University of Essex

Department of Computer Science

MSc Dissertation

"Criteria for the Design of a Platform Independent Distance Learning System"

by

Diego López de Ipiña González de Artaza

Supervisor: **Prof. Simon Lavington**

September 1998

SURNAME:	**López de Ipiña**
NAME:	**Diego**
QUALIFICATION SOUGHT:	**MSc in Computer Science (DIMS)**
TITLE OF PROJECT REPORT:	**"Criteria for the Design of a Platform Independent Distance Learning System"**
SUPERVISOR:	**Prof. Simon Lavington**

Abstract

Recently, efforts have been made to incorporate full-motion, high quality interactive Video into Distance Learning Systems. Initial results have shown that Distance Learning can benefit greatly when good quality video playback is achieved. Development of this type of systems, however, poses great challenges: digital video not only requires significantly more storage space and transmission bandwidth than traditional data services, it must be delivered in time, i.e. isochronously, for continuous playback.

So far, existing Distance Learning Systems have always been developed using proprietary software and constrained to only one client platform. This work addresses a software openness approach to video-based Interactive Distance Learning development, using the latest beta release of an Oracle Server System and up-to-date Java technology. The project analyses the trade-offs taken and difficulties encountered to produce an acceptable performance multi-platform solution.

To my parents for all their love and dedication.

Acknowledgements

I would like to express my most sincere gratitude to my supervisor, Professor Simon Lavington, who helped me throughout this project. I would also like to express my appreciation to Neil Dewhurst and Mathew Hunter for their technical assistance, and Karina Chong, the other MSc student involved in this project, for her support and patience. Finally, I must mention my gratefulness to the Government of my country, The Basque Country, for their sponsorship during this academic year at Essex.

Table of Contents

1. Introduction.

The old model of education is changing from one intensive period in life to a continuous training throughout a career: lifelong learning. Many higher education establishments have realised this fact and are trying to meet this need by offering distance learning activities [Ma-98].

"Distance Learning" refers to teaching and learning situations in which instructors and learners are separated by time and/or space and rely on electronic devices and print material for instruction delivery (e.g. video-cassettes, computer-based training, audio and video-conferencing, etc.). Research comparing distance education to traditional face-to-face instruction indicates that teaching and studying at a distance can be as effective as traditional instruction, when the methods and technologies used are appropriate to the instructional tasks [Willis-96].

The aim of this dissertation project has been to apply streaming full-motion video on demand technology to the domain of Interactive Distance Learning. This domain benefits greatly from good quality video because it enlivens many teaching and learning tasks. Watching video instead of reading a text description improves the learning process in quality and speed.

The usage of streaming video rather than other traditional forms of video delivery bring many benefits but on the other hand require an expensive and complicate distributed architecture. Up to very recently, sound and video were stored and accessed via the Internet in the same way as other media data such as text and graphics; locally-stored copies resulted in very high response times. Distance Learning systems are characterised for being isochronous "on-demand" applications that require recorded sound and video to be sent from the server to possibly many clients, and to be displayed as soon as possible, i.e. as soon as enough data has arrived. Usually a Media Player is required in the client machine responsible for playing out sound or video files, and for running an IP streaming protocol with a server to retrieve the media data [Bolot-96].

Significant progress has been achieved in the last few years in the application of full-motion streamed media to computer-based Interactive Distance Learning Systems [Lavington-97]. However, most of these systems were programmed using proprietary technology and hardly ever could be run on multiple platforms. This dissertation targets the development of a multi-platform client-server system using Java and Oracle Video Server 3.0.3 beta release and details the implementation issues involved in its creation. As a result of this development process, a set of guidelines has been generated to serve as help for future projects of this nature. Furthermore, the criteria to transform the system developed into a fully open system that removes the Oracle technology software layer from the client-side, are discussed.

The resulting product from this project can be described as a multi-platform computer-based Interactive Distance Learning system that applies streaming full-motion video and other multimedia metaphors to develop a Tutorial to instruct in Film Editing and Style concepts. The need for a Tutorial in this field was expressed by the directors of a new MA in Film Studies to start this October at the University of Essex. The system developed aimed to answer this need.

Nevertheless, the core of the work done has been the analysis of the problems found when programming a system with real-time media needs in an interpreted language as Java [Tyma-98]. A detailed discussion of the use of Multithreading to improve performance, and of non-proprietary software alternatives to the Oracle-based technology employed, was also undertaken.

Multi-platform and *software openness* terms will appear frequently in this work. Java's "*write once, run anywhere*" philosophy has been considered throughout the project. A step forward in software openness was undertaken in the present system implementation by the adoption of the new Java Swing GUI Toolkit. Swing usage brought to the Film Studies Tutorial proposed a cross-platform user interface with identical '*look and feel*' no matter what operating system it runs on.

1.1. Project Objectives

The objectives marked at the beginning of this dissertation were:

- Produce a full-motion[1] video-based learning environment with synchronised educational data to assist students of an MA in Film Studies at the University of Essex in their understanding of Film Editing and Style concepts.

- Pursue a software-openness approach to the development of the system, avoiding when possible the usage of proprietary software.

- Generate a set of Guidelines to apply in the development of other future systems with the same real-time media needs as the one proposed.

- Study weaknesses and strengths of the proposed system and analyse alternative technology solutions to it.

[1] The term full-motion will appear throughout this dissertation innumerable times. This is made to emphasise the existing difference between high quality video that can be obtained from a video server such as Oracle Video Server; compared with jerky streamed video we are used to seeing on the Internet by means of Xing's StreamWorks or VODLive.

1.2. Background Work

At the University of Essex, some work in this area has been previously realised [Lavington-97]. There is already an existing pilot system, implemented in a client/server platform through the Oracle products Oracle Video Server 2.0 and Oracle Media Object, that combines Distance Learning and video streaming technology. The Distance Learning System Framework they developed was successfully applied to Foreign Language Learning and Medical Studies.

A past year MSc student in the Department of Computer Science extended this system by adding additional interactive facilities and a messaging system to communicate between students and course tutor [Varonos-97].

This pilot system fulfilled the main goal of providing Interactive Distance Learning by means of the usage of full-motion video on demand. However, two major constraints are inherent to it: the level of interactivity provided is limited and its implementation is completely constrained to an out-of-date Oracle technology.

The purpose of this dissertation has been to produce an Interactive Distance Learning system that provides features which the previous generation system lacks. In this project the Java programming language has been applied to produce a multi-platform system that supports a level of interactivity much higher that its predecessor. Moreover the work done has served to establish the criteria that should be followed to distance the client-side of the system developed from Oracle proprietary software.

At this point, it is pertinent to emphasise that the Film Studies Tutorial developed has been the result of the common effort of a group of people rather than of a single person. The members of this team assumed different responsibilities in its development such as creation of the client side, server side or multimedia content. The author of this work was in charge of programming the client side of the system. The server side was implemented by Karina Chong, another MSc. student at the Department of Computer Science, University of Essex [Chong-98]. The domain-specific tutorial information on Film Analysis was provided by Tara Bacon and Whedbee Mullin (Department of History & Art).

2. Client-Server technologies for streaming Video.

This chapter makes an overview of the support technology that was needed to undertake this project. It includes a ‘tour’ through all the generic concepts associated with Video Server technology, the domain of the application developed. Its objective is to go from the more generic and basic to the more concrete and complicated issues involved in this technology.

2.1. Streamed Media vs. Traditional Media

The traditional approach to video and audio data delivery has been the bulk media transfer or *client-pull* method that consists of the server transferring the entire media file before playback begins. The major drawback of this approach is the lengthy delay before the user can start the media data playback. To reduce the delay to an acceptable level regardless the file size, a streaming protocol is necessary. Streamed video is fed isochronously from a server computer with large storage and delivery capabilities to a client machine that decodes and displays the streamed video in real time as it arrives, without waiting until the entire file has been transmitted to start rendering its content [Serber-96].

For streaming to work, the client side receiving the data must be able to collect the data and send it as a steady stream to the application that is processing the data and converting it to sound or pictures. This means that if the streaming client receives the data more quickly than required, it needs to save the excess data in a buffer. If the data doesn't come quickly enough, however, the presentation of the data will not be smooth. Two of the most popular products supporting this streaming video technology that are used on Internet are Stream Works from Xing Technology corp, and VDOLive from VDOnet corp.

The advantages of Streaming Media over Bulk Transfer can be summarised as follows:

1. In streaming media the file is broken up into many small packets and each packet is transmitted individually. The client can begin the playback after receiving the first few packets rather than waiting for the entire file to be downloaded as in the case of “bulk transfer”.

2. Streamed digital-video media delivers the video as a series of small pieces. As soon as one small piece of video has been sent to one viewer, that piece is available to be sent to another. Thus, streamed video can simultaneously serve many viewers. There is also the possibility of using IP Multicasting.

3. With bulk transfer, the client machine must provide enough storage space to contain the entire downloaded video file. Streamed video requires that the client machine provide only enough capacity to store and display the video portion being viewed, and perhaps a small buffer to provide error recovery capability and smooth out trasfer-delay jitter.

2.2. The need for Video/Audio Data Compression

The necessity of compression in order to support real-time delivery of media is due to the huge length of the files this data is stored in. The required amount of compression is such that only “lossy” compression algorithms can be adopted. Lossy systems are important because accepting a small amount of information loss can give a huge payoff in terms of the compression ratio possible. Moreover, the decrease of quality produced is generally imperceptible to the human eye and ear [Tanenbaum-97].

The MPEG (Motion Picture Experts Group) [MPEG-98a] standards are the main *lossy* compression algorithms used to encode both audio and video. Other popular compression formats used are AVI and QuickTime. The basic idea behind MPEG video compression is to remove *spatial redundancy* within a video frame and *temporal redundancy* between video frames. As in JPEG, the standard for still image compression, DCT-based (Discrete Cosine Transform) compression is used to reduce spatial redundancy (I-Frames). Motion-compensation is used to exploit temporal redundancy (B-Frames and P-Frames). The images in a video stream usually do not change much within small time intervals. The idea of motion-compensation is to encode a video frame based on other video frames temporally close to it. MPEG, therefore, achieves high compression rate by storing only the changes from one frame to another, instead of each entire frame. To compress audio MPEG tries to remove the irrelevant and redundant parts of the signal [VIDEONICS-97].

MPEG files are composed of three layers: video, audio, and system. The *video* and *audio* layers (also called elementary streams) contain the encoded video and audio data, respectively. The *system* layer defines the multiplexed structure for the video and audio data, as well as the timing information required to replay the segments synchronised in real time.

There are two major MPEG standards: MPEG-1 and MPEG-2 [MPEG-98b]. The most common implementations of the MPEG-1 standard are designed to produce bit rates around 1.5Mb/s, with images of size 352x240 at 24-30 frames per second (fps). MPEG-1 produces video quality slightly below the quality of conventional VCR videos. Full screen MPEG-1 following the European standard (PAL) for colour television broadcast signals defines a frame size of 352*288 and is encoded at 25 frames/sec whereas the one following the US standard (NTSC) defines a frame size of 352*240 and is encoded at 30 Frames/sec. MPEG-1 is usually considered of acceptable quality for Distance Learning applications. It was the video encoding format used in the Film Studies Tutorial developed.

A newer standard, MPEG-2, offers resolutions of 720x480 and 1280x720 at 60 fps, with full CD-quality audio. This is sufficient for all the major TV standards, including NTSC, and even HDTV. The ISO standards body is currently working on the new standards MPEG-4 [Javasoft-98b] and MPEG-7.

2.3. Problems found in Real-Time Media Delivery

Among the many different types of media available for retrieval, retrieving full-motion, high quality video in real-time poses the greatest challenges. Digital video not only requires significantly more storage space and transmission bandwidth than traditional data services, it must be delivered in time for continuous playback [Serber-96].

Internet is intrinsically a packet-switched network and was not designed to handle isosynchronous (continuous-time based) traffic such as real-time audio and video. Regardless of the available bandwidth, Internet operates as a *best effort service*, and hence there is no guarantee of *minimum throughput* or *maximum delay*. Internet is notoriously unpredictable when it comes to transmission performance. Heavy traffic load and internal transmission problems can cause delays that are beyond anyone's control. The main challenge when designing a real-time data delivery system is to cope with lost packets and bandwidth limitations while meeting minimum audio and video quality requirements.

Bandwidth limitation can be catered for with the adoption of the lossy compression schemes explained in the previous section. *Loss of packets* could be avoided by the usage of the TCP transport protocol. TCP delivers reliable data stream in a correct order at the cost of long delay (due to large packet headers and retransmissions). However, TCP is not usually applied because real-time media, such as audio and video, cannot benefit from re-transmissions in interactive applications. Real-time media is more concerned with prompt delivery rather than with accurate delivery. This type of media has a maximum delay associated with it, and re-transmitted traffic arrives too late to be played back. However, it is not affected by small percentages of packet loss providing that such losses are rare [Bolot-96]. Latencies in delivering video data are extremely noticeable and manifests in the applications that use it, in the form of glitches such as:

- Misaligned or non-corresponding parts of video.
- Some parts of the video stop moving.
- Audio and video not synchronised.
- Video Screen black for short instances of time.

Since TCP requests retransmission, it is not suitable for real-time traffic. Real-time media do not use TCP, but UDP (UDP provides a very lightweight service, and does not re-transmits lost data). However, media that uses UDP as the underlying transport protocol may suffer from packet loss and does not provide any means to overcome it. It is therefore necessary that real-time media tools provide application layer framing (ALF) for the transport protocol used in order to avoid or at least reduce data lost. The Real Time Protocol (RTP) has been developed within the IETF (Internet Engineering Task Force) and is a standard protocol proposed to deal with this issue [Schulzrinne-96]. It provides facilities such as sequence numbers and timestamps that are required by real-time media.

2.4. Hardware solutions to real-time media delivery requirements

Software solutions such as the Real Time Protocol (RTP) can help in part to deal with real-time data delivery problems, but only the adoption of new network technologies that allow resource reservation and maximum time delivery guarantees will guarantee a final answer. Two hardware solutions can be adopted depending on the environment in which real-time media data is delivered. In the case of the Internet Environment, ATM could be the solution while in the Organisations environment where data is transmitted in a LAN, Switched Ethernet is a solution [Tanenbaum-97].

Asynchronous Transfer Mode (ATM) networking is a reliable mechanism for high-speed delivery of large amounts of highly complex data, using fixed-size delivery packets called *cells*. Because of its switched, connection-oriented nature and its fixed cell size of 53 bytes, ATM is an inherently reliable, scalable networking technology.

ATM can offer several advantages over other networking protocols such as TCP/IP, the protocol stack used in Internet, including [Gaurene-98]:

- scalability, to work at different speeds and on different media
- open-ended growth paths, since ATM is not tied to any physical medium or speed
- a single network for delivery of voice, data, and video, thereby improving efficiency and manageability
- compatibility with existing physical networks (because ATM is not dependent on a specific type of physical transport).

When real-time media data is used only within an organisation LAN, such as an Enterprise environment, the hardware solution to tackle with the problems associated to streamed media delivery is to adopt Switched Ethernet technology. Unlike shared Ethernet in which every computer reads every data packet, switched Ethernet uses a hardware hub that provides a dedicated link to each client. The dedicated link allows the hub to physically map each client IP address to the communication channel between the hub and the client. Each client views only its data and does not spend time examining packets addressed to others on the same communication channel [Linden-98].

2.5. System Architectures used for real-time media delivery

Real-time media delivery systems have three main goals:

- Retrieve large encoded content files from storage
- Stream these files at high delivery rates across a network that may be unreliable
- Reassemble, decode and display the streamed content on a client device.

The components of a real-time media delivery system that make possible the fulfilment of these goals can be seen in Figure 1:

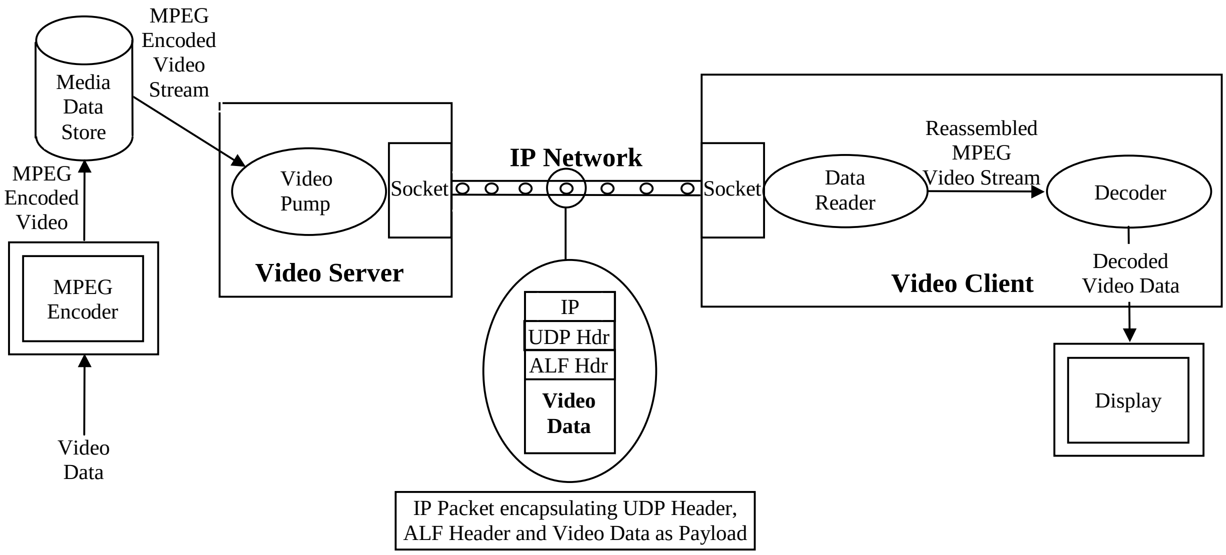


FIGURE 1
Video Data Encoding Decoding Process Overview

- The **encoder** encodes the media (audio or video) data into the compressed format that can be efficiently transmitted over the Internet. This process is usually done off-line and placed in a real-time media server database, named Media Data Store.

- The **Media Data Store** (MDS) is a real-time file system for storing and delivering uninterrupted video in real time. MDS files are stored in MDS volumes, named collections of disks. Each volume has a table of contents that lists the files in the volume and their locations on disk. Usually a technique called *striping* is used for storing the media data. Space *Striping* means dividing a file into pieces called stripes (typically of 32k or 64k each) and storing each of them on a different disk. Striping a file distributes access to file across many disks, rather than concentrating it on one, reducing requests for each disk and improving performance when the file is accessed by many concurrent clients [Lee-98].

- The **Video Server** is responsible for delivering the data, controlling the flow of data, and other tasks related to performance monitoring and control. The Video Pump packetises the encoded video data obtained from MDS and sends it over IP networks in a continuous pace.

- The **Video client** is responsible for obtaining, decoding and displaying the video stored on the server. It contains a Media Player that decodes (decompresses) the data stream into a format that can be played back on the client's computer. The client does not retain the data, once the digital video data has been displayed it no longer exits on the client machine.

It is worth noting that the video content will have to be encoded only once but will be decoded thousands of times. This asymmetry means that is acceptable for the encoding algorithm to be slow and require expensive hardware providing that the decoding algorithm is fast and does not require expensive hardware.

2.6. Approaches to Real-Time Video Delivery: VOD and NVOD.

The two more characteristic approaches to video data streaming are: Video On Demand (VOD) and Near Video ON Demand (NVOD). VOD implies that the video Server can be configured to enable video clients to watch what they want to when they want to, choosing any title that has been digitised and stored for delivery. In contrast, NVOD or scheduled video happens when the Video Server streams a specific video data on a specific channel at a specific time. Clients can then "tune in" to see the video they want with no signals required from the client to the Video Server. Since the server pushes video data to the client at a controlled rate, these approaches are called *server-push*.

2.7. Differences between a Data Server and a Video Server

Database servers are tuned for optimum database performance. The primary goal of a database server is to offer a rich set of operations on structured data and to deliver transaction results quickly and reliably. In most database applications the requirement is to get accurate data; if the data is not accurate, the database is not useful.

In most video applications, however, the key requirement is that the video data arrive on time, so that it can be received and displayed in real-time. To that end, the processes running on a video server have to be optimised to provide real-time guarantees based on resource allocation of CPU, disk, and network bandwidth. The most important resource on a video server machine is its *I/O bandwidth*. Unlike Data Servers which must execute procedures and perform complex sort and join operations, video servers require comparatively less CPU. However, video servers must have efficient disk drives, network cards, device drivers, and a fast system bus, because the primary goal of a Video Server is to read the video data from disk and put it onto the network as efficiently as possible to be displayed on the client [Linden-98].

2.8. A Real-Time Media Delivery System: Oracle Video Server 3.0.3

In the development of this project, Oracle Video Server 3.0.3 in its beta release has been used. This section describes in detail what are the different components of Oracle Video Server architecture and explains how one actual real-time media delivery system works in practice.

The Oracle Video Server System in its version 3.0.3 is composed of three main components [Linden-98]:

- Oracle Video Server
- Oracle Video Client
- Oracle Media Net.

The Oracle Video Server (OVS) function is to store and deliver encoded video data into applications running on PCs or NetWork Computers.

The Oracle Video Client Software enables the development of interactive, video-based multimedia applications such as distance learning using movies on demand. Oracle Video Client applications can receive and display MPEG-1 system streams, MPEG-2 transport streams and Raw Key Frame, including OSF (Oracle Streaming Format) files from the Oracle Video Server [Herrick-98].

Oracle Media Net is a networking infrastructure that enables the OVS and its clients to communicate in a distributed computing environment. Using heterogeneous network protocols, Oracle Media Net enables connectionless communication among the various OVS system components running on different platforms. Oracle Media Net is Oracle's implementation of CORBA [CORBA-98], so heterogeneous services programmed in different languages and distributed over different computers in a network can communicate without concern for each other's location or the details in transporting data or converting data among them [Maring-97].

The client applications that run on the client device communicate through OMN with specialised server applications, or services, that perform specific task on behalf of the client, such as provide the client with information on available videos. Note that the client and server communicate through OMN to set up and control the streaming of video, but the video data itself is transported as real-time data directly to clients using standard network protocols, and is not transported through OMN.

2.8.1. Oracle Video Server 3.0.3 Architecture

Oracle Video Server function is to transmit video data over the network in a paced, isochronous (continuos time-based) fashion. OVS consists of a group of services that work together to provide clients with access to video and other types of OVS data [Fursbush-98]. The Services Oracle Video Server provides are:

- **Session Service.**

  The Session Service establishes and maintains the client/server communication channels and manages a set of OVS resources on behalf of a particular client device. It keeps track of all the clients currently connected to the video server and the server resources that have been allocated. When a client wants to stream video, it first establishes a session with the Session Service that ensures that the resources needed to service that client are available. If no more resources are available (e.g., if the server has reached the maximum number of concurrent video streams it can support), the client request will be denied.

  As a result of a successful session allocation the client will be assigned with:

  - An identifier that uniquely identifies it from the other connected clients.
  - One or more circuits through which control messages and video are transported. Generally a session is composed of a downstream (from Server to Client) channel to deliver video and a bi-directional channel to carry control messages.
  - Several Resources for the client and its connection to the OVS. Two key resources that are required for all clients are a stream object (or stream context) and a Video Pump thread.

- **Content Service**.

