

Image Video-On-Demand System

Antonio Roberto Llanos, MS
Florida Atlantic University
antoniorllanos@yahoo.com

Abstract

In this paper, methods for encoding and delivering a stream of continuous images as a video and text associated with the video are researched. The techniques are combined with a framework for developing a "video on demand" system to query the images and deliver them to the requestor. The paper also investigates using the Java Media Framework (JMF) as a platform for developing the solution. Issues related to the implementation to the system are also discussed.

1. Introduction

1.1. Overview

Many businesses maintain collections of images that are used for survey, research and presentation. Often, these organizations require a method to view these images, which may be in a sequential order or time-lapse order. In addition, information on the image may be required. This information may provide information describing the time, location or other information about the image.

Simple player models such as a flipbook model provide the user with a simulated motion video. While this implementation works, it often requires the user to be connected to a corporate network and is not an efficient method to deliver images.

1.2. Video Technology

Currently, there are many standard video formats that exist that can be viewed by a computer. Formats such as MPEG and Apple's Quicktime can be viewed by anyone on most major PC operating systems. Software such as Windows Media Player and Apple's Quicktime Player provide low cost solutions to viewing digital video. The video consists of a series of

frames played at a specified frame rate. In cases such as MPEG, compression algorithms are used to reduce the size of the video data.

1.2. Java Media Framework

The Java Media Framework (JMF) is a multimedia framework written by Sun Microsystems. The latest implementation is version 2.1.1e. It is a collection of classes and interfaces that allow for software developers to write time-based media applications. The JMF provides an object model and design that allows users to create media players, encoders, decoders, transcoders and write media to disk without having to develop a considerable amount of code. It also provides an implementation of the Real Time Protocol (RTP) to deliver the media to a client.

The JMF object model consists of several interfaces that encapsulate the basic requirements of a video system. A centralized controller manages these objects, and is responsible for creating and controlling the life cycle of these objects.

At the center of the JMF is the Manager class. It is responsible for the creation of many of the multimedia objects. The Manager class is responsible for the creation of an abstraction of a media source, called a DataSource object. The DataSource object represents a stream of media, which was obtained in some fashion, either by file, URL or RTP stream. The Manager utilizes a DataSource or creates one based on parameters passed to it to determine a media player for the source. The JMF contains a registry of media MIME-TYPES, file extensions and associated Java packages that it uses to determine which player should be instantiated. The Manager class uses this registry by examining the URL of the DataSource, and cross-referencing the extension of the URL with the registry. The JMF requires that developers name their players in a specific fashion, such that the corresponding MIME-TYPE of URL's file extension matches the Java

Package name. Furthermore, the class to play the media must be called Handler. In the case of MPEG Video, the developer must create the player class in a package called: (some company name).video.mpeg.Handler for the MIME-TYPE video/mpeg. Custom MIME-TYPES and extensions can be added through the JMF Registry application or at runtime.

In the case that the media is not to be played, but utilized in some form of coding, a Processor object can be created. Processors work like players, however, they do not display or play the media. If the media is to be displayed, a Player object is instantiated. Players contain graphical components that are displayed on the screen. They also may contain control components as well to assist in the starting and stopping of media. Finally, if data is to be written to storage, a DataSink object is created to take the media and write to disk in the format that is specified by the DataSource passed to it.

JMF processing works by using an asynchronous model that posts events to listeners as steps in the process are being completed. Each object in the JMF provides the ability to send events to objects that may be controlling the execution of the framework. Developers simply implement a ControllerListener interface and code to react to specific events that are posted to it.

The JMF supports many of the most popular formats such as MPG, Quicktime, AVI, and Windows WAV audio. The architecture of the framework allows for the implementation of new players, encoders and decoders. IBM has extended the framework to include an MP4 decoder and player [4]. Although the framework has existed for several years, little support exists and many of the most recent advances in Video communication are not supported.

2. Developing a Video-On-Demand System

2.1 Statement of Problem

The goal of the research performed was to investigate methods of encoding a series of graphic images, include information relevant to each frame of video, and display this information in a synchronized fashion to the user. Furthermore, the foundation for this system was to be the Java Media Framework and supporting Java code.

A new framework to allow for the querying, retrieval, encoding, delivery and display of this media was also to be investigated.

A model implementation was developed for each method discovered. This model can serve as a reference for other developers wishing to use the framework for their own application.

2.2 Existing Extensions of the JMF

Since the JMF is a complex API, research was performed to uncover existing implementations that would aid in the development of the system. One such aid was the Abstract Player Framework by Rob Gordon [1]. The Abstract Player Framework (APF) provides an additional layer of abstraction to the JMF and simplifies the development of a custom media player. Gordon also developed a synchronized player called Xsync [1] that allows for the synchronization of disparate media types to be displayed simultaneously.

