Prototyping The VISION Digital Video Library System

by

Kok Meng Pua
pua@eecs.ukans.edu
B.S. University of Kansas, 1993

Submitted to the Department of Electrical Engineering & Computer Science and the Faculty of the Graduate School of The University of Kansas in partial fullfillment of the requirement for the degree of Master of Science

---

Dr. Susan Gauch
(Chairman of the Committee)

Dr. Joseph Evans
(Member of the Committee)

Dr. John Gauch
(Member of the Committee)

Date Thesis Accepted

Abstract

The digital libraries of the future will provide electronic access to information in many different forms. Recent technological advances make the storage and transmission of digital video information possible. This project is to design and implement a digital video library system prototype suitable for storage, indexing, and retrieving video and audio information and providing that information across the Internet. The goal is to allow users to quickly search indices for multiple videos to locate segments of interests and view and manipulate these segments on their remote computers. A minimum of one hour of video, which will serve as the searching information source, will be digitized, segmented, and stored in a database. An initial VISION prototype that handles video clips in KUIM video file format was implemented. It has been modified to handle video data in AVI video file format with JPEG compression. Videos, soundtracks and transcripts are digitized, and information from the soundtrack and transcripts is used to automatically index videos in a clip by clip manner. While this technology would be applicable to any collection of videos, we are aiming at the educational user, providing teachers with the ability to select segments of videos which complement their lessons. This system prototype will serve as the pilot work of the VISION[1] project, in which a complete and fully functioning digital video library system will be implemented and evaluated.

[1] Video Indexing for SearchIng Over Networks

Contents

Chapter 1

Introduction

1.1 The VISION Project Overview

1.1.1 Driving Problem

How does a teacher find a video clip of the Oklahoma City Federal Building bombing? Or a video clip of Joe Montana's first NFL career touchdown throw? Or a video clip of the Kobe earthquake in Japan? Or a video clip showing how to play tennis?

Textual information such as journals and technical reports make up a large number of the items accessible by the Internet, but they represent only a small portion of the volume of data available. Sound recordings, images, videos, and scientific data which are non-textual usually require more storage per item. For instance, a single uncompressed image with a size of 240x320 pixels requires roughly 230 KB, and a second of uncompressed digitized video, which contains 30 frames of images, requires over 7 MB of disk space. The supply and demand for non-textual information sources is likely to grow exponentially in the near future with

the increasing availability of input and output devices on multimedia workstations. In other words, we must develop technologies necessary to efficiently handle this tremendous volume of non-textual information to provide a major breakthrough for the multimedia world of the future.

As new information is created and shared among the millions of Internet users at an awe inspiring rate, it is possible that most of the information for which we are looking exists on the net somewhere. All we need to do is to use an effective searching technique to find them. Unfortunately, the supply of information is increasing far more rapidly than our ability to support effective searching of this huge resource. Therefore, we must first make fundamental advances in how this information can be captured, stored, searched, filtered, and displayed before it grows out of control.

The current technology supporting the search and retrieval of non-textual information lags far behind text based systems. The simplest way to locate non-textual information is to select by name, copy from the database to the user's site, and then examine it. This simple search technique has a few drawbacks; for instance, if the names of items are not chosen carefully, certain items may never be accessed, or other items may be incorrectly retrieved. The retrieval of useless data puts a tremendous load on the communication capabilities of the Internet due to the huge volume of non-textual items such as images and videos which can be 100 to 100,000 times larger than typical text items. Our conclusion is that new technologies for digital video libraries which support intelligent content-based searching and retrieval of video information should be developed and evaluated.

1.1.2 The VISION Project Goals

This project performs important pilot work in developing technologies necessary to provide automatic video indexing and digital video delivery via computer networks. Industrial collaborators such as WGBH and CNN need these capabilities to manage their video libraries for internal use and see this as a new way to deliver their products. This project is to design, develop, implement and evaluate the VISION digital video library system suitable for storing, indexing, searching, and retrieving video and audio information and providing that information across the Internet or the evolving National Information Infrastructure. Implementing VISION will push the limits of technology. Nevertheless, in order to be an effective library, users need to be able to find the video segments they want. Achieving this goal will require ground-breaking research into automatic content-based indexing of videos that will significantly improve the users' ability to access specific video segments of interest. In our approach, videos, soundtracks, closed captions, and transcripts will be digitized, and information from the soundtracks and transcripts will be used to automatically index videos in a frame by frame manner. This will allow users to quickly search indices for multiple videos to locate segments of interest from their remote computer. While this technology would be applicable to any collection of videos, we will target educational users, providing teachers with the ability to select segments of nature and/or current events videos which complement their lessons.

We plan to develop technologies necessary to provide desktop access to video segments stored in remote digital libraries, specifically automatic video indexing and digital delivery via computer network.

We intend to focus on four primary areas:

- The acquisition, digitization, segmentation, and storage of video.
- The indexing of video using scripts, manual techniques, closed captions, and speech recognition applied to the sound track.
- The retrieval of appropriate video clips using information retrieval techniques.
- The access to the video library and distribution of video via the Internet.

The proposed effort will address several fundamental questions:

1. How can large volumes of video data be handled in an efficient manner?
2. Can worthwhile indexing terms be extracted from video sound tracks?
3. Can information from multiple sources be combined for effective indexing?
4. What is the accuracy and precision of retrieving appropriate video segment from an information retrieval system?
5. Can video be delivered to a wide range of users over the Internet?
6. How will the improved access to segments of educational videos be used by teachers and students to promote learning?
7. How will high-speed networks (ATM) perform under large volume of video data traffic?

Our goal is to build a digital video library storing approximately 100 hours of short (2-5 minutes) video segments which can be searched using natural queries

language. To support the storage, retrieval, and transmission of this enormous quantity of digital video, a high performance video storage system must be constructed which utilizes state-of-the-art compression and communication techniques. To support text-based video searching, automatic indices must be constructed based on textual information extracted from the video. To support remote access to VISION by students and educators, a variety of graphical user interfaces must be developed.

1.2 Prototyping VISION

1.2.1 Motivation

This thesis is to design and implement a VISION prototype suitable for storing, indexing, and retrieving video and audio information and providing that information across the Internet. The goal is to allow users to quickly search indices for multiple videos to locate segments of interest and view and manipulate these segments on their remote computers. Approximately one hour of video will be digitized, segmented, and stored in the video database. The final product will be a VISION prototype that handles video clips in both KUIM video file format and AVI video file format with JPEG[1] Compression.

A few benefits that we will learn from the prototype are:

- A working system is available quickly to demonstrate the feasibility and usefulness of the application.
- The prototype serves as a basis for writing the specification for a production

[1]Joint Photographic Expert Group

quality system.

- We may find incomplete and/or inconsistent requirements as the prototype is developed.
- Confusing and difficult-to-implement ideas may be identified and refined.

As a whole, the major goal of this thesis is to perform pilot work on developing technologies needed on this digital video library project. Prototyping is considered to be an important step, which collects all the pilot work together in order to testify that the proposed idea will work and is heading to the right direction.

1.2.2 Accomplishments

The following results were obtained through the design, and implementation of the VISION prototype:

- A valuable digital video library of approximately one hour of video, made accessible from the Internet via a testing client.
- A system and network architecture to support the delivery of video library services on the Internet. The principal level supported is a high quality video service with a maximum transmission rate of 1.5 Mb/s.
- Methods and procedures to playback video and audio in real-time.
- A search and retrieval engine which performs Boolean full-text searching and returns a ranked results.

- A graphical user interface written in Tcl/Tk package using XLib routines was designed and developed to support the use of digital video library. This client GUI is available on Alpha and Suns platforms.

- A video processing system which performs video and audio digitization, segmentation, and compression in real-time.

- Automatic segmentation of video clips based on pixel values.

By prototyping the VISION digital video library system, we gained experience in designing and developing a complete VISION digital video library system. We also learned about the basic technologies needed, including: 1) handling multimedia data in the network, 2) processing the video and audio data such as digitization, segmentation, decoding and capturing them in real-time, 3) Choosing an appropriate database system to build search indices and perform Boolean full-text searching, 4) Handling the synchronization of the video and audio playback, and achieving a near real-time playback rate of 28 frame/sec. Finally, our future work is to implement a full function VISION system by extending the current prototype.

1.3 Related Work

1.3.1 Digital Libraries In General

Technological advances of several kinds are converging to transform the ways in which we generate, store, and use information. Digital libraries are being built which store a wide variety of information and information types: page images of technical journal articles [22], nucleic acid sequence data [3] , geographic information [32], computer science technical literature [Brunei, 1993] to name a few.

Transmission over gigabit networks will allow ubiquitous access from K-12 students, colleges, businesses and government. With regular libraries, the user goes to the information. In the digital realm, the information is delivered to the user, requiring easy to use, easy to learn user interfaces [16], and information servers which can interface with a wide range of client technologies [27]. The ability of users to manipulate retrieved information has fundamentally changed the relationship between the information producer and consumer [35], prompting attention to both the legal and social aspects of this process [17].

Prominent libraries are leading the foray into the electronic world. The Library of Congress has been continuing the digitization efforts begun five years ago by its American Memory pilot project, which has successfully digitized more than 210,000 items. As the home of the US. Copyright Office, the library is currently addressing copyright issues through several projects, one of which is the Electronic Copyright Management System (ECMS) allowing automated copyright registration [Becker, 1995]. Initiatives for Access, a program of 20 development projects at the British Library, was inaugurated in July 1993 to investigate hardware and software platforms for the digitization and subsequent networking of a rang e of library materials. In addition to enhancing library services and facilitating access, the program will establish standards for the storage, indexing, retrieval and transmission of data, and will examine the copyright issues involved with digitization of material and its provision over networks [33].

Large scale collections of video data are getting attention. For instance, AT&T envisions a huge digital library storing a wide range of data, including movies for viewing on demand, interactive presentations, educational materials, marketing

presentations, and news [6]. To make this dream a reality requires research in the basic technologies necessary to implement digital video libraries. Recent efforts have been made in developing the individual components necessary for handling multimedia data [30]. Current digital library prototyping efforts include: content-based document browser, document analysis and conversion. document architectures and standards, search and display.

The emerging ability to digitize and manipulate image, video and audio information is leading to the creation of large scale collections of multimedia data. Recent efforts have been made in developing the individual components necessary for handling multimedia data [30; 34], and building software systems and operating systems designed to handle multimedia data [15; 23]. Launched in 1991, in response to the ACM Publications Board's plans to encourage an electronic archive for computer science, project Envision at Virginia Tech is a digital library effort to develop a multimedia collection of computer science literature with full-text searching and full-content retrieval capabilities [21].

People are looking for a technology to treat collection of digital video segments as a library which can be automatically indexed and searched based on the contents of the video. Given the limited descriptive ability of current computer vision systems and improving accuracy of connected speech recognition systems, the most sensible approach for automatically indexing video is to extract textual descriptions of the video directly from the audio track. The Video Mail Retrieval Using Voice Project at The University of Cambridge represents one effort in this direction [25]. This group is attempting to extract video indexing terms from the sound track and written contents of video mail. We are in contact with this group

through Dr. Karen Sparck Jones, who is on our technical advisory board, and intend to share ideas and successes with them.

Recently, the National Science Foundation funded six digital library initiatives. The Stanford Digital Library Project tries to develop enabling technologies for an integrated virtual library to provide an array of new services and uniform access to networked information collections. The Integrated Digital Library will create a shared environment linking everything from personal information collections, to collections of conventional libraries, to large data collections shared by scientists. The project aims integration at a much higher level than transport-oriented exchange protocols details of accessing diverse sources, an abstraction layer called Information Bus is proposed to mediate communication between different clients, sources and services. The Information Bus protocols will allow users and applications to express their goals in a language that expediently describes information management tasks and objects. High-level concepts and protocols will allow users and developers to interact with diverse library resources in a unified manner [40].

UC Berkeley's Digital Library Project focuses on the following critical technologies: 1) Fully automated indexing and intelligent retrieval. To this end, they are developing and exploiting techniques for computer vision of digital textual and image documents, 2) Developing database technology protocol for client/server information retrieval, 3) Improving data acquisition technology, and tools that facilitate the creation of specialized document recognition systems, and 4) New paradigms of interaction, which is to provide a content-oriented browser in which text segments, images, and other parts of documents will be analyzed for their contents. They are creating a prototype digital library focused on "The California

Environment". It is a large collection of diverse kinds of data about the environment in California [25].

The goal of the Alexandra Digital Library Project at UC Santa Barbara is to develop a distributed system that provides a comprehensive range of library services for collections of spatially indexed and graphical information. While such collections include digitized maps and images as important special components, the Alexandra Digital Library will involve a very wide ranges of graphical materials and will include textual materials. Users of the Alexandra Digital Library will include from school children to academic researchers to members of the general public. They will be able to retrieve materials from the library on the basis of information content as well by reference to spatial location. The architecture of the testbed system includes a user interface supporting simple access to each of library services by combinations of textual and visual languages; a catalogue component providing rapid and appropriate response to user queries, particularly those involving content-based search; a storage component providing storage for, and high-speed access to, large collections of spatially indexed items; and ingestion component allowing libraries and system managers to add new items to the library collection [37].

The University of Illinois is working to bring professional quality search and display to Internet information services. The testbed collection consists of articles from engineering and science journals and magazines which are obtained in SGML format directly from major partners in the publishing industry. The testbed software will support comprehensive search and display of complete contents of articles, including text, figures, equations, and tables [36].

The University of Michigan Digital Library (UMDL) Project's contents emphasize a diverse collection focused on earth and space sciences. The UMDL testbed will consist of a cooperating set of three types of software agents: user interface agents, mediation agents, and collection agents. User interface agents will conduct interviews with users to establish their needs. Mediation agents will coordinate searches of many distinct but networked collections and collection agents will handle searching within specific collections of text, images, graphics, audio and video [8].

1.3.2 Digital Video Library System

The sixth NSF initiative, The Informedia Digital Video Library Project at Carnegie-Mellon University, is the most relevant to this project. They are developing new technologies for creating full-content search and retrieval digital video libraries. Working in collaboration with WQED Pittsburgh, the project is creating a testbed that will enable K-12 students to access, explore, and retrieve science and mathematics materials from the digital video library. The library will initially contain 1,000 hours of video from the archives of project partners. One of the research aspects of the project is the development of automatic, intelligent mechanisms to populate the library through integrated speech, image, and language understanding [7]. The library will use a speech recognizer to transcribe narratives and dialogues automatically. In addition to annotating the video library with text transcripts, the video will be segmented into smaller subsets for faster access and retrieval of relevant information. Segmentation of video clips via visual content will be performed.

Our approaches complement those of the Informedia project. Whereas they are focusing on the application of knowledge-intensive techniques for video indexing, we are exploring the efficacy of low-knowledge statistical approaches (corpus linguistics rather than natural language processing, image analysis rather than computer vision, word-spotting rather than full speech recognition). It is quite possible that knowledge-rich indexing will result in improved search performance. However, for many many video archives, the investment in knowledge engineering will be considered too high a price to pay, making it worthwhile to also investigate lower cost (in effort and money) approaches. For many archives, the choice will be either automatic domain-independent segmentation and indexing or no indexing at all.