  The Content Service provides the client with information on the content available to it from the OVS system. It queries the Media Data Store for a list of video content available for streaming. The video server associates each piece of content with a server-defined structure, called *asset cookie* (or stream id). The contents of the asset cookies are opaque to the client application and are used only by the server.

  A client application obtains asset cookies in response to queries on the content service. To obtain video or other types of content from the video server, the client passes the asset cookie associated with the desired content back to the server, which then uses a content resolver to “resolve” the asset cookie into the actual pieces of content in the MDS.

- **Stream Service**.

  The Stream Service is the process all clients communicate with to control their video streams. It represents the "virtual VCR" for each client, supporting operations for video playback such as open, prepare, play, pause, and seek. The stream service directs a video pump to deliver OVS data to the client device in the form of a real-time stream. Streams are delivered directly from the server to the client over an isochronous circuit.

  To receive a stream, the client application requests a stream object (or stream context) from the Stream Service. When allocating a stream, the playback capabilities of the client device must be specified to the stream service. Once allocated, this context allows the client to open a video and perform playback operation on it, just by passing to the stream service an asset cookie that specifies the content to be streamed. Clients do not communicate with the Video Pump directly, the Stream Service acts as an intermediary, making sure that the Video Pump sends the correct data from the content file. The Stream Service is the only component of OVS that knows exactly what type of content is being streamed

- **Video Pump.**

  The Video Pump is a multi-instance and multithreaded process, that receives streaming commands from the Stream Service, reads the Video Data from the Media Data Store, packetises the data, and then sends it over IP networks to the client.

  In order for the OVS to transport the video data over the network, the video pump must convert the video stream into a series of packets. For streams consisting of data encoded using MPEG-1 System Streams or Oracle Streaming Format (OSF), the video pump divides the video stream into a series of chunks and wraps each chunk with an Oracle Generic Framing (OGF[2]) header. The client uses the information in the OGF headers to reassemble the video data back into a continuous stream.

  When operating in local-area networks using UDP, Oracle Video Server uses a default network packet size of 8K[3] to transmit video data. Depending on the network protocol and components in use, these packets may be fragmented further as they are routed from source to destination [Baltz-98].

  The Video Pump uses double buffering, filling two buffers before beginning to stream to the client. When it has exhausted the first buffer by streaming all the data from that buffer, it fills it again with a request to MDS for more data and streams in the mean time from the other buffer.

[2] Note that OGF is the Application Layer Framing (ALF) added by the OVS to the UDP packets transmitted.
[3] The default size is the maximum UDP packet size = 8192 bytes, however this size is adjustable.

Unlike the Stream Service, the Video Pump does not have any knowledge of the type of content being delivered. It only manages network delivery issues, such as framing and timing.

- **Media Data Storage Services**

Media Data Storage Service stores and provides access to all of the media data in the OVS System. MDS (Media Data store) is the repository used by Oracle Video Server for storage of real-time content. It uses Oracle's patented real-time RAID software technology, which allows Oracle Video Server to stream video to large number of clients even in the event of disk failure. Clients can receive data from OVS in the form of a real-time stream or a binary object, called a BLOB (Binary Large Object). Note that the client receives BLOB data over a Media Net control channel and real-time data directly over an isochronous channel from the server.

Once a content file has been copied into MDS, it must be "tagged". OVS creates and stores a tag file associated with each content file. Tag files are small files (.mpi extension files, typically) that contain indexing information about the content, such as byte offsets of the I-frames in the content. They also contain other information needed by the Stream Service to play the file, such as the encoding type (e.g, MPEG-1) and the bit rate. Tag files are used by the Oracle Video Server for any operation requiring random access to the content files, such as seeking, fast forward, and rewind. All content must have an associated tag file in order to be available for streaming.

Two concepts very important to be understood on the storage logic employed by MDS are the terms Clip and Logical Content. A clip is a logical part of a physical content file. Each clip maps to a specific start and stop position within the file. A Logical Content denotes a collection of titles, each representing a clip. The OVS enables system administrators to assemble these clips as they see fit and then label the assembled clips and videos with a single title, to make retrieval and delivery easier. These clips do not actually need to physically be assembled into a single file and are accessed from their physical locations by OVS's internal logic.

It is worth commenting the constraints OVS 3.0.3 brings regarding clips assembling. Experience during the project development showed that the maximum number of clips allowed being part of a Logical Content is 9. Moreover, all the clips that compose the Logical Content have to be encoded under exactly the same encoding parameters.

2.8.2. Oracle Video Client 3.0.3 Architecture

OVC handles the technical aspects of the video streaming process and provides three different client interfaces around which client applications can be created. The Oracle Video Client (OVC) software consists of two main parts [Herrick-98]:

- The OVI abstraction layer and
- The client interfaces:
  - Oracle Video Web Plug-In,
  - Oracle Video Java Library and
  - Oracle Video ActiveX Control.

The Oracle Video Client Interfaces use the Oracle Video Interface (OVI) to access streaming video from the Oracle Video Server. The separation of client interface from the basic streaming functionality supported by OVI means that the functionality of the Oracle Video Server can be harnessed from a variety of platforms, without worrying about the underlying mechanisms that handle the low level tasks.

The OVI abstraction layer handles the technical aspects of communicating with the server, controlling the real-time stream, and audio-video playback. OVI is a client of the OVS and the client interfaces can themselves be considered clients of OVI. Its functions are:

1) Communicate with the Client Interface, passing requests from the Interface on to the Video Server
2) Handle network communications between the client machine and the Video Server, including managing video and audio streams as they arrive from the Video Server.
3) Communicate with the playback and display technology, such as Oracle Video FrameWork and ActiveMovie.

OVI software layer is itself composed of five components (see Figure 2):

- **Producer.**

  The Producer is the interface between the network and the Stream Manager. It receives video data over UDP from the Oracle Video Server and delivers it into the Stream Manager’s buffer pool. It is also responsible for handling network errors such as missing packets or packets received out of order. The Producer component has no knowledge of the type of content being received.

- **Consumer.**

  The Consumer is the interface between Stream Manager and the video decoding solution, either hardware or software. The Consumer takes video data from the buffer pool and feeds it to the video decoder as needed for display. This component based architecture allows the software to stream to different third-party decoding solutions by writing new Consumers for them.

- **Stream Manager.**

  The Stream Manager is responsible for coordinating the data flow from Producer to Consumer. To do this, it manages a pool of buffers containing the incoming video data. Oracle Video Client buffers a certain amount of data so that delivery to the decoder can be as smooth as possible. The Stream Manager controls the video stream and performs actions such as pausing and resuming the server when necessary to keep the right amount of data buffered.

- **Video Decoder.**

  The Video Decoder receives the compressed video stream provided by the consumer, decodes it and passes it to the Video Player.

- **Video Player.**

  The Video Player receives a sequence of video frames, still images, and displays one frame after another. When dealing with MPEG-1 it has to display at a rate between 24 to 30 frames per second. OVC works with ActiveMovie decoders, so the decoder used must me compliant with ActiveMovie 1.0, no matter if it is hardware or software.

Figure 2 entitled “Video Streaming Process in OVS 3.0.3” synthesises all the concepts explained about OVS 3.0.3 System Architecture. It also shows the steps followed when a client connects to an Oracle Video Server requesting streaming video data.

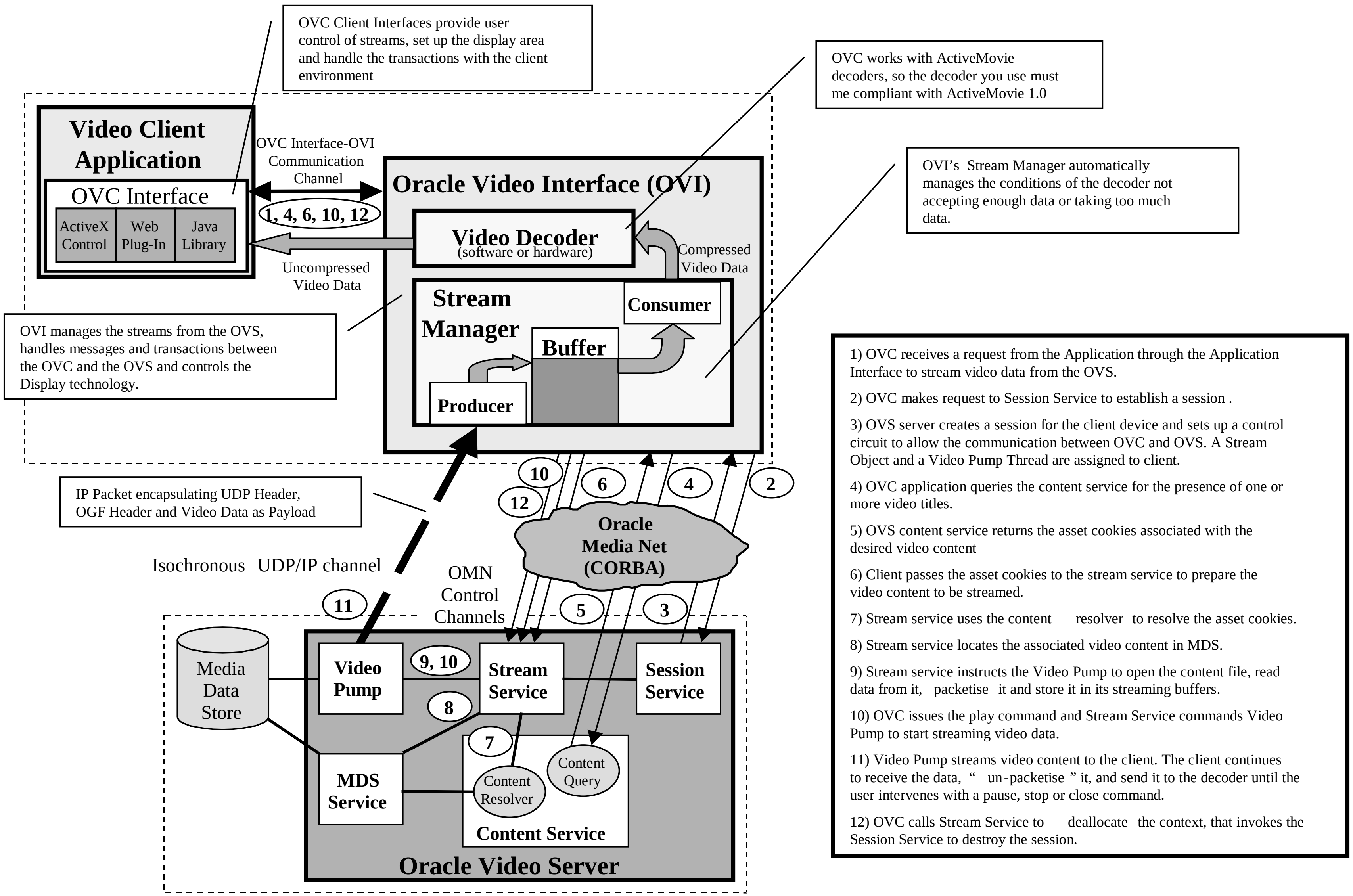


FIGURE 2

Video Streaming Process in OVS 3.0.3

2.8.2.1. The Oracle Video Java Library Interface

The Oracle Video Java Library is the OVC Interface that was chosen to develop the Film Studies Tutorial. This interface enables Java applications to play streaming video and audio from OVS. It provides a number of public classes and interfaces that allow the programmer to playback and control video and audio streams, find out information about the current stream and query the video server for available content titles [Herrick-98].

The **Player Classes** constitute the primary classes and interfaces in the Oracle Video Java Library. These are:

- **Player Class**, which contains all of the functionality needed to directly control media file loading and playback.

- **PlayerFactory Class**, which creates new Player objects.

- The **PlayerListener** interface, which allows your application to be notified of events affecting the Player object.

- **PlayerException Class**, which supports notification of raised exceptions specific to the Oracle Video Java Library.

The **Stream Information Classes** are used to report on the current stream, including:

- Stream position, by means of the class **StmPos**.

- Stream Information, such as the name, transport protocol, description, frame rate, and so on, through the usage of class **StmInfo**.

- Stream statistics, such as the number of data packets received, number of data packets dropped, average frames per second, and so on. The **StmStats** is used for this purpose.

The **Content Query Classes** are used to query the Oracle Video Server for a list of available content files. The classes are:

- **Content**, which performs the actual content query.

- **ContenIter**, which is used to iterate through the returned content entries.

- **ContentException**, which is thrown when an exception occurs during the query operation.

3. Devising a platform independent Distance Learning System

The core of this project has been to develop an Interactive Distance Learning System using Oracle Video Server 3.0.3 and Java. Film Studies was chosen as the domain for which the Distance Learning System would be developed. The main purpose of the system was to produce an Interactive Distance Learning Tutorial System that would allow the students of an MA in Film Studies to be started in October 1998 at Essex, to understand the diverse range of camera techniques applied when a movie is being recorded. The technical name of the knowledge field this Tutorial treats is ‘Basics of Continuity Editing and Style’.

This chapter describes the stages that were followed to device such system:

- *Identify System Requirements.*
  The requirements to produce a system of these features were identified.

- *Produce a System Description*.
  Once the requirements of the system were understood, the system to be produced was devised. At this stage, mock-ups of the GUI the system would support were produced and all the facilities it had to support from the user point of view were studied.

- *Define the System Architecture.*
  At this stage the author focused his attention at what hardware and software architecture was required to produce the Tutorial. This was the first time in which HOW the system would be implemented was considered.

- *Undertake the System Design*.
  After having identified the system needs or facilities and had understood the system architecture needed to achieve them an Object oriented Design took place.

  The outcomes of this process were:

  - A Class Diagram defining all the classes necessary to support the system functionality.
  - Two Flow Diagrams identifying the different states the system would go through during its future execution.
  - A concurrency specification describing the concurrent parts of the system.
  - A Database Design, for the Educational Data associated with the Video Content of the Tutorial.

3.1. System Requirements

The system requirements of the Film Studies Tutorial proposed, a multi-platform Distance Learning system that uses video and multimedia educational data, are:

- *Create and capture the video content*.

  The video material used in the system was selected from the classical movies “The Big Sleep”, “Suspicion” and “Psycho” from the famous British Director Alfred Hitchcock and it was digitalised.

- *Encode the captured video.*

  Suitable fragments of the above three movies were encoded to the MPEG-1 format and with all of them a Logical Content was formed.

  Remember that Logical Content, in OVS terms, can be defined as a ‘logical file’ consisting of a set of different video segments called clips. A clip is another logical storing structure that makes reference to a fragment of a physical video file.

  In common agreement with all the stakeholders of the system, it was decided that the Tutorial Video Content would consist on a OVS Logical Content build from three clips representing fragments of the movies “The Big Sleep”, “Suspicion” and “Psycho” respectively. These 3 clips were encoded into MPEG-1 system stream with interleaved video and audio data. The encoding parameters that were employed followed the PAL standard that defines a frame size of 352*288 pixels, at 25 frames/sec, encoding the audio at 44100 Hz and producing a video/audio bit rate of 1.5 Mb/sec.

- *Associate educational content to the video*.

  In order to provide an effective Distance Learning system it was necessary to associate to the Tutorial Video Content some multimedia educational data that would give explanation to the concepts the video data was intended for. Having in mind the purpose of the learning system, *Continuous editing and style*, it was decided after pertinent meetings among the developers and potential users of the system, that the most sensible way of structuring the Tutorial was by basic movie units called shots. “A shot is the basic building block of a film. An unedited shot corresponds to the amount of film footage exposed by the camera from the time it is turned on until it is turned off. Individual shots are then sequenced and trimmed by the editor to create the finished film” [Prince-95].

  Once the basic Video Content Division Unit was established, the stakeholders of the system indicated that for each shot, a set of Diagrams explaining each of the different camera movements produced inside a shot was required. Moreover, each shot had to be associated with a set of keywords identifying the film editing and style concepts explained in it and a Lecturer Notes Text.

- *Design a framework to integrate and synchronise Video and Educational Data.*

  An application framework was designed to allow the communication between the Distance Learning Video Client and the two Servers that provide data to it, a Relational Database Server with multimedia educational information, and a Video Server with Video Content. This framework had to integrate seamlessly for the user point of view the data coming from these two sources and what is most important it had to synchronise them in order to display the appropriate educational data at the pertinent position of the Tutorial Video Content.

- *Provide real-time interaction facilities to the student.*

  In order to produce a successful Distance Learning Tutorial it was necessary for the system to provide a high degree of user interactivity. The user was required to be able to have full VCR-like control over the Tutorial Demo. Therefore, controls that would allow the user volume control, fast-forward, stop and resume the Tutorial, move to the previous or next shot or simply move to the Tutorial Position the student wanted to focus the attention on were clearly needed. The System Description section gives a detailed explanation of all the interactivity required in the proposed system.

- *Develop a communication system to bridge the 'distance' existing between supervisor and learner.*

  To achieve a complete Distance Learning system, it is necessary to provide a tool that would allow some communication between the student and the supervisor. For this purpose an e-mail system was developed.

- *Deploy the application for one or more platforms.*

  The way to provide a Distance Learning system that could be run in all the platforms without the requirement of source code re-compiling was to implement it using the Java programming language. Java programs can be run in every system that provides the Java Virtual Machine [Kramer-96]. Moreover, a step forward towards software openness was achieved through the use of the Swing JFC component [Andrews-98], that allows the application designed to look and behave (feel) exactly in the same way independently of the platform it is run. With Swing, applications can adapt to user's preferences rather than to the platform in which the application where it is run. Swing is part of the Java software openness revolution in which applications will not have an identifying look and feel due to the platform they run on.

3.2. System Description

The first stage in the development of the Film Studies Tutorial System was to capture the requirements of interactive facilities that the system to be built should provide. Moreover it was necessary to understand how the Film Studies Tutorial would be structured and what would be the different GUI components that would allow the display of all the educational multimedia material involved in the system.

At this stage it was still to be determined what the system proposed would exactly do and look like. After several meeting with all the stakeholders of the proposed system it was decided to divide the functionality of the system in two parts:

- Tutorial Demo
- Glossary

The Tutorial Demo is the essence of the application. It constitutes the framework in which different multimedia elements are combined to produce an attractive Interactive Distance Learning System to the user. Figure 3, shows the layout of the different components that compose the Tutorial Demo Panel or GUI:

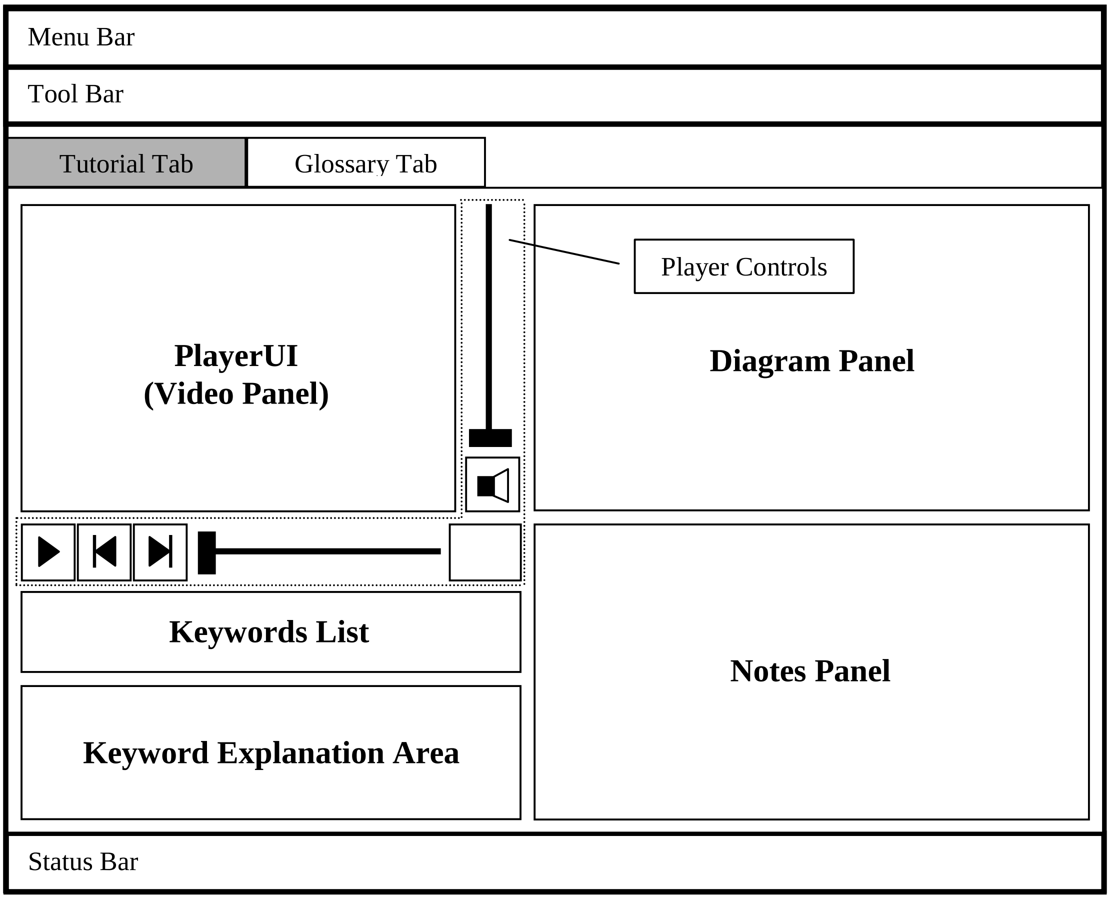


FIGURE 3
Tutorial Demo Panel Layout

- A *Video Panel* in which a movie streamed from the Video Server is played. The Video Content rendered on this panel is composed of three sections or clips, according to OVS terminology as it was explained in the System Requirements.

- A *Diagram Panel*, where concurrently with the movie played a GIF Diagrams Animation is produced while the movie is being played. The aim of this Animation is to explain the student the camera techniques that where employed in the recording of each of the film shots. The Tutorial Film can be stopped by a click of the user over the Diagram Panel. Once the Demo has been stopped, whether because the user clicked over the Diagram Panel or other way of stopping was used, mouse movements over the components contained in the current active diagram will produce Tag Tips to come up, with the name of the component the mouse is on. It should be noted this facility was added in order to help the user distinguish the different elements of a diagram drawn in sketch mode, where only the objects to explain a Film Editing concept, the camera and film characters, are shown with higher definition.

- A set of *Player Controls* (see Figure 4) consisting of some video playback control buttons, two Slider Bars and a Shot Number Display. These control elements provide the full VCR-like control required in the Tutorial. The control buttons allow the user to play or pause the movie, move to the next or previous shot in the movie and turn on/off the volume. The two sliders are a Movie Position Slider and a Movie Volume Slider. The Movie Position Slider allows the user to move flexibly through all the different positions of the Tutorial Demo. The reason to add this control was thought as a means of allowing the user flexibly move to the point in the Tutorial where the students want to work. The Movie Volume Slider purpose is to control the volume of the Tutorial Film. Finally, the Shot Number Display objective is to show the student the position inside a Clip where the Demo is on.

