-based web application. First, the configurable options for identity management in web applications are plentiful these days, reminiscent of the pluggable authentication management (PAM) framework for UNIX logins.

2.4.2 Transport Layer Security

Second, transport layer security (TLS) now has a new name since it has succeeded the patented [7] Netscape secure sockets layer (SSL) protocol ten years ago. But TLS is rock-solid, familiar to users, and has enormous industry support. TLS version 1 came from SSL version 3 with a handshaking protocol so advanced that checksum and cipher algorithms may be relaxed or made stronger on-the-fly within an established session. But casual users would hardly know that the attributes of their cipher suite are being re-negotiated and instead, another “small change” is probably one that has had greater meaning to end users.
From a more practical point of view, the default lists of recognized certificate authorities that browsers and the Java runtime keystore come with has grown. As a specific example, ten years ago a website administrator may have been asked to pay over $800 to have her web server’s certificate signed by a VeriSign [8] CA, or over $600 for a Thawte [9] signature. But today she can get an equivalent GoDaddy signature for $100 [10]. Her users will enjoy the same benefits of secure TLS and they will not be made nervous by any “Untrusted Certificate” warnings from their web browsers.

Figure 6: A browser warning about an untrusted certificate signature
Technical users may often know how to manage their registries of trusted CAs and they will comfortably navigate a browser warning with confidence. Yet casual non-technical users are likely to be confused, unsettled, annoyed or otherwise dissuaded from continuing to a website whose certificate is signed by a CA that their browser does not recognize. Figure 6 shows an example of the type of warning a user may get when a certificate is not recognized. This is a small security-related configuration change that has been ironed-out over the years. The browser’s trusted CA lists continue to grow and improve for users, and these improvements are another important incentive enjoyed by secure browser-based web applications.

2.4.3 Spam Control

Collaborative spam control (#3) in webmail is an innovative way for peer users to submit instant feedback that identifies spam and cooperate in reducing its damage to webmail systems. This supplies critical early warning value to a webmail system that is similar to Google Flu Trends [11], a site that geocodes web search queries for influenza-related answers and estimates the current risk of flu outbreaks for a given geographic region. These are modern “complaint boxes” that can record details during the onset of an attack instantly and from the front line.
2.4.4 Automatic Updates

The webmail security features listed #4 (freedom from software installation) and #5 (automatic vulnerability patching) are also small differences that were touched upon in chapter 1. Software installation and patching are obstacles to end-users' adoption for three main reasons.

They are obstacles first because they are annoying; installation procedures interrupt productivity, they are time consuming and they may require the user to save all work and then reboot. Second- because astute users have been conditioned over time to suspect side-effects to installation and patching: that they might not easily be able to un-install, that something else that they have may be broken, that their system will be slower, and/or that they might be admitting a trojan horse or “spyware” onto their system. And third, they might not be allowed to install or patch software on a public or shared workstation because their account lacks sufficient permission; or they may work daily with diverse systems such as Linux, FreeBSD and/or Solaris for which the installations and/or patches are not available.

Browser-based web applications help users to enjoy effortless security enhancements and automatic vulnerability patching at a few levels that are worthy of some mention here. The web application may be improved incrementally on its server-side, such that the user gets the latest and
greatest version each time she logs-in to get webmail. The built-in web
browsers that come bundled with operating systems are automatically
updated since they are considered system components. And some third-
party web browsers such as Firefox are beginning to include their own
integrated and automatic update mechanism for the core browser and any
installed plug-ins. For example, the Java runtime and applet plug-in has its
own update manager that may be automatic on Microsoft Windows or
manual (push-button) on other operating systems [12].

2.5 Related Work

Here I have listed some works that are related to this thesis topic and can be
grouped into three main areas: multimedia, collaborative systems, and
multimedia architectures.

2.5.1 Multimedia

A system called PECOLE provides portability by using Java and JXTA to keep
systems compatible and well-performing on the widest range of devices
[13]. Second, PECOLE eases on-line teleconferencing and topic presentation
with a layered and peer to peer architecture. Third, PECOLE assists in
locating peers, maintaining sessions, and in locating (or providing)
translations for multilingual content.
Separately, author W. L. Yeung describes some of the current strengths of wiki technology as 1.) students already use wikis and accept wikis as great tools to create and explore content, and as such 2.) the use of wikis by students has been growing quickly. Wikis have potential for accelerated growth through integration of multimedia content including instant messaging and video conferencing to help in holding wiki users' attention [14].

In [15] a detailed description is provided for a system called “mc³” which makes multimedia publishing tasks quicker and easier. First, mc³ helps promote reuse of previous and related publications. Second, mc³ enables on-line access to a search-able and growing collection of learning modules. Third, mc³ facilitates federation between institutions to examine, exchange and enrich research and learning materials. The learning materials apparently tend to be in multimedia, and the mc³ services are designed to help teachers to collaborate. This leads to the next section, where I have listed some works whose primary emphasis seems to be on aiding-in users' collaboration while working with multimedia documents.

2.5.2 Collaboration

Anupam and Bajaj describe their work to support cooperative and accelerated application-level work-group development [16]. They call their
system “Shastra” and it has been documented to coordinate and serialize concurrent substrate transactions which contribute to group-wise scientific and engineering design. The heart of their system is a generic Shastra layer that is extensible and supports a wide variety of collaborative applications.

Mobile generation and representation of multimedia content is provided by a system built by Prabhu and Gadh [17]. One key to their design was a unified file format with delivery via message queue middle-ware. The middle-ware described in their work is intended to help direct migration from desktop collaborative systems toward mobile collaborative systems.

A work that explores both collaborative and architectural aspects of a multimedia system is described in [18]. Their methods for creation, revision and collaborative exchange of multimedia documents with high performance are kept flexible by developing a Jabber service oriented architecture (SOA). The article elaborates on an emphasis for keeping the multimedia document state consistent, and in keeping concurrent document changes coherent in their appearance for up to six concurrent end users.

2.5.3 Architecture

Requirements traceability during design, implementation and maintenance of large scale systems may be aided by adopting a SOA service strategy
planning space, which helps to map problems to solutions [19]. In their perspective, there seems to be a conceptual problem space above the service strategy planning space which contains a domain area, a context, and one or more business drivers. Below the service strategy planning space, their solution space contains engineering, business, and operations aspects of a SOA. Their interesting “service strategy” seems to be a set of functions that relate a domain of problems to a range of solutions in a traceable way that will succinctly indicate precise service changes required to accommodate healthy changes in the problem and solution spaces.

Separation of concerns for authoring services and publishing services of multimedia content is exemplified in the LimSee3 platform architecture [20]. According to Mikáč, Roisin, and Le Duc, adoption of new technologies is simplified and deployment of authoring services is facilitated by having an intermediate export format. This separation of concerns helps to insulate authoring tools from document formats and makes it easier to tailor authoring services to users' custom needs.

An experimental media sharing application for mobile phones called “Sandboxes” seeks to promote interactivity, flexibility, and cohesiveness for mobile authors of multimedia [21]. Sandboxes allow users to compose
messages that mix multiple media into a single document. The Sandboxes creators refer to this as a “multimedia collage on a shared 2D canvas.”

These advances in multimedia, collaborative, and architectural technologies would help to sustain and entice a user base that is eager to create and exchange multimedia about their lives. So the Internet continues to grow with more personal computers and hand-held devices being added each minute. Now there are new possibilities opened to bring last mile broadband to subscribers with government help. Rural wireless "third pipes" that are both fat and cheap over 802.11n or 802.11y may spur growth of green shoots on grass roots networks that are fed by ubiquitous and instant authors of community home grown content [22][23].
3. SYSTEM DESIGN AND ARCHITECTURE

Some of the technologies planned for use in my approved thesis proposal were found to be unfit for one reason or another. And some goals proved to be too optimistic for my pace of development in the time that I had. My prototype still has a lot of growing to do if it will become an everyday tool. Both the design and implementation of the PushToShow have been kept as small and simple as possible in order to make it easier for adoption by others in their work.