Given the limited descriptive ability of current computer vision systems [20], the only imaging techniques available to index the videos, especially without any domain information, are the extraction of basic image features from representative frames in each segment. The QBIC system developed by IBM Almaden Research Center is one system which indexes images by features, allowing retrieval based on color percentages, color layout, shape etc. This approach may be sufficient for collections of images which have one main subject in each picture, but it seems less applicable to video frames containing many objects of interest. Also, what is said is a better indicator of content than what is in the picture. How many movie scenes could be described as picturing a human face? However, we will explore this approach by extracting a collection of image features which are automatically extracted in order to compare these methods with the audio-based indexing.

With the improving accuracy of connected speech recognition systems [38], our main approach for automatically indexing video is to extract textual descriptions of the video directly from the audio track. The Video Mail Retrieval Using Voice project at The University of Cambridge represents one effort in this direction [25]. Alternatively, the MIT VsBrowser system [2], extracts textual information from closed captions, on which it does simple pattern matching. This allows it to deal efficiently with a dynamic database (CNN Newsline reports are acquired every half hour) but does not allow it to provide high-quality search available using full-text retrieval techniques. We too will experiment with using the contents of the closed captions but we will use them for indexing with a full-text retrieval engine. To further distinguish our approach, unlike VsBrowser, we will segment the video sources we acquire.

Chapter 2

VISION Prototype Design

2.1 System Overview

2.1.1 System Design Requirements

The VISION system is designed to support access to relatively short video segments. This is fundamentally different from video on demand systems which are designed to transmit entire movies. In particular, its storage and retrieval system must be designed to support random access to short video segments. In addition, the system must be able to support a variety of user interfaces supported over communication networks with different transmission properties. The requirements for creating an effective VISION prototype with these properties are:

- We will need to acquire a collection of production quality educational videos which will be used to evaluate the effectiveness of the VISION prototype design and implementation. Educational videos typically contain a number of segments on topics which can stand on their own without viewing the entire video. These video clips can often be viewed in any order and still retain educational value.

- Since at present, most of educational videos are stored on analog magnetic media, we will need to include video and audio digitization capabilities. The source of material must be digitized from high quality video media in order to ensure highest possible fidelity to the original videos. It must have enough processing capacity and bandwidth to digitize video and audio in real time.

- A video storage system that stores digitized video and audio streams is needed. It will also provide an interface for retrieving, processing, updating, and storing additional video and audio streams. In this video storage system, handling disk space effectively will be a major concern.

- There should be a subsystem that has sufficient processing capacity to implement video and audio compression and image processing functions in real time.

- The goal of the VISION project is to support searching of a digital video library using content-based queries. Hence, a user-friendly and interactive user interface must be designed and implemented.

2.1.2 Overview of The VISION Software Components

In order to simplify the discussion of the system requirements mentioned in the previous subsection, VISION is decomposed into the following five primary components.

1. Video Storage System(VSS)

The VSS stores video segments for processing and retrieval purposes. Since our objective is to provide intelligent access by multiple users to portions of a video rather than entire videos, the VSS must be capable of delivering

numerous short video segments simultaneously.

2. Video Processing System(VPS).

The VPS consists of video processing programs to manipulate, compress, compact, and analyze the video and audio components of a video segment. In particular, the VPS contains a component to recognize keywords from the soundtrack of video segments.

3. Information Retrieval Engine(IRE)

The IRE is used to store indices extracted from video segments and other information about the video segments, such as source, copyright, and authorization. The IRE will be capable of supporting Boolean search queries.

4. Query Server(QS)

The QS processes video queries from the Remote Clients and communicates with the IRE and VSS to enable users of the digital library to extract video data and create multimedia representations of the information of interest.

5. Client

The Client is a graphical user interface which resides on the user's computer. It includes interfaces for conducting structured and Boolean full-text search and a simple video editor.

Figure 2.1 shows the overview of the VISION software components. As can be seen from Figure 2.1, these five components are highly interrelated and support three very different VISION functions:

- The creation of the VISION archive.

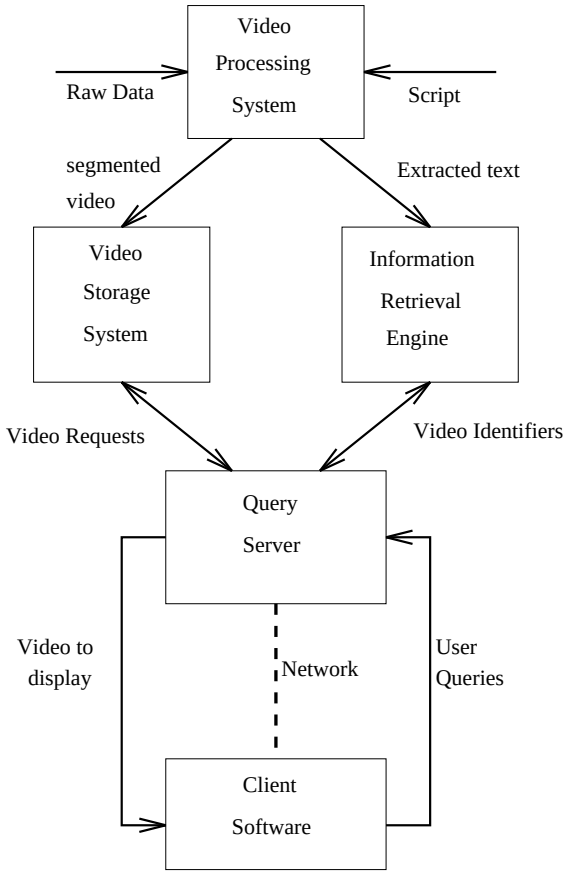

Figure 2.1: Overview of the Digital Video Library Software Components

- The processing of video in the VISION to build automatic indices.
- The access of the VISION by users of the testbed system.

The design of each of these components will be discussed in the next section.

2.2 The Design

2.2.1 Design Strategy

Figure 2.2 shows the system design process necessary to produce a useful prototype. The first step, the system design requirements, were established in the previous section. Since this is a prototype design, producing a working prototype quickly and cost effectively is our first priority. Hence, adaptation of existing software and hardware components is essential in this prototyping process.

The software requirements of the initial prototype are:

- A database management system running on the Video Storage System or Information Retrieval Engine to handle structured data, building indices and searching.
- A full-text search capability located at the Information Retrieval Engine to process textual queries.
- A software package that will handle all the video and audio processing tasks.
- A query server whose job is to communicate with clients and control signal and data traffic between database and clients.

- A graphical user interface as the client. It should includes interfaces for conducting Boolean searching and a simple video editor.

On the other hand, the hardware requirements are :

- Fast-speed disk capacity to store data.
- A UNIX workstation which is used as the server and runs VPS, IRE, and VSS.
- A UNIX workstation which is used as the client.
- There will be a J300 board mounted on both server and client machine for the purpose of digitizing, compressing, and decompressing video data for storage and display after retrieval.
- Both server and client machines connected via Ethernet or ATM.

Top-down design in conjunction with functional decomposition strategy was used to produce design part shown in Figure 2.2. Using this design strategy, the problem is recursively partitioned into sub-problems until smaller tractable sub-problems are identified. As shown in Figure 2.1, the system is decomposed into five main software components. Refinement of each components will be carried out according to the system design requirements and specifications.

2.2.2 Hardware Design

This project does not require us to design any new hardware components, but the organization and configuration of hardware components of the system play

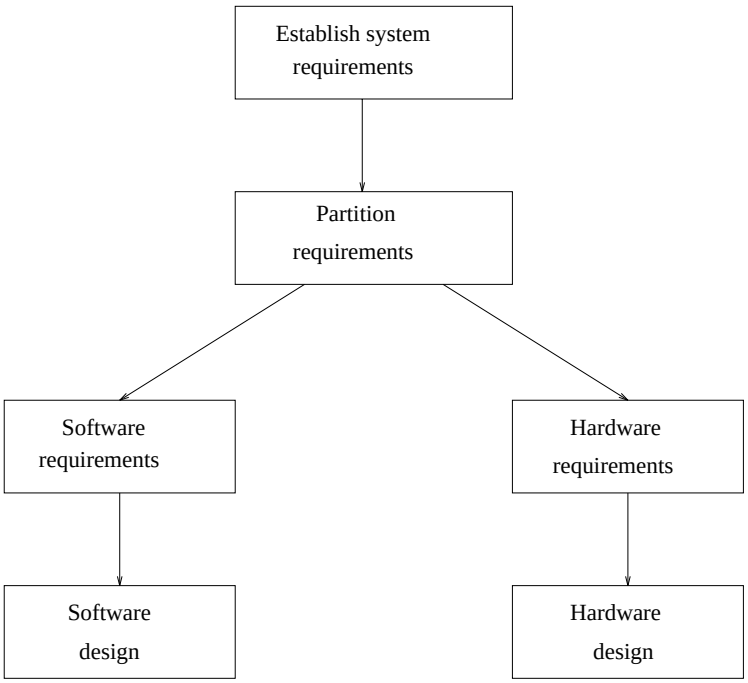

Figure 2.2: The system Design Process

a major role in achieving system effectiveness. We are using existing hardware components such as Alpha workstations to be part of the VISION prototype. The major hardware exploration is identifying, adapting and purchasing suitable hardware components such as video boards for digitizing purposes.

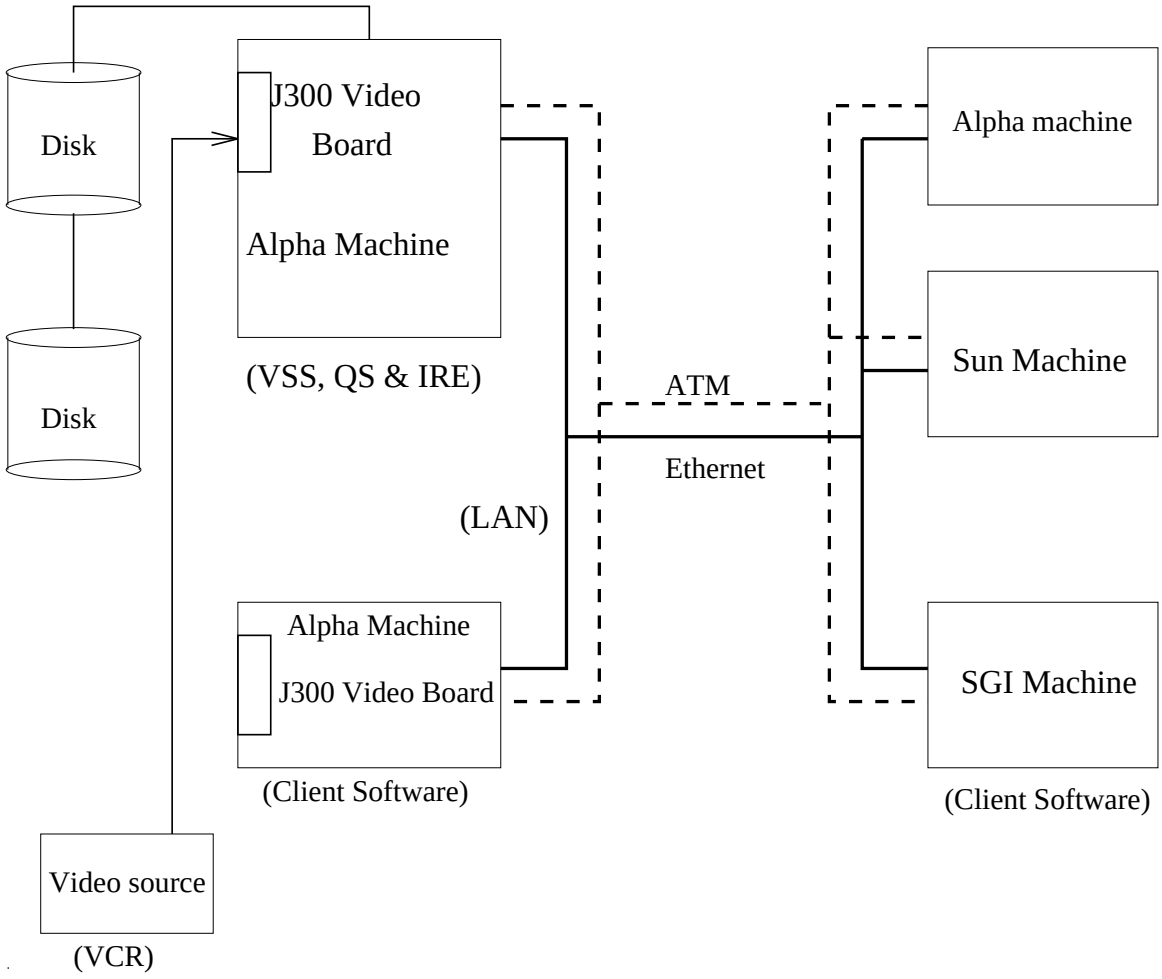

Figure 2.3: The VISION prototype hardware and network system setup

Figure 2.3 shows the configuration of hardware and network system setup of the VISION prototype. An Alpha machine will be the server and run the VPS, IRE, and VSS. All the digitized video segments will be stored in high-speed disks which are mounted directly to the server machine. With this disk space, an hour of digitized video data will be stored for searching and retrieving demonstrations. In order to produce digitized video segments from analog sources, a J300 video board mounted on an Alpha machine, together with the necessary software, will create video segments file in both KUIM[1] file format and AVI[2] video file format.

Different workstations will serve as the Clients handling the user interactions. These workstations include an Alpha workstation in the same local area network as the main server machine and several Sun and DEC workstations. Macintosh and Pentium machines are also available. The whole network is connected by Ethernet and ATM.

Again, the primary goal of this thesis is to design and implement a working prototype for proof-of-concept purposes; hence, it is sufficient that the available hardware components provide the following functions:

- Disk space to store and handle an hour of digitized, compressed video and audio data, together with their indexing information.
- An Alpha machine as the client with a J300 video board which can decompress the video and audio data fast enough in order to match the standard video playback rate.
- An Alpha machine as the server with a J300 video board which can digitize

[1] A video file format that a chunk of 1024 bytes of header followed by raw video data
[2] Audio and Video Interleaved

and encode video data in the specified format.

- An Ethernet connection between the Client and the Query Server with external connections from the Quer Server to the Internet.

2.2.3 Software Design

Figure 2.4 shows the system functional flow diagram. Each primary component (dashed boxes), is decomposed into a few primary activities which are the top-level algorithm specifications for implementing the prototype modules.

A high-level system data flow diagram is also shown in Figure 2.5. It consists of the data flow inside and between each of the five primary software components. This system data flow diagram is a sub-product of the system functional flow diagram shown in the Figure 2.5. The main purpose of having this data flow diagram is to identify the data format to be handled and produced by each of the primary software components. These system functional and system data flow diagrams will be further decomposed into smaller sets as the design process continues.