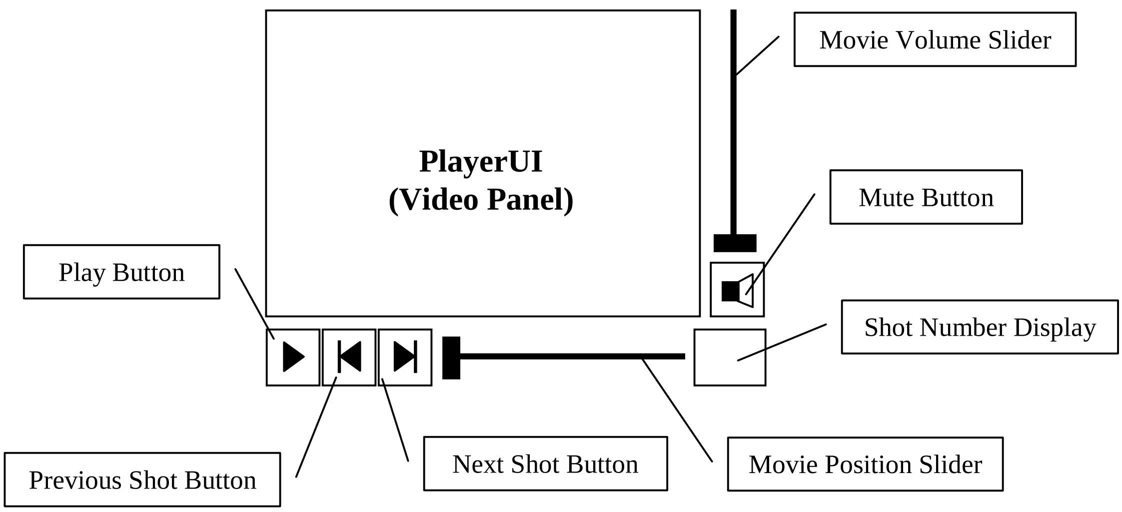


FIGURE 4
Tutorial Demo Player Controls

- A *Keywords List* where keywords associated to each of the shots of the Tutorial Film are displayed. The function of this control is to point out the Film Editing and Style concepts that are employed at the current Shot of the Tutorial Film.

- A *Keyword Explanation Panel*. Whenever the user clicks a Keyword List Item, the Demo stops, in case it was running, and the explanation of the selected keyword is displayed in the Keyword Explanation Area.

- A *Notes Panel*. This panel contains the notes associated to each shot. This notes only appear when the Demo has been stopped. The notes can contain two different kinds of links: *Diagram Links* and *Keyword Links*. The user click over a Keyword Link brings about a tip window to pop up explaining the keyword the link corresponds to. A click over a Diagram Link results in a new Diagram appearing in the Diagram Panel. This is necessary to explain certain concepts in the notes when reference is done to previous shots and comparison is needed with the current Active Shot in the Tutorial Film.

The Glossary (see Figure 5) of the Film Studies Tutorial was created to allow the student to revise the different Film Editing and Style concepts (keywords) the Tutorial gives explanation to. The Glossary GUI or Panel is formed by the following two components:

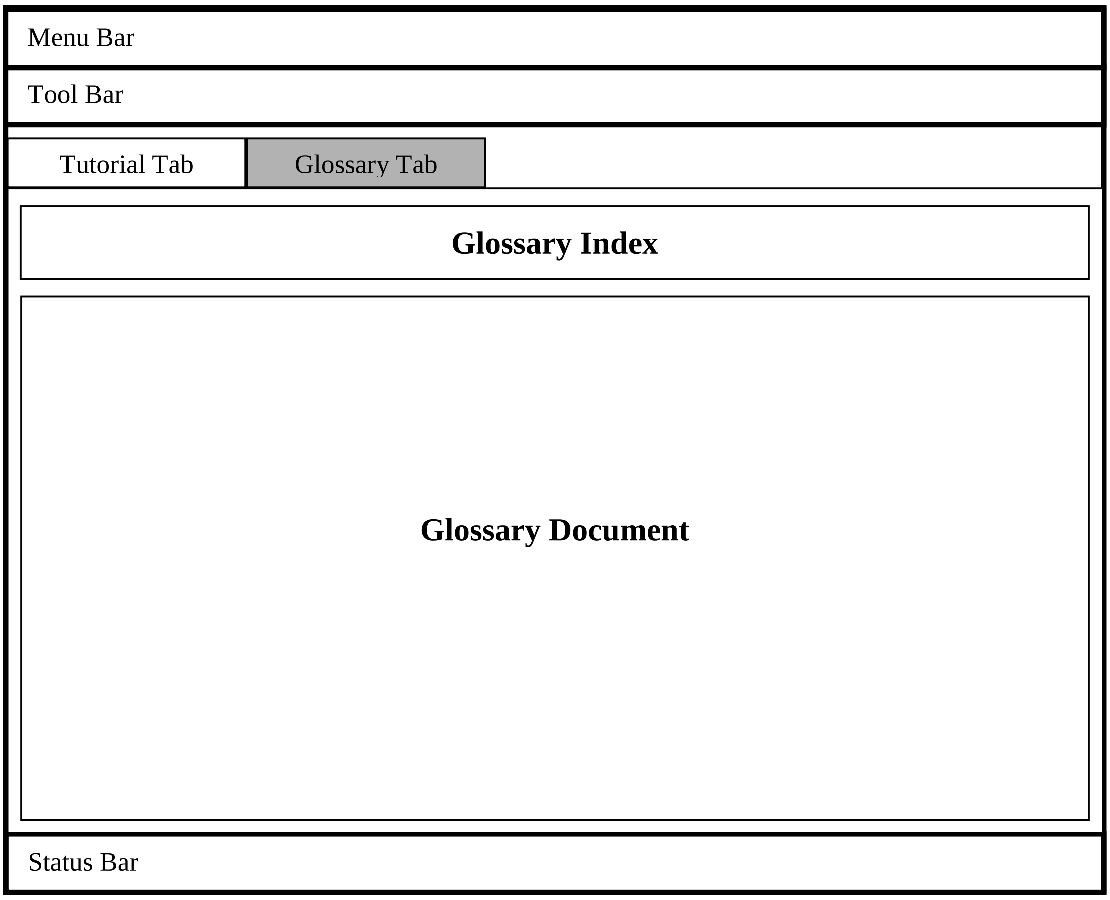


FIGURE 5
Glossary Panel Layout

- A *Glossary Index* consisting on an click-able alphabetical Index to provide quick access to the beginning of the section of the Glossary where Keywords starting by the letter clicked are displayed.

- A *Glossary Document* containing all the keywords explained in the Tutorial Demo together with their explanation and link to their bibliography references. The keywords are listed alphabetically. When the user clicks over a Bibliography link, the bibliography references associated to the keyword are displayed.

3.3. System Architecture

Once the facilities the system had to provide were identified, the next stage in the development process of the Film Studies Tutorial was to identify the software and hardware architecture needed to make it possible.

The system architecture proposed was a client-server one (see Figure 6). In the Client Side the Java Distance Learning application resides together with the interfaces that allow the client system gain access to the Server Side. In the Server Side two different Information Servers are found. On one hand, the Oracle Video Server attends to the requests of the clients for streaming video, whereas on the other one an Oracle 7.3.2 Relational DBMS stores the multimedia educational data associated to the encoded-video placed at OVS's Media Data Store. The hardware networking infrastructure that makes possible the communication between the two sides of the system is a Fast Switched Ethernet (100 MHz) LAN.

The client communication with the Oracle Video Server takes place through OVI (Oracle Video Interface). This Oracle Video Client Software Component deals with low level details concerning the reassemble of the Video Data Packets that arrive from the Server Side and the decoding of them for its posterior visualisation. The actions carried out by OVI are completely hidden to the programmer. During the implementation of the client side in Java, the Player class provided in the Oracle Video Client Java Library Interface was used to deal with the basic interactive operations over the streaming video data sent from OVS.

The communication between the client stand-alone application and OVS takes place through two different channels: a TCP control channel and an UDP isochronous (continuous time-based) data channel. The TCP control channel functionality is to request OVS services by means of Oracle Media Net, the communication infrastructure between OVC and OVS. The isochronous data channel is a downstream channel (from server to client) that enables the delivery of the UDP packets in which the MPEG-1 encoded data is packetised by the Video Pump component of OVS.

On the other hand, the Distance Learning Video Java Client gains access to the Relational DBMS by means of the JDBC (Java Database Connectivity) [Guan-98] API. The communication between the Java Client and the DBMS takes place through a native-protocol all-Java JDBC Driver (Type 4). See "Appendix A: JDBC Notes" for further details about JDBC drivers.

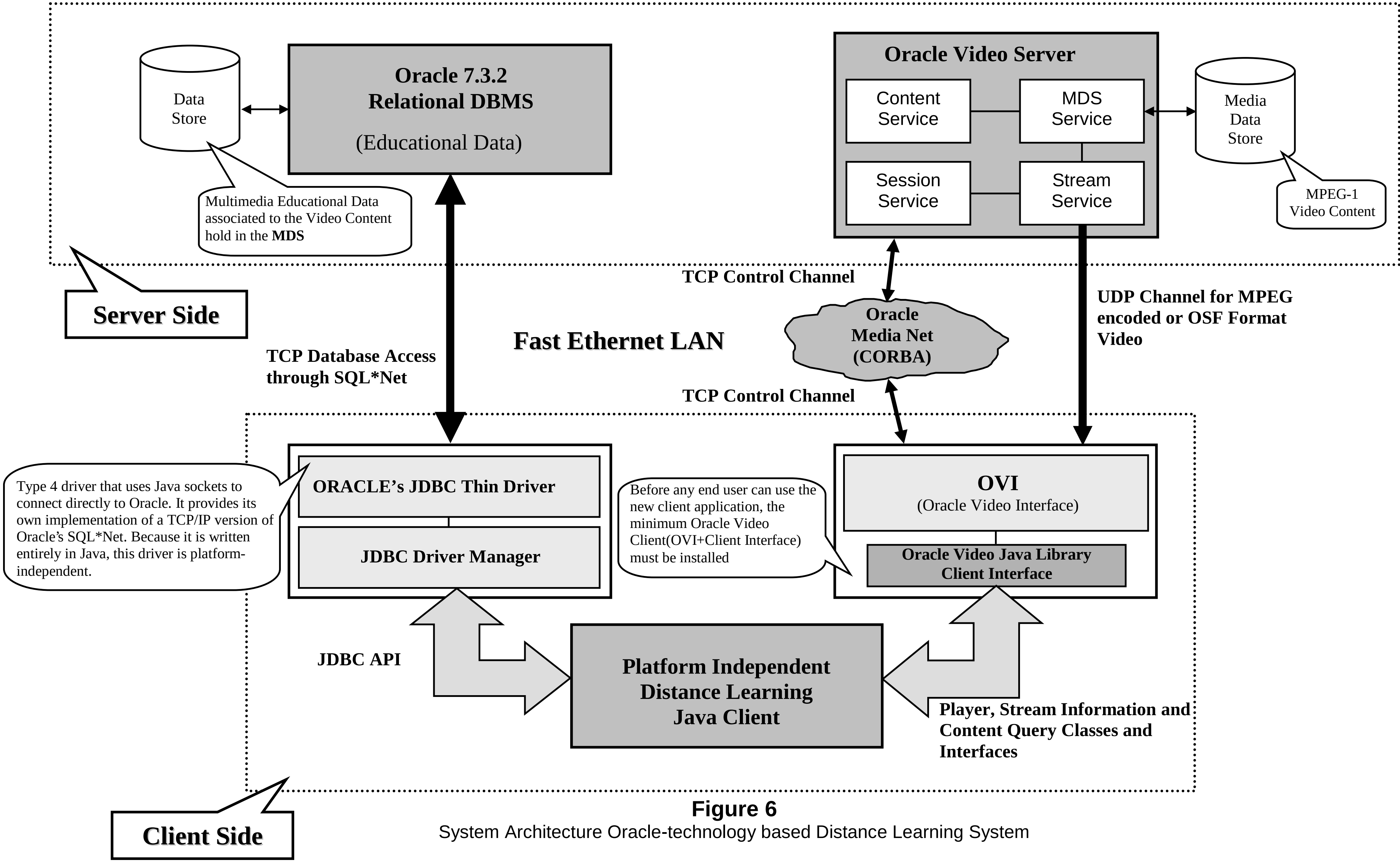


Figure 6

System Architecture Oracle-technology based Distance Learning System

3.4. System Design

Having described WHAT facilities the system should provide to the student and WHAT system architecture will support them, the next stage was to determine HOW the system could be implemented. For this purpose the following stages where followed:

1. Develop a Class Diagram containing all the classes that will be needed to implement the Film Studies Tutorial functionality. An object-oriented approach to design was taken because the language chosen to implement the system, Java, is a fully object-oriented language.

2. Analyse the flow of interaction between the user and the system, producing flow diagrams. This study was realised to the sophisticated and complex level of interactivity required in the proposed system.

3. Analyse the concurrency elements of the system proposed and produce a Concurrency Specification of them. Seamless synchronisation of educational data and video content is a key issue in the Film Studies Tutorial that requires parallel execution of tasks.

4. Design the database that will hold the multimedia educational data associated with the Tutorial Demo Video Content.

3.4.1. Film Studies Tutorial Class Diagram

Taking into consideration the rich user interface required to implement the Film Studies Tutorial, the classes defined were largely based on the different GUI elements. Figure 8 depicts the Class Diagram obtained using Booch notation [Booch-93]. It defines all the classes considered necessary to implement the Tutorial. Figure 7 explains the different notation elements employed in the diagram.

- **FilmStudiesTutorial.**

  This is the main class of the system. This class is in charge of building the framework of the application and establishing the connection with the Relational Database System. It defines the MenuBar, the ToolBar and the StatusBar of the system. Events taken place on any of those GUI elements are handled by a FilmStudiesEventListener class instance, contained in the FilmStudiesTutorial. To establish the connection to the RDBMS, the FilmStudiesTutorial class creates an instance of the class SQLHandler. FilmStudiesTutorial contains an instance of each of the classes that implement the whole functionality of the Tutorial, the TutorialDemo, the Glossary class and the Mailer Class.

- **FilmStudiesEventListener.**

  This is the class to which the FilmStudiesTutorial class delegates the events that occur over its GUI elements.

- **SQLHandler.**

  This class handles all the interactions between the Film Studies Tutorial and the Relational Database System. It defines the methods that allow the system to retrieve the multimedia educational data associated to the Tutorial Film.

- **TutorialDemo.**

  The TutorialDemo class holds the functionality essence of the interactive distance learning system. It creates the UI that holds the Interactive Tutorial Demo. Its attributes correspond to all the GUI elements that form it. The PlayerUI Panel is used to display the video. The PlayerControls are a set of buttons and slider bars, that hold all the interactive operations that can be applied to the Tutorial Demo, such as stop or resume it, go to the next or previous shot, seek to a random position in the Demo, turn on/off the Volume and control the volume level. The DiagramPanel is a panel to contain the Diagram animation produced for each shot of the movie. The KeywordList is a list where all the keywords associated to a shot are displayed. The KeywordExplanationArea is where the explanation of the keyword clicked is shown. Finally, the NotesPanel is the area where the notes of a shot are displayed. It also provides all the methods that allow the manipulation of the GUI elements mentioned. All the events that happen over these GUI elements are handled by an instance of the class TutorialDemoEventListener.

TutorialDemo's most important attributes are an instance of the VideoPlayer class and another of the DemoWorker class. The VideoPlayer handles the interaction with the Video Server and provides to TutorialDemo VCR-like control over the Logical Content streamed. The DemoWorker class instance controls the synchronisation between the Logical Content played and the educational data associated to it.

- **Glossary.**

  This class creates the GUI of the Glossary, composed of the GlossaryIndex and the GlossaryDocument. All the events that can happen in the Glossary such as click over and index letter or bibliography link of a keyword are handled by an instance of the class GlossaryEventListener. The Glossary contains a list of instances of Keyword class that define all the terms contained in the Glossary.

- **Mailer**.

  This class implements the proposed communication system between learners and course tutor. Its function is to provide the Film Studies Tutorial with email facilities.

- **TutorialDemoEventListener.**

  This class is the class used by TutorialDemo to delegate the events that occur over its GUI elements.

- **VideoPlayer.**

  This class implements the Player that loads the Video Content from the Video Server and controls its playback. It defines methods to provide the system full VCR-like control over the video streamed (play, stop, pause, resume, forward and rewind methods). It also supports other kind of control methods to know details about the video streamed, getInfo(), the current position, getPos() and the state of the player, getState().

- **DemoWorker.**

  This class holds the most intelligent element of the system defined. It carries out the control of all the Tutorial Demo execution. It provides methods that allow the Demo to be started, stopped and interrupted. The doWork method invokes the play method of the VideoPlayer class to play the video, tracks each of the shots the video passes through and for each shot it produces a diagram animation representing the camera movements produced within it and displays the keywords associated to it. When stopped it shows the notes associated to the shot where the movie was stopped.

- **LogicalContent.**

  This class defines the attributes associated to the Tutorial Film. It has a set of instances of the clip class. In the case of the FilmStudiesTutorial it actually has 3 clips, as it was explained in the section System Description.

- **Clip.**

  This class contains the details of each of the clips, the LogicalContent is composed of. A Clip instance contains several Shot class instances.

- **Shot.**

  This class holds all the details of the shots contained in a clip. A shot contains 0 to n Diagram class instances, 0 to n keyword class instances and 0 to 1 Notes Text.

- **Diagram.**

  This class represents each of the Diagrams a shot is represented by. A Diagram has to 0 to n Tag class instances.

- **Keyword.**

  This class represents the keywords a shot is associated to and the keywords the Glossary is composed of. A Keyword has 1 to n Bibliography class instances.

- **Tag.**

  This class represents each of the tags associated to the different components of a diagram.

- **Bibliography.**

  This class represents each of the bibliography references associated to a Keyword.

- **Notes.**

This class is used to give a text explanation of all the film editing concepts explained through a shot. It contains 0 to n links of the two existing link types: Keyword and Diagram link.

- **NotesKeywordLink.**

This class represents each of the links contained in the Notes of a shot that refer to film editing and style keywords.

- **NotesDiagramLink.**

This class represents each of the links contained in the Notes of a shot that make reference to a previously represented shot. The click of this type of links produces a Diagram to be displayed in the Diagram Panel.

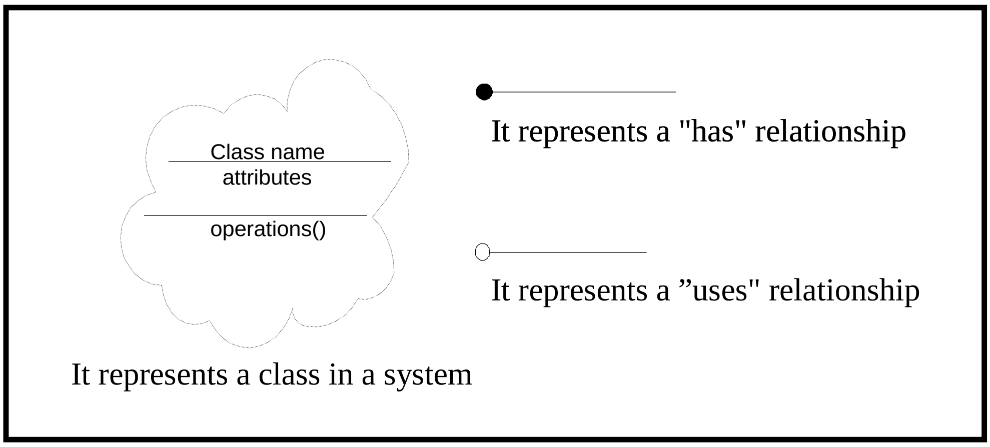


FIGURE 7
Booch Object Oriented Methodology Notation

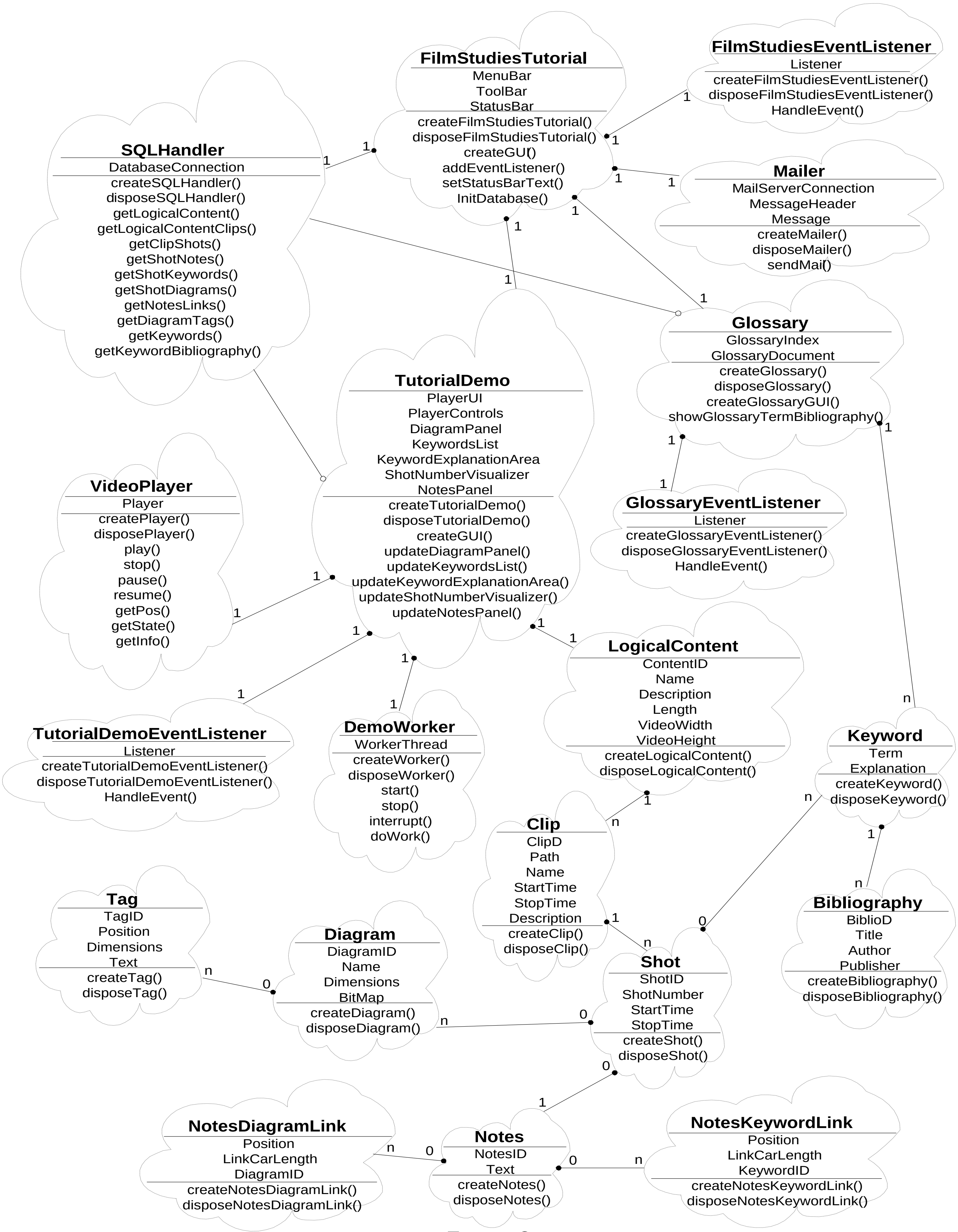


FIGURE 8
Film Studies Tutorial Class Diagram

3.4.2. Flows of interaction between the user and the system.

Due to the rich and complex range of interactivity facilities that was identified during the System Requirements capture, it was deemed necessary to do a thorough study of how interactivity manifests in the proposed system. For that purpose, Figures 9 and 10 were produced. These diagrams define the flow control of the Film Studies Tutorial, by synthesising the different stages the Tutorial passes through when user interaction occurs. These figures were very helpful when the programming phase of this project took place.