3.1 Design Overview

The PushToShow prototype was built with a simple three-tier architecture that consists of a server-side with an application server and database that supports browser-based clients over HTTP:
As figure 7 shows, on the client-side there are three web pages to list moments, view moments, and capture moments. These HTML pages incorporate JavaScript, the QuickTime and Java plug-ins respectively. The ListMoments page uses JavaScript to open a new tab to view each moment that is clicked. The TakeMoment page uses the QuickTime plug-in to play the video portion of each moment. And the ShareMoment page uses the Java plug-in to capture, compress, and upload new moments.

All end-user features are browser-based, and all of the traffic between the client and server is over HTTP. Displayed output is downloaded over HTTP as HTML, images and video. Newly created moments are uploaded from the end-user's browser to the server over HTTP by using multi-part upload for the video portion and object serialization for the textual portions of shared moments. The term moment is used here to describe a new type of audio-visual document which is composed and published by a PushToShow author, such that a PushToShow recipient may then view the moment upon request.

### 3.2 Architecture

The following figures 8 and 9 show the UML class diagrams for the PushToShow Applet and web services.
Figure 8: Class Diagram of the Client Side
The following figures 10 and 11 show UML sequence diagrams for recording and sending a moment.

**Figure 9: Web Service Classes**

**Figure 10: Record moment sequence diagram**
Figure 11: Send moment sequence diagram
4. DESCRIPTION OF PROTOTYPE

This chapter describes ordinary use cases for the PushToShow prototype, how to setup its development environment, and an overview of its design.

4.1 Send a Moment

A moment is like an e-mail message that includes sound and video. And like e-mail messages, moments must be composed and sent. PushToShow makes composing moments easier with a browser-based WYSIWYG authoring tool that includes a real-time graphical web-cam "mirror:"

![Figure 12: The ShareMoment web page](image)

*It has a large web-cam mirror and is used to compose and send moments.*
4.1.1 Capture the Video Portion

Here are the steps to capture your moment with PushToShow:

2. Verify video input, compression type, and audio input settings
3. Click the “Record” button to begin video capture
4. Click the “Stop” button when finished with a video “take”

4.1.2 Compose the Moment

Life's best moments must be composed with care. With PushToShow, it is easy to retry video takes and add a little side note to your moment:

1. The last take is the take that counts
2. Video takes are saved to the home directory with a time-stamp in the form psv_YYYYMMDD_hhmmss.mov
3. It is easy to edit the video portion with QuickTime Pro before sending
4. Type a caption or text message into the large text area on the right-hand side of the ShareMoment page

4.1.3 Send the Moment

It is time to share the moment that you have captured and composed:

1. Address the recipient
2. Enter the subject line
3. Click the “Send” button
Receive a Moment

It is fun and easy to take a moment with PushToShow, and saving moments locally is the same as saving ordinary web pages:

4.2.1 Select a Moment


2. Moments are listed chronologically with sender ID and subject line (see figure 13)

3. Click an entry to view a moment
4.2.2 View a Moment

The viewer has plain text fields for meta-data and each URL is of the form:

http://localhost:8080/pss/TakeMoment.jsp?id=123573625140463

Beneath the image, the compact video playback controls may be used to:

1. adjust sound volume,
2. start or pause playback,
3. scan through frames of video,
4. single-step forward or backward,
5. and save the video portion of the moment to disk.
4.2.3 Save a Moment

Each moment is an ordinary web page, and may be saved to disk locally:

1. Use your built-in browser “Save page as..” feature
2. Keep your locally saved moments organized with browser bookmarks

4.3 Environment and Hardware Notes

Faster computers today have more storage and gigabit network connections that helped to make this prototype feasible. Providing video capture with compression and a simulcast viewing window each add up to a computing cost that has recently become affordable, even for video quality that approaches that of analog television.

The PushToShow prototype was built with many free and open source tools. And PushToShow is designed to work on any modern Mac. But here I want to quickly list my hardware and software suggestions for great video and low cost. Here is a parts list with early 2009 price estimates:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22-inch DVI monitor</td>
<td>$250.00</td>
</tr>
<tr>
<td>2</td>
<td>Logitech QuickCam Vision Pro</td>
<td>$150.00</td>
</tr>
<tr>
<td>3</td>
<td>Apple Mac Mini</td>
<td>$900.00</td>
</tr>
<tr>
<td>4</td>
<td>USB Mouse Controller</td>
<td>$20.00</td>
</tr>
<tr>
<td>5</td>
<td>USB Keyboard</td>
<td>$30.00</td>
</tr>
<tr>
<td>6</td>
<td>QuickTime Pro (video editing software)</td>
<td>$35.00</td>
</tr>
</tbody>
</table>
4.3.1 Mac Mini

PushToShow was developed-on and tested-with “Mac Mini” PCs from Apple, purchased new in August of 2007. This version of the Mac Mini has a 64-bit 1.8GHz Intel Core 2 Duo processor which has 2GB of memory and a front-side bus frequency of 667MHz. While testing PushToShow, I found that moments occupied about 1GB per hour of recorded video and that PushToShow uses about one megabit of network bandwidth per user.
The Mac Mini is the cheapest Mac offered by Apple, and it is the most versatile. By “versatile,” I mean that the Mac Mini is compatible with PC hardware and can run Microsoft Windows or Linux in a dual-boot method with a new Mac OS 10.5 feature called “boot camp.” The Mac Mini’s small size and light weight make it portable between school, work, and home; the Mini has very low power consumption and it is virtually silent. The tamper-resistant case for the Mac Mini is wrapped with a heat-sink which helps keep required airflow (and dust) minimal and acts as a tough aluminum “bumper.” Also the legendary security of Mac OS 10.5 helps to keep the Mac Mini safe against attack and “student resistant” to damage. So I think that these attributes make the Mac Mini a good PC for a classroom environment.

4.3.2 Logitech UVC Webcam

The Mac laptop models from Apple come with a built-in camera positioned just above the screen. But in my showroom testing (at BestBuy) they only seemed to capture about 4 frames per second at a resolution of 640x480 pixels. Still, those built-in cameras performed better than most of the external USB web-cams that I tested. Because in my testing I found that the performance specifications listed on the side of each box were wildly overstated. Often the cameras could not be set at the resolutions that were claimed, and rarely did I get the frame rates that were promised. Luckily they are quite cheap though and I was able to test quite a few cameras.
The Logitech QuickCam Vision Pro is a new Universal Serial Bus Video Class (UVC) camera that debuted about twelve months ago, in March of 2008. It is the only UVC camera that Apple would permit an Apple-logo for, from Logitech. I found that the QuickCam Vision Pro captures 15 frames per second of 640x480 (VGA) resolution video, about three time more video than any other USB camera that I tried; and it is compatible with both Microsoft Windows and Mac. This is the first and only UVC camera that I tried, since it lived up to its advertised performance. It would be interesting to try some other UVC cameras because this first UVC camera works so well.

Figure 16: Some cameras did not perform as advertised
The details of a movie that was captured using PushToShow with the Logitech QuickCam Vision Pro are shown in figure 17.

![Movie Details](psv_20090227_184812.mov)

**Figure 17: PushToShow achieves high video quality**
5. CONCLUSION AND FUTURE WORK

For this thesis I designed and built a prototype of a small tool that might help users to express themselves. My research and development took about ten months and I found several ideas for features that did not seem to work yet, but they might pan-out in later revisions. The video capture portion seemed more difficult than imagined and the web-cam “mirror” is the part that I am most proud of.

The questions that I intended to explore are centered around how pleasant this chore could be made for users. And the PushToShow prototype is roughly as pleasant as I can imagine in that it is easy to use and it produces the best quality moments feasible with ordinary consumer equipment. The PushToShow prototype is clearly a big step forward from my earlier MomentShare project, both in terms of performance and ease of use.