The distribution of system control signals is another important part of designing the system. A sketch of the control signals between the five primary software components is shown in Figure 2.6. Notice that the client software acts as a trigger to the query server which controls the Video Storage System and Information Retrieval System. Note that the Video Processing System is isolated from the other four primary components since it is only a preprocessing unit. The data and control signal flow between this video processing system and video storage system or information retrieval system is only needed to create and update the database and searching indices. As shown in Figure 2.6, there are total of 6 main

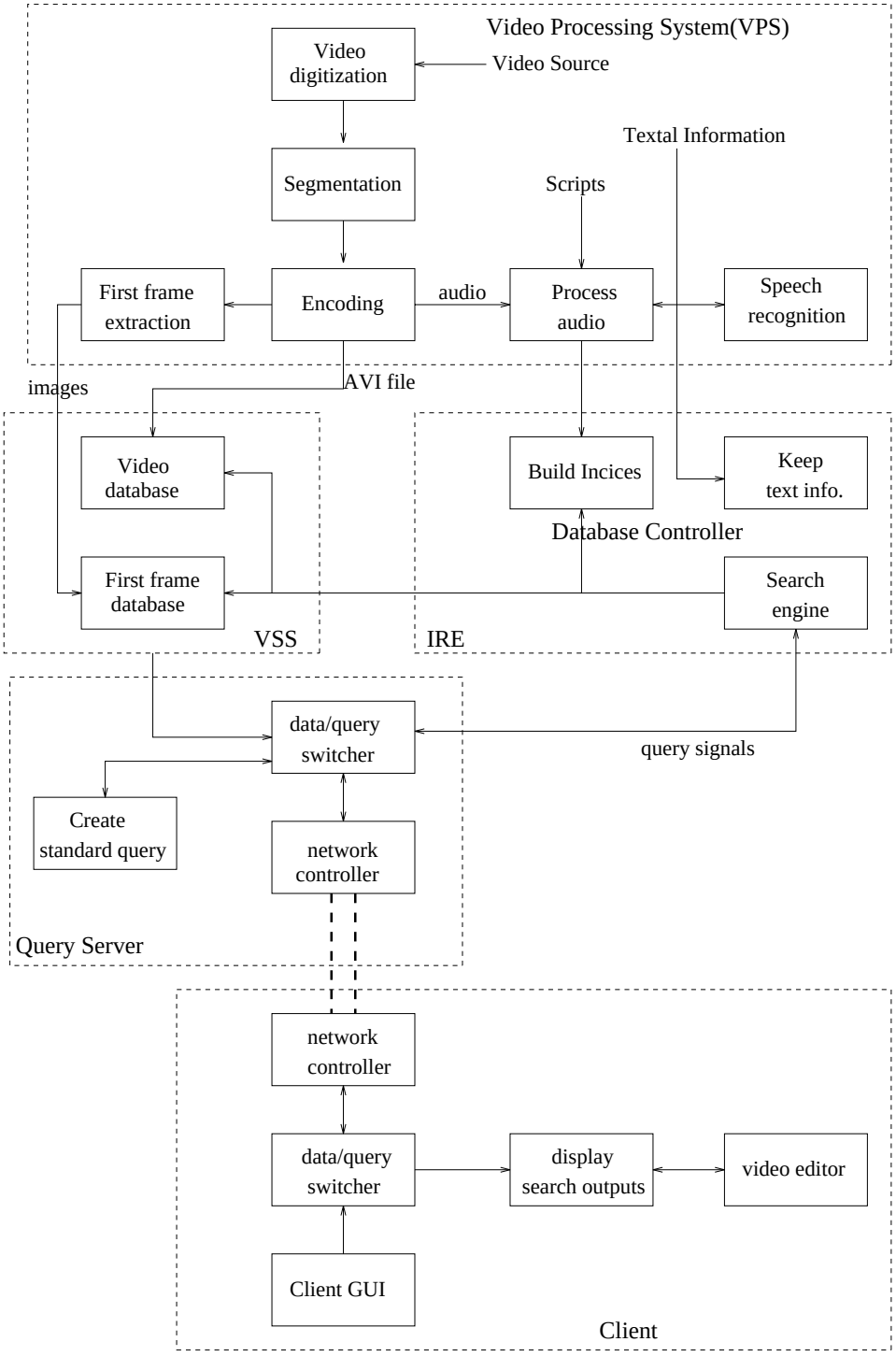

Figure 2.4: System Functional Flow Diagram

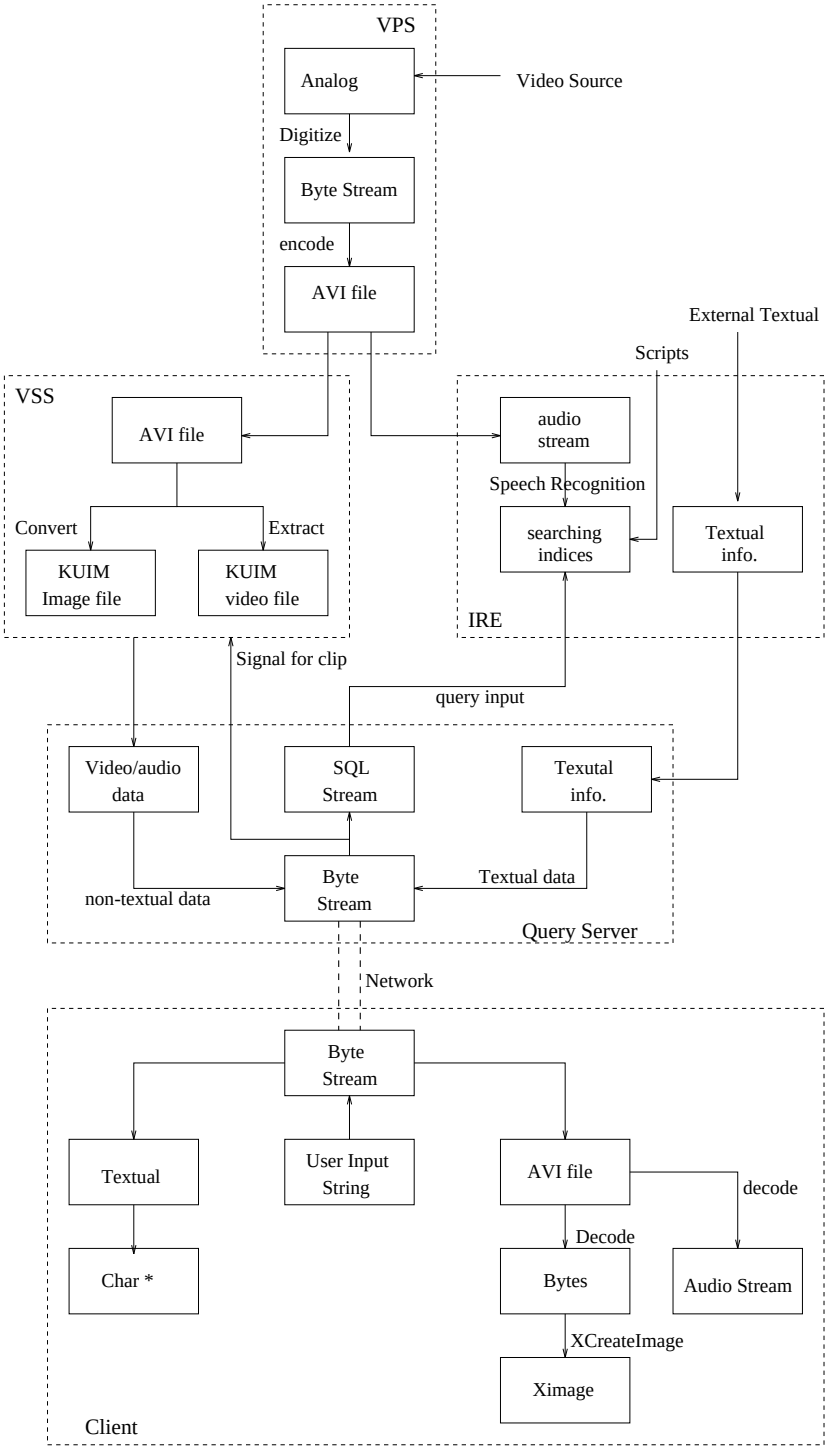

Figure 2.5: System Data Flow Diagram

internal control signals found in the system. The following is a description for each.

1. Signal to trigger the video processing unit when there is a need to update the database and searching indices. This control signal could be omitted from the control system because the video processing unit can be isolated as a stand-alone unit. That is to say, updating the database and searching indices can be done externally without any acknowledgement from the system itself.

2. Retrieval signal used to trigger the retrieval of the first-frame of a video clip in the video database.

3. Combination of query input signal and message processing signal. This is the main control signal from the Query Server to access the Information Retrieval Engine and the Video Storage System.

4. Retrieval signal asking for the delivery of a chosen video clip. This signal controls the transfer of a video clip from the video database through the network to the Client.

5. User input signals and searching control signals which will be the trigger source for either control signal 4 or 3.

6. Feedback or return control signals from the server. It includes error messages or return signals of the network system software package.

In the following few sections, the internal design of each primary component will be discussed in detail. The data structure design and algorithm design for each of these components will be discussed in greater detail in chapter 3.

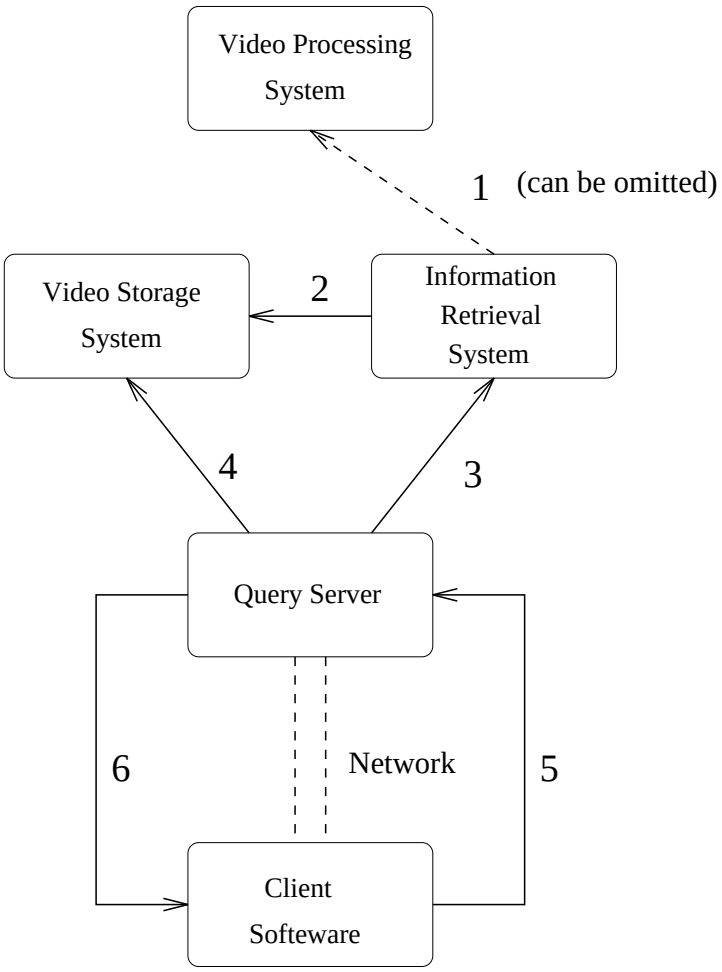

Figure 2.6: System Control Signal Flow Diagram

2.2.3.1 Video Processing System(VPS)

Because we are constructing a digital library, the source material must either be acquired in digital form or digitized from high quality video media. The main sequence of functions carried out by the video processing unit is digitization, segmentation, encoding, and speech recognition. An additional sub-system which performs the extraction of first-frame of each segmented video clips and saves them in KUIM image file format is added. These extracted images are used as icons to represent retrieved video clip in the client. Figure 2.7 shows the functional flow diagram of this video processing system.

In the encoding process, the Joint Photographic Experts Group (JPEG) proposal specifies an algorithm for compressing single frames based on the discrete cosine transform (DCT). This is a lossy compression algorithm which operates by discarding and quantizing DCT coefficients for each 8x8 (16x16) blocks of pixels in the image. By varying the degree of information loss (and image quality), JPEG can achieve compression rates between 5:1 and 50:1, or higher. Reasonable speed is achieved by partitioning of the image into 8x8 (0r 16x16) pixel block, and by using look up tables in the quantization and coding of DCT coefficients. The major drawback of this approach is the blockiness of decompressed images when too many coefficients are discarded. At compression rates of approximately 10:1, we expect JPEG compressed images to be almost original quality.

2.2.3.2 Video Storage System (VSS) and Information Retrieval Engine (IRE)

Since the functionality of Video Storage System and the Information Retrieval Engine are interrelated, it is convenient to treat these two systems as one unit

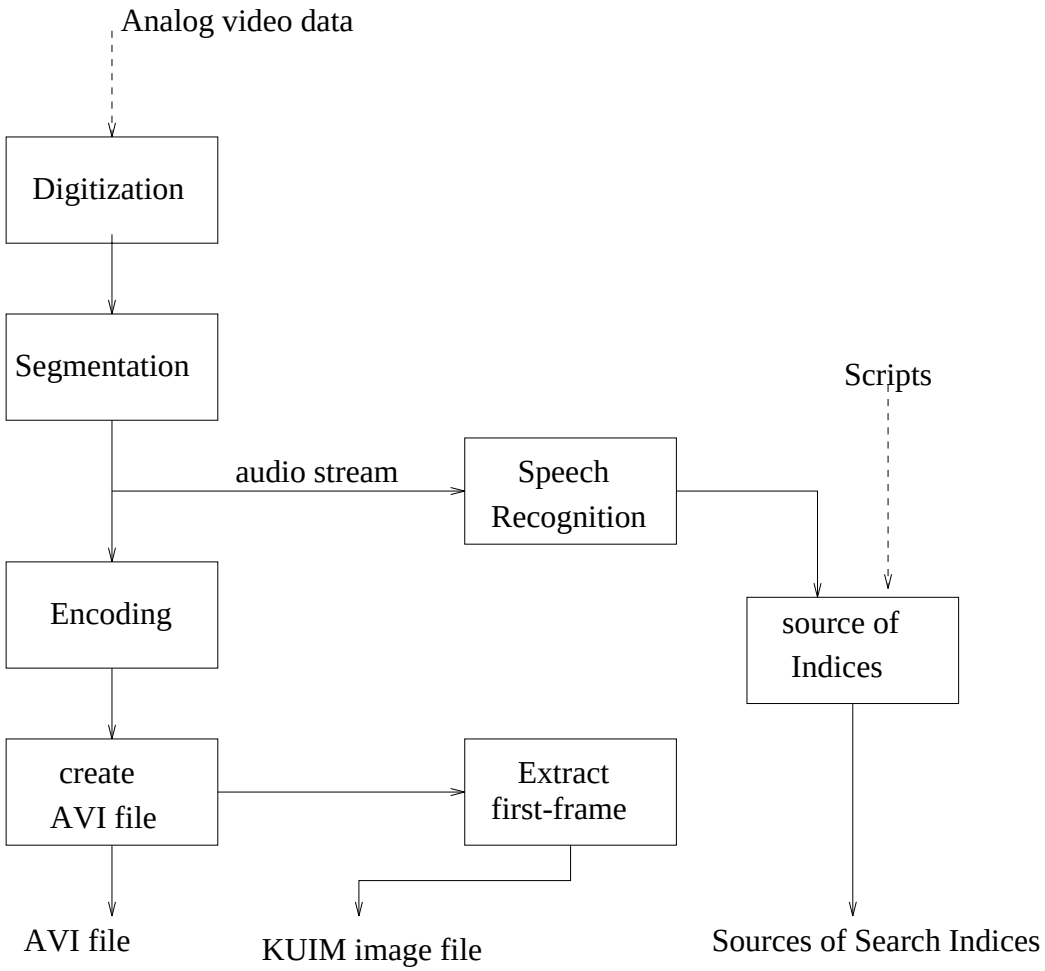

Figure 2.7: Functional Flow Diagram of VPS

in the design phase. Figure 2.8 shows the sub-system functional flow diagram for both the Video Storage System and the Information Retrieval Engine. The Video Storage System simply consists of two different subdirectories on a local disk directly mounted on the VISION server machine. One subdirectory is used to store digitized video clips in AVI format. The other subdirectory is used to store the icons representing each video clip. In VISION, we are dealing with four types of information. There are: images (icons), video clips, full-text, and structured textual data (DBMS). This data can be accessed through the database controller that resides in the Information Retrieval Engine. In other words, the Information Retrieval Engine serves not only as the information retrieval system controller, but also as the access to the information database. The indexing for the VISION will be based on audio information in the video which is extracted manually. The full system should index video segments based on speech recognition, scripts, and closed captions.