Figure 9 explains the different stages the system goes through once the Film Studies Tutorial is started. In first place, the GUI of the Tutorial is created and connections with the two Media Servers are established. Then the system enters in a loop were events generated by the user interactions are handled. Remember this system was conceived as being composed by a Tabbed Pane containing the Tutorial Demo Panel and the Glossary Panel (see Figure 5). Events occurring in these two panels are handled independently as it can be seen in Figure 9. When the user decides to finish the Film Studies Tutorial execution, the connections with the two servers are closed.

Figure 10 explains in greater detail the flow control when the Tutorial Demo Panel is active. Special attention should be paid to the diagram's left hand side, to the Diagram Area between Play Tutorial Film and Pause Tutorial Film. The two parallel lines mean the actions carried out inside them are undertaken in parallel. So, as it can be observed in Figure 10, while the frames of a shot are retrieved and rendered, the keywords and diagrams associated to it have also to be retrieved and displayed. Note as well the complicated set of operations that have to be carried out when the demo is interrupted by an event, see right hand side of the diagram. Observer that when the Tutorial Demo is running and an event takes place, a test is necessary to check the kind of event that occurred and determine if after the event is handled the Demo execution should be resumed or not. If the Demo is not running, the processing involved when an event happens is much easier.

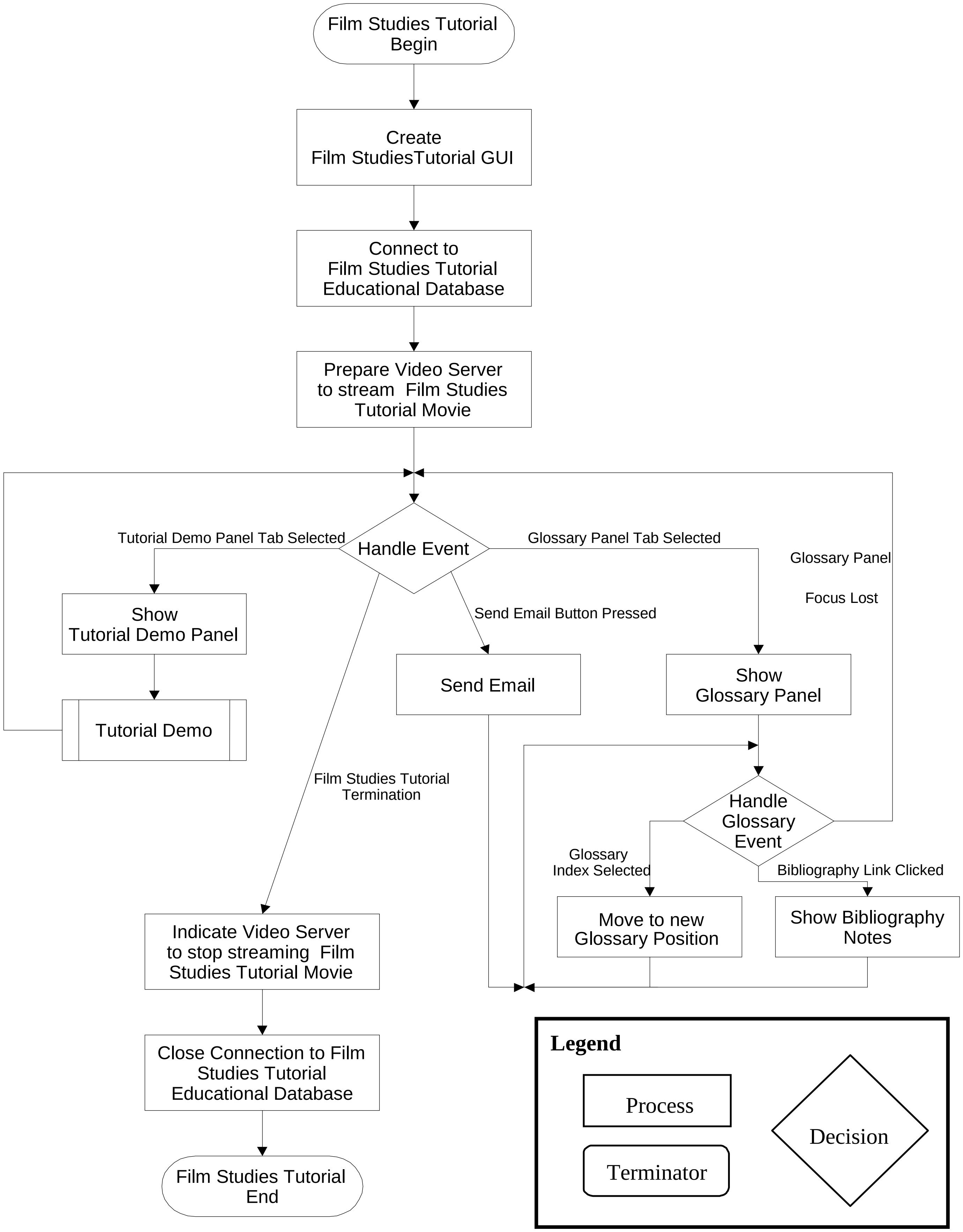


FIGURE 9
Film Studies Tutorial Flow Diagram

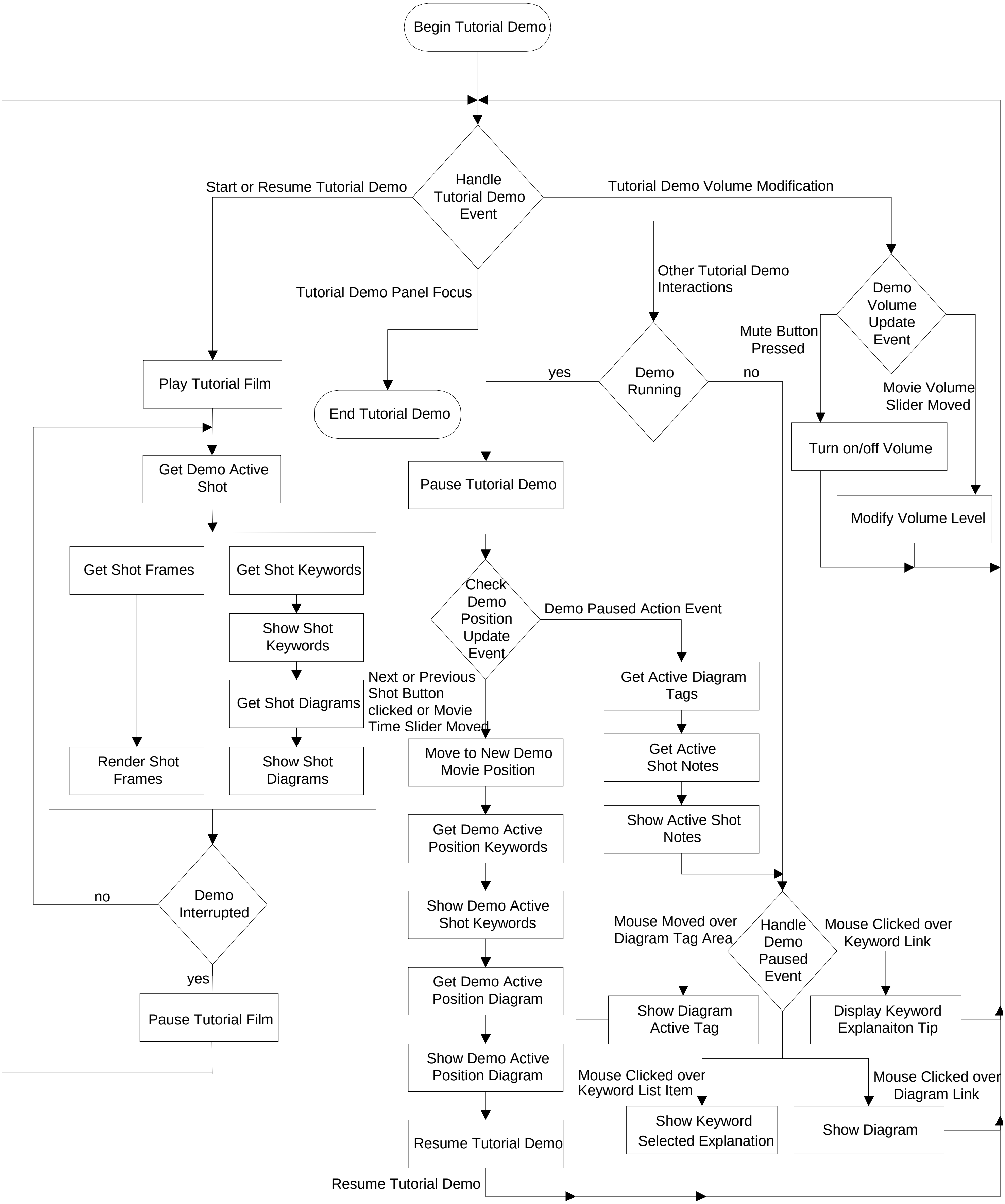


FIGURE 10
Tutorial Demo Flow Diagram

3.4.3. Concurrency Specification and Thread Control

While undertaking the Object oriented design of the system, it was understood the most complicated part of this system implementation would be the synchronisation of the VideoPlayer and DemoWorker objects execution (see Figure 8). The VideoPlayer Object duty is to carry out the displaying of the movie whereas the DemoWorker Object needs to track the Shot in which the film is in and for each shot display the Diagrams and keywords associated to it. There can be several Diagrams for a given Shot, and therefore the requirement of the system is to produce a simple kind of animation with them, displaying each of the diagrams for a specified time, that corresponds always with the time elapsed between two camera movements.

In order to understand this process better, the concurrent execution of the VideoPlayer process and the DemoWorker process was specified using the CSP (Communicating Sequential Processes) notation, encountered at [Mett-96].

```
TUTORIAL_DEMO_ON = PLAYER_ON || DEMO_WORKER_ON

PLAYER_ON =
  player.getShot → player.getShotFrames → (FramesIndex := 0);
  ((FramesIndex < NumShotFrames) *
   (playerUI.showCurrentFrame → FramesIndex := FramesIndex+1)
  ); PLAYER_ON

DEMO_WORKER_ON =
 ( player.getShot → Database.getShotKeywords → Database.getShotDiagrams →
   ShotNumberDisplay.showShotNumber → KeywordsList.showShotKeywords  →
   DIAGRAM_ANIMATION
 ); DEMO_WORKER_ON

DIAGRAM_ANIMATION =
  CurrentDiagram := 0;
  ( (currentDiagram < NumShotDiagrams) *
    (Database.getCurrentDiagram → DiagramPanel.showCurrentDiagram →
      ( (CurrentDiagramStopTime  < player.getPos) *
        (MovieTimeSlider.updateToCurrentPos)
      ); currentDiagram := currentDiagram+1
    )
  )
```

It is important to note the specification of the process PLAYER_ON, it is only an approximation of what actually happens when the Video Player Object is playing video. The implementation of this object is hidden from the programmer, it is provided within Oracle Video Java Library, and therefore the only thing the programmer knows is that there is a thread that carries out the rendering of film frames. In contrast, all the steps specified in the CSP notation of the processes DEMO_WORKER_ON and DIAGRAM_ANIMATION were fiercefully followed during the system implementation.

As it is seen in the specification of TUTORIAL_DEMO_ON process, when the Tutorial Demo is running two recursive processes (or threads) are executed in paralell (||), DEMO_WORKER_ON and PLAYER_ON.

Due to the need of displaying the movie withouth any glitching, the programmer should assign maximum priority to the PLAYER_ON process. This is in fact, the only manipulation the programmer is able to do over this process, because as it was mentioned, its implementation is hidden within the Player class of the OVC Java Library.

The DEMO_WORKER_ON recursive process has to be synchronised with the execution of the process PLAYER_ON. A main requirement is that the execution of the DEMO_WORKER_ON must not delay the PLAYER_ON process. The DEMO_WORKER_ON process although having to be concurrently executed with the PLAYER_ON one, should be assigned a lower priority in order to maximise the CPU resources allocated to PLAYER_ON process. It should be realised the vast amount of processing involved by DEMO_WORKER_ON process. The operations that have to be carried out by this process are:

1. Track the beginning of a new shot.
2. Obtain the keywords associated with the active shot from the Educational Database.
3. Obtain the diagrams associated with the active shot from the Educational Database.
4. Show the active shot number in the Shot Number Display element of the Tutorial Demo GUI.
5. Display the set of Keywords associated to the given shot in the Keyword List element of the Tutorial Demo GUI.
6. Run an animation with the set of diagrams associated to the active shot, displaying those images in the Diagram Panel contained in the Tutorial Demo GUI. It is necessary to display a new Diagram for each different camera movement produced inside a shot. Therefore the programmer has to keep track of the position of the player within the current shot in order to change the active diagram when necessary.
7. Update the Movie Time Slider Bar indicating the film position during the shot display time.

3.4.4. Educational Database Design

This section describes the Design of Database created to associate Educational Data to the encoded video contained in the MDS of OVS. The database design carried out was divided in two stages:

1. Develop an Entity Relationship Diagram in Chen Notation representing the main entities of the proposed system.

2. Derive the Relational Database Schema associated to the ERD and proceed to its normalisation in 3rd Normal Form to obtain the Database Tables.

It is convenient to highlight that the actual implementation of the Educational Database was carried out by Karina Chong, another MSc student in the Department of Computer Science, University of Essex, in charge of the Server side of the system [Chong-98]. She implemented the Database using SQL scripts and produced an Editing Tool to load data in. The author of this work limited to collaborate with her in the Database Design. It was, however, thought that an overall description of this Database Design would be helpful for a thorough understanding of the client system developed by the author. Note the second stage of this database design is found in "Appendix B: Relational Database Schema Derivation".

3.4.4.1. Entity Relationship Diagram of the Educational Database

Figure 12 shows the main entities identified in the Film Studies Tutorial System, as well as the relationships that exist among them. In what follows a detailed explanation of all the entities as well as complex relationships, (M:N) cardinality type relationships, is given.

The entities identified are:

- **Logical Content**.
  The *Logical Content* entity has been named in this way after the OVS way of denoting a collection of titles, each representing a series of parts of physical content files (clip). The Film Studies Tutorial System uses a single *Logical Content* created in the OVS. It is described in the RDBMS by its identifier, *ContentID*, its *name*, a *description* of its contents, the time duration of it (*length*) and the Dimensions of the frame size employed in its encoding (*Frame Width* and *Frame Height*).

- **Clip**.
  The *Logical Content* associated to the Film Studies Tutorial System *contains* a collection of clips. In OVS terminology a clip maps to a specific start and stop position within a single physical file. However, in order to ease the implementation of the system, each clip was considered to represent an entire physical file. The DBMS holds for each *Clip*, an identifier, *ClipID*, the *path* of the directory inside the MDS where the physical file has been stored, the *name* of physical file the clip refers to and a *description* explaining the clip contents.

- **Shot.**

Each *Clip* is *divided into* several shots. A *Shot* is a cinema term that refers to each of the basic blocks in which a movie is built. For each shot a shot identifier, *ShotID*, the *shot number* within the clip, and the *start* and *stop time* with regards to the *start time* of the containing clip is stored.

- **Notes.**
  Each shot *is explained in* zero or one *Notes* text. Each *Notes* is described in the Database by a notes identifier, *NotesID*, and a *text*. It is possible that several shots have exactly the same *Notes* text.

- **Diagram.**
  Each *Shot is represented by* a set of *Diagrams*. Each *Diagram* is stored as an identifier, *DiagramID*, the *name* of the Diagram, the dimensions of it, its *width* and *height*, and the *BitMap*, the binary representation of the *Diagram* content.

- **Keyword.**
  Each *Shot gives explanation to* zero or more *Keywords*. A *Keyword* is represented with the attributes *Term*, the Keyword itself, and the *Explanation* of it.

- **Tag.**
  A *Diagram has* a set of *Tags* associated to it. It could happen that a Diagram has no *Tag* related to it. A *Tag* is described by a Tag Identifier, *TagID*, a *text* field and the area in the *Diagram* over which the *Tag* should be displayed when the mouse pointer is moved in. The area of display of the *Tag* is specified by the attributes Position, denoting the Top Left Corner (*X*, *Y*) and Dimension, the *width* and *height* of the area.

- **Bibliography.**
  Each *Keyword is found at* a set of *Bibliography* notes. A *Bibliography* note is denoted by a Bibliography Identifier, *BiblioID*, the *title* of the bibliography reference, the *author* and the *publisher*.

The complex relationships denoting N:M cardinality are:

- **“contains”.**
  This relationship was created to handle the case in which a *Clip* could be contained more than once in the *Logical Content* associated to the Film Studies Tutorial System. Moreover it is necessary to keep track of the position or positions within the *Logical Content* in which a *Clip* is included. The attributes *Start Time* and *Stop Time* are used for this purpose.

- **“is represented by”.**
  This N:M relationship was created because the same *Diagram* can be reused for various different *Shots*. Within a given Shot the Film Studies Tutorial System requires to know when inside a given Shot a new Diagram must be displayed. The *Start Time* attribute, therefore, indicates at what instant in the Shot Displaying Time the *Diagram* should be displayed. A Diagram is displayed until a new Diagram *Start Time* in a given Shot is reached or the first Diagram corresponding to a new Shot is displayed. The *Start Time* attribute represents the absolute time from the beginning of the clip the shot it refers belongs to.

- **"contains link to".**
  This N:M relationship is used to enable the inclusion of *links* in the *Notes* text associated to a *Shot*. Two different kinds of *links* can take place in Notes text. The first one, Keyword Link, allows the user to see an explanation of a *Keyword* when the link is clicked. This kind of link is possible through the relationship established between Notes-Keyword entities. The other type of links, Diagram Link, allows the user to invoke the display of a Diagram when the link is clicked. This type of link is possible through the relationship between Notes-Diagram. The Film Studies Tutorial System knows where inside the Notes text the links have to be displayed by means of the attributes of this relationship, *Position* and *Length*. *Position* indicates in which character position from the beginning of the text a link starts, whereas *Length* indicates, how many characters the link is formed of.

- **"is found at".**
  This N:M relationship is created to enable the same bibliography note to be used by several Keywords. The *Page Number* attribute of this relationship allows the specification of the page number within a bibliography reference in which a keyword can be found.

"Appendix B: Relational Database Schema Derivation", explains how the Relational Database Schema shown in Figure 11 was obtained form the ERD displayed at Figure 12.

Logical Content (ContentID, Name, Description, Length, VideoWidth, VideoHeight)
LogicalContentContainsClip (*ContentID, ClipID*, StartTime, StopTime)
Clip (ClipID, PathName, Description)
Shot (ShotID, ShotNumber, StartTime, StopTime, *ClipID*)
DiagramRepresentingShot (*ShotID, DiagramID*, StartTime)
Diagram (DiagramID, Name, Width, Height, Bitmap)
Tag (TagID, X, Y, Width, Height, Text, *DiagramID*)
Notes (NotesID, Text, *ShotID*)
NotesDiagramLink (*NotesID, DiagramID*, Position, Length)
NotesKeywordLink (*NotesID, Term*, Position, Length)
Keyword (Term, Explanation)
KeywordFoundAtBibliography (*Term, BiblioID*, PageNumber)
Bibliography (BiblioID, Title, Author, Publisher)
ShotExplainsKeyword (*ShotID, Term*)

FIGURE 11
Relational Database Schema

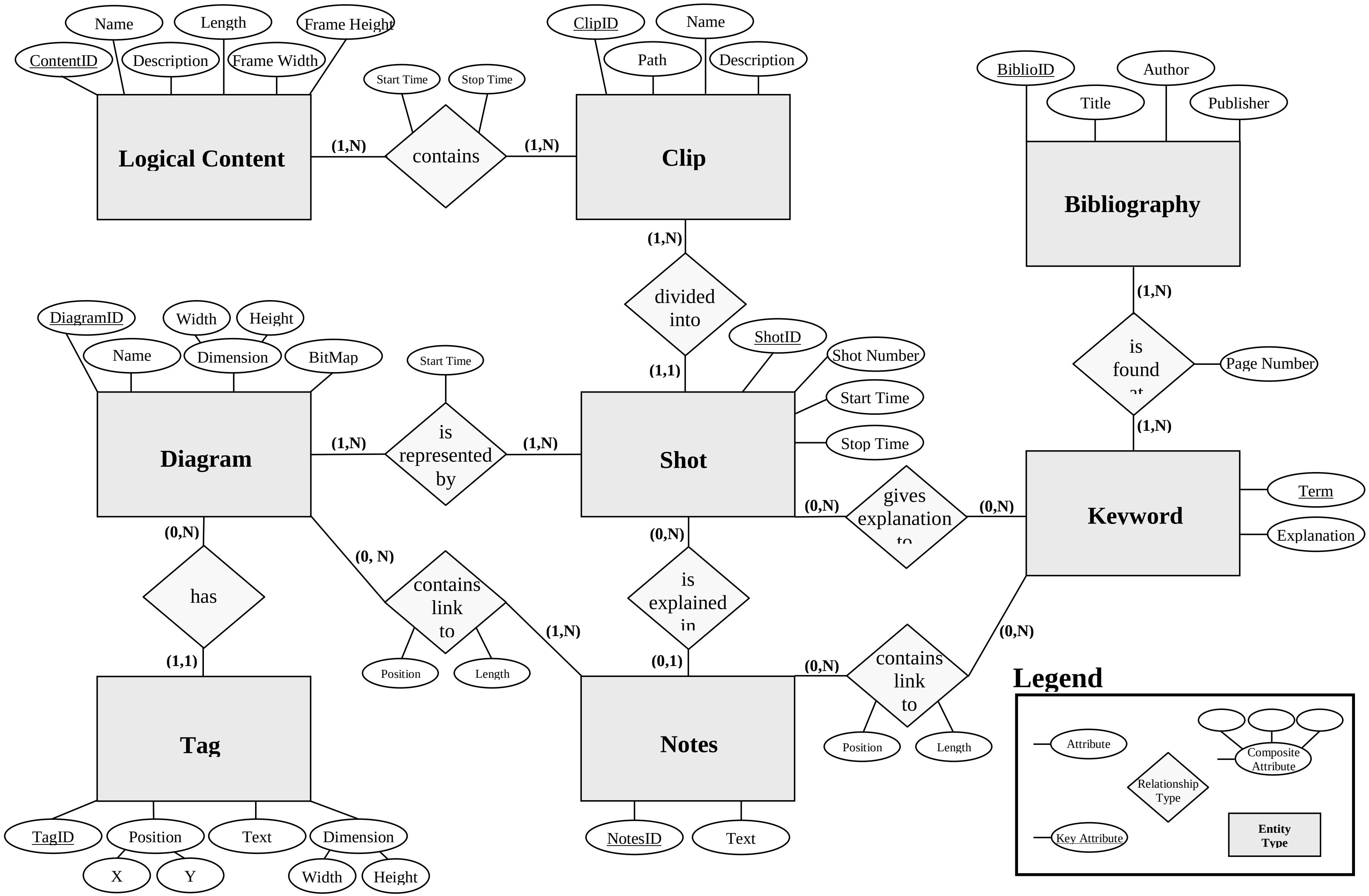


FIGURE 12
Film Studies Tutorial Entity Relationship Diagram

4. Implementation Issues.

Once the Design phase of the Film Studies Tutorial was concluded the next stage was to proceed to its implementation. This chapter makes in first place, an overview of the technological background required to understand how the system was implemented. In second place, it describes the most significant problems that were identified in the system programming and the solutions that were adopted.