5.1 Future Work

With another iteration this prototype will be ready for incubation as an open source project with educational value. After the PushToShow project is opened up for collaborative scrutiny and development then I believe that some features may be added to make PushToShow fit for an enterprise setting.
Priority number one will be to configure PushToShow to work with a web-based single sign-on authentication package like OpenSSO. This will give the senders of moments identity. With TLS and OpenSSO, PushToShow will have enterprise-class authentication and transport layer security that is easy for end-users, especially if they are already authenticated with another OpenID friendly web application. Other future work should include efforts to make PushToShow portable to Microsoft Windows and Linux by using the Java Media Framework (JMF) for which there are free native “performance packs” available to download and distribute from Sun Microsystems. Also, some usability features ought to be added such as better status messages and progress bars where applicable. And it would make PushToShow more practical if it were a drag-n-drop target for end-users' external home-made videos and/or screen-casts in addition to capturing video from their web-cams.

5.2 Development Environment

I am hopeful that others will gain interest in PushToShow and so for due diligence here I would like to describe how to set up its development environment. The development environment consists of a NetBeans IDE workspace with three projects, a designated directory to store video files and a small PostgreSQL database.
All of the application-layer source code is contained in the three small NetBeans projects, named with the three-lettered acronyms (TLAs) psa (applet) pss (servlets) and psv (video store.) There is a minimal source
code base consisting of three JSPs and eight Java files, with a pair of deployment descriptors for the **pss** and **psv** web applications. The **psv** multipart upload service and video store was separated from the **pss** web application because I kept accidentally deleting movies that were in the video store when I performed a “clean” task. With the **psv** code separated from **pss**, I only need to backup my video files when I make changes to the **VideoUploadSvc.java** source code file. Most of the Java code is for the **PushToShow applet**, and runs in the end-user's web browser when she visits the **ShareMoment.jsp** web page to capture a new moment. And to simplify
the NetBeans workspace, all eight of the Java source code files are gathered into the edu.gsu.cs.ps package, which is located in the psa project.

The PushToShow database schema is also made to be about as simple as I could have imagined, with just one table to hold the moments, a pair of indices, and three simple stored procedures. The indices are on two table columns that appear in query WHERE and ORDER BY clauses, for recipients and sent-time. Three stored procedures SELECT moments for the ListMoments and ViewMoment web pages, and INSERT moments for the ShareMoment page. Database performance is improved by cheaper record lookups with the indices and by reducing SQL parsing time with the stored procedures. Also, the stored procedures were added because (as parameterized queries) they reduce web application exposure to SQL-injection style attacks and serve as security precautions.

To work with the PushToShow project on Mac OS 10.5 developers need to install three free and open-source tools, download the PushToShow source code, create the database and open the NetBeans workspace. The tools required are NetBeans IDE version 6.5 (figure 19), the PostgreSQL version 8.2 relational database management system (RDBMS) and the PgAdmin III graphical database client utility (figure 18).
Each of these tools has an intuitive installation procedure and their websites are well documented, so their download and installation should only take about an hour of time. In my experience, the only “gotcha” I had was that my Mac needed to be restarted for PostgreSQL to be started as a new service. Since then, PostgreSQL has automatically started every time I have booted my Mac PC. When the tools are installed, developers should download the PushToShow source code from:

http://codd.cs.gsu.edu/~swatson13/thesis/PushToShowSrc.zip

or

http://www.pushtoshow.net

After unpacking the project source code file there will be two files called “readme.txt” and “PushToShow.sql” and a subdirectory called “ps.” It should only take about an hour to browse the “readme.txt” file, load the “PushToShow.sql” database DDL file into a PostgreSQL database, and open the three small Java projects in the “ps” directory with the NetBeans IDE.

Then the deployment descriptors (called web.xml) will need to be reviewed and maybe edited in the pss and psv projects, to revise database connection settings and designate a directory for storing moments' video files. This will also be detailed in the readme.txt file and should just take a few minutes to do. Finally, the three projects may be highlighted in
NetBeans IDE and a “clean and build” task invoked. When the pss and psv projects are highlighted and at last their “run” or “debug” tasks are invoked, the ShareMoment and LisMoments pages should be available at:

http://localhost:8080/pss/ShareMoment.jsp

and

http://localhost:8080/pss/ListMoments.jsp

This project comes from me watching scenes of people receiving important messages that they watched and listened-to on screens or as holograms in futuristic science-fiction movies. It comes from my understanding of previous and current attempts to make communication quicker, easier, and more accurate for ordinary people. And it comes from my practice in learning dozens of tiny elements of modern computing science over the past four years in Atlanta at Georgia State University.

It would mean the most to me if I could string together the little pieces of my understanding to organize a voluntary and educational project whose free and open-source software products help people to communicate and record their experiences. And this thesis came from my desire to be someday recognized for serving others by contributing to some slow sweet progress for the human race.

[2] Mosaic -- The First Global Web Browser,
http://www.livinginternet.com/w/wi_mosaic.htm

http://www.dilbert.com/blog/entry/confusopolies/


http://patft.uspto.gov/netacgi/nph-Parser?
Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetacgi%2FPTO
%2Fsrchnum.htm&r=1&f=G&l=50&s1=5657390.PN.&OS=PN/5657390
&RS=PN/5657390


[10] Choose your SSL Certificate!, GoDaddy,


[12] Java Control Panel,
    http://java.sun.com/j2se/1.5.0/docs/guide/deployment/deployment-guide/jcp.html


The Applet class is the main point of entry for the PushToShow application's ShareMoment page.

```java
/* Susan Gentner
 * PushToShow Masters Thesis
 * Georgia State University
 * March 2009
 */
package edu.gsu.cs.ps;

import javax.swing.*;
import java.awt.Component;
import quicktime.*;

public class Applet extends JApplet {
    protected WebCam camera;
    protected final String className;
    protected CapturePanel capturePanel;

    public Applet() {
        className = this.getClass().getName();
        System.out.println( (new java.util.Date()).toString() + " - " +
            className + ",Applet() called.");
        try {
            QTSession.open();
        } catch (QTException ex) {
            ex.printStackTrace();
        }
        System.out.println( (new java.util.Date()).toString() + " - " +
            "QTSession.open() called.");
        System.out.println( (new java.util.Date()).toString() + " - " +
            "QTSession.canDoFullScreen() = " +
            QTSession.canDoFullScreen() );
        System.out.println( (new java.util.Date()).toString() + " - " +
            "QTSession.hasSecurityRestrictions() = " +
            QTSession.hasSecurityRestrictions() + "\n" );
    }

    public void init() {
        // init code here
    }

    public void destroy() {
        // destroy code here
    }
}
```

Figure 20: Applet class diagram
```java
System.out.println((new java.util.Date()).toString() + " - " + className + ".init() called.");
try {
camera = new Webcam();
camera.setPreview();
camera.startCapture();
camera.setPreview();
camera.startCapture();
camera.popDeviceSelections();
camera.popCodecs();
camera.setMsgUploadURL(getParameter("MsgUploadServletURL"));
camera.setFileUploadURL(getParameter("FileUploadServletURL"));
System.out.println("camera.getFileUploadURL()= '" +
camera.getFileUploadURL() + "'");
System.out.println("camera.getMsgUploadURL= '" +
camera.getMsgUploadURL() + "'\n");
Component imageComponent = camera.getMirror();
capturePanel = camera.getCapturePanel();
capturePanel.setImageComponent(imageComponent);
} catch (Exception e) {
e.printStackTrace();
}
this.setBackground(new java.awt.Color(255, 255, 255));
this.setSize(1005, 600);
this.getContentPane().add(capturePanel);
}

public void stop() {
System.out.println((new java.util.Date()).toString() + " - " +
className + ".stop() called.");
camera.stopGrabbing();
System.out.println((new java.util.Date()).toString() + " - " +
"WebCam.stopGrabbing() called.\n");
}

public void destroy() {
System.out.println((new java.util.Date()).toString() + " - " +
className + ".destroy() called.");
QTSession.exitMovies();
System.out.println((new java.util.Date()).toString() + " - " +
"QTSession.exitMovies() called.\n");
QTSession.close();
System.out.println((new java.util.Date()).toString() + " - " +
"QTSession.close() called.\n");
}
```
CapturePanel is the main GUI component in the ShareMoment web page.

```java
package edu.gsu.cs.ps;

import java.awt.*;
import javax.swing.*;
import java.util.Date;
import java.text.SimpleDateFormat;
import java.awt.event.ActionListener;
import org.netbeans.lib.awtextra.*;

public class CapturePanel extends JPanel {
    
    protected final JButton btnSend;
    protected final JPanel imagePanel;
    protected final JLabel lblPostage;
    protected final JLabel lblAudioInput;
    protected final JLabel lblDateSent;
    protected final JLabel lblRecipient;
    protected final JLabel lblSubject;
    protected final JLabel lblVideoInput;
    protected final JTextArea taBodyText;
    protected final JTextField tfSubject;
    protected final CardLayout cardLayout;
    protected final JTextField tfRecipient;
    protected final JScrollPane spBodyText;
    protected final JToggleButton btnRecord;
    protected final JComboBox cboAudioInput;
    protected final JComboBox cboVideoInput;
    protected final JLabel lblVideoCompression;
    protected final JComboBox cboVideoCompression;
    protected final SimpleDateFormat simpleDateFormat;

    public CapturePanel() {
        
    }
}
```