Sun Microsystems also provided several examples that assisted in the development of the system. Examples such as “Generating a Movie File from a List of JPEG Images” [3] assisted in the understanding of how custom DataSource objects could be defined and encoded to a specified video format.

2.3 Approach to Solution

The approach to developing an encoding solution was devised by first developing an object model to support the management of image and corresponding data. This data must be stored and have the ability to be queried to allow the user to select a subset of images for encoding. Once the selected images (or references to the images) were retrieved, the images are to be processed and displayed with a corresponding player. Figure 1 shows an overall design architecture on how the system should work. A client makes a request to an application server, which requests images from a datasource. The application server then encodes the images and metadata and returns the media to the client.

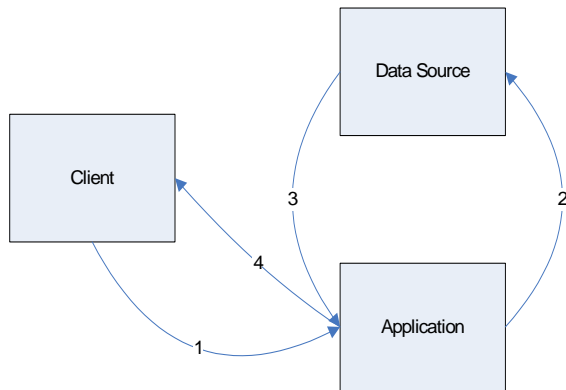


Figure 1: System Architecture

The first major problem to be solved was how 2 disparate media types, a graphical image and plain text were going to be displayed simultaneously. The second problem was how to utilize the JMF to perform these tasks as well as meet the restrictions and limitations of the JMF. The third was if no such solution could be found, what method could be implemented to meet the requirements.

3. Three methods for the implementation of an encoding scheme

3.1 Custom VOD File

The first solution devised was the creation of a custom video format, called a VOD. The VOD is a custom encoding scheme that includes both the image and text data combined. The VOD file format is based on the Rob Gordon's MIV file format [1].

To implement the VOD frame format, a customer encoder and player were developed. The player adheres to the Abstract Player Framework [1] and decodes the image and text data one frame at a time. The player contains a custom GUI component to display the image and text data. The VOD file contains other relevant information, such as image size, duration and frame rate. The frame rate is used to calculate a "sleep" time between the displays of each frame.

Although the VOD file format is not standard, it allows for the most flexibility. The file format may be extended without restrictions of standards or players that cannot be modified. One fallback of the VOD file format is that it cannot be used with standard players such as Windows Media Player.

3.2 Quicktime and Custom Text Player

A second solution was to modify Sun Microsystems example code [3] to encode the images as a Quicktime file. The issue that arose is how to encode the text data associated with the image. A custom text player was developed, utilizing the Abstract Player Framework [1]. The idea behind the text player was to develop a file format that presented the frame metadata in such a way that it could be read and displayed by a player. Since the file was a plain text file, an encoder was created to write the information in such a way that a player could decode the file and display the information properly. Since time-based media depends on a frame rate, a frame rate is included in the file, along with size and the actual frame text, repeated for each frame.

The custom text player follows much of the implementation of the VOD Player. It reads the text stream and "sleeps" for the time between frames.

The main problem to solve with this solution is how to display both media types together and synchronized. The JMF provides the ability to "chain" players together. Since all media in the JMF is considered "time based", the composite player's responsibility is to determine an equivalent time base for both players, control them to start in synchronization and display both players together. Rob Gordon's Xsync [1] player implements a manual method of synchronizing players.

The most important issue is that both players must operate at the same frame rate and play for the same period of time. Since the Video-on-Demand framework is responsible for managing the encoding, it enforces the frame rate and length.

3.3 MPEG and Custom Text Player

Since the size of the images can be quite large, a more efficient method of encoding the images was researched. An attempt was made to encode the images as either MPEG-1 or MPEG-4. Since the JMF provides both an MPEG-1 and MPEG-4 compliant decoder from IBM [2], the same method that was used to display Quicktime and the custom text player together could also be used.

The process for encoding the files is as follows. The first step in the process was to transcode the image to YUV. A YUVEncoder class was created to take the series of images, convert them to YUV 4:2:0 format,

create a sequence of the images and write them to a file. The file contains the image sequences in QCIF (176x144) format. The encoder is based on Michiko Fujiwara's YUV applet [4].

For the target of an MPEG-1 encoding, the YUV sequence was transcoded to AVI, using yuvtoavi.exe. The AVI file was then transcoded to MPEG-1 using TMPGEnc by Pegasys, Inc.

For the target of an MPEG-4 encoding, the YUV sequence was encoded using the ISO/IEC 14496 reference encoder provided by Microsoft. This encoder created a CMP file. This CMP file was then converted to MPEG4 using the open source tool, MP4CREATOR60 [5].

