Client-Server Web Cam Image Chat

Coms 319 Portfolio 1

By Mike Croskey

Table of Contents

Project Overview:

The goal of this project was to extend my original implementation of a server-client chat service into a chat service capable of sending images captured from a web cam. I initially hoped to include the capability to live stream video between the clients, except, due to time limitations and Java limitations I was unable to achieve video/audio feed.

The project consist of two basic parts a server and a client that communicate with each other to provide the chat service. Each client has the capability of sending messages to a public chat log that is visible in the active window. The user is able to see a live feed from their camera in addition to having the ability to capture and send an image to a public image log. The public image log is also viewable on the client user interface. The most challenging aspect of the project was figuring out which java API was capable of capturing images from a web camera.

This project required an extensive amount of research into different java APIs that are capable of capturing web camera data. I was forced to tests a handful of API options repeatedly until I was able to discover a platform that allowed a 64bit development environment. This was very challenging and completely new to my understanding. I had never attempted to get a web camera to work from within a java application.

After I was able to get a functioning capture API working, I needed to develop a way of determining which kind of information was being transferred between the client and the server. Basically I needed a way to make sure chat messages were received as chat message data and images properly as image data. This provided a significant challenge and lead to the need for specialized formatting of data before passing thru a socketed data stream. The complexity of these requirements forced me to spend large amounts of time learning and testing the project in order to produce a chat service that was functional.

I had to create the interface design for displaying the interaction data in a manner that was useful to the user. I examined countless documentation pages looking at different options for accomplishing what I needed. I had to create all of my own testing platforms used for collecting data on image dimensions versus actual data size. This data then allowed me to create my design once I had examined the details of my tests.

I had to examine different options for rendering images from a web cam and determine which appeared best suited for my application. The resulting tests I performed allowed me to determine which resolution would make it possible to send and image over a connection socket without causing transfer issues. I analyzed the data result and determined I needed an additional level of formatting which allows handling the image data correctly.

Many of the challenges I faced lead to the discovery of new and fascinating information because I would start looking for one solution and unintentionally find another possibility. I had to judge which solution would provide the best results and what result could be produced in the allotted period of time. Most of the project design required a mixture of creating/analyzing/evaluation. Each successful step would lead to a new round of creating an idea followed by researching the idea for potential options and evaluation of the resulting accomplishment. Thus, I was able to develop a working image chat service by applying these methods.

Class Related and New Material:

In order to bring this project to realization I had to figure out a number of rather difficult and extremely frustrating steps. Each step progressed my eventual design and created the individual components required for the final design.

I began working on the individual components for this project during the first homework assignments. I wanted to take small aspects from each and create a project that makes use of these different components. The initial stem of inspiration came from designing the simple UI components during the first few home works.

I gained new understanding of swing, reinforced past experiences with server-client design and discovered the usefulness of threads inside of swing applications. Using this newly acquired knowledge I began to investigate the possibility of expanding my Server-Client design from a previous homework assignment.

I finally decided to create a chat client using the Server-Client Design and is capable of allowing the client to send a picture captured from a web cam. This captured image is set to the server from the client then relayed from the server to each of the clients using a socket connection. The relayed image is then displayed in a public image log visible on the client UI.

I expanded the simplistic Server-Client code even further from the extra credit additions I had made to the HW assignment. I added a live camera feed to allow the client to view active feed from installed camera. This active feed was accompanied by a send image button that when clicked uses a button action listener to capture the current image and invoke the send and receive methods for the client. Upon click a small image is made visible in the frame allowing the client to view the image sent to public image log. The client UI has the ability to both send and receive txt or images to any client connected to the current server.

The server and the client both utilize socket connections to transfer information between the multiple program entities. This requires dual purpose for each of the send/receive socket streams. In order to have this dual functionality I designed a simple flag based system shifting the parsing routines within the looping structure used for connections. The client and the server broadcast a preceding flag value into the socket stream. The socket stream is then written with the appropriate data to allow the other member to properly read the correct amount of data.

In addition to learning new ways of using java swing I was able to utilize multithreading in my project design. I use threads to allow multiple clients to connect to the server without bogging down the server. Threads allow the server to address each client individually and continue to accept new incoming connections. The thread acts like an individual process that is able to terminate without closing the main server when a client disconnects.

The Client UI has three main components that handle user interactions; Login, Chat, Cam. The login is a simple Frame with a JPanel containing an input area from entering client user name. The Login has a button with an action click listener that calls the main client frame and closes the login. The Chat frame utilizes a custom Jlist, listmodel and cell list render for both public chat log and the img log. The list uses a model that stores all the objects which contain the data relayed from the server. The Cam module is an extended jPanel class that allows me to create and drop the Camera component into a panel or frame at my will.