Figure 21: CapturePanel class
lblVideoInput = new JLabel();
lblAudioInput = new JLabel();
cboVideoInput = new JComboBox();
cboAudioInput = new JComboBox();
lblVideoCompression = new JLabel();
cboVideoCompression = new JComboBox();
spBodyText = new JScrollPane();
taBodyText = new JTextArea();
tfRecipient = new JTextField();
lblRecipient = new JLabel();
tfSubject = new JTextField();
lblSubject = new JLabel();
btnSend = new JButton();
btnRecord = new JToggleButton();
lblDateSent = new JLabel();
lblPostage = new JLabel();
cardLayout = new CardLayout();
imagePanel = new JPanel(cardLayout);
simpleDateFormat = new SimpleDateFormat("h:mm a z    E dd MMM, yyyy");
setBackgroundColor(new Color(255, 255, 255));
setMinimumSize(new Dimension(920, 560));
setPreferredSize(new Dimension(930, 560));
setLayout(new AbsoluteLayout());

lblVideoInput.setBackground(new Color(255, 255, 255));
add(lblVideoInput, new AbsoluteConstraints(177, 510, -1, -1));

lblAudioInput.setBackground(new Color(255, 255, 255));
add(lblAudioInput, new AbsoluteConstraints(745, 510, -1, -1));

cboVideoInput.setBackground(new Color(255, 255, 255));
add(cboVideoInput, new AbsoluteConstraints(259, 505, -1, -1));

cboAudioInput.setBackground(new Color(255, 255, 255));
add(cboAudioInput, new AbsoluteConstraints(827, 505, -1, -1));

cboVideoCompression.setBackground(new Color(255, 255, 255));
add(cboVideoCompression, new AbsoluteConstraints(461, 510, -1, -1));

cboVideoCompression.setBackground(new Color(255, 255, 255));
add(cboVideoCompression, new AbsoluteConstraints(553, 505, -1, -1));

spBodyText.setPreferredSize(new Dimension(100, 84));
taBodyText.setColumns(20);
taBodyText.setRows(5);
taBodyText.setPreferredSize(new Dimension(100, 80));
taBodyText.setLineWrap(true);
spBodyText.setViewportView(taBodyText);
add(spBodyText, new AbsoluteConstraints(645, 190, 335, 290));
tfRecipient.setText("bob");
add(tfRecipient,
    new AbsoluteConstraints(740, 130, 240, -1));

lblRecipient.setBackground(new Color(255, 255, 255));
lblRecipient.setHorizontalAlignment(SwingConstants.RIGHT);
lblRecipient.setText("To:");
lblRecipient.setHorizontalTextPosition(SwingConstants.RIGHT);
add(lblRecipient,
    new AbsoluteConstraints(710, 130, -1, 20));

lblSubject.setBackground(new Color(255, 255, 255));
lblSubject.setHorizontalAlignment(SwingConstants.RIGHT);
lblSubject.setText("Subject:");
lblSubject.setToolTipText("Subject");
lblSubject.setHorizontalTextPosition(SwingConstants.RIGHT);
add(lblSubject,
    new AbsoluteConstraints(680, 160, -1, -1));

tfSubject.setText("sharing a moment with ya...");
add(tfSubject,
    new AbsoluteConstraints(740, 160, 240, -1));

btnRecord.setBackground(new Color(255, 255, 255));
btnRecord.setText("Record");
btnRecord.setToolTipText("Record a video clip");
add(btnRecord,
    new AbsoluteConstraints(0, 505, -1, -1));

btnSend.setBackground(new Color(255, 255, 255));
btnSend.setText("Send");
btnSend.setToolTipText("Send the complete message");
add(btnSend,
    new AbsoluteConstraints(90, 505, -1, -1));

lblDateSent.setBackground(new Color(255, 255, 255));
lblDateSent.setHorizontalAlignment(SwingConstants.RIGHT);
lblDateSent.setText(simpleDateFormat.format(new Date()));
add(lblDateSent,
    new AbsoluteConstraints(740, 70, 240, 80));

ImageIcon keyFrame =
    new ImageIcon(getClass().getResource("Postage.jpg"));
lblPostage.setIcon(keyFrame);
add(lblPostage,
    new AbsoluteConstraints(640, 5, 343, 104));

add(imagePanel,
    new AbsoluteConstraints(0, 0, 640, 480));
}

public void addActionListener( final ActionListener actionListener )
{
    btnRecord.addActionListener( actionListener );
    btnSend.addActionListener( actionListener );
    cboAudioInput.addActionListener( actionListener );
cboVideoInput.addActionListener( actionListener );
cboVideoCompression.addActionListener( actionListener );
}

public void setImageComponent( final Component imageComponent )
{
    imagePanel.add( imageComponent, "capture" );
}

public JToggleButton getRecordButton()
{
    return btnRecord;
}

public JComboBox getCboAudioInput()
{
    return cboAudioInput;
}

public JComboBox getCboVideoInput()
{
    return cboVideoInput;
}

public JComboBox getCboVideoCompression()
{
    return cboVideoCompression;
}

public JButton getSendButton()
{
    return btnSend;
}

public JTextArea getTaBodyText()
{
    return taBodyText;
}

public JTextField getTfRecipient()
{
    return tfRecipient;
}

public JTextField getTfSubject()
{
    return tfSubject;
}

public JPanel getImagePanel()
{
    return imagePanel;
}
FileUploader is used by the applet to upload the video portion of the moment.

```java
package edu.gsu.cs.ps;

import java.io.File;
import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.multipart.*;
import org.apache.commons.httpclient.params.HttpMethodParams;

public class FileUploader implements Runnable
{
    protected final String targetURL;
    protected final String fileName;

    public FileUploader( final String _fileName,
                        final String _targetURL )
    {
        fileName = _fileName;
        targetURL = _targetURL;
    }

    public void run()
    {
        System.out.println( "Transferring File: '" +
                            fileName + "' 

                        );
        try
        {
            File transferfile = new File( fileName );
            PostMethod filePost = new PostMethod( targetURL );
            filePost.getParams().setBooleanParameter( HttpMethodParams.USE_EXPECT_CONTINUE, false );
            try
            {
                final Part[] parts =
                {
                    new FilePart( "userfile", transferfile )
                };

                filePost.setRequestEntity( new MultipartRequestEntity( parts,
                                                            filePost.getParams() ) );

                final HttpClient client = new HttpClient();
                client.getHttpConnectionManager().
                        getParams().setConnectionTimeout( 5000 );
        }
    }
```
int status = client.executeMethod( filePost );

if( status == HttpStatus.SC_OK )
{
    System.out.println("'" + fileName +
            '" - Upload complete, " +
            "response=" +
            filePost.getResponseBodyAsString() +
            ");
    System.out.println("Transfer resulted: " +
            filePost.getResponseBodyAsString()
            + "\n\n");
}
else
{
    System.out.println("'" + fileName +
            '" - Upload failed, response=" +
            HttpStatus.getStatusText( status ) +
            ");
    System.out.println("Transfer resulted: " +
            HttpStatus.getStatusText( status )
            + "\n\n");
}