2.2.3.3 Query Server

The Query Server processes video queries from the Remote Client. It serves as the master controller to the Information Retrieval Engine and the Video Storage System. All signals from the Remote Client go through this Query Server first before they are distributed to their final destinations. Also, all the return signals, including the data stream from the database, pass through the Query Server before they are being transmitted to Remote Client through the network. Figure 2.9 shows a functional flow diagram of the Query Server. As can be seen from this figure, the Query Server is a signal controller and distributor between the Remote Client, the Information Retrieval Engine and the database.

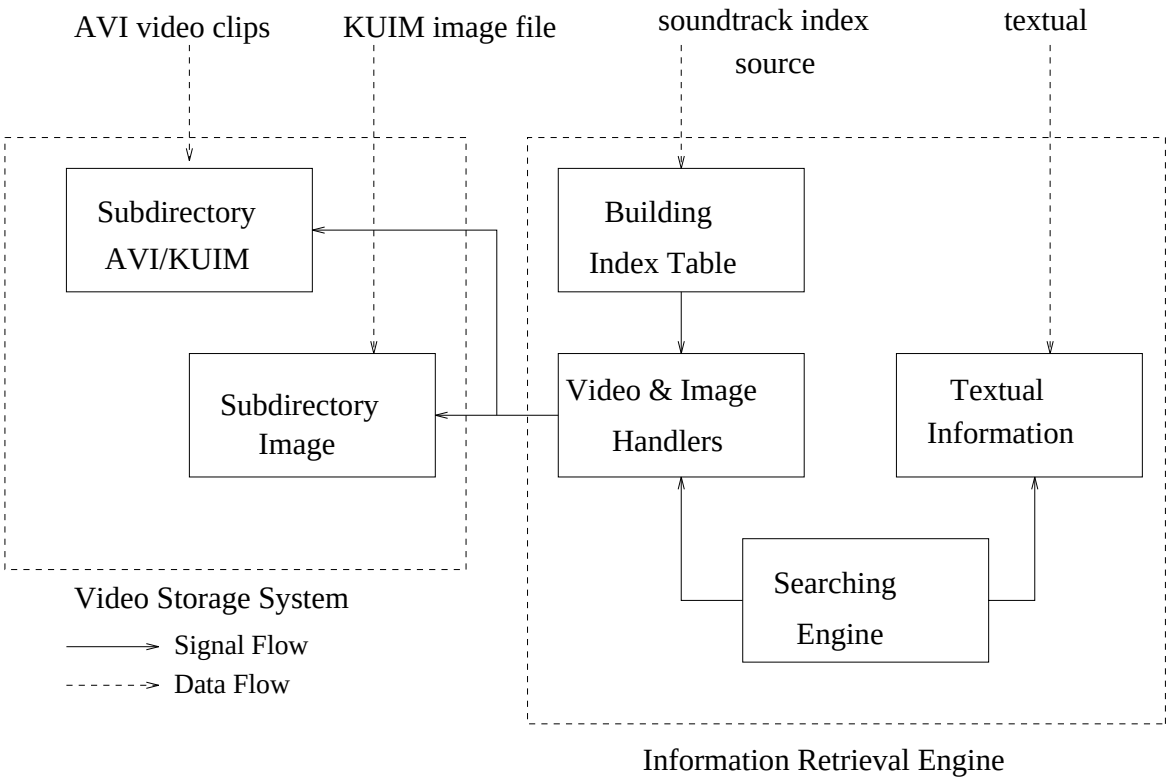

Figure 2.8: Functional Flow Diagram of VPS and IRE

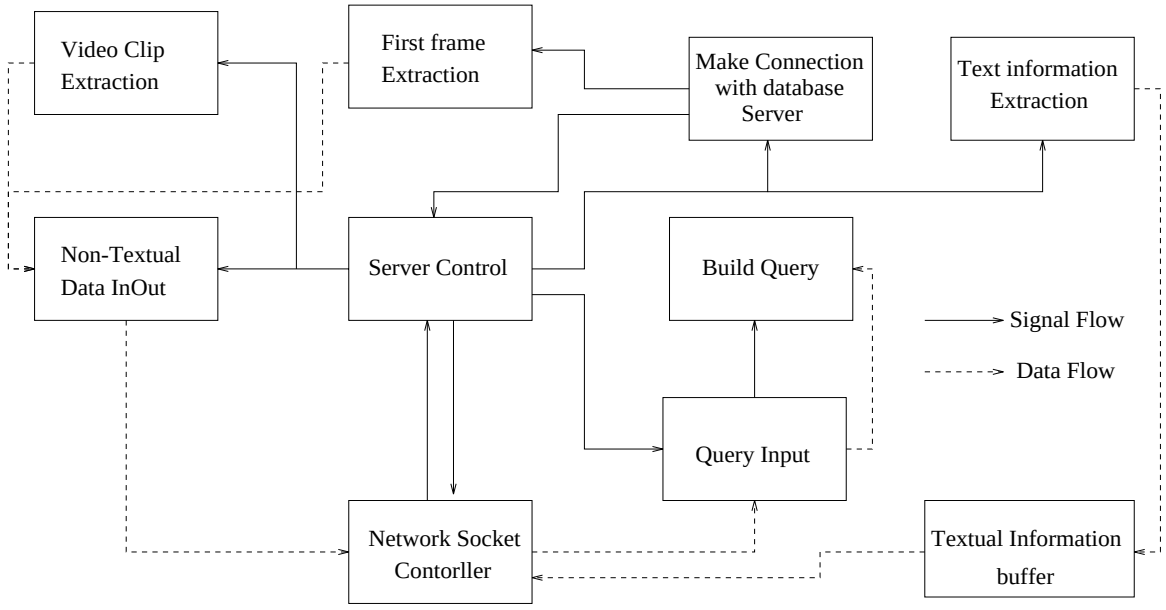

Figure 2.9: Functional Flow Diagram of The Query Server

2.2.3.4 The Remote Client

Figure 2.10 shows the system flow diagram of the Remote Client. This Remote Client consists of six primary sub-processes.

1. A network controller which sets up the communication between the Remote Client and the Query Server. It transmits the signals from the users to the Query Server and sends the received return signals to a signal distributor. In addition, it handles all the data flow from the database and transfers it to the correct location for storage and processing.

2. A signal distributor which accepts input signals from the user and forwards them to the network controller. It distributes the necessary triggers to activate different sub-processes.

3. A non-textual sources collector which receives video streams from the network controller and saves them in local disk.

4. A textual source collector that collects the returned textual information in local memory.

5. A video display and editing feature that saves video clips in KUIM or AVI video file format. A simple editing utilities such as selecting an image from a video clip is also provided by this video editor.

6. A client GUI that provides utilities for displaying search returns such as icons and textual information, and a graphical user interface for the Boolean search process.

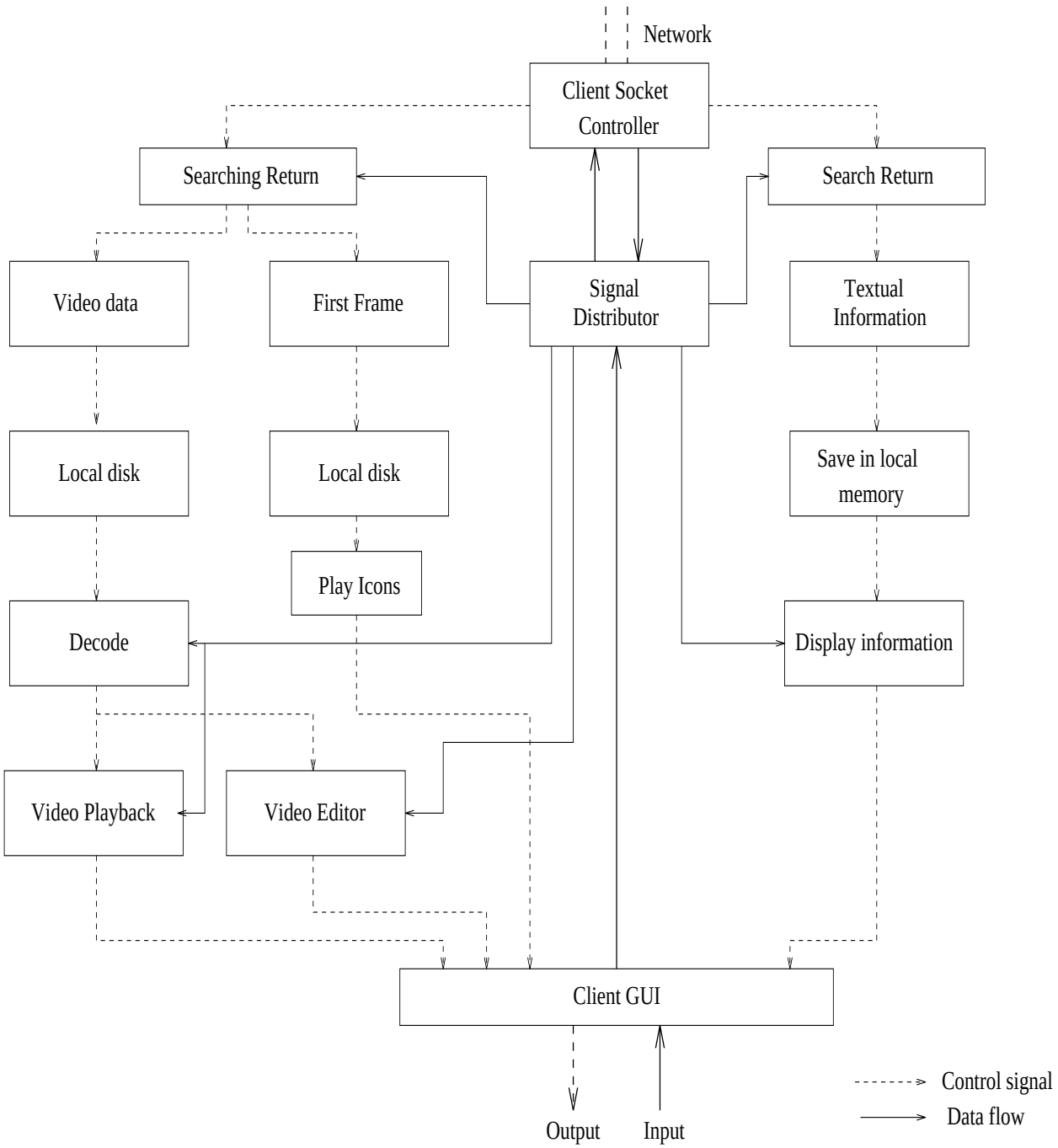

Figure 2.10: Functional Flow Diagram of The Client Software

2.2.3.5 Networking

The main function provided by the networking software is to handle different formats of data to transfer them over the Internet, and vice versa. A connection-oriented protocol called Berkeley socket is used to set up the communication between the Query Server and the Remote Client. Figure 2.11 shows the functionality provided by this network software. As can be seen from this figure, all the data is converted into a byte stream before it is read from or written to the socket. Hence, a socket with TCP/IP protocol which is the SOCK_STREAM type is created to handle this data communication. Once the socket is created and a connection is ready between the Query Server and the Remote Client, all the data flow through this connection is controlled by the Query Server on the Server, and by the client software on the Remote Client. The implementation of the socket will be discussed in detail in the Chapter 3.

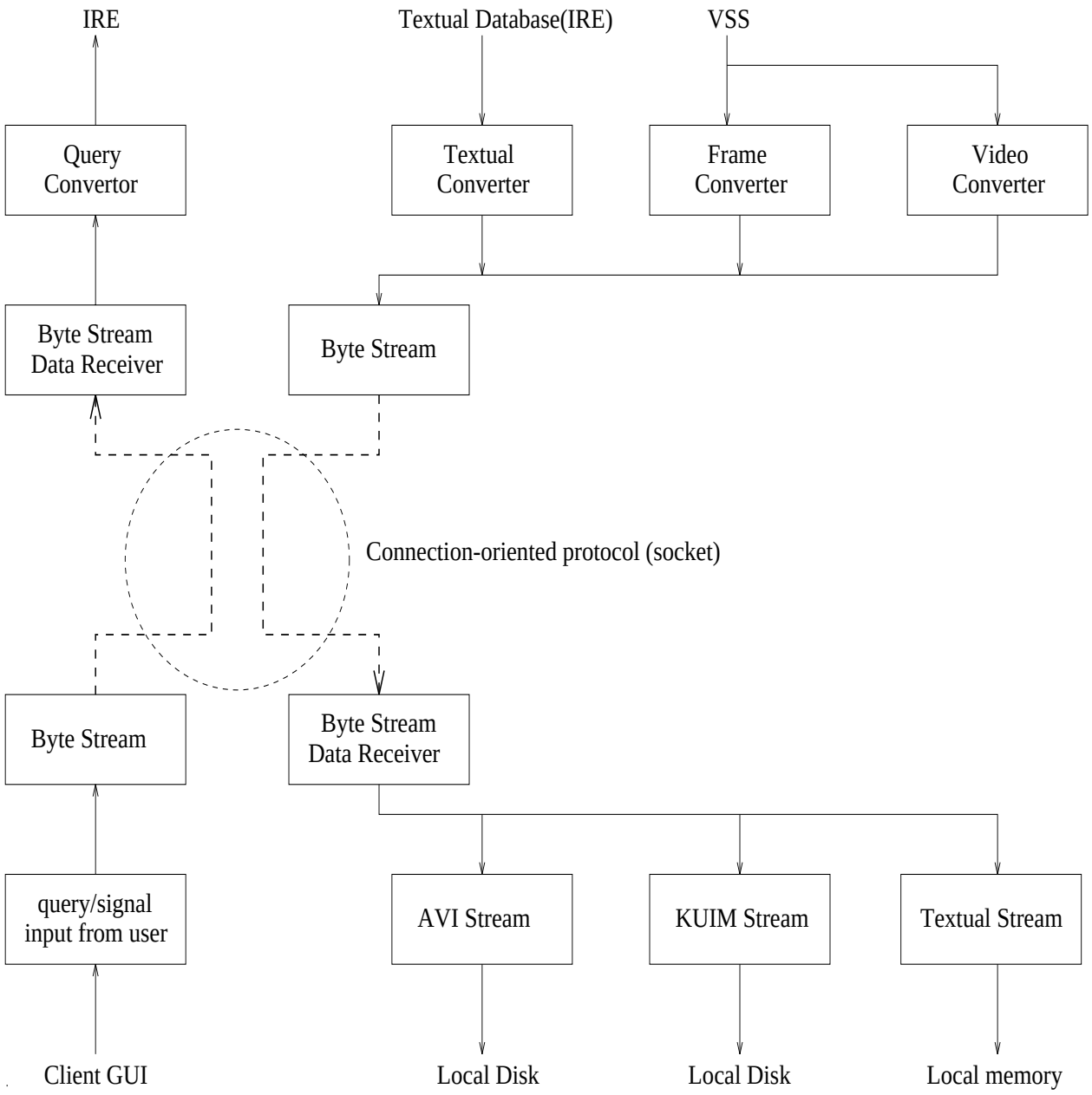

Figure 2.11: Functional Flow Diagram of The Networking Software

Chapter 3

Implementation

This chapter focuses on the implementation of each software component discussed in the previous chapter. The design of each component is refined in the implementation process in order to suit the system requirements and to simplify the implementation process of the prototype.