4.1. Technical background

In this section, an introduction to the most relevant features of the Java programming language that were used in the Film Studies Tutorial implementation is given. Java Multithreaded Programming and Java Foundation Classes [JavaSoft-98a] are briefly described.

4.1.1. Multithreaded programming in Java

As it was seen in the System Design (Section 3.4.3), the Film Studies Tutorial needs several processes (or threads) to be executed in parallel in order to produce the synchronisation of the data coming from the two different Information Servers. The means Java provides to achieve this effect is Threads usage. Appendix C makes an overview of Multi-threaded programming in general. This section focuses its attention on how Java performs multitasking.

MULTITHREADING DEFINITION

A *multithreaded program* is an executable entity containing two or more parts that run concurrently [Gosling-96]. Each part of such a program is called a *thread*, and each thread defines a separate path of execution. The main advantage of multithreading is that allows writing very efficient programs that make maximum use of CPU, because idle time can be kept to a minimum. Java provides the `Thread` object to support multithreading.

THREAD PRIORITIES

Thread priorities are used to decide when to switch from one running thread to the next, i.e. when a *context switch* takes place. Java assigns a priority (`setPriority` method of `Thread` Class) to each thread that determines how that thread should be treated with respect to the others. The rules that determine when a context switch takes place are:

- A thread can voluntarily relinquish control. The highest priority thread that is ready to run is given the CPU.

- A thread can be preempted by a higher-priority thread. This is call preemptive multitasking.

When two threads with the same priority are competing for CPU cycles, the situation is a bit complicated. Windows 95 Java threads of equal priority are time-sliced in a round-robin fashion. However, for other operating systems, such as Solaris 2.x, threads of equal priority must voluntarily yield control. It must be noted, therefore, that implementations of Java have radically different behaviour when it comes to scheduling.

When programming threads special care has to be paid to tasks that are CPU intensive. Such threads will dominate the CPU. For these types of threads, it will be recommended to yield control occasionally so that other threads can run.

THREADS SYNCHRONISATION

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time, i.e. they need synchronisation. In Java the model of synchronisation employed is the monitor. The monitor is a control mechanism that was first defined by C.A.R. Hoare in his famous paper of 1974 [Hoare-74]. A *monitor* is an object that is used as a mutually exclusive lock, or *mutex*. Only one thread can own a monitor at a given time. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor. Java uses the `synchronized` keyword to synchronise the access to methods and objects. Please refer to for more information to [Deitel-97].

A special type of error that can happen when monitors are used for synchronisation is called *deadlock*. Deadlock occurs when two threads have a circular dependency on a pair on synchronised objects.

A given library function is said to be **thread-safe** when it has been implemented in such a manner that it can be executed by multiple concurrent threads of execution.
The major problem with explicitly programmed thread support, as in Java, is that is always difficult to be sure the locks needed have been acquired and released again at the right time. If a method returns prematurely, for instance, or if an exception is raised, for another instance, the lock that was imposed would not be released and deadlock would be the result.

THREAD LIFE CYCLE IN A JAVA PROGRAM

The goal of this point is to explain the stages a thread passes through since a new Thread type object is born by means of `new Thread()`Java statement till it dies. It is impossible to explain how to deal with threads in Java in such a small space, but with this brief description it was just pretended to give an idea on how it is done. Please refer either to [Cornell-96], [Deitel-97] or the source code (Appendix F) of the application to see actual examples.

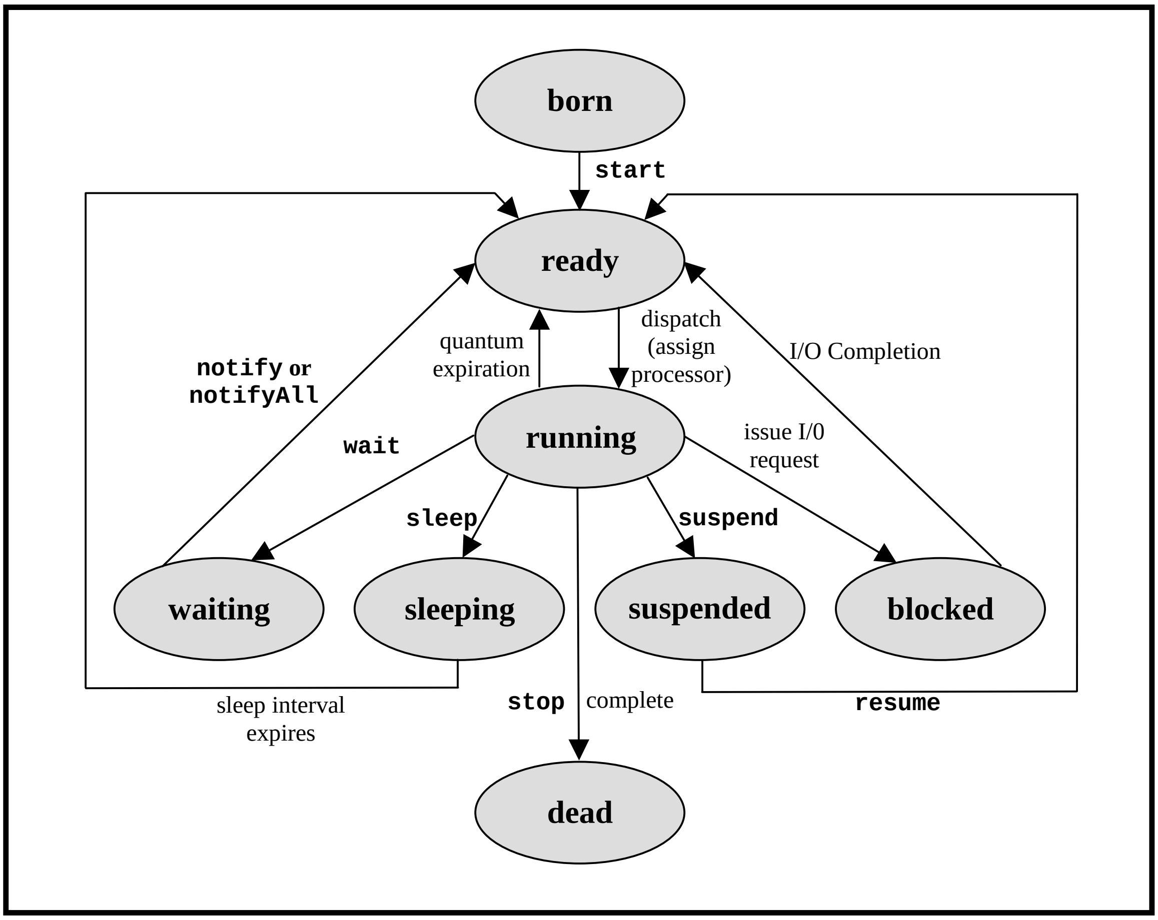


FIGURE 13
Thread Life Cycle in a Java Program

At any time a thread is said to be in one of the several thread states shown above. If the thread has just been created is in the *born* state. The thread remains in this state until the thread's `start` method is called; this causes the thread to enter in the *ready* state and to move its point of execution to the beginning of its **`run`** method. The highest priority ready thread enters the *running* state when the system assigns the CPU to the thread. When a thread is running is simply executing the code contained in its `run` method.

A running thread enters the *blocked* state when the thread issues an input/output request. A blocked thread becomes ready when the I/O operation it is waiting for completes. A blocked thread does not consume CPU resources even if they are available.

When a *running* thread invokes the `sleep` method, that thread enters the sleeping state. In a sleeping state a thread yields control for an specified period of time, not consuming CPU resources during that period. A sleeping thread becomes ready after the designated sleep time expires.

When a running thread's `suspend` method is called, that thread enters the *suspended* state. A thread is in its suspended state when its execution is stopped momentanealy at a given execution point in its **`run`** method. A suspended thread cannot use the CPU even if it is available untils its `resume` method is called by other thread. A resumed thread continues its execution within its **`run`** method at the point in which it was suspended.

When a running thread calls `wait` the thread enters a *waiting* state where it waits in a queue associated with the particular object on which `wait` was called. The first thread in the wait queue for a particular object becomes ready on a call to `notify` issued by another thread. Every thread in the wait queue for a given object becomes ready on a call to `notifyAll`.

A thread enters the dead state when its `run` method completes or its `stop` method is called.

4.1.2. Java Foundation Classes: A unified UI Framework.

A very important part of the work done to implement the Film Studies Tutorial was to provide an outstanding GUI that would be highly user-friendly to the student and that would encourage its use. The Abstract Windowing Toolkit (AWT) API, the default GUI toolkit provided by Java, doesn't provide a very rich range of GUI visual components, such as Tabbed Panes, Progress Bar, Sliding Bars, etc. If sophisticated graphic user interface elements want to be used, the programmer is forced to code his own GUI classes. Fortunately, the appearance of the Swing API contained in the recently released Java Foundation Classes was the answer to the GUI needs of this project.

WHAT ARE THE JAVA FOUNDATION CLASSES?

The Java Developer's Kit (JDK) release 1.1 introduced the first installation of the most significant cross-platform graphical user interface technology since the advent of the windowing system: the Java Foundation Classes(JFC) [Nelson-98].

The Java Foundation Classes extend the original Abstract Windowing Toolkit[4] (AWT) by adding a comprehensive set of graphical user interface class libraries. Java and the AWT freed developers from being tied to a specific or proprietary platform for GUI application development and deployment. However, sophisticated applications for the enterprise could not still be produced due to the lack of a rich range of visual components. JFC solves this issue.

The Java Foundation Classes empowers developers on all platforms, and brings the future of portable GUI development here today: Write Once, Run Anywhere. It is composed of the following APIs:

- **"Swing" GUI Components**
  These components are written in the Java language, without native window-system specific code. This facilitates a customisable look and feel without relying on the native windowing system, and simplifies the deployment of applications.

- **Accessibility API**
  The Accessibility API is an interface that provides assistive technologies such as screen magnifiers and audible text readers. These technologies are designed to help people with disabilities to interact and communicate easily with GUI components.

- **Java 2D API**
  Java 2D is a set of classes for advanced 2D graphics and imaging. Java 2D supplies Java applications with many different paint styles, mechanisms for defining complex shapes, and classes and methods for fine-tuning the rendering process.

- **Drag and Drop API**

[4] Java default Graphic User Interface Toolkit.

Drag and Drop is a technology that enables data transfer across Java and native applications, across Java applications, and within a single Java application.

SWING

From an architectural point of view, the Swing component set extends the Abstract Windowing Toolkit (AWT) [Fowler-98]. Swing is a graphical user-interface (GUI) toolkit formed by a set of object-oriented classes designed to help programmers write graphical user interfaces for their Java programs.

Swing comes with a special cross-platform look and feel, named *metal*[5] or the Java look and feel, that always presents a set of components with the same appearance and behaviour, no matter what operating system the application is running on. With Swing your applications can not only run in all the platforms but they can also provide you the same look and feel (way of interaction with them) no matter which platform you are working with. However, you can still make a program developed with Swing look and feel specifically for the computer platform it resides on, without any kind of modification in the source code [Andrews-98].

SWING VS. AWT

The approach followed by AWT to implement a portable GUI environment is to layer an API or class library on top of the native toolkit. Also known as the “peer” model, each Java component class “wraps” the peer or native implementation.

The AWT greatest success was to shield developers from each platform idiosyncrasies and yet produce 100% portable code with a native look and feel. However, GUI developers soon complained that the AWT’s peer model was too restrictive in rendering and event handling and that it was difficult to extend or override different aspects of a component.

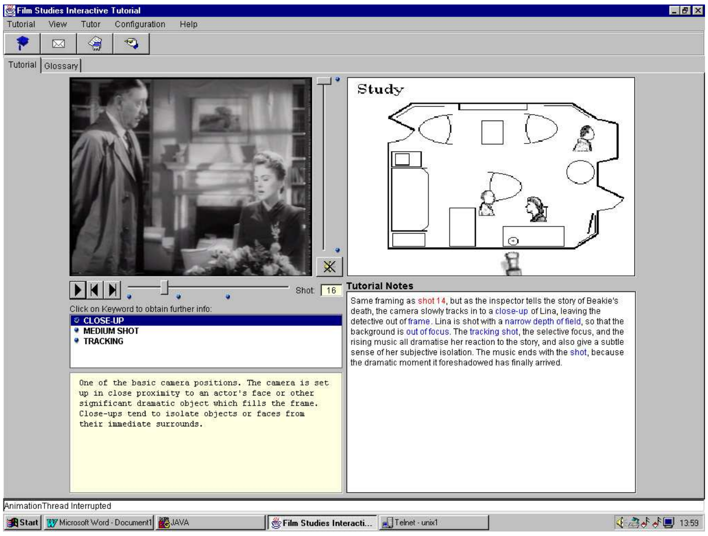
The approach followed by Swing to implement a portable GUI environment is to provide a single toolkit and then emulate the native look and feel on each platform.

Swing set contains far more components than the pre-Swing AWT toolkit did, and Swing components have many new features and capabilities that old-style AWT components did not have. For instance, with Swing a button can have associated a picture apart from the typical text. The most important difference between Swing and the pre-Swing AWT toolkit is that Swing components don't borrow any native code from the platforms on which they run. In other words, Swing does not use any platform-specific implementations or “peer”. Instead, Swing creates its components using pluggable look-and-feel (PL&F) modules that are written from scratch and don't use any native code at all.

[5] The Film Studies Tutorial was implemented to have by default “*metal*” look and feel. However, the look and feel of the Tutorial can be easily changed through a Menu Choice.

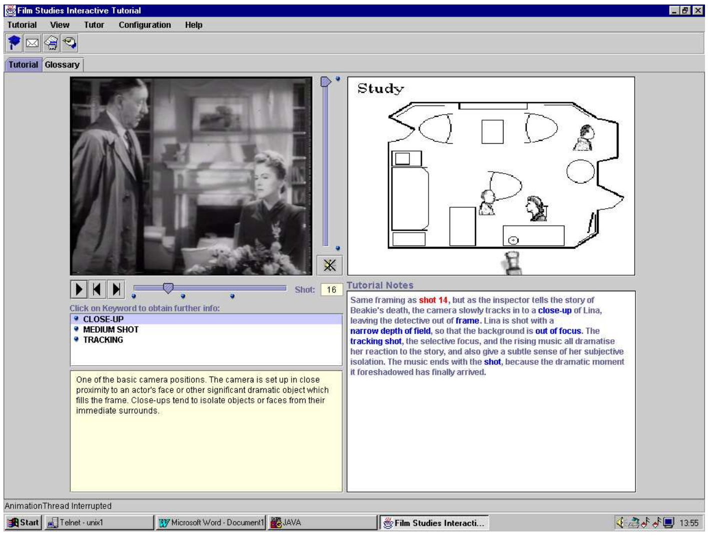
The most outstanding feature of Swing is its support of a platform-independent pluggable look and feel. This feature gives a user the ability to switch the look and feel of an application dynamically, at runtime, while your program is being executed. Swing allows an application running for example in a Windows platform to have the look and feel of a Motif environment, or to have the platform independent Java look and feel called *metal*. Change from one environment look and feel to the other, can be easily produced just by selecting at run-time a menu choice.

Figure 14 shows the result of implementing this feature in the Film Studies Tutorial. This feature was implemented through a Menu Choice in the Menu View that allows selection among Metal, Motif or Windows Look and Feel.



Window Look and feel

Java cross-platform Look and Feel “Metal”



Motif Look and feel

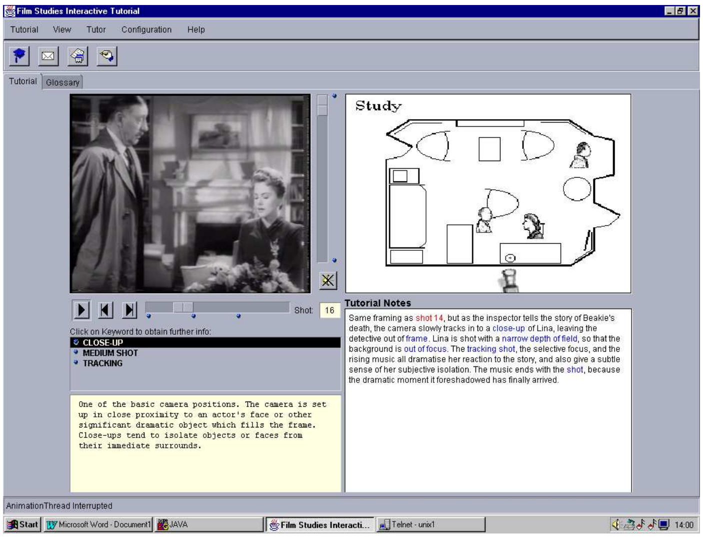


FIGURE 14

Swing Pluggable Look and Feel

4.2. Implementation Problems and Solutions

In this section a tour is made through all the problems faced during the implementation of the system. The goal of this section is to summarise problems identified in the development of this Interactive Distance Learning system and to discuss the solutions adopted. It is hoped this section will serve as reference for future developers of systems of this nature to reduce the time invested in their software implementations.

- **Size of the Video Panel and Diagram Panel**

  An important conclusion that can be drawn when the design of a multimedia system, as the one proposed, is faced is that the GUI framework built to integrate the different media will have to be designed on the basis of the special features of the data. If the GUI is first designed and later the media data (video and graphics) has to be accommodated to them numerous problems will be found.

  This situation arose when the programming of the Tutorial System was initiated. A video size that would suit nicely with the rest of the GUI components of the system was being looking for. It was even initially considered to resize the GUI component (panel) in which the video was displayed. However, we overlooked one of the special features of MPEG-1 (PAL) encoded video, the video format used in this Tutorial. This feature is that its frame size is 352*288 pixels. Displaying the video in a panel whose size was different to this resulted in poorer quality video because a scaling procedure had to be applied to the video frames. The solution was therefore to develop a Video Panel of fixed size, 352*288 pixels, in order to avoid the mentioned scaling algorithm.

  A similar problem occurred when visualising the GIF images representing the Diagrams that give explanation of how the movie is recorded. Once again was obviated that if images have to be displayed in a size that is not their actual one, a scaling algorithm has to be applied. This procedure usually results in a lower image quality unless a high CPU consuming smooth scaling method is applied. Thus, in order to produce the most lightweight application it was decided to make all the diagrams of the same size and display them in a fixed size panel of those dimensions.

- **GUI development challenges**

  As mentioned in the above point, the creation of the application GUI began with a main constraint, the size of the displaying areas of video and diagrams had to be fixed. A considerable amount of time was invested to achieve this effect in the user interface. The solution to this problem was the new Layout Manager, *BoxLayout* that comes with the last release of the Swing GUI Components of JFC [Nelson-98].

  Once the components of the GUI were properly arranged, the problem passed to the development of some special GUI components that were not found neither on the default AWT (Abstract Windowing Toolkit) nor on the Swing Toolkit. The

Film Tutorial proposed demanded levels of interactivity that could be satisfied only by the programmer implementation of new GUI classes. In what follows a an explanation of two new GUI that had to be created is given:

- *Notes Panel containing text with Diagram and Keyword Links.*

  This GUI component required was difficult to implement because there was not found any built-in GUI component in Java that allows text display with click-able words. A new class, called *TextAreaWithLinks* (see Appendix F: Source Code), had to be developed to achieve the required effect. The artifice employed was to implement the Notes text as a set of Labels (for normal text) and buttons without borders that looked like normal labels (for the links).

- *Pop-Up Tip Window appearing after user click on Keyword Link.*

  The component required was a Windows-help like component that appears when the user clicks over a Keyword Link and disappears when the user clicks over the component, or somewhere else making the element lose the focus. The KeywordTip class was created to solve this problem. Its implementation is based in a Java Mouse Listener.

- **Synchronisation between Video Content and Educational Data.**

When the media to synchronise with the video played is not highly resource demanding in terms of network communication, CPU and memory, synchronisation is relatively easy. For example, if text has to be synchronised with video, text can be retrieved from a database and be displayed soon enough to appear to the user that the text is synchronised with the video position. A small buffer is used sometimes to keep the text to be displayed subsequently. However, if images have to be synchronised with video, they definitely need to be retrieved in advance and loaded to memory due to their bigger size and CPU resources needed when rendered.

Moreover, another problem that occurs when synchronising media with video is that the video time does not necessarily coincide with the system time. A second of video does not necessarily elapses one system second, but usually a bit more. Therefore the system time can not be taken as the “wall clock” to synchronise video and the other educational media data but the video time should be considered instead. It is worth remembering that the educational data associated to the video has timestamps that indicate at what instant during the video playing that data should be shown.

Synchronisation in the Film Studies Tutorial was achieved by creating a thread, *DemoWorker* (see Appendix F: Source Code) that uses a polling technique in which the position of the movie is tracked with certain time frequency. Taking into consideration that a PAL MPEG-1 encoded film needs to be played at a rate of 25 frames per second, to achieve good quality video, it was decided to enquiry the movie player position every 10 frames time (400 ms). The DemoWorker process can be send to *sleep* for 400 ms, and in this way release the CPU for being used by other threads in the mean time. This time was experienced as a good trade-off between system performance and synchronisation.

- **Mail System.**

To implement the mail system, a thorough understanding of the SMTP [Postel-82] protocol was needed and of Java sockets programming [Deitel-97]. The Mailer class in the source code (Appendix F) corresponds to the email system implementation.

- **Application Performance**

The main problem found when the first running version of the system was completed is that despite all the functional requirements of the system were met, the performance of the Demo distanced much from appropriate. Its main inconvenience was that due to all the processing required during the display of the educational data associated to the video content, the movie playback was interrupted occasionally and produced an unpleasant jerking effect.

The solution thought was to give the thread (Player object) in charge of playing the movie the highest system priority and to the thread in charge of synchronising the educational data (DemoWorker) with the video content minimum priority. Moreover all the stages the DemoWorker thread passes through where high processing is involved were re-programmed in a way that would always allow the CPU to be yielded to the Player thread to minimise the glitching effect.

Despite all the effort paid in this issue, the author still could not avoid this problem completely. Java thread programming was studied carefully in order to provide the most optimum solution, however the final result although rather good is not the one the author would have wished. The only reason it can explain this unresolved problem is that Java is interpreted and its performance still distances quite a lot from other compiled programming languages that produce native code. As indicated in the further work section, a compilation of the Java code to native code is suggested as a further performance improvement.

The approaches followed to reduce the time spent during each DemoWorker thread stage were:

- *Load to memory all the Educational Data needed to be displayed in real-time.* In the Tutorial Initialisation all the educational data that needs to be displayed concurrently with the video (keywords and Diagrams) is stored into memory. This results in a delay during the application initialisation. However, the DemoWorker thread does not have to access the database while the Demo is ON. This would produce the DemoWorker to be blocked until the database operation is completed. In the mean time the Player thread would continue rendering the video, making that when the database operation finally returns and the DemoWorker is assigned the CPU by the dispatcher the movie is now in a position where the educational data retrieved is not valid any more.

A more intelligent approach could have been taken by having in memory only the currently needed educational data and the one that will be needed in the following seconds of the demo. A low priority thread could have been employed to control the educational data buffered and retrieve new information when needed. Time limitation disallowed proceeding in this task and has been suggested as a further improvement of the system.