} catch( Exception ex )
{
    System.out.println("ERROR: " + ex.getClass().getName() +
            "" + ex.getMessage() );
    ex.printStackTrace();
    finally
    {
        filePost.releaseConnection();
    }
    System.out.println( "Transfer complete.\n\n" );
    System.out.println( "Drop files here. " );
    return;
} catch( Exception e )
{
    e.printStackTrace();
}
}
Moment is the metadata for a PushToShow moment, it does not include the video portion but it does have the video file name.

```java
public class Moment
{
    protected long id;
    protected long sentTime;
    protected String senderName;
    protected String recipients;
    protected String subject;
    protected String bodyText;
    protected String videoName;

    public Moment()
    {
    }

    public Moment( final ResultSet rs ) throws SQLException
    {
        id = rs.getLong( "id" );
        sentTime = rs.getLong( "sentTime" );
        senderName = rs.getString( "senderName" );
        recipients = rs.getString( "recipients" );
        subject = rs.getString( "subject" );
        bodyText = rs.getString( "bodyText" );
        videoName = rs.getString( "videoName" );
    }

    public String toString()
    {
        return ( "id = " + id +
            ", sentTime = " + ( new Date(sentTime) ).toString() +
            ", senderName = " + senderName +
            ", recipients = " + recipients +
            ", subject = " + subject +
            ", bodyText = " + bodyText +
            "");
    }
}
```

Figure 23: Moment class
        ", videoName = '" + videoName + '" 
    
    public String getRecipients()
    {
        return recipients;
    }

    public void setRecipients(String recipients)
    {
        this.recipients = recipients;
    }

    public String getSubject()
    {
        return subject;
    }

    public void setSubject(String subject)
    {
        this.subject = subject;
    }

    public String getBody()
    {
        return bodyText;
    }

    public void setBody(String body)
    {
        this.bodyText = body;
    }

    public String getVideoName()
    {
        return videoName;
    }

    public void setVideoName(String videoName)
    {
        this.videoName = videoName;
    }

    public long getId()
    {
        return id;
    }

    public void setId(long id)
    {
        this.id = id;
    }

    public long getSentTime()
    {
        return sentTime;
    }
```java
public void setSentTime(long sentTime)
{
    this.sentTime = sentTime;
}

public String getSenderName()
{
    return senderName;
}

public void setSenderName(String senderName)
{
    this.senderName = senderName;
}
```
MomentUploader uploads the moments to the server.

```java
package edu.gsu.cs.ps;

import java.net.URL;
import java.io.IOException;
import com.oreilly.servlet.HttpMessage;

public class MomentUploader implements Runnable {
    protected final Moment moment;
    protected final String msgUploadURL;
    protected final String fileUploadURL;
    protected final FileUploader fileUploader;

    public MomentUploader( final Moment _moment,
        final String _msgUploadURL,
        final String _fileName,
        final String _fileUploadURL ) {
        moment = _moment;
        msgUploadURL = _msgUploadURL;
        fileUploadURL = _fileUploadURL;
        fileUploader = new FileUploader( _fileName, _fileUploadURL );
    }

    public void run() {
        System.out.println("Uploading moment with subject line '" +
            moment.getSubject() + "\n\tto URL '" +
            fileUploadURL + "' ");
        try {
            System.out.println("Starting to upload the file");
            fileUploader.run();
            System.out.println("No exceptions after upload. ");
            URL url = new URL( msgUploadURL );
            HttpMessage msg = new HttpMessage(url);
        }
    }
}
```

**Figure 24: MomentUploader class**

MomentUploader uploads the moments to the server.
msg.sendPostMessage(moment);

try {
    msg.sendPostMessage(moment);
}

} catch ( IOException e )
{
    e.printStackTrace();
}

System.out.println("Done uploading moment with subject line ", moment.getSubject() + ";
};
MomentUploadSvc - This class implements a web service that receives new moment meta-data by object serialization over HTTP. As each new moment is received, a matching record is created in the “moments” database table.

```java
/* Susan Gentner
 * PushToShow Masters Thesis
 * Georgia State University
 * March 2009
 */
package edu.gsu.cs.ps;

import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.DataSource;
import javax.naming.InitialContext;

public class MomentUploadSvc extends HttpServlet {
    protected void processRequest(
            final HttpServletRequest request,
            final HttpServletResponse response )
        throws ServletException, IOException {
        System.out.println( "In MomentUploadSvc.processRequest() ..." );
        Connection cxn = null;
        CallableStatement stmt = null;
        final String senderName = "pounce@cs.gsu.edu";
        final long id = System.nanoTime();
        final long sentTime = System.nanoTime();

        try {
            final ObjectInputStream inputStream =
                    new ObjectInputStream( request.getInputStream() );
            final Moment moment = ((Moment)inputStream.readObject());
            final String recipients = moment.getRecipients();
            final String subject = moment.getSubject();
```
final String body = moment.getBody();
final String videoName =
    moment.getVideoName().substring(
        moment.getVideoName().lastIndexOf("/")+1);

final DataSource dataSource =
    (DataSource)((new InitialContext()).lookup(
        "java:/comp/env/jdbc/postgres" ));

cxn = dataSource.getConnection();
stmt = cxn.prepareCall( "{ call add_moment( " +
        "?, ?, ?, ?, ?, ?, ?) }" );

stmt.setLong( 1, id);
stmt.setLong( 2, sentTime);
stmt.setString(3, senderName);
stmt.setString(4, recipients);
stmt.setString(5, subject);
stmt.setString(6, body);
stmt.setString(7, videoName);

System.out.println("recordsUpdated = "+
    stmt.executeUpdate());
}

try
{
    if( stmt != null )
    {
        stmt.close();
        stmt = null;
    }
}

catch( SQLException e )
{
}

finally
{
    try
    {
        if( cxn != null )
        {
            cxn.close();
            cxn = null;
        }
    }catch( SQLException e )
    {
    }
    try
    {
        if( cxn != null )
        {
            cxn.close();
            cxn = null;
        }
    }catch( SQLException e )
    {
    }
}
protected void doGet(final HttpServletRequest request, final HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

protected void doPost(final HttpServletRequest request, final HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

public String getServletInfo() {
    return "A web service to receive moment meta-data";
}
VideoUploadSvc – This class implements a web service that receives the video portion of a new moment from the end-user's web browser, and saves the video file to disk on the server-side. The new video files are received by multi-part upload method over HTTP.

```java
/* Susan Gentner
* PushToShow Masters Thesis
* Georgia State University
* March 2009
*/
package edu.gsu.cs.ps;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

public class VideoUploadSvc extends HttpServlet {

    protected void processRequest(
            final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");

        final PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Multipart File Upload Servlet Test</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Multipart File Upload Servlet Test Results "+
                "from " + request.getContextPath () + "</h1>");

        try
```
final ServletContext servletContext = getServletContext();
final String msgsDirectory = 
    servletContext.getRealPath("/
    + servletContext.getInitParameter("VideosDir")
    );
out.println("Server-side real path to upload directory is '" + msgsDirectory + "' <br>");
out.println("Acquiring an iterator for the uploaded file(s) " +"<br>");

final DiskFileItemFactory factory = new DiskFileItemFactory();
final ServletFileUpload upload = new ServletFileUpload(factory);
final List fileItems = upload.parseRequest(request);
final Iterator files = fileItems.iterator();

out.println("Now, to receive each file.. <br>");
while ( files.hasNext() )
{
    final FileItem fileItem = ((FileItem)files.next());
    final String fileName = fileItem.getName();
    final InputStream fileInputStream = fileItem.getInputStream();
    final File newFile = new File( msgsDirectory, fileName );
    final FileOutputStream fileOutputStream = new FileOutputStream( newFile );
    final BufferedOutputStream bufferedStream =
        new BufferedOutputStream( fileOutputStream );

    long fileSizeSize = 0;
    int fileByte = fileInputStream.read();
    final long fileStartTime = System.currentTimeMillis();
    while ( fileByte > -1 )
    {
        fileSizeSize++;
        bufferedStream.write( fileByte );
        fileByte = fileInputStream.read();
    }

    final long elapsedMillis =
        System.currentTimeMillis() - fileStartTime;
    final long byteRate =
        fileSizeSize * 1000L / elapsedMillis;

    out.println( "&nbsp; &nbsp; &nbsp; &nbsp; " );
    out.println("Received file '" + fileName + "' in " + elapsedMillis + " mS, " + fileSizeSize + " bytes, " + byteRate/1024 + " KB/sec. <br>");

    bufferedStream.flush();
    bufferedStream.close();
    fileInputStream.close();
}
out.println("Done receiving file(s) <br>");