Project Complexity:

It was very challenging finding an API that would run on a 64 bit system and compile under a 64 bit compiler. Many of the available API options are left over from years gone by and do not support the use of a 64 bit JRE. The project I created was designed using JMF (JAVA MEDIA FRAMEWORK). This requires the installation of a 32bit JRE as well as configuring eclipse for project specific application design. Without specification the system will assume use of a default environment and the webcam will not open for capture under the JMF API functions.

Finding a working framework was not the only issue I encountered. Unfortunately, I was also forced to buy a new webcam after fighting with the imbedded camera on the laptop screen. The drivers used to communicate with many of these types of “onboard” cameras do not play nicely with the basic capture drivers used by JMF. Troubleshooting the hardware

compatibility issues added a significate amount of frustration to this project. Thus, dealing with legacy software and stubborn hardware made the start of this project extremely difficult.

After managing to get the Camera functioning, I faced the next challenge of learning how to capture information from the camera. I had to design a class that implemented the camera and placed it in its own frame with a specific size to match the captured image. I spent a significant amount of time reading blogs and docs in order to figure out how exactly an image must be rendered to a JPanel. I created a specific class to handle the camera frame, capture event button and sending captured camera data to server.

Integrating the functionality of a camera and a public chat room turned out to be one of the most challenging design features on this project. The captured image had to be converted into a byte[] array which contained a converted format of the original image. I was able to get to dual function by first initializing a data type flag. The flag is fallowed with either a message for sending chat or a sized buffer for sending a byte array containing an image.

This lead into the next major obstacle I had to tackle. I mentioned above a size buffer follows transmission of a flag indicating msg or img data. This buffer is used by the next call to read a set number of bytes from the designated input stream. I had to design extensive error checking routines in order to guarantee the server and the client do not get out of sync with each other. If the image is incomplete or has a failure during the transfer then the program needed to be able to correct itself and must continue to accept incoming images and chat after the failure occurs.

This project required a simple solution to a number of rather complex problems and as I mention before caused me to produce a strategy for handling these issues. In turn I was able to find a fix allowing me to run the 32bit JMF on a 64bit operating system. I established a functional development environment within eclipse by altering project environmental settings. I pirated through countless online tutorials and code examples because I needed to learn how to detect and capture from an attached web camera. I cracked the mystical pattern of capturing and converting an image into a useable data format. I developed a system for handling the transfer of multiple types of data patterns for text and images across a client-server platform. Lastly I mastered a technique for sending and displaying the captured webcam image and created a UI for interaction.

Blooms Taxonomy:

-Synthesis and Analysis

This project offered a large number of challenging aspects because of the need for multiple user interaction. After some extensive looking I decided upon my initial project design. I spent a few days researching different ways of using java swing to implement some sort of video chat. I manage to find a solution that was basically a plug and play implementation. This did not leave me with much to work with for development purposes so I abandoned the idea for live video streaming. I moved on with research to investigate the possibility of using java swing to capture and manipulate images from a camera.

Many of the available options have fallen into legacy and no longer receive updates or support. This was one of the greater project challenges because it required extraneous trial and error. I spent significant time reading about different open source and distributed packages that offered capture support in JAVA. After learning more about the compatibilities of available API options I was forced to decide between a small listing of options.

This was very interesting to say the least, mainly due to issues with platform specific performance. I began research on a handful of potential frameworks by use of google and the resulting search returns. My first attempt at using java to operate and capture was with Sarox webcam-capture API. This is a java platform API that is capable of communicating with a multitude of camera types and formats. I was unfortunately unable to get this API to compile due to incompatibilities with 64 bit development environment.

I investigated a few more open source/distributed java API possibilities such as JMF, FMJ, VLCJ, Xuggler and finally JavaFX. Interestingly JMF has been out of circulation for the longest of all my options and it turned out to be the only viable solution for creating the type of application I wanted.

As mentioned JMF has fallen to the wayside of things with the advancements in coding platforms and hardware. FMJ has also become old school in the view of bleeding edge programmers because of 64 bit platform limitations and due to inability to prevent security issues within created applications.

The platform I had originally investigated for creating a video chat application was VLCj because it allows for the creation of formatted audio/video streams. I was able to get this functioning and was able to set up a rough client to client type video stream. Unfortunately, this type of environment is basically self-contained. VLC must be installed on both systems and

it relies heavily on behind the scenes actions within the VLC player platform. In order to make the application function I would have basically embed a VLC player into a frame and have added some simple UI options for interaction. I felt this was far too simplistic and did not require the creation of much of anything. With this in mind I abandoned the VLCj platform.