3.1 Video Processing System (VPS)

Figure 3.1 shows the working block diagram of the video processing module. All the functional blocks designed in the previous chapter have been refined to their corresponding functional implementation in software or hardware. To digitize video and audio, we chose the J300 video board mounted on an Alpha machine, together with a software package called Multimedia Services for DEC OSF/1 AXP Software. This software helps developers create multimedia end-user applications and enhances existing applications using the Sound and Motion J300 option module.

The process of digitizing the audio and video source, segmenting and creating

the AVI file for each video clip was implemented by Wei Li, a post doctoral researcher. Hence, in term of designing and implementing the Video Processing System, this thesis only covers the process of first-frame extraction, soundtrack processing, and preparing the one-hour of video data using Dr.Li's programs.

3.1.1 The Sound and Motion J300 Video Board

The Sound and Motion J300 option module permits the users to add multimedia capabilities to the Alpha AXP system by installing one two-board TURBOchannel option module. The Sound and Motion J300 video board also allows developers to receive and transmit video and sound using laser-disc players, video cassette recorders, video cameras, speakers, and more. It was chosen for our project because it can:

- handle real-time video capture of video signals in NTSC (640 x 480), PAL, or SECAM (768 x 576 pixels) format.
- support full-frame video data storage.
- perform JPEG compression and decompression of the video signal in real-time.
- scale and filter the video signal before compression.
- dither from 24 bits/pixel to 2-to-256 colors.
- capture video (video-in) in 4:2:2 YUV format, in 8-bit pseudo-color or 8-bit grayscale.
- sample multiple audio at rates from 8-bit (8K Hz) to 16-bit

For prototyping purposes, we are digitizing the video in real time using 8-bit pseudocolor format JPEG compression. The compression ratio to be achieved will

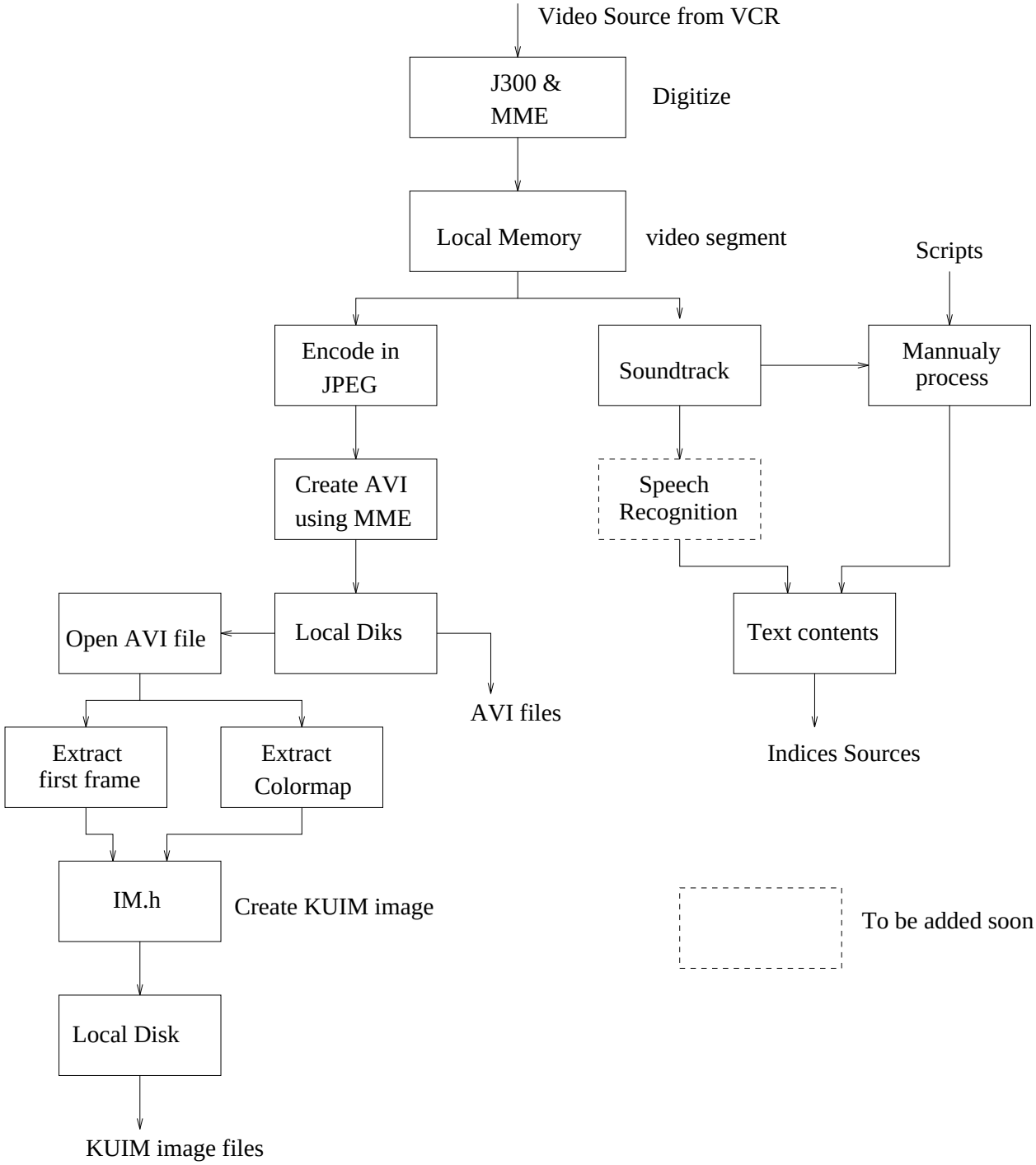

Figure 3.1: Implementation Outline of Video Processing Module

be around 1-to-10 and 1-to-50. There will be another J3000 video board mounted on the client machine in order to support the decompression process in real-time. The J300 requires an Alpha machine, with a minimum memory of 32 MB running the DEC/OSF1 operating system Version 1.3 or higher.

3.1.2 Multimedia Services for DEC OSF/1 AXP Software

Multimedia Services for DEC OSF/1 AXP is the multimedia audio and video enabling software for DEC 3000 Alpha AXP systems. It is Digital's standard application programming interface (API) to multimedia functionality on Alpha workstations. It provides a set of software services and tools for developing applications that use multimedia features on these systems. Multimedia Services for DEC OSF/1 AXP is designed to minimize the device-specific code that application designers must include in their multimedia application programs. It includes low-level device drivers, device-independent run-time services, and the API. The API and file formats are compatible with multimedia programming interface defined by Microsoft for Window and Window NT. This compatibility eases porting of multimedia applications from Windows platforms to Alpha systems and vice versa by reducing the programmer's learning curve and the coding changes required for a successful port.

Multimedia Services for DEC OSF/1 AXP has a client/server architecture that uses shared memory transport to communicate between the client applications and the server controlling the multimedia hardware. The multimedia services include the following:

- **Waveform audio recording and playback**

An application uses waveform audio services to manage the recording and playback of waveform audio. The primary waveform audio functions give an application the ability to record audio (encode audio signals and capture digitized audio in buffers) and to play audio (generate output audio waveforms from digitized audio contained in buffers). The waveform audio services also include supporting functions that allow an application to query device capabilities and to open, close, and configure devices.

- **Video capture and playback**

  An application uses the low-level video services for video capture and playback. The video capture services allow an application to capture incoming video signals, either single frames or real-time streams, in memory buffers.

- **Video compression and decompression**

  An application uses video compression and decompression services to compress or decompress video data. These services provide access to multimedia hardware and software support for JPEG compression and decompression and for dithering on the Sound and Motion J300 video board. The video compression and decompression services also include functions for managing devices and for controlling the compression parameters that affect the quality of compressed video image data.

- **Multimedia file I/O**

  An application uses file I/O services in conjunction with the other multimedia functions to create and write disk files to store audio and video data. The file are in RIFF[1] file format which includes the Waveform Audio (WAVE) file format and the Audio/Video Interleaved (AVI) file format.

[1] Microsoft Resource Interchange File Format

3.1.3 Processing Soundtrack For Indices Source

For this prototype, the text content for each video clip, which is used as the indexing source, is manually entered based on the scripts provided. If there is no scripts provided, the text content is created by watching to the video clip and transcribing its soundtrack. A speech recognition software package will be installed in the video processing unit in the near future to recognize the soundtrack of each video clip. However, the manual transcription will be a valuable resource to measure the accuracy of the speech recognition process.

3.1.4 Extracting An Image From The Video Clip

A program written in the C language uses MME[2] service routines to extract the first frame of each video clip in AVI file format to be used as an icon at the Remote Client. The size of the extracted frame is 100 x 100 pixels. Figure 3.2 shows the flow diagram of internal functional calls in the extraction program. The main sequence of functions in this program is open an avi file, set up the header, get the first frame chunk and colormap, decode them, and finally store them in a 100x100 pixels KUIM image file.

3.2 Video Storage System (VSS) and Information Retrieval Engine (IRE)

The Video Storage System (VSS) stores digitized video and audio in disk files. These are indexed by a table in the database used by the Information Retrieval Engine. The VSS has a total of 9 gigabytes of disk space which is more than

[2]Multimedia Services for DEC OSF/1 AXP

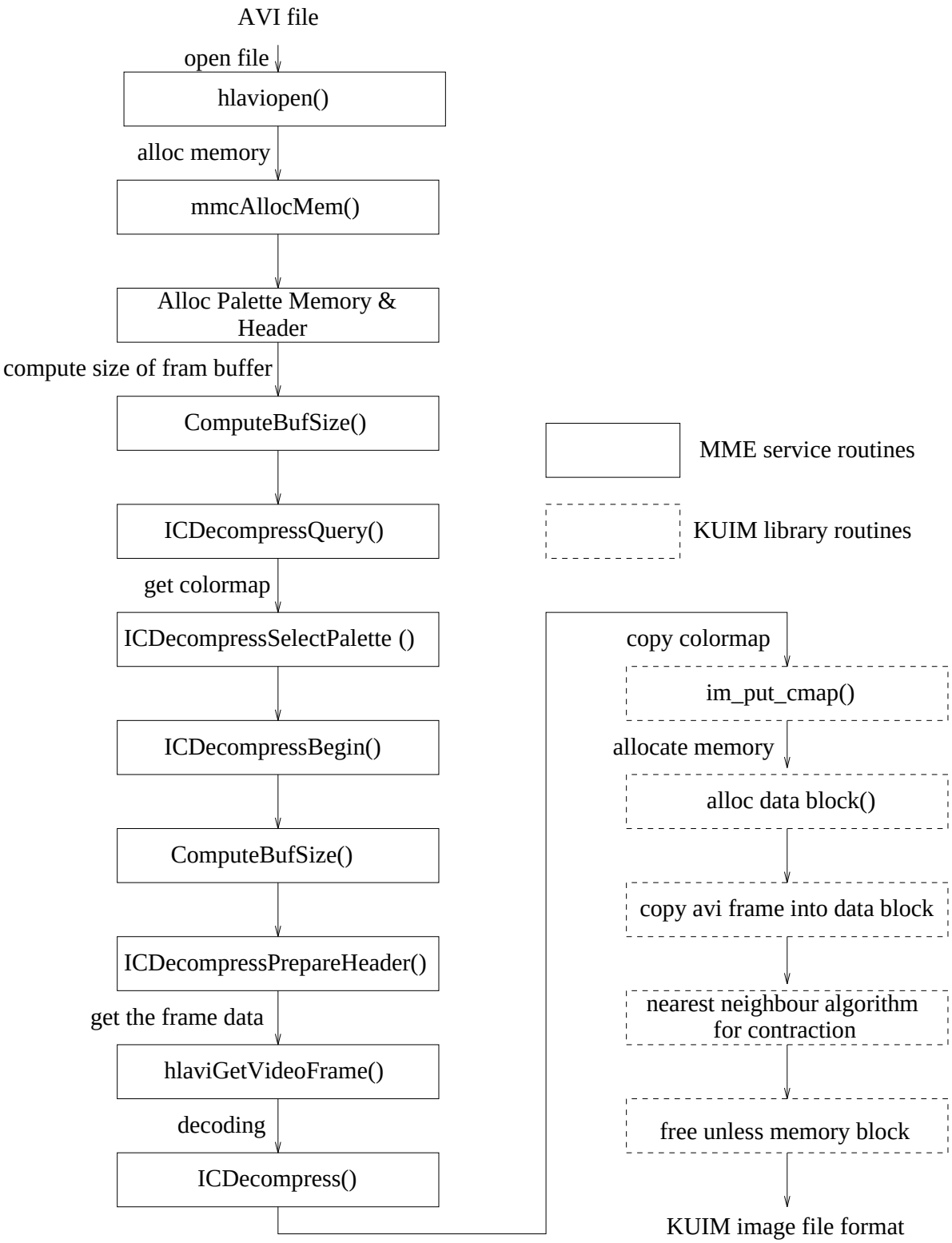

Figure 3.2: Internal functional calls of the extraction program

sufficient to store one hour of video and audio data. The Information Retrieval Engine, together with the Query Server, has the full control of this video storage system which is in fact consists only of a subdirectory. Video and audio data are stored in AVI file format, identified by a unique filename.

The IRE consists of a search engine and a database which is used to keep the search indices, the id (unique filename) of each video clip and its icon in the VSS. The IRE also stores the textual information and structured data which are the search indices for each video clip. Figure 3.3 shows the flow diagram of the internal structure of each sub-component found in both VSS and IRE.

3.2.1 Video Storage Unit

The basic video storage system consists of a 9 gigabytes of disk space which is directly mounted on the server machine. For real time video playback (30 frames per second), each frame consists of 480 rows and 640 columns of color pixels. With 2 bytes per pixel, the size of an hour of uncompressed video data will be around 66.2 GB. For the audio data, we will be dealing with a sample rate of 8000 samples per second, where each sample is 8 bit in Mu-Law format. Hence, the total size of an hour of audio data is close to 0.2 GB. By applying a compression ratio of 10:1 to both the video and audio, we need 6.64 GB of disk space for a full hour of compressed video and audio data. Therefore, a size of 9 GB disk is sufficient. Of course, there is a tradeoff between compression ratio, image quality and amount of disk needed which must be evaluated on an application by application basis.

The only task of this video storage system is to store video clips, generated in VPS, in AVI file format and their first frame images in KUIM file format.