} catch (FileUploadException e) {
    e.printStackTrace(out);
}

finally {
    out.println("</body>");
    out.println("</html>");
    out.flush();
    out.close();
}

protected void doGet(final HttpServletRequest request,
        final HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

protected void doPost(final HttpServletRequest request,
        final HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

public String getServletInfo() {
    return "A web service to receive the video portion of a moment";
}
WebCam – This class initializes and receives video from the attached webcam on the end-user's PC.

```java
/* Susan Gentner
* PushToShow Masters Thesis
* Georgia State University
* March 2009
*/

package edu.gsu.cs.ps;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.Date;
import java.text.SimpleDateFormat;
import quicktime.*;
import quicktime.qd.*;
import quicktime.io.*;
import quicktime.std.*;
import quicktime.std.sg.*;
import quicktime.std.image.CodecNameList;

public class WebCam implements ActionListener
{
    protected int[] pixelData;
    protected boolean grabbing;
    protected String fileName;
    protected String msgUploadURL;
    protected String fileUploadURL;
    protected BufferedImage image;
    protected QDRect cameraImageSize;
    protected QDGraphics qdGraphics;
    protected SGVideoChannel videoChannel;
    protected SGSoundChannel soundChannel;
    protected SequenceGrabber seqGrabber;
    protected SimpleDateFormat simpleDateFormat;
    protected final CapturePanel capturePanel;
    protected final SequenceGrabber seqGrabber;
    protected final SimpleDateFormat simpleDateFormat;
```
public WebCam() throws Exception
{
    capturePanel = new CapturePanel();
    seqGrabber = new SequenceGrabber();
    capturePanel.addActionListener( this );
    simpleDateFormat = new SimpleDateFormat( "yyyyMMdd_HHmmss" );
    initSequenceGrabber();
    initBufferedImage();
}

private void initSequenceGrabber() throws Exception
{
    soundChannel = new SGSoundChannel( seqGrabber);
    soundChannel.setUsage( StdQTConstants.seqGrabPreview | StdQTConstants.seqGrabRecord );
    videoChannel = new SGVideoChannel( seqGrabber);
    videoChannel.setCompressorType( 1483297896 );
    qdGraphics = new QDGraphics( cameraImageSize );
    seqGrabber.setGWorld( qdGraphics, null );
    videoChannel.setBounds( cameraImageSize );
    videoChannel.setUsage(quicktime.std.StdQTConstants.seqGrabRecord|
    quicktime.std.StdQTConstants.seqGrabPreview |
    quicktime.std.StdQTConstants.seqGrabPlayDuringRecord );
    videoChannel setFrameRate( 0 );
}

private void initBufferedImage() throws Exception
{
    final int intsPerRow = qdGraphics.getPixMap().getPixelData().getRowBytes() / 4;
    final int size = intsPerRow * cameraImageSize.getHeight();
    pixelData = new int[size];
    final DataBuffer dataBuffer = new DataBufferInt(pixelData, size);
    final ColorModel colorModel = new DirectColorModel( 32, 0x00ff0000,
    0x0000ff00, 0x000000ff, 0x000000ff );
    final int[] masks = {0x00ff0000, 0x0000ff00, 0x000000ff};
    final WritableRaster raster = Raster.createPackedRaster( dataBuffer,
    cameraImageSize.getWidth(),
    cameraImageSize.getHeight(),
    intsPerRow, masks, null );
    image = new BufferedImage( colorModel, raster, false, null );
}

public String getFileUploadURL()
{  
    return fileUploadURL;
}

public void setFileUploadURL( String _fileUploadURL )
{
    fileUploadURL = _fileUploadURL;
}

public String getMsgUploadURL()
{
    return msgUploadURL;
}

public void setMsgUploadURL( String _msgUploadURL )
{
    msgUploadURL = _msgUploadURL;
}

public CapturePanel getCapturePanel()
{
    return capturePanel;
}

public boolean isAvailable( final SGDeviceName deviceName )
{
    return((deviceName.getFlags() & StdQTConstants.sgDeviceNameFlagDeviceUnavailable) == 0);
}

public void setPreview() throws QTException
{
    seqGrabber.setDataOutput( null,
        StdQTConstants.seqGrabDontMakeMovie );
}

public int numberAvailableDevices( final SGDeviceList sgDeviceList )
    throws QTException
{
    int numDevsAvailable = 0;
    for( int i=0; i<sgDeviceList.getCount(); i++ )
    {
        if( isAvailable( sgDeviceList.getDeviceName( i ) ) )
            numDevsAvailable++;
    }
    return numDevsAvailable;
}

public void setRecord() throws QTException
{
    fileName = System.getProperty( "user.home" ) + "/psv" + simpleDateFormat.format( new Date() ) + ".mov";
    System.out.println( "Movie file is called '" + fileName + '"" );
    final java.io.File javaFile = new java.io.File( fileName );
javaFile.deleteOnExit();
QTFile movieFile = new QTFile( javaFile );
seqGrabber.setDataOutput( movieFile,
    StdQTConstants.seqGrabToDisk );
}

public void startCapture() throws StdQTException
{
    seqGrabber.prepare( false, true );
    seqGrabber.startRecord();
    grabbing = true;
    Runnable idleCamera = new Runnable()
    {
        public void run()
        {
            try
            {
                while( grabbing )
                {
                    Thread.sleep( 25 );
                    synchronized( seqGrabber )
                    {
                        seqGrabber.idle();
                        seqGrabber.update( null );
                    }
                }
            }
            catch( Exception ex )
            {
                ex.printStackTrace();
            }
        }
    };
    (new Thread( idleCamera )).start();
}

public void stopGrabbing()
{
    try
    {
        grabbing = false;
        if( seqGrabber != null )
            seqGrabber.stop();
    }
    catch( StdQTException e )
    {
        e.printStackTrace();
    }
}

public Component getMirror()
{
    final Component component = new Component()
public void paint( Graphics g )
{
    super.paint( g );
    g.drawImage( image, 0, 0, this );
}

final Runnable imageUpdate = new Runnable()
{
    public void run()
    {
        try
        {
            while( true )
            {
                synchronized( seqGrabber )
                {
                    qdGraphics.getPixMap().getPixelData().
                    copyToArray( 0, pixelData, 0, pixelData.length );
                    component.repaint();
                }
                Thread.sleep( 10 );
            }
        }
        catch( Exception ex )
        {
            ex.printStackTrace();
        }
    }
    (new Thread( imageUpdate )).start();
    return component;
}

public void popDeviceSelections()
{
    try
    {
        int i = 0;
        int numAvailableDevices = 0;
        String defaultSelection = null;
        final SGDeviceList audioDevList =
        soundChannel.getDeviceList( 0 );
        final int numAudioDeviceNames = audioDevList.getCount();
        final int numAudioDevicesAvail =
        numberAvailableDevices( audioDevList );
        String[] audioDevNames = new String[numAudioDevicesAvail];
        for( i = 0; i < numAudioDeviceNames; i++ )
        {
            final String devicename =
            audioDevList.getDeviceName( i ).getName();
        }
if( isAvailable( audioDevList.getDeviceName( i ) ) )
{
    if( deviceName != null )
    {
        System.out.print( "audio #" + i + " is '" +
                        deviceName + "'");

        if( deviceName.contains( "Camera" ) )
        {
            defaultSelection = deviceName;
            System.out.println( " <-- SELECTED " );
        }
        else
        {
            System.out.println( "" );
        }
        audioDevNames[numAvailableDevices++] =
            audioDevList.getDeviceName( i ).getName();
    }
}
}
capturePanel.getCboAudioInput().setModel(
    new DefaultComboBoxModel(audioDevNames ));

if( defaultSelection != null )
{
    capturePanel.getCboAudioInput().setSelectedItem(    
        defaultSelection );