- *Use the MediaTracker to load all images to memory to save displaying time.* The MediaTracker class provided by the AWT API allows the programmer to control when an image has been completely loaded to memory. The retrieval of an image from disk in Java is done asynchronously and the usage of the MediaTracker class is the only way to ensure that when an image needs to be displayed has already been loaded into memory.

- *Prioritise Player Thread with respect to DemoWorker Thread.*
  As it has been explained, threads with higher priority receive more ‘quantum’s of CPU that the ones with lower priority. Therefore, with this assignation of higher priority to the thread in charge of displaying the movie most of the CPU time is invested in this processing.

❑ **Multi-thread programming and Swing.**

Several problems in the stability of the system were found as a consequence of combining multiple-threads and Swing GUI Components. Swing components bring many benefits in GUI development, as it has been commented, but have associated to them the following drawback:

“Once a Swing component has been realised[6]*, all code that might affect or depend on the state of that component should be executed in the event-dispatching thread*[7]*”* [Muller-98a]

The problem is that Swing Components Library is not thread-safe. A given library function is said to be **thread-safe** when it has been implemented in such a manner that it can be executed by multiple concurrent threads of execution. Swing was not implemented in this way because usually most post-initialisation GUI work occurs in the event-dispatching thread[7]. Once the GUI is visible, most programs are driven by events such as button actions or mouse clicks, which are always handled in the event-dispatching thread.

However, there are some situations in which programs need to perform non-event driven GUI work after the GUI is visible. For instance, the programs whose GUI must be updated as the result of non-AWT events. This is the case of the system proposed where the GUI must be modified when the Tutorial Demo is ON, by updating the Diagram Panel with new Diagrams, the Keyword List with new Keywords, the Movie Time Slider with new Video Positions, and displaying the

[6] A Swing component is realised when the component has been displayed.
[7] The event-dispatching thread is the thread that executes drawing and event-handling code.

new Shot number in the Shot Number Display. The problem identified was that if another thread like the DemoWorker in this system was updating Swing GUI components while an event occurred, two threads (SwingWorker and the event-dispatching thread) could have uncontrolled access to the same component. Due to the fact the methods in Swing components are not synchronised a *race condition* could be experienced and an unpredictable situation would be reached leading the program to crash in some occasions.

The SwingUtilities class provides two methods to help the programmer run GUI updating code in the event-dispatching thread and in this way solve the non-thread safe problem associated to Swing. See [Muller-98b] for more details. The two methods that can be applied for this purpose are:

- invokeLater(), requests that some code be executed in the event-dispatching thread. This method returns immediately, without waiting for the code to be executed.

- invokeAndWait(), acts like invokeLater(), except that this method waits for the code to be executed.

5. Testing & Experimental Results.

After the implementation of the system was concluded the important testing phase was initiated. Black Box and White Box Test techniques described in [Pressman-93] were employed for carrying out the tests. Black Box Testing techniques consist of testing the fulfilment of the specific functions the system was developed for. White Box testing, on the other hand, is concerned with ensuring the reliability of the system, even in the non-common situations the system might pass through.

Bearing in mind the real-time nature of the system, due to its real-time media needs and multithreading features, special tests had to be undertaken to control the synchronisation of both media and multiple thread execution. As result of this process, several performance measurements were made, the results of which are included in this section.

5.1. Functional Testing (Black Box Testing)

During this stage, the functional requirements of the system were tested. The focus of attention was to check that the functional requirements established at the beginning of the project were achieved after its implementation. Moreover, the tests carried out helped to identify:

1. Incorrect or overlooked functions.
2. GUI bugs.

Due to the time in which the development of the system took place, summer time, it was not possible to carry out these tests with the actual targets of the system, the students of the MA in Film Studies. However, staff of the Department of Art, History & Theory of the University of Essex, organisers of the MA in Film Studies, voluntarily offered to test the system and evaluate if the system fulfilled their initial requirements.

Its work was to test and evaluate the following three aspects of the application:

1. Test the integration of the different media elements employed in the Tutorial and evaluate the degree in which the educational purpose of the system was achieved through them.

2. Test the interactivity of the system and evaluate its appropriateness to assist the students in the learning process.

3. Test the email system provided by the system and evaluate its usefulness as a means to communicate the students with the course tutor.

The first aspect of this evaluation, the *integration of the different media elements*, was estimated as really outstanding. The synchronous and simultaneous visualisation of movie, diagram animation and keywords when the Tutorial Demo is running was considered as highly comprehensive. The suitable layout of Video Panel, Diagram Panel and Keywords List was highlighted as paramount to easily follow the Tutorial, despite the wide range of dynamic elements (Movie, Diagram Animation, Keywords) on it. Figure 15 shows the outlook of the Film Studies Tutorial Demo while running.

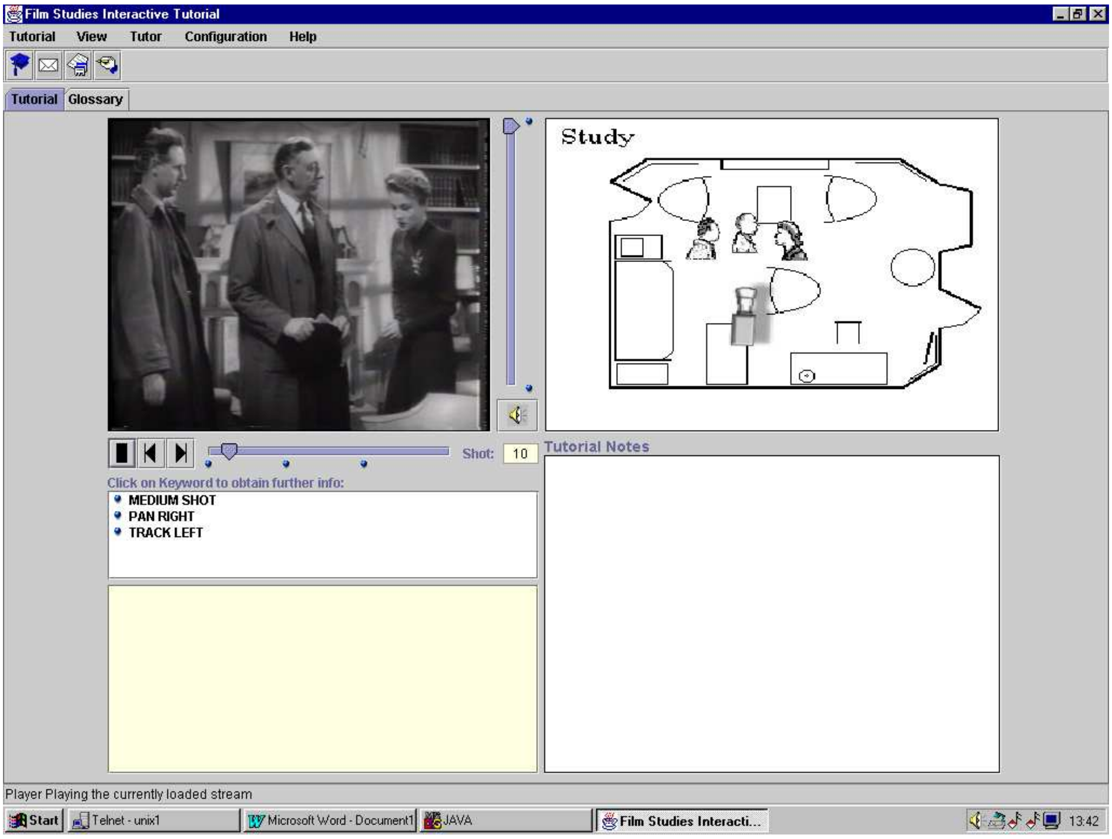


FIGURE 15
Film Studies Tutorial when Tutorial Demo is ON

The second aspect tested was the *interactivity of the system*. Film Studies Staff pinpointed the user friendliness and rich sort of interactivity facilities provided by the system. The Tip Tags (see Figure 16) that appear when the user moves the mouse over the different Film Studies Tutorial Controls (Player Controls, Tool Bar, etc.), were considered as very helpful to make the user prompt learn how to use the system. The rich sort of facilities provided for the full VCR-like control over the Demo Film (play/pause, next and previous shot, fast forward or rewind, Volume Control), not only fulfilled their requirements but exceeded their expectations. It was considered as very logical the capacity of allowing the user pause the application by clicking, the Diagram Panel, the Keywords List or the Player Controls.

Positive comments were received as well about the interactive facilities provided when the Tutorial is paused. The facility of receiving Keywords explanation whether through a click over a Keyword List Item or a highlighted keyword on the Tutorial Notes (see Figure 17) by means of the Keyword Explanation Area and a pop-up Tip Window respectively, were considered as very useful. The click facility over Diagram Links (see Figure 16) resulting in a new Diagram displayed in the Diagram Area to allow the comparison between the current shot and previous ones was deemed as highly instructive as well. Figure 16 and 17 show the situation in which the Tutorial Demo is stopped in two different instants inside the shot 14 of the Tutorial. Different operations are executed in each case.

FIGURE 16

Tutorial Demo paused, Diagram Link clicked and Mouse Moved over Resume button.

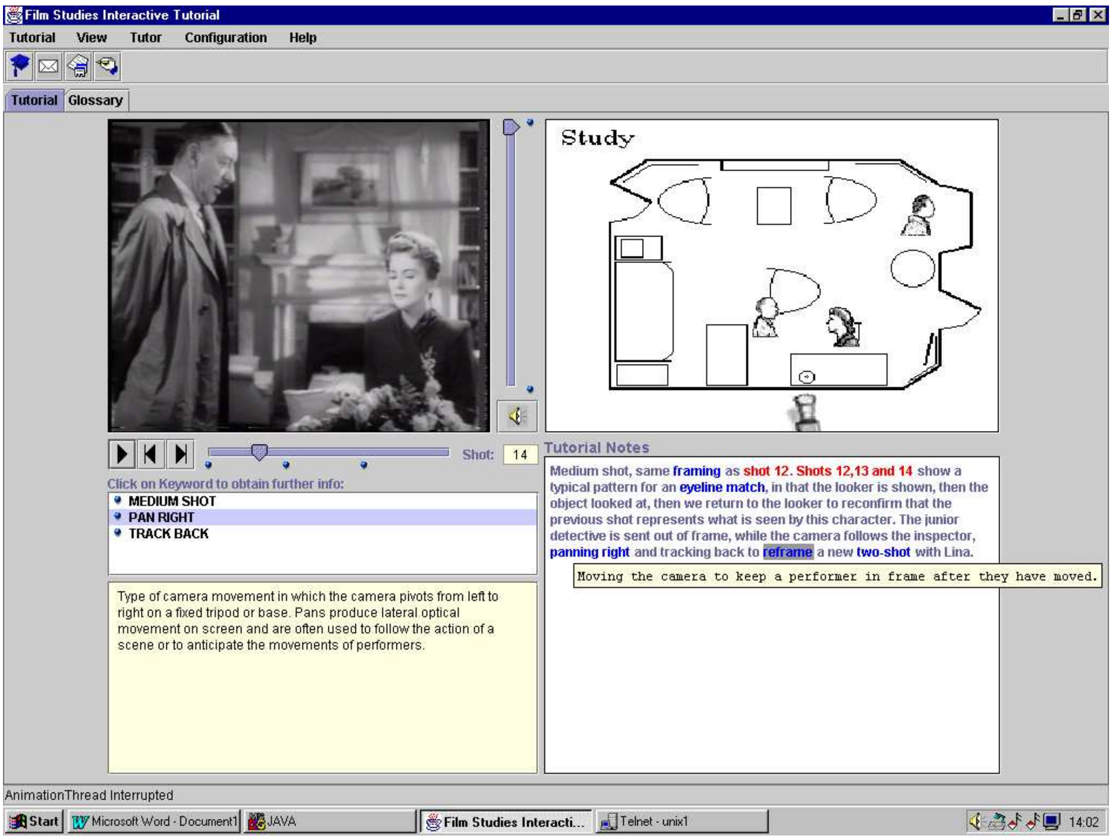


FIGURE 17

Tutorial Demo paused, Keyword List Item and Keyword Link clicked.

The Glossary and its quick access index facility was judged as very useful to help the student remember Film Editing terms and provide access to bibliography sources where further information about a term can be found. Figure 18 shows the situation in which the student has clicked the 'M' letter to have rapid access to all the Film Editing and Style terms that start by that letter.

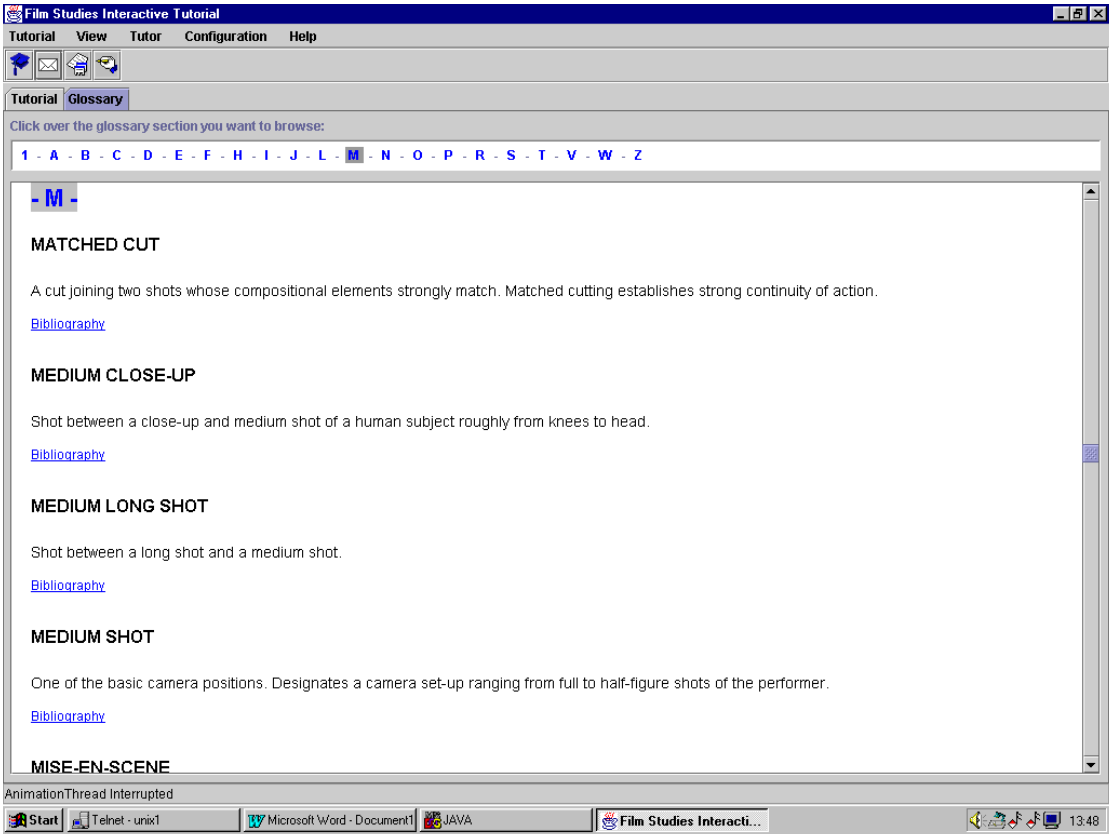


FIGURE 18

Glossary panel after user clicks over 'M' letter index.

Finally the usefulness of the e-mail facility (see Figure 19) as a means to communicate students and course tutor was evaluated. Tutorial testers deemed it as a very practical tool to provide the course tutor feedback of the difficulties encountered by the students during the Tutorial execution. By means of this e-mail system student and course-tutor which are de-coupled in space and time are brought together. The student does not feel alone in front of a machine any more. Thanks to the authoring tool[8] provided as a complement of this system, the Tutor could improve the educational data contents associated to the Demo Movie, by for example rewriting a keyword explanation where the students found problems.

[8] This authoring tool has been developed by Karina Chong, another MSc student in the Department of Computer Science, University of Essex.

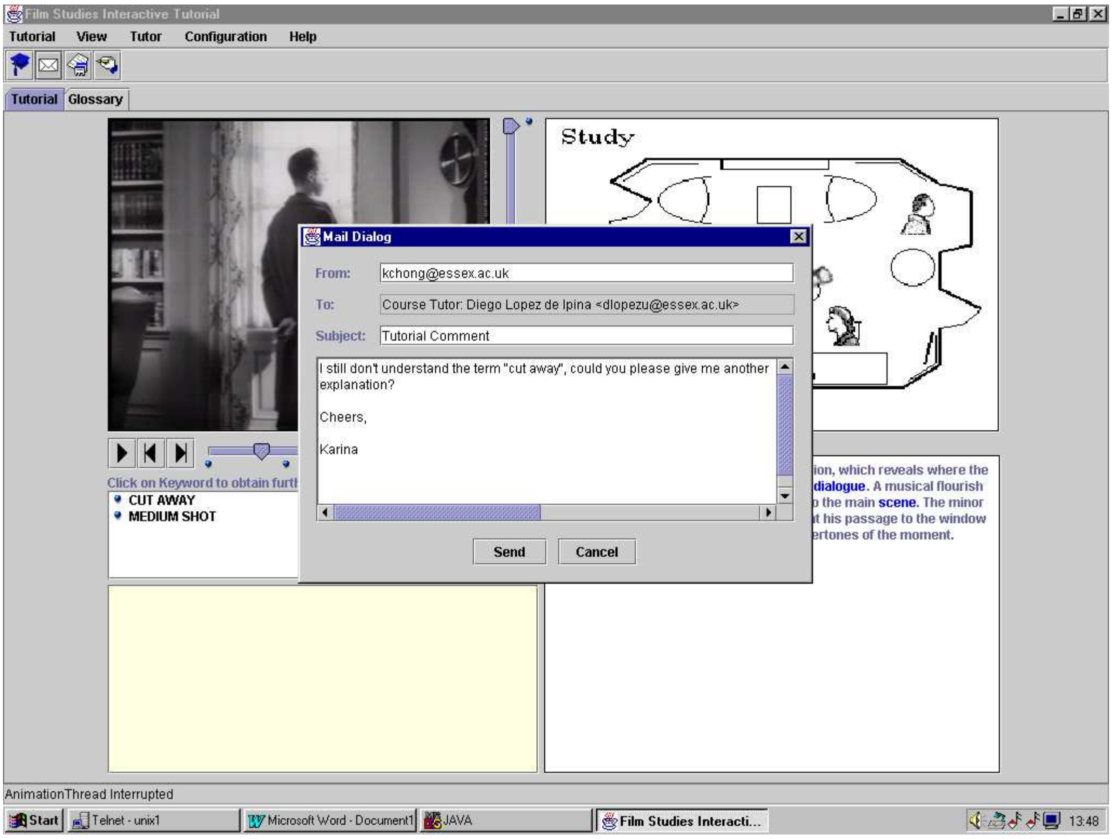


FIGURE 19
Film Studies Tutorial Email facility.

5.2. Reliability Testing (White Box Testing)

Due to the real-time and multi-threaded features of the system developed, two major problems were faced in the production of a reliable and stable system:

- Synchronisation of the different media elements.
- Thread synchronisation.

Synchronisation was a critical issue in this system implementation because of the need of integrating data coming from two different Information Servers, a Relational Database Management System and a Video Server. Data obtained from the RDBMS contains timestamps that indicate where during the video playback associated educational data has to be displayed. One of the main objectives of the system produced was to achieve a seamless integration of the media in a way that the different origin of the data would never be perceived by the final user.

The solution to the synchronisation of the different media was achieved by means of Java threads. Java threads allow concurrent execution of tasks within a process. In the case, of the system produced, it was necessary to have two different threads, the thread that plays the movie and the thread that controls the movie position and ensures the pertinent educational data is displayed. Multithreaded programming improves program's performance but on the other hand makes debugging much more difficult. When several threads share a resource and have access to it simultaneously, *race conditions* can arise making the application crash. Therefore, during this reliability testing all the code sections run by these two threads were carefully studied. The `synchronized` Java keyword was used in all the code and data sections of the program where a synchronisation anomaly between threads could have occurred.

During the implementation phase, the programmer made numerous tests, examining all critic situations where anomalies could occur. Those critic situations were when a VCR-like control operation is carried out by the user, i.e. when pausing, resuming, moving to a new position of the Demo, etc. In all these cases, the media synchronisation, after the proper bug corrections, was achieved.

The stability of the system was highly affected by the usage of threads. As it was described in the Implementation Issues section of this work, when an event occurs, a conflict can arise between the thread in charge of attending the event, and the thread in charge of controlling the demo execution. Moreover, the usage of Swing made it even more difficult due to its no thread-safe nature. Fortunately, the papers [Muller-98a] and [Muller-98b] helped a lot on the solution of this problem.

The users that tested the system pointed out its high reliability. During their hands-on sessions, no system crash was experienced. The synchronisation of video and media data achieved was completely seamless according to them.

5.3. Performance Testing

The needs of real-time media in the Film Studies Tutorial required that the Performance Testing stage should be considered in great detail. To achieve an acceptable level of performance was without doubt the most complicate and challenging part of this system development.

To begin with, the client/server architecture of this system should be considered. To achieve a homogeneous level of performance in a system with these features is obviously much more complicate that to do it in a system that runs on one machine and does not need to communicate with others. Network delays, and the delays with which servers attend requests from clients are factors that influence in the overall system performance. Therefore, during the performance testing ample problems were experienced because the results obtained were lower than expected not due to bad programming, but to problems far beyond it, such as Sever processing overload due to other tasks or network hardware problems.

The hardware description of the client/server architecture in which the system was tested is:

- A Sun SPARCUltra 4 under Solaris 2.5 with 64 Mbytes of memory, where the Relational Database Server and Video Server are run.
- Three client machines, Pentium II at 233 MHz with 64 MB under Windows 95.
- A Fast Ethernet LAN at 100MHz that allows the communication between the client machines and the server.

On the other hand the features of the video streamed in the Film Studies Tutorial are:

- Video Format: MPEG-1 System
- Full Length: 13 minutes and 29 seconds
- File Size: 116.954.070 bytes
- Frame rate: 25 fps
- Bit Rate: 1650800 bps
- Normal Size: 352 * 288 pixels

The Video Server transfers full-motion video in a constant bit rate being the size of the UDP packets employed of 8,192 bytes[9]. Providing that the video bit rate is 1650800 it can be drawn that approximately 25 UDP[10] packets/sec are transmitted from server to client, with a 40 ms gap between them. However, due not only to the network delays but also to the physics of the server's hard disk, latency and seek times have also to be considered, the actual UDP packets rate sent is lower.

The client machine has a default video buffer of 0.5Mb to receive the video streamed, i.e. it stores approximately 2.5 seconds[11] of video. If the server has to send full-motion video to many client machines then each client should serve a large enough buffer to hold enough data to compensate with the increased possibility of network delays.

[9] This is the maximum UDP size
[10] 1650800 bits per second/ 8 bits per byte/ 8192 byte per UDP packet = 25,189 packets/second
[11] 512K * 1024 bytes/K / (1650800bits / 8 bits/byte) = 2.54 seconds

Moreover the Tutorial System needs to load to Dynamic Memory the Educational Data needed in real-time, an estimated amount of data corresponding to 8MB. This is due to the fact that in order to speed up the execution of the system it is required to load to memory all the keywords and diagrams associated to each shot of the movie. Appendix D: Dynamic Memory Storage Needs gives a detailed description of the Heap Memory storage needs estimation that was made.