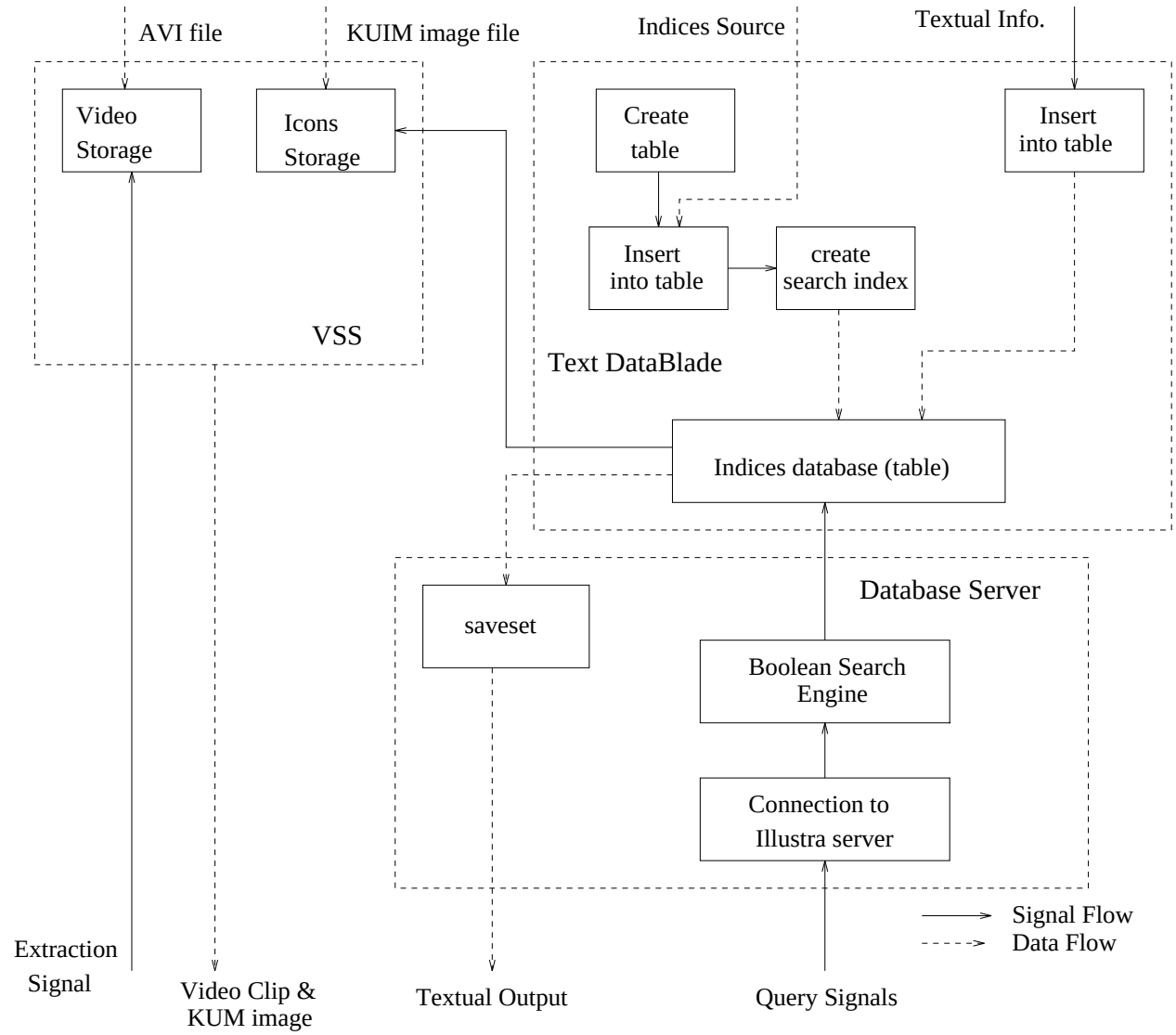

Figure 3.3: Internal Structure of the VSS and IRE

Each AVI file and KUIM file has a unique filename which is stored in a textual database at IRE. In the other word, these video clips and images are identified by their unique filenames.

3.2.2 Information Retrieval Unit

The information retrieval unit consists of two parts: 1) a textual database and 2) the database server/the search engine. In this prototype, an Object_Relational DBMS[3] called Illustra is used as the database system to index both structured and free-text information. The basic system can store, index, and retrieve data. An optional extension to this is the Text DataBlade, which has the capability to store full-text documents, build search indices and perform Boolean search on their contents.

Creating Video Database Using Illustra Database System

The Illustra Database is an object-relational database management system. The architecture of this ORDBMS is shown in Figure 3.4. ORDBMS combines the data modeling advantages of ODBMS and RDBMS to provide good performance in terms of speed and data handling. The features of this ORDBMS include the following:

- Table-driven database system
- Extensible with user-defined datatypes, user-defined functions, and new access methods.
- Able to add new datablades with user-defined new datatypes.
- Unique object identifiers (OIDs) which can be used for indexing.

[3]Database Managements System

The Illustra database is based on Client/Server system architecture, in which the server spawns processes for each client connection. Thus, an Illustra Server is capable of supporting multiple clients simultaneously.

The video database for the VISION prototype consists of two subdirectories and the Text DataBlade module, which is an optional datablade in the Illustra Database System. The first subdirectory is used to store files of video clips which are indexed by unique filenames. The second subdirectory is used to store image files of icons for each video clip stored in the first subdirectory. The files of icons are identified by assigning them the unique filename for each icon. The Text DataBlade module in the Illustra Database System is used to create a table which is used to store search indices information of the video database. Each row of the table stores the search index information for a video clip. This search index information includes the filename and the path of the video clip and icon, the size, the scripts and etc. The description of this table will be covered in the following section. The Boolean searching will be performed on this table.

Building Search Indices Using Text DataBlade

The Text DataBlade module, which is one of the optional datablades in the Illustra Database System, defines and supports a single data type called *doc*. The *doc* data type stores a document in an object of the *large_object* data type that is part of the basic Illustra system.

A table stores document text in columns of the *doc data* type. Figure 3.5 represents the structure of the table created in the Text DataBlade. It also shows the link of each column entry, which is the textual data of each video clip, to

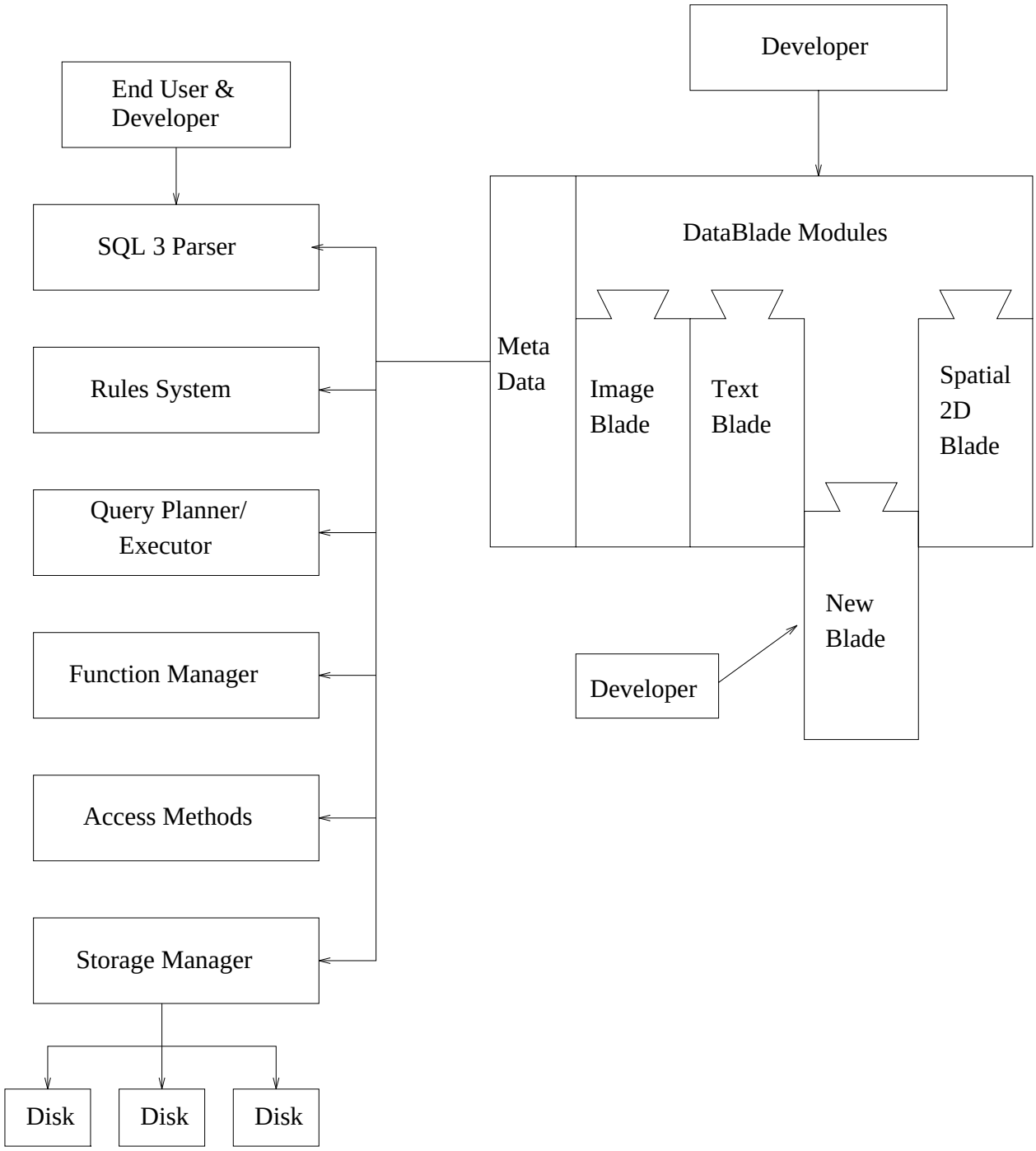

Figure 3.4: Illustra Object-Relational DBMS Architecture

its location of video clip and icon in the Video Storage System. The following SQL statement creates the table named *invertfile* in which each row contains a text column called *id* for the filename of a video clip with a few extra columns containing textual information about this video clip. There is also a *doc* column called *doc* which stores the full-text document of each video clip. This *doc type* column is used to create the index table, on which the search index will be based.

```
create table invertfile (
id text,
size int,
frame int,
copyright text,
format text,
audio text,
doc doc );
```

The following SQL statement inserts textual information and its search document into a row of table *invertfile*:

```
insert into table invertfile values (
'winter.IM',
617852,
90,
'CNN',
'KUIM',
'/users/winter.au',
'/users/winter.doc' );
```

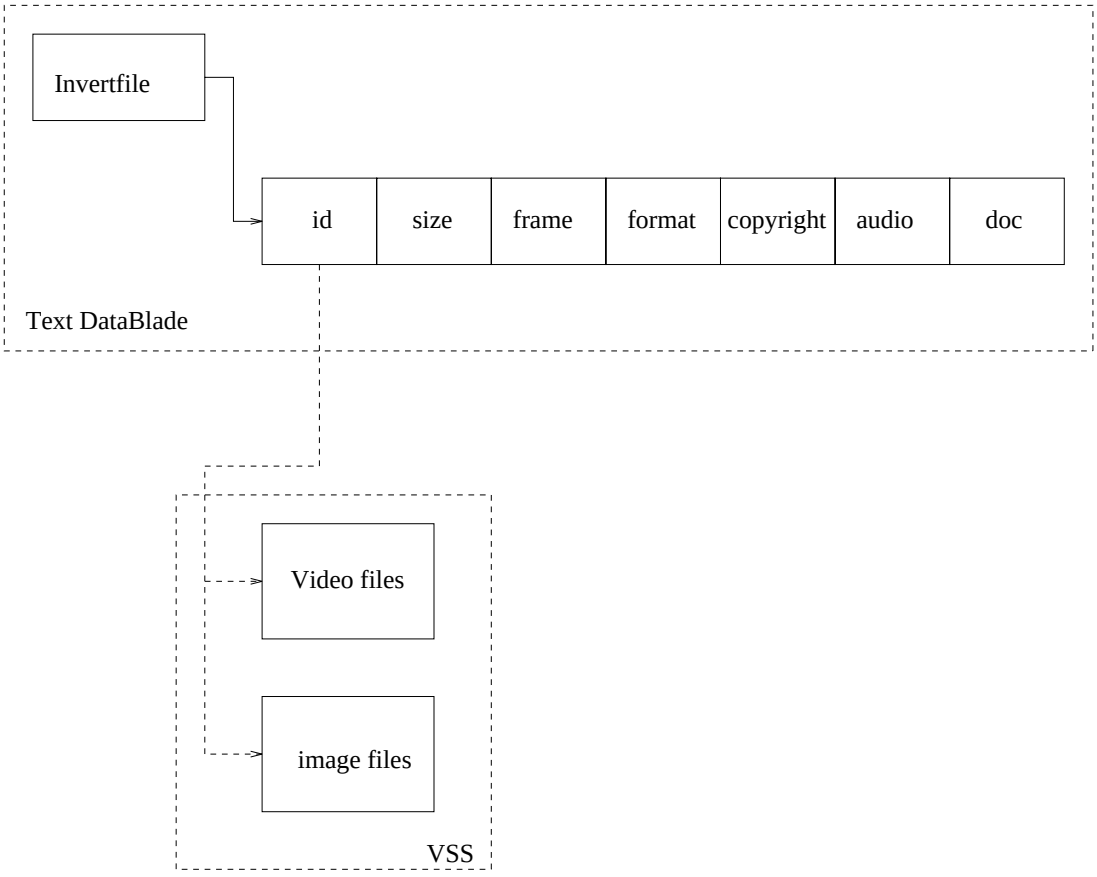

Figure 3.5: Index Table Structure Organization

The most powerful feature of the Text DataBlade is its support of text searches through its *dtree* access method. Before the text search feature can be used, a *dtree* index must be created on the *doc* column that stores the documents to be searched. *Dtree* indices are only available with the Text DataBlade. The syntax for creating a *dtree* index is:

create index index_name **on** table_name
using dtree (column_name[op_class])

The olny *op_class* that is applicable to a *dtree* index is 'text_ops'.

When a document is inserted into the table as an object of the *doc type*, the file is parsed into individual words. The Text DataBlade then looks for the stem, or root, of each word in the Word Net tables which are provided as part of the Datablade module. The Text DataBlade is case-insensitive. A list of the documents' stemwords is compiled. After the words have been stemmed, they are compared to a list of stopwords contained in the *stopwords* file, also provided as part of the Text DataBlade. If a word is a stopword, it is removed from the list of stemwords. The stemwords that survive this elimination process are stored in the large object that contains the document, following the document's text. This list of stemwords forms the basis of the document's *dtree* index which is then used for text searches of documents based on their contents. In our system, each document describes a specific video clip.

Text DataBlade Boolean Search Function

The Text DataBlade search function that is used for Boolean search is called **Contains()**. The syntax of this search function is **Contains(doc, text)** which returns Boolean. The text argument is an *input string* that contains the word or words being searched for and the *doc* argument is the specific document id. **Contains()** returns whether or not the words specified by an *input string* are contained in a specified document. It is typically used in a search condition to return a list of documents that match the Boolean expression in the *input string*. Documents are returned in descending order of how well their contents match the *input string*.

The following SQL statement is used for searches of documents which leads to the return of video clip's id:

select * from invertfile where Contains(doc,'*input string***);**

Figure 3.6 shows the sequences of steps on how a query input and its return result is being processed by the IRE. In this figure, all the routines whose names start with a **"mi"** phrase are built-in functions in Libmi.c standard library provided by Illustra System. Libmi allows user-written procedures access Illustra database system.