    System.out.println( "Set soundChannel device to: '" +
                        defaultSelection + "'");
}

numAvailableDevices = 0;
defaultSelection = null;
SGDeviceList videoDevList = videoChannel.getDeviceList( 0 );

final int numVideoDeviceNames = videoDevList.getCount();
final int numVideoDevicesAvail =
    numberAvailableDevices(videoDevList);

String[] videoDevNames = new String[numVideoDevicesAvail];
for( i = 0; i < numVideoDeviceNames; i++ )
{
    final String deviceName =
        videoDevList.getDeviceName( i ).getName();
    if( isAvailable( videoDevList.getDeviceName( i ) ) )
    {
        if( deviceName != null ) // usually print this
        {
            System.out.print( "video #" + i + " is '" +
                                deviceName + "'");
            if( deviceName.contains( "USB" ) )
            {
                defaultSelection = deviceName;
            }
        }
    }
}
null
System.out.println("Set videoChannel device to: "+
defaultSelection + \\
"");
}
}

if( defaultSelection != null )
{
capturePanel.getCboVideoInput().
setSelectedItem( defaultSelection );
System.out.println("Set videoChannel device to: "+
defaultSelection + \\
"" );
}


344 try
345 {
346 CodeNameList codecNameList = new CodeNameList( 0 );
String[] codecNames = new String[codecNameList.getCount()];
String defaultSelection = null;

349 for( int i = 0; i < codecNameList.getCount(); i++ )
{
    final String codecName =
    codecNameList.getNth( i ).getTypeName();
    if( codecName.contains("Xiph") )
    {
        defaultSelection = codecName;
        System.out.println("codec algorithm "+ i + " is "
            + defaultSelection +
            " <-- SELECTED ");
    }
    else
    {
        System.out.println("codec algorithm "+ i + " is "
            + codecName + \\
            ");
        codecNames[i] = codecName;
    }
capturePanel.getCboVideoCompression().setModel(

new DefaultComboBoxModel(codecNames));

if( defaultSelection != null )
{
    capturePanel.getCboVideoCompression().
        setSelectedItem( defaultSelection );
}

}  

}

catch( QTException qte )
{
    qte.printStackTrace();
}

}

public void actionPerformed( ActionEvent e )
{
    if( e.getSource() == capturePanel.getRecordButton() )
    {
        stopGrabbing();
        try
        {
            if( capturePanel.getRecordButton().getText().
                equals("Record") )
            {
                grabbing = true;
                capturePanel.getRecordButton().setText( "Stop" );
                capturePanel.getCboAudioInput().setEnabled( false );
                capturePanel.getCboVideoInput().setEnabled( false );
                capturePanel.getCboVideoCompression().
                    setEnabled( false );
                capturePanel.getSendButton().setEnabled( false );
                setRecord();
            }

            else if( capturePanel.getRecordButton().getText().
                equals("Stop") )
            {
                grabbing = false;
                capturePanel.getRecordButton().setText( "Record" );
                capturePanel.getCboAudioInput().setEnabled( true );
                capturePanel.getCboVideoInput().setEnabled( true );
                capturePanel.getCboVideoCompression().
                    setEnabled( true );
                capturePanel.getSendButton().setEnabled( true );
                setPreview();
            }

            startCapture();
        }
        catch( QTException e1 )
        {
            e1.printStackTrace();
        }

    } else if( e.getSource() == capturePanel.getCboAudioInput() )
    {

System.out.println( "cboAudioInput changed to : " +
capturePanel.getCboAudioInput().
.getSelectedItem().toString() + "" );

try
{
  if( seqGrabber.isPreviewMode() || seqGrabber.isRecordMode() )
    stopGrabbing();

  soundChannel.setDevice( capturePanel.getCboAudioInput().
      getSelectedItem().toString() );
  startCapture();
}
catch( Exception ex )
{
  System.out.println( "Could not set the sound device" );
  ex.printStackTrace();
}

else if( e.getSource() == capturePanel.getCboVideoInput() )
{
  System.out.println( "cboVideoInput changed to : " +
      capturePanel.getCboVideoInput().
      getSelectedItem().toString() + "" );

  try
  {
    if( seqGrabber.isPreviewMode() || seqGrabber.isRecordMode() )
      stopGrabbing();

    videoChannel.setDevice( capturePanel.getCboVideoInput().
        getSelectedItem().toString() );
    startCapture();
  }
  catch( Exception ex )
  {
    System.out.println( "Could not set the video device" );
    ex.printStackTrace();
  }
}

else if( e.getSource() == capturePanel.getCboVideoCompression() )
{
  System.out.println( "cboVideoCompression changed to : " +
      capturePanel.getCboVideoCompression().
      getSelectedItem().toString() + "" );

  try
  {
    final CodecNameList codecNameList = new CodecNameList(0);
    final int ctype =
        codecNameList.getNth( capturePanel.
            getCboVideoCompression().
            getSelectedIndex() ).
            get CType();
  

if(seqGrabber.isPreviewMode() || seqGrabber.isRecordMode())
    stopGrabbing();

    videoChannel.setCompressorType( ctype );
    startCapture();

    }

    catch( Exception ex )
    {
        ex.printStackTrace();
    }

    }

else if( e.getSource() == capturePanel.getSendButton() )
{
    System.out.println( "Send button was clicked." );

    final Moment moment = new Moment();
    moment.setSenderName( "pounce@cs.gsu.edu" );
    moment.setRecipients( capturePanel.getTfRecipient().getText() );
    moment.setSubject( capturePanel.getTfSubject().getText() );
    moment.setBody( capturePanel.getTaBodyText().getText() );
    moment.setVideoName( fileName );
    momentUploader = new MomentUploader( moment, msgUploadURL, fileName, fileUploadURL );

    (new Thread( momentUploader )).start();
    System.out.println( "Started thread to upload message with "+
                        "subject line " +
                        moment.getSubject() + ":" );

}

else if( grabbing )
{
    try
    {
        seqGrabber.idle();
        seqGrabber.update( null );
    }

    catch( QTException qte )
    {
        qte.printStackTrace();
    }
}

}
ListMoments.jsp – This web page lists all moments that are addressed to a specified recipient. The moments are presented in an HTML table that has click-able rows. When the user clicks a row, a new tab will be opened in the web browser with a matching ViewMoment page for the moment entry that was clicked.