To the Dynamic Memory Storage Needs we have to add the memory needed to load the Application code and the non-dynamic data. Therefore, it can be concluded that the memory needs of the Film Studies Tutorial are very high. The Film Studies Tutorial should be run in system with 64 MB or more.

During the performance testing of the system the most complicate problem that was faced was to avoid the unpleasant jerking effect that occasionally appears during video playback. Multiple factors were determined to influence in this glitching effect. In what follows these factors are enumerated and when applicable the solution that was or could be adopted:

1. **Interpreted nature of the language in which the Tutorial has been programmed** [Kramer-96]**.**

   Java language programs performance, even when JIT[12] compilers are used, can not be compared with the performance of other programming language programs where native code is generated. In the execution of the Film Studies Tutorial the JIT compiler that comes with the JDK 1.1.6 is used. However, it is expected that if the application would be compiled to native code some performance improvement could still be achieved. IBM has recently released a Java compiler that suits the Java programs compiling necessity. IBM High Performance Compiler for Java allows to obtain optimised native code from Java applications. Read [IBM-98] for further information. It is freely downloadable from `http://www.alphaWorks.ibm.com/formula`.

   Experimental test where made with the simplest possible application that can be written in Java to enable video playback from OVS. The goal was to determine the video quality difference experienced when using the default Video Player that comes with the installation of Oracle Video Client 3.0 instead of the simple application implemented by means of the Oracle Video Client Library. The results reflected that the quality obtained with the Java application, although acceptable, were poorer in quality than the ones obtained with the default OVC Player. These results could be explained arguing that the interpreted nature of Java makes applications developed on it run slower that they would do with other non-interpreted languages.

[12] Short for just-in-time compiler, a code generator that converts Java bytecode into machine language instructions. Java programs compiled by a JIT generally run much faster than when the bytecode is executed by an interpreter.

2. Work Load of the server that is providing the video.

An important factor in the resulting performance of the video playback is whether the server machine streaming the video is performing other time consuming tasks or not. When other processes of heavy load are running on the server host, apart from the Oracle Video Server process, the video playback is affected because the server is not capable of streaming UPD video encoded packets to the clients, with the required frequency.

3. Platform on which the system is running

Windows 95 multitask handling is not as good as Window NT. Windows NT was especially designed to host applications that require critical use of resources. A proposed suggestion that would also help to increase the video playback quality would be to run the Tutorial on a Windows NT platform rather than a Windows 95 one. Java programmers using multiple threads in their programs have experienced significant performance enhancements in their applications when switching the platform in which they run them from Windows 95 to Window NT.

4. Lack of memory assigned to the Java Virtual Machine.

Some of the tests carried out demonstrated that some occasional video jerks occurred after hard disk access when new Diagram visualisation takes place. These disk accesses are apparently due to memory swapping from disk to memory. The explanation should be that the memory space assigned to the Java Virtual Machine [Kramer-96] is exhausted.

As it was highlighted before, the memory resources required by the Film Studies Tutorial are very high. It needs 8 Mbytes only for the Educational Data that needs to be stored at Dynamic Memory in the initialisation of the system, not taken into consideration the memory required for the code and non-dynamic data of the Tutorial. A solution to this intensive need of memory should be to increase the memory resources of the machine in which the Tutorial is run to 128 MB. A partial solution was taken by assigning a bigger size of heap memory to be used by the Java Virtual Machine by means of the `java`[13] interpreter switch `–mx`.

5. Network problems

Problems in the Server network card resulted in that the Fast Switched Ethernet LAN existing to link client machines to server, did not actually operate to the expected 100 Mbps, but to 10 Mbps instead, and in half duplex mode. To operate in half-duplex mode was especially awkward because if communications needed to take place in both senses from server to client and on the other way round, the

[13] The Film Studies Tutorial is run using the line command:
`java –mx16000000 FilmStudiesTutorial`
The –mx switch indicates the necessity of assigning 16MB for Dynamic Memory Allocation to the Java Virtual Machine.

transmission time should be shared between the traffic of each direction. A solution of this networking hardware problem is considered would help in great measure to the improvement of the Film Studies Tutorial overall performance.

6. **Polling mechanism to track current Video Position.**

In order to track the position inside the Film where the Demo is at a given instant it is necessary to continuously enquiry the Server about the stream position it pumped last time. Due to the network problems explained in the previous point, when this polling mechanism was done too continuously, UDP video packets coming from the server where forced to share the bandwidth with the control packets sent from both peers of the system, to enquiry and give answer to the current video position. The effect of this delay in video delivery resulted in that the video client buffer was eventually emptied and therefore the application execution had to be stopped briefly to fill the video buffer back. The solution found to tackle this problem was to increase the period of time between two continues polls in order to reduce the traffic generated to enquiry for video positions. Initially, this polling mechanism was programmed to occur every 80 ms, but finally it was thought that in order to increase the overall system performance should be set up in 400 ms. This period of time was taken as a suitable trade-off between performance and media synchronisation. If bigger periods of time would have been chosen the performance would have been increased even more, but the synchronisation of the media would have suffered as consequence.

7. **Quality of the streamed video**

When experimental tests where made with different videos it was observed that depending on the video, the final video playback quality obtained in the client side varied significantly. Therefore, it can be concluded the video encoding quality has as well its influence in the condition video playback occurs.

8. **Multi-threaded nature of the system.**

The jerking effect was also produced in certain measure by the concurrent execution of multiple threads inside the Film Studies Tutorial. Experiments were done to run the Film Studies Tutorial with no threads apart from the one in charge of playing the video. Slight improvements of video quality were achieved. This problem could be due as well to the way in which Windows 95 handles multitasking. As it was described in the section “Implementation Issues” of this work numerous approaches were taken to enhance the overall performance obtained by the several threads parallel execution.

9. **Oracle Video Server 3.0.3 Beta Release**

Numerous stability problems were found in the Beta OVS release that was used. It is hoped that these problems will be solved when the actual release was implemented. It is possible that with the final release of the product, deficiencies identified in the Oracle Video Java Library will have been corrected.

6. Further Work & Research.

This section summarises some pending improvements that could have been applied to the system developed and some alternative approaches to be followed in its solution. These alternative approaches could be taken as suggestions for future MSc Dissertations in Video Server technology domain.

- **Compile the code**

  In the Performance Testing Section of this work it was pinpointed that a way to improve the performance of the system would have been to generate native code for the Tutorial. Some popular Java development environments such as Symantec Cafe already provide the facility of generating 'exe' files from Java source code. However, there is only one compiler available that supports the Swing[14] package, the IBM High Performance Compiler for Java [IBM-98]. It is hoped that in short more compilers supporting the Swing Package will come to market.

- **Test the application on different platforms.**

  Time constraints disallowed the system be tested on different platforms to the one in which it was developed, Windows 95. It would have been worth to prove the multi-platform nature of the produced system by running in on a Solaris platform. Moreover, it would have been very interesting to evaluate the different performance results obtained when the application is run on different platforms such as Windows NT or Solaris. Note that potential problems would have been identified when porting the application to UNIX platforms due to the different way in which Java threads are handled in this environment.

- **Memory Usage Improvement**

  A more intelligent approach to the Memory Management of the Film Studies Tutorial could have been taken by buffering in memory only the currently needed educational data and the one that will be needed in the following seconds. The approach followed was to load all the Educational Data needed in real-time during the system initialisation. A low priority thread could have been used to control the Educational Data buffered and retrieve new information when needed. Time limitation disallowed to compare this approach with the one used and determine the most cost-effective one. Note, however, that system overall performance could suffer by adding a new thread of execution. Memory usage would be improved but system performance might decrease.

- **Develop a Web-based System**

  The Film Studies Tutorial stand-alone application could be easily mapped to an applet to be included into an HTML document and in this way be run through

[14] Swing Package was released last March.

Web Access. However, the following considerations should be taken before proceeding:

- Due to Applet Security constraints the Web Server providing the Web Page where the applet would be embedded should reside in the same machine where the Video Server and the Educational Database Server are. This means that both OVS and RDBMS will have to be running in the same machine where the Web Server is located, since one of the security restrictions of an Applet is that it only allows the creation of network connections to the computer from which it was loaded.

- Java applets created with the Oracle Video Java Library can be run from the command line, in an applet viewer, or in browsers that permit the execution of unsigned applets, such as Sun's HotJava and the JDK appletviewer utility, but not in Netscape Navigator or Microsoft Internet Explorer.

- The network bandwidth needs of the Film Studies Tutorial are very high. As it was explained in the chapter Client/Server technologies for streaming Video, Internet is not still suitable for the delivery of high quality real-time media, such as full-motion video.

Despite the two constraints just mentioned it would still be worth considering a Web-based Oracle-technology based Film Studies Tutorial, because hypermedia capabilities could be added to the system. These facilities would allow the students have access to other related educational material located at remote Web sites.

- **Remove the Oracle-based technology from the client-side of the system.**

The Oracle Video Client uses the Oracle Video Interface (OVI) to access streaming video from the Oracle Video Server. The separation of the client interface from the basic streaming functionality supported by OVI means that the functionality of the Oracle Video Server can be harnessed from a variety of platforms, without worrying about the underlying mechanisms that handle the low level tasks.

As a consequence, to be able to run an application developed by means of any of the Client Interfaces provided by OVC, it is first needed to proceed to a minimum installation of Oracle Video Client. This installation includes OVI and as well the client interface the application was developed in. For instance, in the case of the Film Studies Tutorial, the Oracle Video Java Libraries should be installed in the client side of the system. In essence, Oracle Video Server 3.0.3 provides powerful and relatively easy ways of producing video-on demand applications, but at the same time forces the developer to distribute together with the application the Oracle Video Client product. This necessity ties up the Film Studies Tutorial developed to Oracle proprietary software.

An interesting research topic for future efforts in OVS streamed video-based applications development would be to try to remove the Oracle client side software layer and employ alternative technologies to achieve the same effect. The main issue would be to eliminate the OVI part of the Oracle Video Client. The alternative software provided should handle the technical aspects of communicating with the server, controlling the real-time stream, and audio-video playback.

Two main aspects should be considered in order to replace OVI functionality:

- *Communicate client and server and control the video streaming process.*
  Oracle Media Net architecture allows the communication between OVC and OVS. It is basically an Oracle implementation of the CORBA [CORBA-98] Distributed Object Architecture that allows the invocation from OVC of remote services provided by OVS. A detailed description of the services provided by OVS was given in Section 2. A Java implementation of CORBA would be necessary to communicate the client application with the OVS by means of OMN. Alternatively, the Java Native Interface could be employed to gain access to OMN through native code.

- *Decode the streamed video and render it in the application user interface.*
  The standard Java Media Framework (JMF) [JavaSoft-98c] API provided by JavaSoft could have been used for this purpose. JMF is a collection of classes that enable the display and capture of multimedia data within Java applications and applets. It specifies a unified architecture, messaging protocol and programming interface for playback, capture and conferencing of compressed streaming and stored timed-media including audio, video, and MIDI across all Java enabled platforms. Java Media Framework package is freely available and can be downloaded from the Java Media Framework Home Site at http://www.javasoft.com/products/java-media/jmf/index.html.

  Providing the communication between client and server and the video streaming process was solved, once the streamed video arrived to the client machine it would be relatively easy to de-packetise it, decompress it and render to the screen thanks to the powerful Player class found at JMF. Some experiments were done with JMF library with very promising results, however the lack of time to implement the communication between the client and server through OMN disallowed going any further.

- **Generate a new Tutorial using Apple's QuickTime Architecture.**

An interesting suggestion for future work would be to follow an alternative technology approach to the solution of the problem, the QuickTime leading edge technology [APPLE-98].

Apple Computer's QuickTime is a software architecture that makes possible to create, integrate and publish all types of digital media. Quicktime has been designed to simplify the task of working with and integrating the widest possible range of digital media types, not just sound and video.

QuickTime is composed of three distinct elements: the QuickTime Movie file format, the QuickTime Media Abstraction Layer and a rich set of built-in QuickTime media services. QuickTime Media Abstraction Layer is an advanced component-based software architecture that provides software and hardware developers full access to the built-in QuickTime services. With QuickTime users are assured that all digital media content can be authored in a common file format which also supports real-time video and audio streaming. In essence, QuickTime eases synchronised media streams delivery over the Internet. The QuickTime SDK can be downloaded from http://www.apple.com/quicktime/developers/qt3sdk.html.

QuickTime in its latest version, 3.0, includes built-in support for 10 different media types (video, audio, text, timecode, music, animation, tween, MPEG, VR, 3D). QuickTime Movie file format has been adopted as the starting for an MPEG-4 standard [JavaSoft-98b]. Last generation Education On Demand systems are starting to employ Apple Computer's QuickTime technology because it eases a lot the synchronisation of video and educational data [Ma-98].

7. Conclusion.

This work has provided a study of how full-motion streamed video can be applied to Interactive Distance Learning through a practical case, a Film Studies Tutorial. From an end-user's view, the practical outcome of this project reflects the benefits that a broadband multimedia environment can bring to speed up the learning process for students. The originality of the work done with respect to other existing video-based Distance Learning systems has been the analysis and design of a high-performance open software solution to computer-based Distance Learning.

Java and its leading edge Swing GUI Toolkit were used to provide a platform-independent client system that applies Java philosophy: *write once, run anywhere*, and goes one step forward by supplying a cross-platform GUI look and feel. The Film Studies Tutorial developed presents GUI components with the same appearance and behaviour, no matter what operating system the application is run on.

The adoption of Java as the deployment language in which the system was produced brought many benefits, for example multi-platform and GUI richness, but on the other hand innumerable problems were encountered in achieving acceptable levels of performance on Windows 95 platforms. Seamless synchronisation of media obtained from two different sources, an Oracle Video Server 3.0.3 and an Oracle 7.3.2 Relational Database System, imposed critical performance requirements for the Film Studies Tutorial. The interpreted nature of Java and the need to have multiple threads of execution running concurrently highlighted several issues, as it was analysed in Chapters 3 and 4.

Java language is a powerful, flexible and fully object-oriented language that eases programmer's life. Nevertheless, its interpreted nature, despite the JIT compiler adoption, does not provide the high performance requirements to recommend its usage to systems with real-time needs. It is hoped that further work in this area will help future implementers achieve the ideal of high-performance real-time multi-platform Java applications.

References.

[Andrews-98]
Mark Andrews, "Introducing Swing", The Swing Connection, JavaSoft WebSite, March 1998
http://www.javasoft.com/products/jfc/tsc/swingdoc-static/intro.html

[APPLE-98]
"QuickTime White Papers & Fact Sheets", Apple Computer, Inc. 1998
http://www.apple.com/quicktime/news/facts.html

[Baltz-98]
Nancy Baltz and Christian Bedford. "Oracle Video Server Administrator's Guide and Command Reference Release 3.0 for UNIX", Oracle Corporation, February 2, 1998

[Bolot-96]
Jean Bolot and Philip Hoschka, "Sound and Video on the Web", INRIA-Sophia Antipolis, 1996

[Booch-93]
Grady Booch, "Object-oriented Analysis and Design with Applications", 2nd edition. Benjamin Cummings, Redwood City. ISBN 0-8053-5340-2, 1993

[Chong-98]
Karina Chong, "Managing data for interactive full-motion video for distance Learning". MSc Dissertation for the MSc in Computer Science at University of Essex, September 1998.

[CORBA-98]
"CORBA/IIOP 2.2 Specification", Object Management Group, 1998
http://www.omg.org/corba/corbiiop.htm

[Cornell-96]
Gary Cornell and Cay S.Horstmann. "Core Java", The SunSoft Press, Java Series, A Prentice Hall Title, 1996 ISBN 0-13-565755-5

[Deitel-97]
H.M. Deitel and P.J. Deitel. "Java, how to program", Prentice Hall, 1997
ISBN 0-13-263401-5

[Elmasri-95]
R. Elmasri/S. B. Navathe, "Fundamentals of Database Systems". "Relational Database Design using E-R to Relational Mapping", pages 172-174. The Benjamin Cummings Publishing Company, 1995
ISBN 0-8053-1753-8

[Fowler-98]
Amy Fowler , "An Overview Of Swing Architecture. The Inside Story on Swing Component Design", JavaSoft WebSite, April 1998
http://www.javasoft.com/products/jfc/tsc/swingdoc-static/swing-arch.html#

[Fursbush-98]
Gordon Furbush and Denise Stone, "OVS Developer's Guide Release 3.0", Oracle Corporation, 1998

[Gaurene-98]
Eugenio Gaurene, Paolo Fasano, and Vinicio Vercellone. "IP and ATM Integration Perspectives", IEEE Communications Magazine, Vol. 36, No. 1, pp 74-80, January 1998.

[Gosling-96]
James Gosling, Bill Joy and Guy Steele, "The Java Language Specification. Chapter 17 – Threads and Locks", Sun MicroSystems, 1996
http://www.javasoft.com/docs/books/jls/html/index.html

[Guan-98]
Huiwei Guan, Horace H. S. Ip and Yanchun Zhang. "Java-based approaches for accessing databases on the Internet and a JDBC-ODBC implementation". Computing & Control Engineering Journal, April 1998

[Hamilton-97]
Graham Hamilton & Rick Cattell, "JDBC: A Java SQL API", JavaSoft, January 1997
http://www.javasoft.com/products/jdk/1.2/docs/guide/jdbc/index.html

[Herrick-98]
Rick Herrick, "Oracle Video Client Developers Guide Release 3.0", Oracle Corporation, 1998
ftp://voyager.oraclevideo.com/pub/documentation/ovs3.0/nt/ovcdev.pdf

[Hoare-74]
C.A.R Hoare, "Monitors: An Operating System Structuring Concept", Communications of the ACM, Vol. 17, No. 10. pp. 549-557, October 1974

[IBM-98]
"IBM High Performance Compiler for Java: An Optimizing Native Code Compiler for Java Applications"
http://www.alphaWorks.ibm.com/graphics.nsf/system/graphics/HPCJ/$file/highpcj.html

[JavaSoft-98a]
"Java Foundation Classes: Now and the Future ", JavaSoft WebSite, March 1998
http://www.javasoft.com/marketing/collateral/foundation_classes.html

[Javasoft-98b]
"ISO ADOPTS QUICKTIME FILE FORMAT AS STARTING POINT FOR DEVELOPING KEY COMPONENT", JavaSoft Press Release, February 11, 1998
http://www.javasoft.com/pr/1998/02/pr980211.html

[JavaSoft-98c]
"Java Media Players", Sun Microsystems, Inc., Silicon Graphics, Inc., and Intel Corporation, May 1998
http://www.javasoft.com/products/java-media/jmf/forDevelopers/playerguide/index.html

[Kramer-96]
Douglas Kramer , "The Java Platform: A White Paper", JavaSoft 1996
http://www.javasoft.com/docs/white/platform/CreditsPage.doc.html

[Lavington-97]
Simon Lavington, Richard Gamble and Neil Dewhurst. "Pilot experiments in broadband interactive video using OVS". Department of Computer Science, University of Essex. Internal Report CSM-283, February 1997

[Lee-98]
Jack Y.B. Lee , "Parallel Video Servers: A Tutorial", The Chinese University of Hong Kong, IEEE MULTIMEDIA, Vol.5 No.2 pp. 20-28, 1998

[Linden-98]
Brian Linden and Matt Prather, "Introducing Oracle Video Server, Release 3.0", Oracle Corporation, February 1998.
ftp://voyager.oraclevideo.com/pub/documentation/ovs3.0/nt/intro.pdf

[Ma-98]
Wei-hsiu Ma, Yen-Jen Lee, David H.C. Du, and Mark P. McCahill, "Video-Based Hypermedia for Education-on-Demand", University of Minessota, IEEE Multimedia, pp 72-83, January-March 1998.

[Maring-97]
Sheryl Maring, "Oracle Media Net Developers Guide Release 3.3", Oracle Corporation, 1997
ftp://voyager.oraclevideo.com/pub/documentation/ovs3.0/nt/omndev.pdf

[Mett-96]
Percy Mett, David Crowe, and Peter Strain-Clark, "Specification and Design of Concurrent Systems". The McGraw-Hill International.
ISBN 0-07-707966

[MPEG-98a]
"The MPEG Data Archive" http://www.mpeg.org/index.html/.
A very good Internet knowledge source for MPEG standards.

[MPEG-98b]
The MPEG Home Page
http://drogo.cselt.stet.it/mpeg/#General_documents

[Muller-98a]
Hans Muller and Kathy Walrath, "Threads and Swing", JavaSoft, April 1998
http://www.javasoft.com/products/jfc/tsc/swingdoc-archive/threads.html

[Muller-98b]
Hans Muller and Kathy Walrath, "Using a Swing Worker Thread", JavaSoft, April 1998, http://java.sun.com/products/jfc/tsc/swingdoc-archive/swing_worker.html

[Nelson-98]
Matthew T. Nelson, "Java Foundation Classes", McGraw-Hill, 1998
ISBN 0-07-913758-X

[Postel-82]
Jonathan B. Postel, "Simple Mail Transfer Protocol", RFC 821, Network Working Group, Information Sciences Institute, University of Southern California, August 1982, http://freesoft.org/CIE/RFC/821/index.htm

[Pressman-93]
Roger S.Pressman, "Software Engineering. A Practisioner's Approach". The McGraw-Hill International, ISBN 0-07-050814-3

[Prince-95]
Stephen Prince , "Movies and meaning: an introduction to film". Allin&Bacon
A Viacom Company Needham Heights, 1995, ISBN 0-02-396806-0

[Schulzrinne-96]
Schulzrinne H., Cogner S., Frederik R. and Jacobson. "RTP: A Transport Protocol for Real Time Applications" RFC 1889, IETF, 1996.

[Serber-96]
Ron Serber, Eli Brosh and Shlomy Chaikin. "Video over the Internet". RAD Data Communications, 1996.
http://www.rad.com/networks/1996/video/video.htm#index

[Tanenbaum-97]
Andrew S.Tanenbaum. "Computer Networks", Prentice Hall, 1997

[Tyma-98]
Paul Tyma. "Why are we using Java?", Communications of the ACM, Volume 21, Number 6, pp 38-42, June 1998.

[Varonos-97]
Stavros Varons
"Personalised Interactive Video for Distance Learning". MSc Dissertation for the MSc in Computer Science at University of Essex, September 1997.

[VIDEONICS-97]
Videonics Corporation. "Video on the World Wide Web", 1997.
http://www.videonics.com/videos/about-web-video.html

[Willis-96]
Barry Willis, "Distance Education at a Glance", A Series of Guides Prepared by Engineering Outreach at the University of Idaho Moscow, ID 83844-1014, 1996
http://www.uidaho.edu/evo/distglan.html

Appendix A. JDBC Notes.

WHAT IS JDBC?

JDBC stands for Java Database Connectivity and it is a Java API for executing SQL statements. One can write a single program using the JDBC API, and the program will be able to send SQL statements to different kind of databases [Hamilton-97].

JDBC architecture is composed of three different components:

- JDBC API
- JDBC Driver Manager
- JDBC Driver

The JDBC Driver Manager is in charge of receiving requests from the JDBC API and sending them to the JDBC Driver that implements them. It is used to open a connection to a database via a JDBC driver, which must register with the driver manager before the connection can be formed. When a connection is attempted, the driver manager chooses from a given list of available drivers to suit the explicit type of database connection. After a connection is formed, the calls to query and fetch results are made directly with the JDBC driver. The JDBC driver must implement the classes to process these functions for the specific databases.