3.3 Query Server (QS)

The query server software component controls the data and signal traffic between the Remote Client and the Information Retrieval Engine. It serves as the gateway between the database system and Remote Client. A Remote Client first establishes

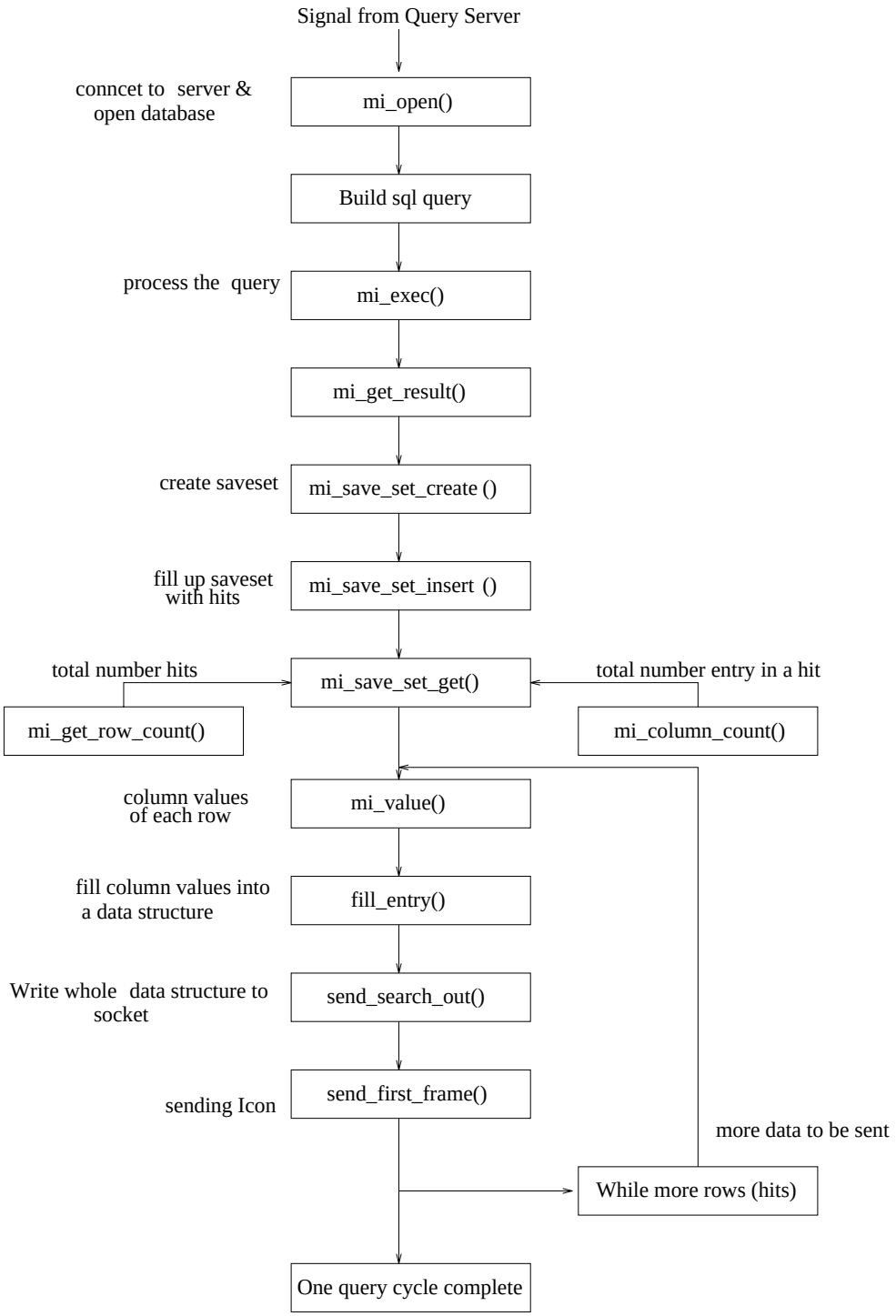

Figure 3.6: Flow Diagram of Query Input and Output Processing

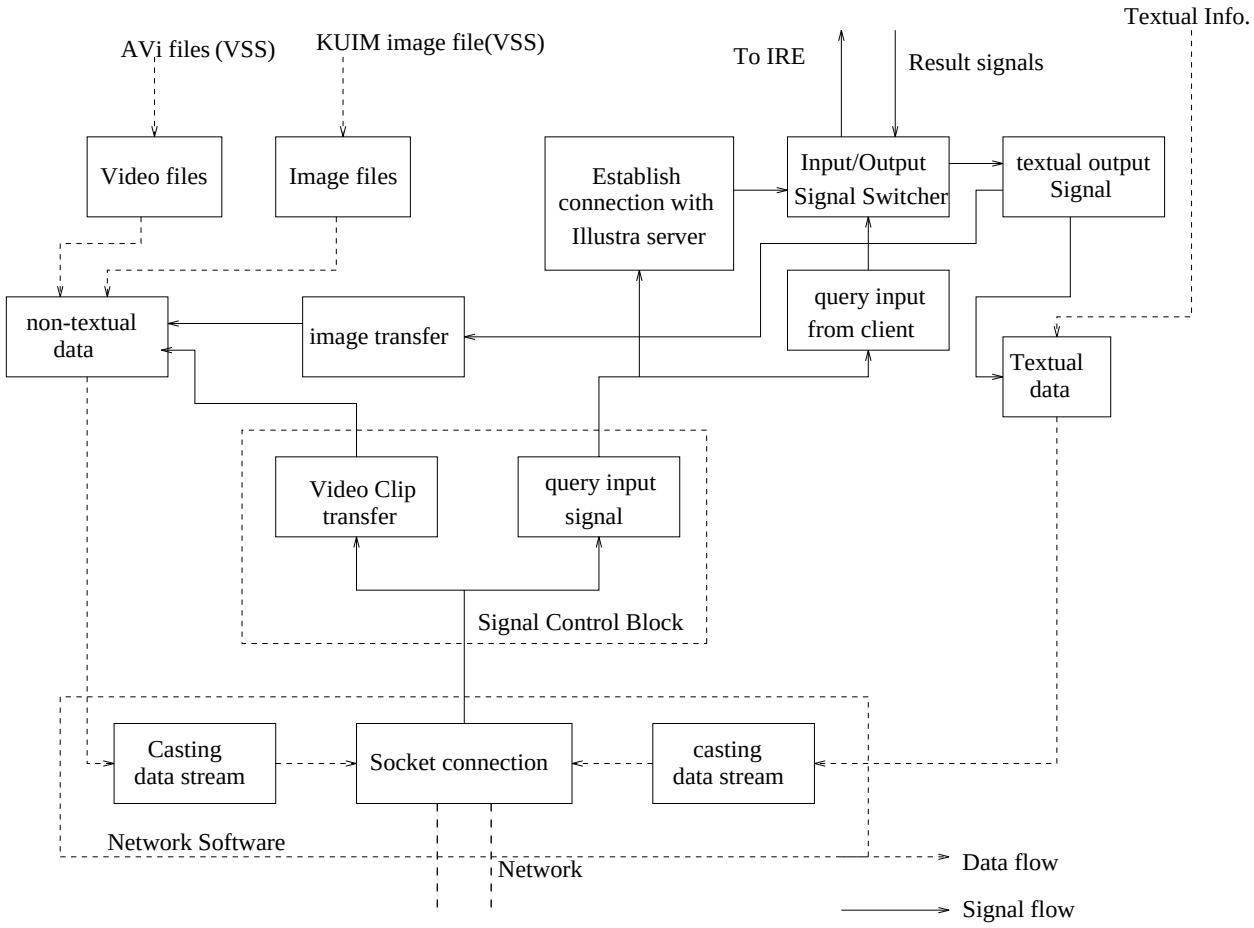

Figure 3.7: Function Block Diagram of the Query Server

a connection with the network software residing in the Query Server before it communicates with the IRE or VSS. All the signals from Remote Client come to Query Server first before they are forwarded to their corresponding destinations. Another important task of Query Server is to establish a connection with Illustra Database Server as needed in order to perform Boolean searches. Figure 3.7 shows function block diagram of the Query Server. As can be seen in this figure, all the data and signals are converted into byte streams before they are sent over through the network. The network software in this query server is a connection-oriented protocol implemented using Berkeley socket. This server socket listens at a specific port number, waiting for a connection from a remote client.

3.4 Client

As discussed in the previous chapter, the Client consists of six major sub-components. The implementation of each sub-component is included in the following:

3.4.1 The Network Software

This is the client network software implemented using a Berkeley socket. This connection-oriented protocol handles the data transfer in streams of bytes. Its major function is to establish a network connection to the Query Server for searching activities and data transfer. The implementation detail of this network software will be discussed later in section 3.5.

3.4.2 The Signal Distributor

This is a signal controller that distributes signals generated from the client GUI to their corresponding processes. It collects the return signals from Query Server and forwards them to the appropriate sub-component as a trigger signal. In terms of C source code, this signal controller is implemented as **case**, **loop** and/or **if** control statements which checks a given input and call for a specific function if it matches and fulfills the required condition. Although not a separate module, the signal distributor was treated as an entity in the design process to help us identify all the necessary control structures.

3.4.3 The Textual Source Collector

This textual source collector reads all the textual information relating to a video clip, one hit (or retrieved clip) at a time. This information includes the filename and size of the video clip and image which are used for video data file transferring

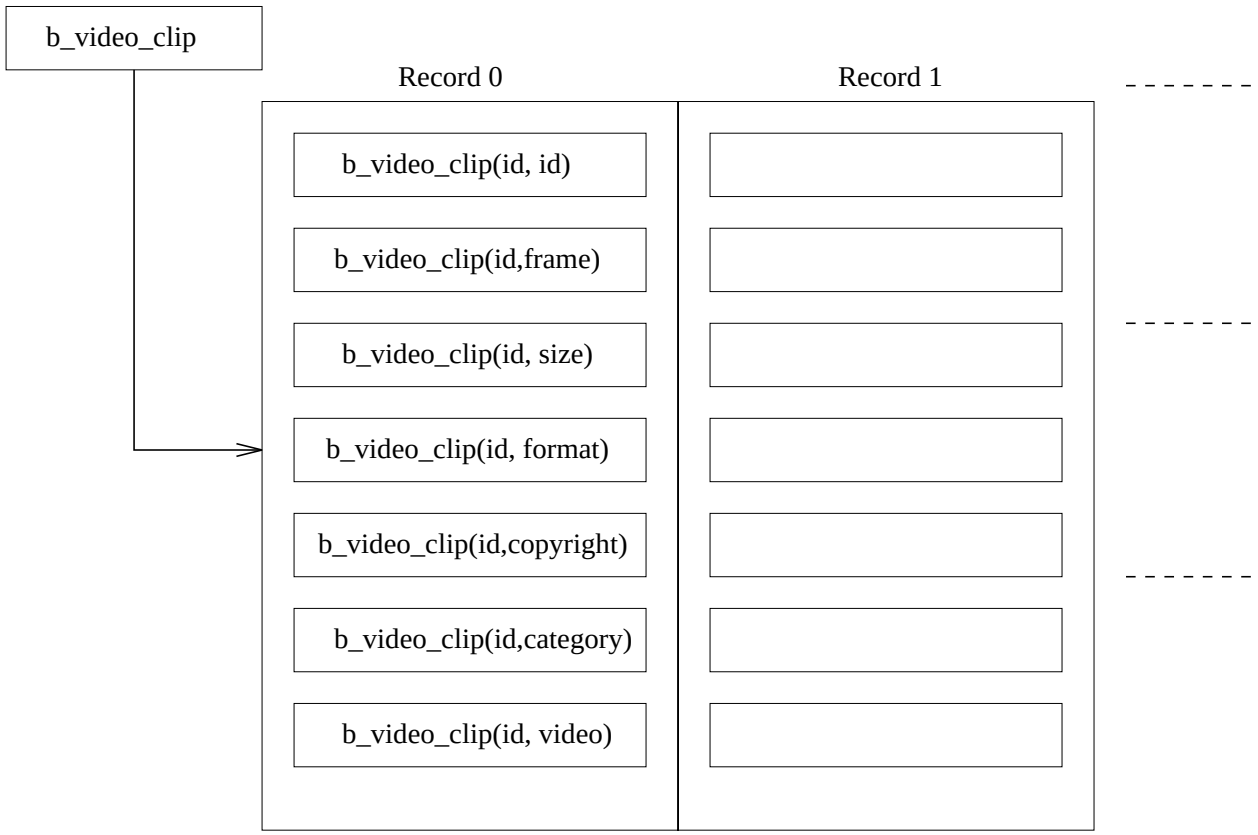

Figure 3.8: The Textual Information Array in Tcl/Tk environment

purposes. The textual information array is implemented as an array of records, with each record storing the information for a specific video clip (See Figure 3.8). One field of the record indicates whether or not the video clip is already cached on the local disk.

3.4.4 The Non-textual Sources Collector

This sub-component performs two functions: 1) it receives data streams of video clips from the socket and 2) it receives data streams of images from the socket. As shown in Figure 3.9, the program first reads the filename which identifies a video clip or an image. Then the size of the file is read from the local array discussed in section 3.4.3. The program then reads the required number of bytes from the

specified socket and writes that data into a newly created file. In other words, this is a simple file transfer process from the server to the client.

3.4.5 The Video Display and Editing Feature

Our goal on video playback is to make sure that the video can be played back at a near real-time rate (say 25-30 frames per second). In order to achieve this playback rate, it is necessary to use hardware to decompress the video and audio data. The playback rate using software is around 10-20 frames per second. To achieve the real-time playback rate, the J300 video board with Multimedia Services for DEC OSF/1 AXP are used to decompress the video and audio data and map the decompressed video data into an Ximage. The Ximage is then displayed using XLib routines which are implemented as low-level code used by Tcl/Tk video widgets. Concurrently, the decompressed audio is sent to the audio playback driver which converts digitized audio data into an analog signal that is forwarded to speakers.