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 TRANSITIONAL//EN">
<!--
Susan Gentner
PushToShow Masters Thesis
Georgia State University
March 2009
-->%
<%@ page import="java.sql.*" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="javax.sql.DataSource" %>
<%@ page import="javax.naming.InitialContext" %>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>PushToShow Inbox</title>
<link rel='stylesheet' type='text/css' href='sample.css' />
</head>
<body>
<form>
<center>
<table width="1005px" height="550px" border="6px" cellpadding="1px">
<tr valign="top"><td>
<table border="1px" width="100%">
<thead>
<tr>
<th width="20%">Date</th>
<th width="20%">Sender</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<th width="20%">Date</th>
<th width="20%">Sender</th>
<th>Subject</th>
</tr>
</tbody>
</table>
</td>
</tr>
</table>
</center>
</form>
</body>
</html>
```
try {
    final ServletContext servletContext = this.getServletContext();
    final int serverPort = request.getServerPort();
    final String serverName = request.getServerName();
    final String path = request.getServletPath();
    final String takeMomentURL = "http://" + serverName + ":" + serverPort + servletContext.getContextPath() + "/TakeMoment.jsp?id=";
    final String shareUrl = "http://" + serverName + ":" + serverPort + servletContext.getContextPath() + "/ShareMoment.jsp";
    final String recipient = request.getParameter("recipient");
    DataSource dataSource = (DataSource)((new InitialContext()).lookup("java:/comp/env/jdbc/postgres"));
    try {
        cxn = dataSource.getConnection();
        pst = cxn.prepareStatement("SELECT * FROM list_moments('" + recipient + ")" );
        rs = pst.executeQuery();
        while(rs != null && rs.next()) {
            out.println("<tr onmouseover='this.style.color="red";" + "style.cursor="pointer";this.style.background="lightblue"' onMouseOut ="this.style.color="black";this.style.background="white"" + "onclick="window.open('" + takeMomentURL + rs.getString("id") +"');return false;">");
            out.println("<td" + simpleDateFormat.format(new Date(Long.parseLong(rs.getString("senttime"))/1000000)) + ";</td>";
            out.println("<td" + rs.getString("sendername") + ";</td>";
            out.println("<td" + rs.getString("subject") + ";</td>";
            out.println("</tr>";
        }
    } catch(SQLException e) {
        e.printStackTrace();
    }
    finally {
        try {
            if(rs != null )
                rs.close();
        } catch(SQLException e) {
        }
    }
}
try
if ( pstmt != null )
    pstmt.close();
}
catch ( SQLException e )
{
}

try
{
    if ( cxn != null )
        cxn.close();
}
catch ( SQLException e )
{
}

</tbody>
</table>
</table>
</p>
<%= new java.util.Date() %>
<% if ( shareUrl != null ) { %>
<input type = "button" name="compose" value="Compose New Moment"
onclick='window.open("<%=shareUrl%>");'>
</p>
</center>
</form>
</body>
</html>
ShareMoment.jsp – This web page allows the end user to capture, compose, and send a moment with just a couple of quick mouse clicks.

```html
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<!--
Susan Gentner
PushToShow Masters Thesis
Georgia State University
March 2009
-->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Push to Show Applet</title>
</head>
<body>
<center>
<table width="1005px" border="6px" cellpadding="1px">
<tr><td>
<table>
<tr>
<td>
```java
String userAgent = request.getHeader("user-agent");
boolean isMac = userAgent.toLowerCase().
contains("mac os x");
if(isMac)
{
}%
```<applet archive="psa.jar"
code="edu.gsu.cs.ps.Applet"
width="1000"
height="550">
</applet>

This application needs a Java-enabled browser to run.
</applet>
```java
```
else
{
   %>
   Sorry this application is currently <b>Mac</b> only.
   %>
   </td>
   </tr>
</table>
</td></tr></table>
<p>
<%=
new java.util.Date()
%>
</p>
</center>
</body>
</html>
TakeMoment.jsp – This web page displays a moment for a recipient.

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 TRANSITIONAL//EN">
<!--
Susan Gentner
PushToShow Masters Thesis
Georgia State University
March 2009
-->%

<%@ page import="java.sql.*" %>
<%@ page import="javax.sql.DataSource" %>
<%@ page import="javax.naming.InitialContext" %>
<%@ page import="edu.gsu.cs.ps.Moment" %>

Connection cxn = null;
PreparedStatement pstmt = null;
Statement stmt = null;
Moment moment = null;
ResultSet rs = null;
String senderName = null;
String recipients = null;
String subject = null;
String body = null;
String videoName = null;

try {
    final ServletContext servletContext = this.getServletContext();
    final long id = Long.valueOf(request.getParameter("id"));

    DataSource dataSource = (DataSource)(new InitialContext()).lookup("java:/comp/env/jdbc/postgres");
    cxn = dataSource.getConnection();
    pstmt = cxn.prepareStatement("SELECT * FROM get_moment(" +
    id + ");");
    rs = pstmt.executeQuery();

    if ( rs.next() )
        {
            moment = new Moment( rs );
            senderName = moment.getSenderName();
            recipients = moment.getRecipients();
            subject = moment.getSubject();
            body = moment.getBody();
            videoName = moment.getVideoName();

            if ( body.contains("\n") )
                body = body.replaceAll("\n", "<br/>");
        }
```
catch ( SQLException e )
{
    e.printStackTrace();
}

finally
{
    try
    {
        if ( rs != null )
            rs.close();
    }
    catch ( SQLException e )
    {
    }

    try
    {
        if ( pstmt != null )
            pstmt.close();
    }
    catch ( SQLException e )
    {
    }

    try
    {
        if ( cxn != null )
            cxn.close();
    }
    catch ( SQLException e )
    {
    }
}

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<title>%% moment.getSubject() %></title>
<link rel='stylesheet' type='text/css' href='sample.css' />
</head>
<body>
<form>
<center>
<table width="1005px" border="6px" cellpadding="1px">
<tr><td>
<table border="1px" width="100%">
<tr>
<td rowspan="5" width="640">
<div class="classname">
<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
    width="640" height="490"
    codebase="http://www.apple.com/qtactivex/qtplugin.cab">

<param name="autoplay" VALUE="false"/>
<param name="controller" VALUE="true"/>
<embed src="<%= "http://" + request.getServerName() + ":" + request.getServerPort() + "/psvmsgs/" + 
videoName%>"
width="640" height="500"
controller="true" fullscreen="full">
</embed>
</object>

</td>
</tr>
<tr height="25px">
<td align="right"><b>Subject:</b></td>
<td><%= subject %></td>
</tr>
<tr height="25px">
<td align="right"><b>From:</b></td>
<td><%= senderName %></td>
</tr>
<tr height="25px">
<td align="right"><b>To:</b></td>
<td><%= recipients %></td>
</tr>
<tr>
<td colspan="2" valign="top">
<%= body %>
</td>
</tr>
</table>
<p><%= new java.util.Date() %></p>
</center>
</form>
</body>
</html>
SQL to create the moments table and indexes

-- Table: moments

CREATE TABLE moments

(
    id bigint NOT NULL,
    senttime bigint NOT NULL,
    sendername character varying NOT NULL,
    recipients character varying NOT NULL,
    subject character varying NOT NULL,
    bodytext character varying NOT NULL,
    videoname character varying NOT NULL,
    CONSTRAINT messages_pkey PRIMARY KEY (id)
)

WITH (OIDS=FALSE);

ALTER TABLE moments OWNER TO postgres;

-- Index: recipients_idx

CREATE INDEX recipients_idx

ON moments

USING btree

(recipients);

-- Index: senttime_idx

CREATE INDEX senttime_idx

ON moments

USING btree

(senttime);
web.xml is a deployment descriptor for the pss web application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <servlet>
    <servlet-name>MomentUploadSvc</servlet-name>
    <servlet-class>edu.gsu.cs.ps.MomentUploadSvc</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MomentUploadSvc</servlet-name>
    <url-pattern>/MomentUploadSvc</url-pattern>
  </servlet-mapping>
  <resource-ref>
    <description>PostgreSQL Datasource example</description>
    <res-ref-name>jdbc/postgres</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
  <welcome-file-list>
    <welcome-file>ListMoments.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```
web.xml is a deployment descriptor for the psv web application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <context-param>
    <description>Video upload directory</description>
    <param-name>VideosDir</param-name>
    <param-value>msgs</param-value>
  </context-param>
  <servlet>
    <servlet-name>VideoUploadSvc</servlet-name>
    <servlet-class>edu.gsu.cs.ps.VideoUploadSvc</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>VideoUploadSvc</servlet-name>
    <url-pattern>/VideoUploadSvc</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```