JDBC DRIVERS

JDBC drivers fit into one of four categories:

- *JDBC-ODBC bridge (Type 1)*
  The JDBC-ODBC bridge provides JDBC access via most ODBC drivers. The applications are written by using the JDBC API. These JDBC calls are passed to the JDBC-ODBC bridge and converted into ODBC APIs with C language. The ODBC calls are then passed to an appropriate ODBC driver for the back-end data store.

  The advantage of this method is that applications can easily access databases from multiple vendors by choosing an appropriate ODBC driver. However, if using this type of driver, the specific ODBC driver for the remote database server must be preinstalled in the client machine.

- *Native-API or partly-Java driver (Type 2)*
  This JDBC implementation converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. These drivers are typically written in some combination of Java and C, as the driver must use a layer of C code in order to make calls to the vendor libraries. They have the same software maintenance problem as the JDBC-ODBC bridge approach because they require the vendor library to be installed in the client.

- *Net-protocol and all-Java driver (Type 3)*

The software implementation of the JDBC driver translates JDBC calls into a database-independent net protocol, which is then translated into a database-specific protocol by a middle-tier server. This type of driver can entirely be written in Java and does not require any kind of pre-installation in the client side. However, a translation from the net protocol to ODBC and from ODBC to the RDBMS protocol is still required.

- *Native-protocol and all-Java driver (Type 4)*
  This type of JDBC driver converts JDBC calls directly into the network protocol used by the specific database vendor, allowing a direct call from the client machine to the DBMS server. These drivers can be written entirely in Java. As these drivers translate JDBC directly into the native protocol, without the use of ODBC or native SQL APIs, they can provide for very high performance database access.

  With this kind of driver approach, the Java applets with the JDBC driver can be downloaded into a browser without any prior client-software installation.

Appendix B. Relational Database Schema Derivation.

In order to obtain the Relational Database Schema from the Entity Relationship Diagram shown at Figure 12, the ER-to-Relational Mapping Algorithm, found at [Elmasri-95] was used. The process followed was:

1. For each regular Entity Type E in the ER schema, a new relation R was created including all the simple attributes of E. When a Composite Attribute was found only the simple attributes of it were included. One of the key attributes of E was chosen as Primary Key for R. For example, in the case of the Entity Type *Diagram*, a new relation called *Diagram* was created with the attributes *DiagramID*, *Name*, *Width*, *Height* and *Bitmap*.

2. For each binary 1:N relationship type R, the relation S that represented the participating entity type at the N-side of the relationship type was identified. The primary key of the relation T representing the other entity type participating in R was included as foreign key in S. For example, in the case of the relationship "*divided into*", the N-side of this relationship, the relation *Shot*, was identified and the primary key of the other relation, *Clip*, was added as Foreign Key to the relation *Shot*.

3. For each binary M:N relationship type R, a new relation S to represent R was created. The primary keys of the relations that represent the participating entity types were included as foreign key attributes in S forming their combination the primary key of S. Also any simple attribute of the M:N relationship type was included as attributes of S. For example, in the case of the relationship between the relations *LogicalContent* and *Clip*, "*contains*", the new relation *LogicalContentContainsClip* was created, including as foreign key attributes in it, the primary keys of *LogicalContent* and *Clip* relations, *ContentID* and *ClipID* respectively. The combination of these two attributes constituted the primary key of the new relation. On the other hand, the attributes of the relationship "*contains*", *StartTime* and *StopTime* passed to form part of the attributes of the newly created relationship.

The Relational Database Schema obtained after applying the process explained above to the Entity Relationship Diagram was:

Logical Content (ContentID, Name, Description, Length, VideoWidth, VideoHeight)
LogicalContentContainsClip (*ContentID*, *ClipID*, StartTime, StopTime)
Clip (ClipID, PathName, Description)
Shot (ShotID, ShotNumber, StartTime, StopTime, *ClipID*)
DiagramRepresentingShot (*ShotID*, *DiagramID*, StartTime)
Diagram (DiagramID, Name, Width, Height, Bitmap)
Tag (TagID, X, Y, Width, Height, Text, *DiagramID*)
Notes (NotesID, Text, *ShotID*)
NotesDiagramLink (*NotesID*, *DiagramID*, Position, Length)
NotesKeywordLink (*NotesID*, *Term*, Position, Length)
Keyword (Term, Explanation)
KeywordFoundAtBibliography (*Term*, *BiblioID*, PageNumber)
Bibliography (BiblioID, Title, Author, Publisher)
ShotExplainsKeyword (*ShotID*, *Term*)

Note that primary keys are underlined and foreign keys are in italics.
It can be observed that the resulting Relational Database Schema is in 3NF.

Appendix C. Multithreaded Programming.

This appendix purpose is to introduce the reader in basic multithreaded programming concepts. In the Technical Background section of the Implementation Issues Chapter multithreaded programming in Java is described. This section was included with the objective of giving some background information to readers for which multithreaded programming is something new.

Process-based multitasking vs. Thread-based multitasking

A *multithreaded program* is an executable entity containing two or more parts that run concurrently. Each part of such a program is called a *thread*, and each thread defines a separate path of execution. Thus, *multithreading* is just a specialised form of multitasking.

All modern operating systems virtually support multitasking. There are two distinct types of multitasking to be distinguished:

- Process-based multitasking
- Thread-based multitasking

Process-based multitasking is the most common form of multitasking. A process is, in essence, a program that is executing. Thus, *process-based multitasking* is the feature that allows your system to run two or more programs concurrently. For example, process based multitasking allows you to run a text-editor at the same time you are using an Internet browser. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler. The scheduler can be defined as the part of the operating system in charge of deciding when more than one process is runnable which one to dispatch.

In a *thread-based multitasking environment*, the thread is the smallest unit of dispatchable code. A multithreaded program performs two or more tasks at once. For instance, you can be typing text in a text-editor at the same time you are printing a document, as long as these two actions are being performed by two separate threads.

The basic difference between *thread-based* and *process-based multitasking* is that the *former one* extends the idea of the *latter one* by taking it one level lower: individual programs have the ability to run multiple computations at the same time. The main advantage of multithreading is that allows writing very efficient programs that make maximum use of CPU, because idle time can be kept to a minimum. Idle time is produced for example when I/O operations are carried out, local file system resources are read and written at a much slower pace than they can be processed by the CPU and this time could be used to undertake other kind of tasks.

Thread Definition

A *thread* is similar to a sequential program in the sense that it has a beginning, an execution sequence, and an end. Moreover, at any given time during its runtime there is a single point of execution as well. However, a *thread* itself is not a program, it cannot run on its own. Rather, it runs within a program. Figure 20 shows this relationship.

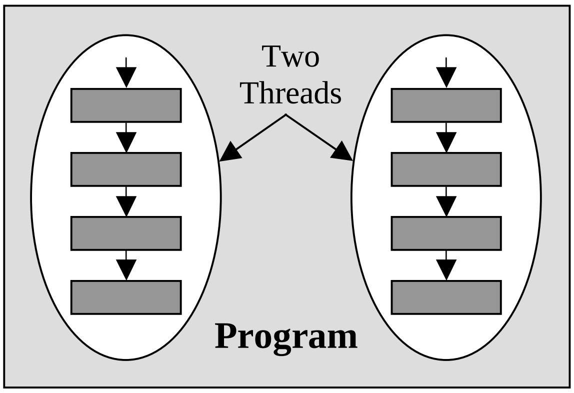


FIGURE 20
Concurrent execution of two threads inside a program

A thread is a single sequential flow of control within a program. A thread is considered as *a lightweight process* because it runs within the context of a program and takes advantage of the resources allocated for that program and the program's environment. A process has a complete set of its own variables, but a thread shares data in the program in which it lives.

Multitasking threads require less overhead than multitasking processes. Processes are heavyweight tasks that require their own separate address spaces. Inter-process communication is expensive and limited. Context switching from one process to another is also costly. Threads on the other hand are lightweight. They share the same address space and cooperatively share the same heavyweight process. Interthread communication is inexpensive, and context switching from one thread to the next is low cost.

Thread/Process Scheduling

Note that in this point the term process and thread will be used interchangeably, because as far as the scheduler it is concerned they are treated similarly.

In a Round Robin scheduler, the most normal kind of scheduler, each process is assigned a time interval, called its *quantum* or *time-slice,* which it is allowed to run. If the process is still running at the end of the quantum, the CPU is preempted and given to the next process in queue. If the process has blocked or finished before the quantum has elapsed, CPU switching is done. Every time a process is granted the CPU, a *context switch* occurs, which adds overhead to the process execution time.

Sometimes process dispatching can get more complicated when priorities are assigned to the processes. A process priority is used to decide when to switch from one running process to the next, i.e. when a *context switch* takes place. When priorities are employed the processes that are ready to be run have to be ordered by its priority in the waiting queue. However, the operating system has to prevent low priority processes to remain in the runnable queue forever. This situation is usually handled increasing the priority of a process in the queue proportionally to the time it has resided in it.

Synchronisation in multithreaded environments

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. For example, if you want two threads to communicate and share a complicated data structure, such as a linked list, a way is needed to ensure that they don't conflict with each other. It has to be prevented that one thread is writing data while another thread is in the middle of reading it. If a situation of this type has occurred it is said that a *race condition* has been reached.

The process by which race conditions can be avoided is called synchronisation and the way to solve it can be through any concurrency control tool such as semaphores, barriers, monitors... Programming languages such as C and C++ need to utilise operating system primitives in order to synchronise threads. Java, on the other hand, implements synchronisation through language elements. It uses Hoare's monitors.

Appendix D. Dynamic Memory Storage Needs.

During the initialisation of the Film Studies Tutorial, two Java dynamic storage structures are used to load into memory all the data that needs to be displayed at the same time the movie playback is carried out. These two structures are:

- **Vector**.

  The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

- **Hashtable.**

  This class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value.

As it can be seen in SQLHandler class in the code several vectors and hashtables are loaded with Educational Data when initialising the application. At the time this calculations were made it was not still known the total amount of educational data needed for the whole Tutorial. Only complete information about 1 of the 3 parts the Tutorial is composed of was known. Therefore the results obtained are only estimations of the total amount of memory that will be required for the final application, when all the educational data was available. The result obtained for the section corresponding to the “suspicion” clip are simply multiplied by 3.

Clips Vector Size

- *Clip Vector Element Size*
  - NameClip (String-type of maximum length 16) = 16 bytes
  - FirstShotNumer (int-type) = 4 bytes
  - StartTime (long-type) = 8 bytes
  - StopTime (long-type) = 8 bytes

  36 bytes
  *
- *Clip Number* 1 clip

  36 bytes

Shots Vector Size

- *Shot Vector Element Size*
  - ShotID (int-type) = 4 bytes
  - ShotNumer (int-type) = 4 bytes
  - StartTime (long-type) = 8 bytes
  - StopTime (long-type) = 8 bytes

  24 bytes
  *
- *Shot Number* 22 shots

  528 bytes

Keywords Vector Size

- *Keyword Vector Element Size*
  - word (String-type of maximum length 32) = 32 bytes

    32 bytes

    *
- *Keyword Number* 113 Keywords

  3616 bytes

Glossary HashTable Size

- *Glossary HashTable Element Size*
  - Keyword (string-type of maximum length 32) = 32 bytes
  - Definition (string-type of maximum length 552) = 552 bytes

    584 bytes

    *
- *Glossary Terms Number* 113 Glossary Terms

  65992 bytes

BitMap HashTable Size

- *BitMap HashTable Element Size*
  - FileName (string-type of maximum length 16) = 16 bytes
  - ScaledImage BitMap of maximum size 60000 bytes = 60000 bytes

    60016 bytes

    *
- *BitMap Number* 43 BitMaps

  2580688 bytes

ShotKeywords HashTable Size

- *ShotKeywords HashTable Element Size*
  - ShotID (int-type) = 4 bytes
  - Keywords Vector
    - Keyword (string-type of maximum length 32) = 32 bytes

      *
    - Maximum number of keywords = 4 keywords

      128 bytes

    132 bytes

    *
- *Shots Number* 22 Shots

  2904 bytes

ShotDiagrams HashTable Size

- *ShotDiagrams HashTable Element Size*
  - ShotID (int-type) = 4 bytes
  - Diagrams Vector
    - AnimationID (int-type) = 4 bytes
    - StartTime (long-type) = 8 bytes
    - StopTime (long-type) = 8 bytes
    - FileName (string-type of maximum length 16) = 16 bytes

      36 bytes

      *
    - Maximum number of Diagrams = 4 Diagrams

      144 bytes

    148 bytes

    *
- *Shots Number* 22 Shots

  3256 bytes

Dynamic Allocated Data Structures	Size
Clips Vector Size	36 bytes
Shots Vector Size	528 bytes
Keywords Vector Size	3616 bytes
Glossary HashTable Size	65992 bytes
BitMap HashTable Size	2580688 bytes
ShotKeywords HashTable Size	2904 bytes
ShotDiagrams HashTable Size	3256 bytes
"Suspicion" Section Estimated Allocated Memory	2657020 bytes
	* 3 Sections
Film Studies Tutorial Total Dynamic Memory Estimated	7971060 bytes

Appendix E. Film Studies Tutorial Manual.

In chapter 5, “Testing and Results”, during Functional Testing, some snapshots of the Film Studies Tutorial were already shown (Figures 15-19) to explain how the system fulfils the functionality needs expressed in the System Requirements. Reference to those snapshots will be made in this user manual. This appendix goal is to summarise all the functionality the Film Studies Tutorial provides and to explain the basic steps to follow to install and configure the system.

Film Studies Tutorial Installation

Before the installation of the Tutorial itself is carried out the following steps have to be followed:

- Install JDK 1.1.6 and Swing 1.0.3 in the client machine. JDK 1.1.6 can be downloaded from:
  http://www.javasoft.com/products/jdk/1.1/jre/download-jre-windows.html,
  whereas Swing 1.0.3 is found at:
  http://java.sun.com/products/jfc/index.html#download-swing

- Install a JDBC driver to allow communication between the client application and the back-end RDBMS. During system development the JDBC Thin Driver (Type 4) provided by Oracle at:
  http://www.oracle.com/products/free_software,
  was installed because an Oracle 7.3.2 RDBMS was employed.

- Proceed to a minimum installation of Oracle Video Client 3.0.3 in the client-machine. This minimum installation includes OVI and the Oracle Video Java Library. For more information about how to install Oracle Video Client, look at “Oracle Video Client Installation Guide” at:
  http://www.oraclevideo.com/documentation/ovs3.0/nt/index.html

- Ensure that Oracle Video Server 3.0.3 has been correctly installed in the Server machine and the Logical Content the to be used in the Film Studies Tutorial is prepared. For information about how to install Oracle Video Server, look at “Oracle Video Server Installation Guide” at:
  http://www.oraclevideo.com/documentation/ovs3.0/solaris/index.html

- Install the Educational Database in a Relational Database Management System. During the System Development an Oracle 7.3.2 RDBMS was used.

- Copy the FilmStudiesTutorial.class, FilmStudiesTutorial.bat, Tutor.cfg[15] and RDBMS.cfg files provided to a new directory in your hard drive, called for example Film Studies Tutorial.

- Double click over FilmStudiesTutorial.bat. If all the installation were followed, the system should start. Note that the content of the file FilmStudiesTutorial.bat is

[15] Note these .cfg extension files contain the Configuration Details of Tutor and RDBMS respectively.

just the command: `java FilmStudiesTutorial`, to indicate the Java Virtual Machine to start running the application.

Film Studies Tutorial Configuration

Two configuration operations are necessary before the Film Studies Tutorial is finally ready to be used by the students:

1. *Set up the Relational Database Details (RDBMS.cfg).*
   Select Configure RDBMS Menu Item from Configuration Menu or simply click over Configure RDBMS toolbar button. Fill in the fields of the Relational Database Configuration Dialog (Figure 21), with the details of the RDBMS that contains the Educational Data associated to the Film Studies Tutorial.

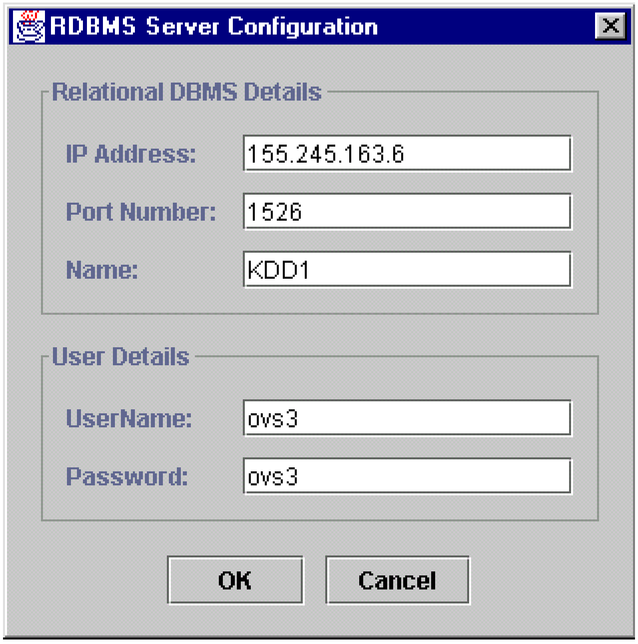


FIGURE 21
Relational Database Configuration Dialog

2. *Set up Course Tutor Details(Tutor.cfg).*
   Select Configure Tutor Details from Tutor Menu or click over toolbar button provided. Fill in the details of the Course Tutor Configuration Dialog (Figure 22).

FIGURE 22
Tutor Details Configuration Dialog

Once these two tasks are carried out the system will be ready to be started.

Film Studies Tutorial Operation

The following three facilities are provided by the Film Studies Tutorial:

1. Tutorial Demo
2. Glossary
3. Email to Course Tutor.

- **TUTORIAL DEMO FACILITY**

  The Film Studies Tutorial is initiated by default with the Tutorial Demo Panel active. If not just select Tutorial Demo Tab for the Tabbed Pane shown. The operation of the Tutorial Demo is very simple and assisted by Tool Tips appearing when the mouse is moved over each of the different controls of the Tutorial. The following operations can be done inside the Tutorial Demo Panel:

  - *Start the Tutorial Demo.*
    Click on the Play/Resume button. Figure 15 shows the Tutorial Demo while it is running.

  - *Stop The Tutorial Demo.*
    A click on the Pause Button, Diagram Panel or Keyword List Item will make the Tutorial Demo to be stopped. When the Tutorial Demo is stopped the Notes associated to the Current Shot will appear. If the click is done over a Keyword List Item an explanation of the Keyword Item Clicked will appear. Figures 16 and 17 show Tutorial Demo when stopped.

  - *Move to the Next/Previous Shot in Tutorial Demo.*
    A click over the Next Shot button will move the Tutorial Demo to the Next Shot position. If the Demo was ON, it will continue running, if not, the new Notes and Keywords associated to the new shot will be displayed. In case that shot has Diagrams, the first Shot Diagram will be shown in the Diagram Panel. Previous Shot button carries out the analogous functionality.

  - *Seek to a new Position in the Tutorial Demo.*
    By means of the Movie Time Scrolling Bar, the user will be allowed to move to different Tutorial Demo Positions. If the Demo was ON, the Demo will continue running, if not, the new Notes and Keywords associated to the new Demo Position will be displayed. In case that new Active Shot has diagrams, the first Shot Diagram will be shown in the Diagram Panel.

  - *Turn on/off the volume.*
    By clicking on the Mute Button, the Tutorial Demo Volume will be set on or off.

- *Control the Volume Level.*
  The Volume Level can be controlled through the Movie Volume Slider.

- *Obtain Keyword Explanation.*
  A click over a Keyword List item will stop the Tutorial, in case it was running, and display the explanation of the keyword clicked.

  A click over a blue colour link (Keyword Link) in the Tutorial Notes provided when the Demo is stopped, will result in a pop-up tip window explaining the Keyword. Figure 17 shows the two ways to obtain a Keyword Explanation.

- *Obtain Diagram Explanation.*
  Once the Demo has been stopped, whether because the Pause button, a Keyword List Item or Diagram Panel was clicked, mouse movements over the Diagram Panel will cause Tips to come up explaining the different components of the Diagram.

- *Obtain Diagram refered in Notes.*
  A click over a red colour link (Diagram Link) in the Notes will produce the application to display a new diagram in the Diagram Panel, to give explanation to some concepts explained in the notes. Figure 16 shows the effect of clicking a Diagram Link.

❑ GLOSSARY FACILITY

To start the Glossary Facility click over the Glossary Tab. Once the Glossary Panel is active three operations can be carried out:

- *Move to the beginning of a Glossary Section.*
  By clicking over an index in the Glossary Panel the position of the Glossary Document will be moved to the position where the first Keyword starting by the index letter clicked is given. Figure 18 shows the outlook of the Glossary when an index is clicked.

- *Obtain Keyword Bibliography References.*
  Click over the Bibliography Link that appears at the end of each Keyword Explanation to obtain the references of a Keyword.

- *Move to a random position in the Glossary.*
  Use Page Up, Page Down Keys or the Scrolling Bar provided to move wherever you want in the Glossary.

❑ EMAIL FACILITY

To send an email to the Course Tutor, click over the Send Mail button in the Tool Bar or the Send Mail Menu Item in the Tutor Menu. Write the message in the Dialog that will appear and click whether Send Button to send it or Cancel to cancel the message. Figure 19 shows the system email facility.

Appendix F. Source Code.

The Film Studies Tutorial has been implemented by means of the following Java files:

Source File Name	Description
AboutDialog.java	Class implementing About Box Dialog.
AnimationMouseListener.java	Class to handle events over the Diagram Panel.
BibliographyNote.java	Class containing details about a Bibliography reference.
Clip.java	Class containing details about a clip.
ConfigurationDialog.java	Class implementing Database Configuration Dialog.
CourseTutor.java	Class containing details about the Course Tutor.
Diagram.java	Class containing details about a diagram.
FilmStudiesTutorial.java	Main Class of the system. Creates the UI and connect to the Database.
GlossaryPanel.java	Corresponds to the Glossary class of the Design. It implements the Film Studies Tutorial Glossary functionality.
IntroWindow.java	Class implementing the Window that presents the application.
Keyword.java	Class representing a shot keyword.
KeywordChooser.java	Class to handle events over the Keyword List.
KeywordTip.java	Class implementing a Tip Window appearing when a Keyword Link is clicked.
MailWindow.java	Class implementing the window used to send email.
Mailer.java	Class implementing the mail system of the Tutorial.
MoviePositionSliderListener.java	Class to handle events over the Movie Time Slider.
MyUtil.java	Miscellaneous class containing several useful methods.
Notes.java	Class containing details about the notes of a shot.
NotesLink.java	Class representing a link in the Notes.
PlayerController.java	Class to handle events over all the buttons contained in the Player Controls.
RDBMSConnectData.java	Class containing details about the RDBMS the system connects to.
Shot.java	Class containing details about a Demo Film Shot.
SMTPException.java	Class defining exceptions occurred during the email delivery.
SQLHandler.java	Class defining all the interactions with the RDBMS.
SwingWorker.java	Class to implement the background thread the Demo Worker is inspired on.
Tag.java	Class containing details about a Diagram Tag.
TextAreaWithLinks.java	Class implementing the Tutorial Notes with links.
TutorDialog.java	Class implemented the dialog to change the Tutor Details.
TutorialPanel.java	Corresponds to the TutorialDemo class of the design. It implements the Tutorial Demo functionality.
MovieVolumeSliderListenerl.java	Class to handle events over the Movie Volume Slider.