I then quickly looked into Xuggle and spent a significant amount of time attempting to understand how to even use the platform. The extreme difficulty of setup and lack of relevant tutorials or documentation caused me to decide against this platform. Finally, I looked into javaFX for creating the project application and decided it was also not the correct fit for the type of project I want to create using java Swing

Trying to figure out how to simply get java to access a camera either via USB or onboard was an extremely frustrating step in the design process. Java does not talk very well with low level hardware using simple API's. After trying each of the available options I was finally able to get an API to detect and capture from an external web camera purchased from Wall-Mart. This offered a rather frustrating challenge of which I managed to concur by making some specific changes to my system configuration. I spent a few hours trying to get each of the available options working before I realized that I needed to be running a 32 bit JRE.

I read through forum after forum and I just happened to notice a trend of people who were attempting to use these API options. I finally realized they were all using 32 bit systems as development platforms. Once I corrected this simple mistake I restated my search and began testing each platform again. Trial and error finally lead to the discovery of one forum explaining how to configure a 64 bit environment to run JMF. This allowed me to move on to creation of the user interface.

-Analysis and Evaluation

I deiced early on that I was going to abandon the idea of streaming video thru a java swing application because of the lack of implementable APIs. I discovered that many of these options just did not have what I needed in order to produce what I wanted.

After analyzing all of the possibilities at my disposal I fell back on my original designs from a basic chat server with multiple clients. I had used threads to produce this type of client-server behavior before and I felt this project was an opportunity to gain more experience. I recognized that I could utilize multithreading to enable a single server to connect and send images between clients. I tested this theory by developing a server with a forwarding system that works with swing user interactions. Initially I was unsure the setup would work because of the

large amount of data each picture contained. I had to perform sample test with the camera at different resolutions in order to discover an average size for expected images.

Prior to this lab I had not spent much time working with cameras using java. I knew that java would be able to handle the parsing of data but I needed to evaluate its capability of managing image data. I began by writing a simple looping structure that captured images from the web cam. Along with capturing an image the algorithm calculated the byte size of different resolutions. I examined the results from the algorithm and determined that larger size resolution may produce issues when transferring without creating a buffering system. I debated for a small time on whether the implementation of a data buffering system would be necessary. I decided the amount of time it would take and the benefit made it a non-viable option for this project design. I needed to figure out a way of taken the large image files and packing them into the transfer stream.

I analyzed documentation for image type objects and learned they are a base class for many different image classes. This allows the image class the easily be ported into different formats and or converted into byte arrays. I had to test a few different methods that were recommended for handling Image objects. I found out the Image can be wrapped as a byte array and this allows the image to be transferred over a set data size. I had to evaluate the resulting array sizes after wrapping different resolution to determine how large I needed to make images I would then transfer between the server and clients.

Client byte size :: 8731 Received 240x320:

Sample 1:

I evaluated the resulting array sizes an example data output can be seen in Sample. Using this data I determined Java could send an image that was 240x320 without using a large array. This could reduce the chances of delay errors with large files over sockets. I implemented this idea and ran a set of test to determine in fact my prediction was correct and I could quickly send images between the server and the client. Throughout this process I learned a great deal about how the capture device worked and how much information was actually necessary to produce low quality images. I originally assumed the quality would be rather lacking but after running the program test I was pleased to discover a rather decent quality image could be produced when relay between the server as a byte array. This follows a general pattern many systems use by taking the raw data and manipulating into a format best suited for the specific application.

I faced another decision that had to deal with user interaction and the resulting interface between the users. I wanted to create something that was functional yet still simple enough to

create in the limited time given. I tried a number of different methods for many different designs and decided to go with design features of which I was somewhat familiar. I decided to investigate the JList component to figure out how much it was capable of doing. I analyzed the documentation for Jlist, custom cell render and models because I wanted to see if I could display images inside of the jlist cells. I learned that Jlist has the ability to hold jpanels and in addition each panel can have an icon and name set as a standard parameter. I immediately tested the function of this customized jlist and was very pleased with the resulting scrollable image log I was able to produce from this discovery.

I spent a significant amount of time developing/ testing the initial server to client and client to server interaction. I constructed the basic design after learning as much as I could about possible API options. I was able to learn enough about the JMF API in order to create a swing application for sending the images between the clients. I feel this allowed me to apply my skills as a programmer while discovering a new API and learning a lot about how web camera capture works from within a java development environment.