The series of steps to create the MPEG-4 encoding proved to be very difficult. The encoding and track extraction proved to be too much of a manual process and much of the software was still in beta. In addition, the IBM MPEG-4 Player for JMF requires that the MPEG-4 video be wrapped in an AVI file format. This is an additional step that was required. As of the writing of this paper, IBM has developed a new Toolkit for MPEG-4 [6], which does not rely on the JMF.

3.4 File Size Comparisons

The final experiments consisted of retrieving 250 frames of images and encoding them into VOD, MOV and MPG files. The VOD file was 11MB, the MOV file was 11MB and the MPG was 7MB.

4. Delivery of Media

There are 2 methods for delivering the media generated. The first is a simple HTTP stream. This stream is by default handled by the JMF. To play the media, the application wraps the URL of the location of the media in a DataSource object. This object is then passed to the manager and the application plays the media normally.

The second method is using the Real Time Protocol (RTP). The JMF provides a mechanism to deliver media over RTP. This method is not as easy as delivery over HTTP. The additional step required is to transcode the media into an RTP format. Formats for RTP in the JMF are RAW, H.261, H.263, and JPEG [7]. Once the media is transcoded, it can be

transmitted using the RTP classes in the Java Media Framework. The video data requires no additional work, since JMF supported video is handled directly through the JMF. The custom text data requires special code for the transmission over RTP. This was examined but not implemented.

5. Issues uncovered during development

Many issues were uncovered in the research and development of the system. Below describes the resolution to some of these issues.

5.1 Reference issues

There are very few references on the Java Media Framework. The best reference found was the book Essential JMF, by Rob Gordon and Stephen Tally [1]. This contained an excellent reference model for implementing new players and included the Abstract Player Framework [1] that was used as a model for the development of the text player. The book also contained excellent references for synchronizing the players. In the search for tools, several free and trial programs were found. The main issue is that the majority of these software programs run on Unix based systems and the target platform is Windows.

5.2 Transcoding issues

For the final solution, an MPEG based viewer, the task of converting the raw YUV 4:2:0 sequences to MPEG proved to be a very difficult task. With the implementation Operating System being a Microsoft Windows Based System, it was difficult to find tools that allowed for command-line based encoding of YUV to MPEG. Furthermore, command-line based encoders require extensive configuration files that are unique to each file created. This is due to the various settings and options that are needed during MPEG encoding.

5.3 Player issues

In order to enforce the use of the Java programming language, a special MPEG-4 player was installed using IBM's AlphaWorks MPEG-4 Player for the Java Media framework [2]. Due to the poor support of MPEG video in the JMF, only MPEG-1 encoding was actually performed. The JMF does not support MPEG-2 encoding and therefore, no attempt was made to encode the video as MPEG2.

An additional problem was discovered during the experiments with MPEG encoded video. Due to the various transcoding operations, the video lost important reference information required to synchronize the player with the text player. To resolve this issue, the order in which the Xsync [1] player was initialized was set to use the text source first (to drive the overall time) and then force the video to conform to the text source time.

6. Future Work

The work performed in this research proves that the Java Media Framework can be utilized to display synchronized video and metadata. The framework is very extendible, however, requires a considerable amount of setup and support files. This leads to a need for a more compact and portable solution.

One possible solution is to imbed or overlay the metadata within each video frame. This removes the requirement of having to create multiple players, multiple datasources and synchronize the players. This also provides a standardized format to deliver the media so that the user may view the video offline.

Further research must be performed to uncover a more efficient method of generating MPEG encoded videos. The current process, which requires manual intervention, cannot be performed in an on-demand fashion. Ultimately, the final solution is to utilize IBM's Toolkit for MPEG-4 [6] to create customized player utilizing sequences that contains the metadata overlaid on the image.

7. Conclusion

It can be concluded that the work performed here proves that both text and video data can be displayed simultaneously. Furthermore, the JMF provides a layered framework to allow for multiple encoding formats to be utilized without the need for custom coding. The fallback of the framework is that it has little support and does not support some of the most recent advances in video technology.

8. Resources

[1] Gordon, Rob, Talley, Stephen, *Essential JMF: Java Media Framework*, Prentice Hall, Upper Saddle River, NJ, 1999.

[2] IBM Corp, <http://www.alphaworks.ibm.com/tech/mpeg-4>

[3] Sun Microsystems, <http://java.sun.com/products/java-media/jmf/2.1.1/solutions/JpegImagesToMovie.html>

[4] Fujiwara, Michiko, http://quattro.phys.sci.kobe-u.ac.jp/Java_demo/YUV/YUV.java

[5] Sourceforge, <http://mpeg4ip.sourceforge.net>

[6] IBM Corp, <http://www.alphaworks.ibm.com/tech/tk4mpeg4/>

[7] Terrazas, Alejandro, Ostuni, John, Barlow, Mike, *Java Media APIs – Cross-Platform Imaging, Media, and Visualization*, SAMS, Indianapolis, IN, 2002.