Mechanism of Audio and Video Playback

The idea of video frame playback is to decompress one frame of video data at a time using hardware or software. The frames are them mapped into Ximages for display. At the same time, audio frame playback requires a more complex mechanism. The audio frame is first decompressed using hardware or software and then stored in an audio buffer which can take up to sixteen frames of decompressed audio data. The decompressed audio data is concurrently read from this buffer by an audio playback driver which converts digital audio into an analog signal and sends it to the speakers. The fetching of audio data from the audio buffer into audio playback driver is considered continuous as long as there is new audio data

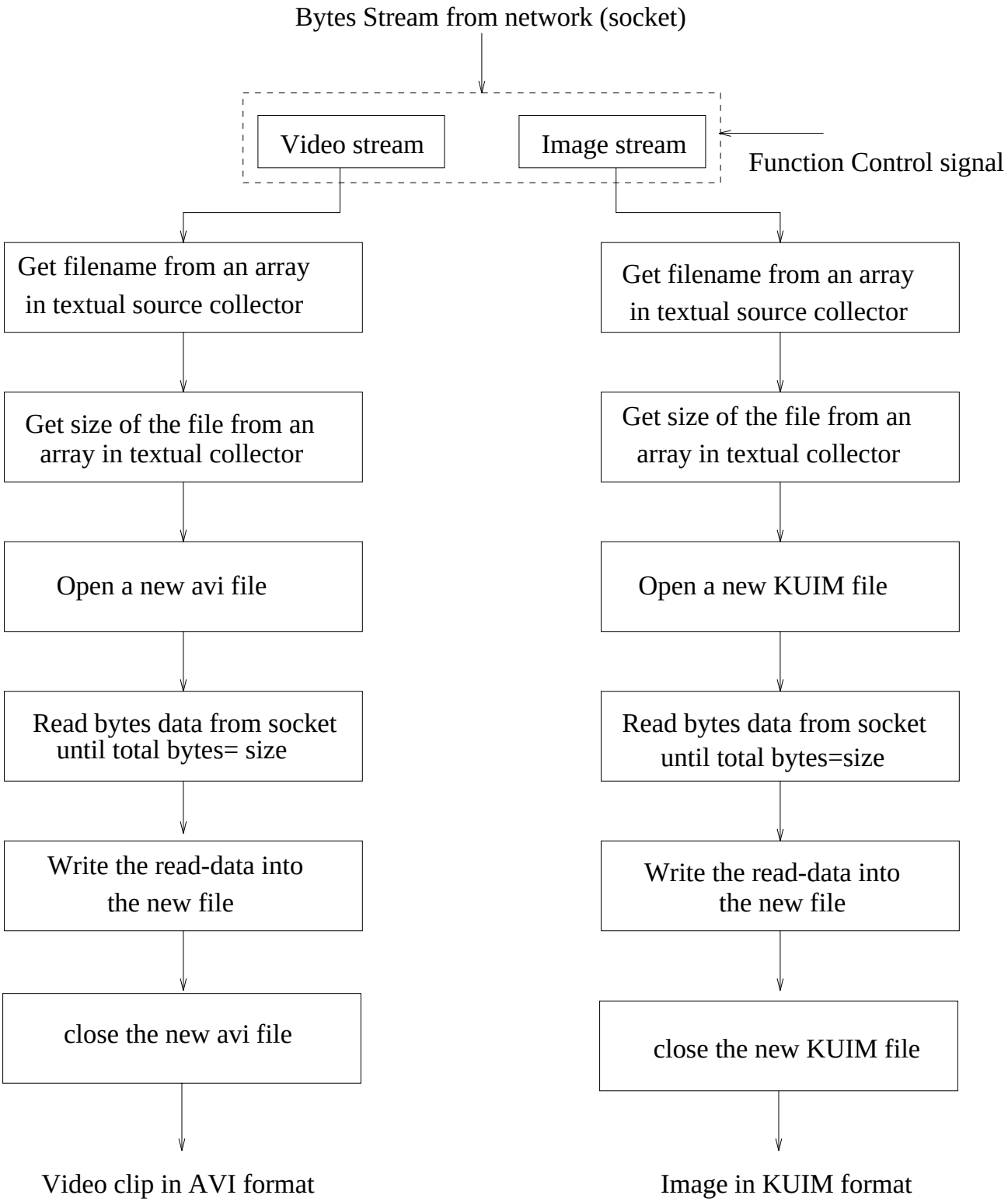

Figure 3.9: Steps of Processes of the Non-Textual Collector in the Client Software

available in this audio buffer. A discontinuous audio playback occurs when the audio buffer runs out of data. This produces a discontinuous analog audio stream which is noticeable by human ears. Hence, it is desirable to make sure that the audio buffer always has data available for the audio driver. This is achieved by having a large buffer size and a continuously running process to fill the buffer. The only drawback of this practice is the limitation of the memory found in certain client machines.

The video and audio playback mechanism is shown in Figure 3.10. This figure shows that we are using the audio stream to control the rate of video frame playback. Since there is a lag between audio playback and video frame mapping into Ximage format, a few frames of video data are skipped occasionally to synchronize the video to the audio. The video is allowed to lag a maximum of 6 frames (0.2 second of a second) behind the audio. When this maximum is exceeded, the video stream skips ahead to resynchronize with the audio stream.

The video editing feature includes taking a snapshot of an individual frame of the playback video and storing it in KUIM image file format. The user can also select their favorite video clip and store it in either AVI or KUIM format. The video playback widget can be resized for larger or smaller video clips. The maximum widget size that can be handled is 678x960 pixels.

Data Transfer Mechanism Over The Internet

This VISION prototype transfers video data in AVI files. That is to say, video data is transferred from the database to the remote client as video data files which are decompressed at the remote client using hardware or software for playback.

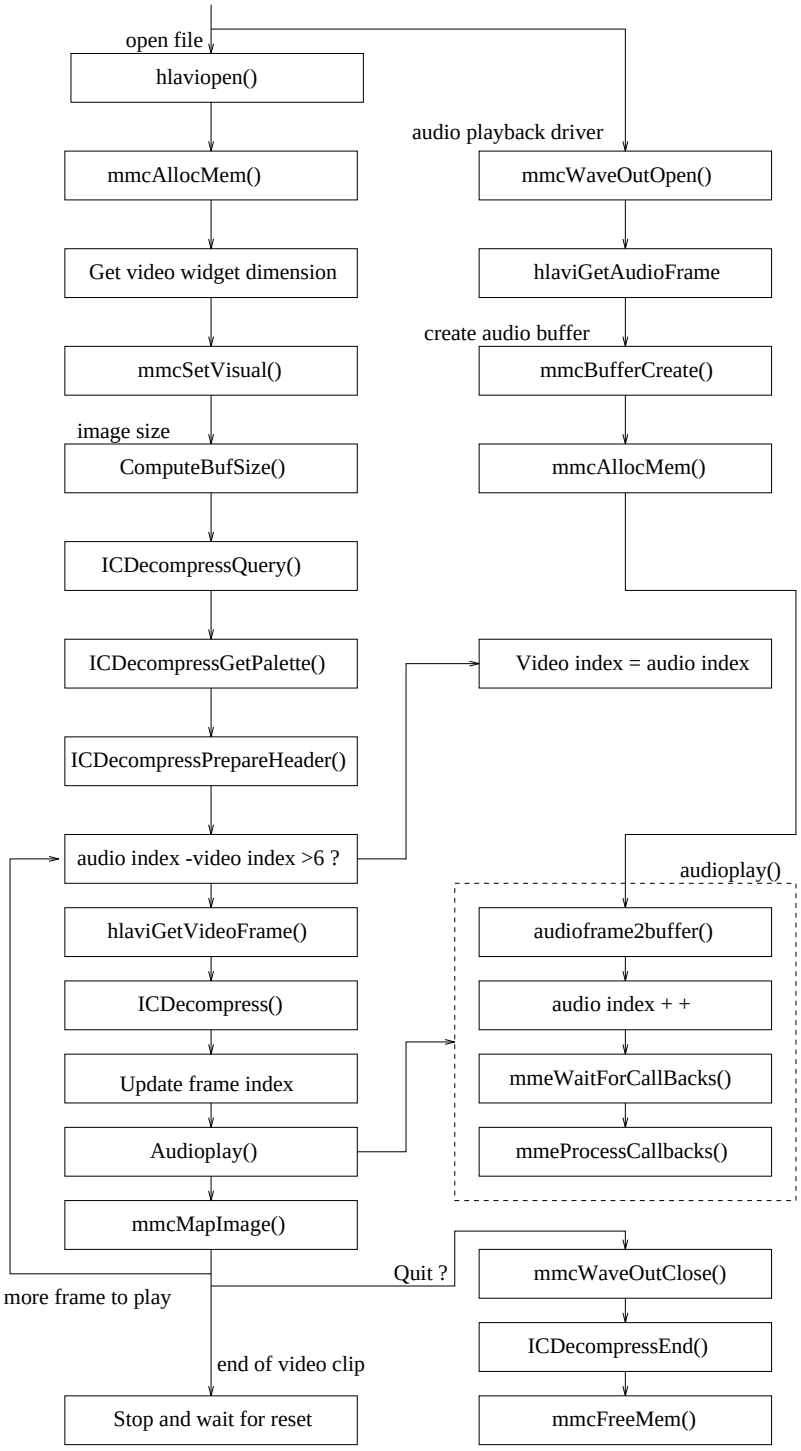

Figure 3.10: Implementation of the Video Display Function

Using the existing Ethernet connection between the Query Server and the Remote Client, a 6 MB data file compressed at a ratio of 2:1 takes about 60 seconds to be transmitted from the database to the Remote Client for playback. This data transfer rate is far behind from our goal of achieving real time data streaming mechanism. One of the solutions to this low data transfer rate is to use ATM network, in which high data transfer rate can be achieved. We could also compress the video and audio at a higher ratio, decompress it in the Client's hardware, but this leads to lower quality video.

Hardware and Software Data Decompression

In this prototype, a video or audio data stream can be decompressed using either hardware or software. The quality of the decompressed data for both methods does not differ. The standard size of a video frame that is used in this prototype is 360x480 pixels. The hardware decompression, which can achieve a speed of up to 25 frame/sec, is needed to playback the video in real time. On the other hand, software decompress can in the range of 10-15 frame/sec. Therefore, software decompression is used only when hardware decompression is not available.

3.4.6 The Client GUI

The main function of the client GUI is to provide an easy to use graphical user interface. It provides the interactive interface between the users and the Query Server. Figure 3.11 shows the system flow diagram of this client GUI.

A screen dump of the Client GUI is shown in Figure 3.12. The Boolean query entry menu is shown at the top part of the main widget at the left. Right below

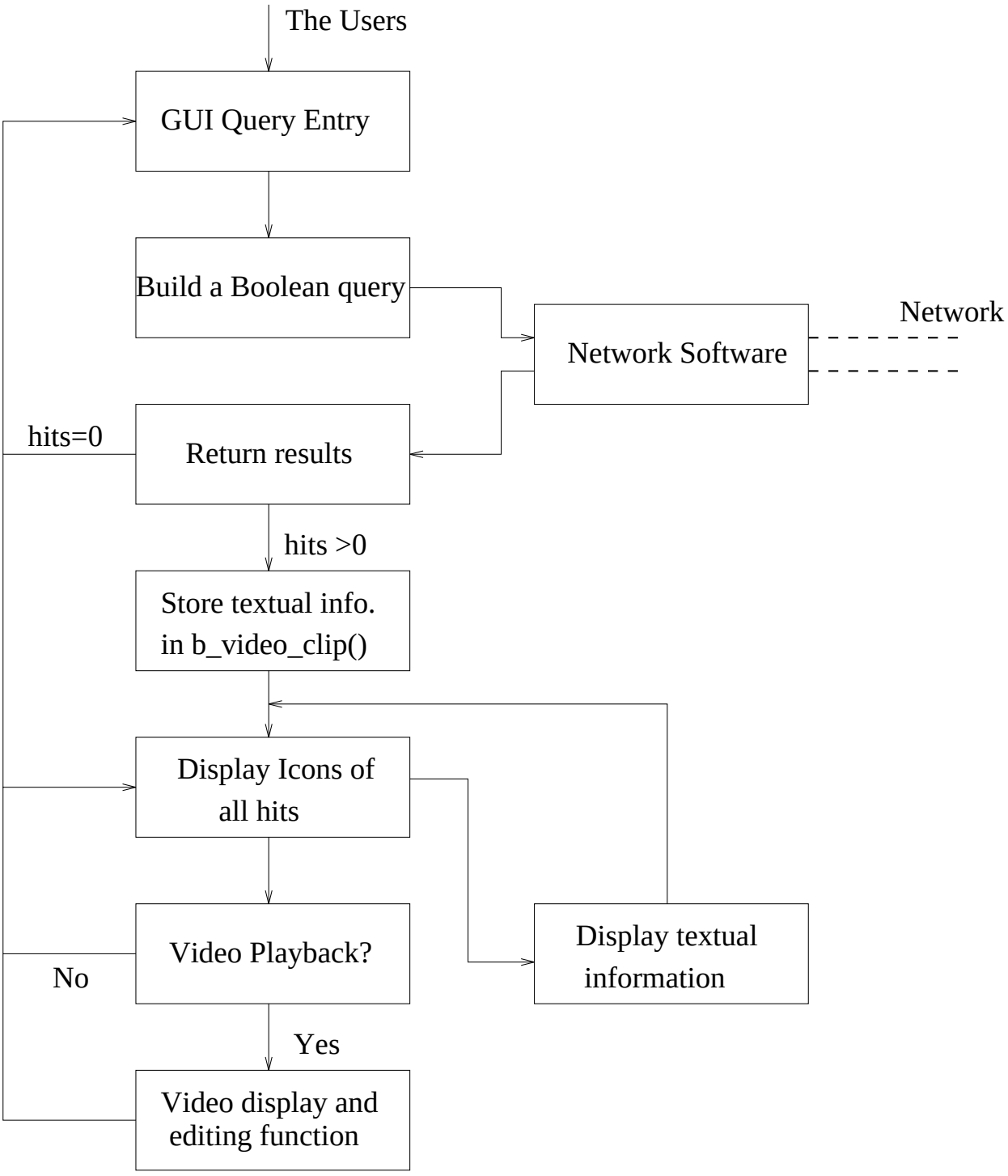

Figure 3.11: The Client GUI System Flow Diagram

the bottom of the query entries are the **Send** and **Quit** command buttons. The **Send** command button is used to send queries to the Query Server. The **Quit** command button is used for closing the Boolean search process. The bottom half of the main widget is used to display the icons of the retrieved video clips. There is a textual information button for each icon which will create a window displaying textual information for that video clip. An example of this textual information display widget is shown at the bottom right of Figure 3.12.

The maximum number of hits allowed for a query input is 25. The Boolean search query does not automatically transfer any video clips from the video database. Video clips are only transferred to the local disk of the client when video playback signal is sent. This two phase process reduces the load of hit transfers tremendously. To playback a specified video clip found in the icons display widget, the user double clicks on the video button at the bottom right of that icon. The Client creates a signal which is sent to the Query Server to transfer the AVI video file of that video clip in the Video Storage System to the Client for playback. Once the AVI file is at the Client's local disk, the J300 video board is used for video and audio decompression which creates the data stream for movie playback.

An example of a movie playback widget is shown at the top right of figure 3.12. A movie playback widget consists of a video screen with a default size of 360x480 pixels and a set of movie control buttons. The movie control buttons are play forward, play backward, fast forward, fast backward, forward and backward stepping, stop and reset. A few video editing features are also available via the menu buttons at the top of the movie playback widget. These video editing features are resizing the video screen, retrieving an image from the video clip, re-

seting the video screen, turning the soundtrack on or off, and storing the video clip.

Tcl/Tk Software Package for GUI

Two packages called Tcl and Tk provide a programming system for developing and using graphical user interface applications. Tcl stands for "tool programming language" [10] which is a simple scripting language for controlling and extending applications. It provides generic programming facilities which are useful for a wide range of applications, such as variables and loops and procedures. Furthermore, Tcl is embeddable; its interpreter is implemented as a library of C procedures that can easily be incorporated into applications, and each application can extend the core Tcl features with additional commands specific to that application.

One of the most useful extensions to Tcl is Tk. It is a toolkit for the X Window System. Tk extends the core Tcl facilities with additional commands for building user interfaces, so that you can construct Motif user interfaces by writing Tcl scripts instead of C code. Tk is implemented as a library of C procedures so it too can be used in many different applications. Individual applications can also extend the base Tk features with new user-interface widgets and geometry managers written in C.

Tcl makes it easy for any application to have a powerful scripting language. Once few new Tcl commands are written to provide the basic features of a new application, the application can be linked with the Tcl interpreter to produce a full-function scripting language that includes both the commands provided by Tcl and those implemented by the application. The second benefit of Tcl/Tk is rapid

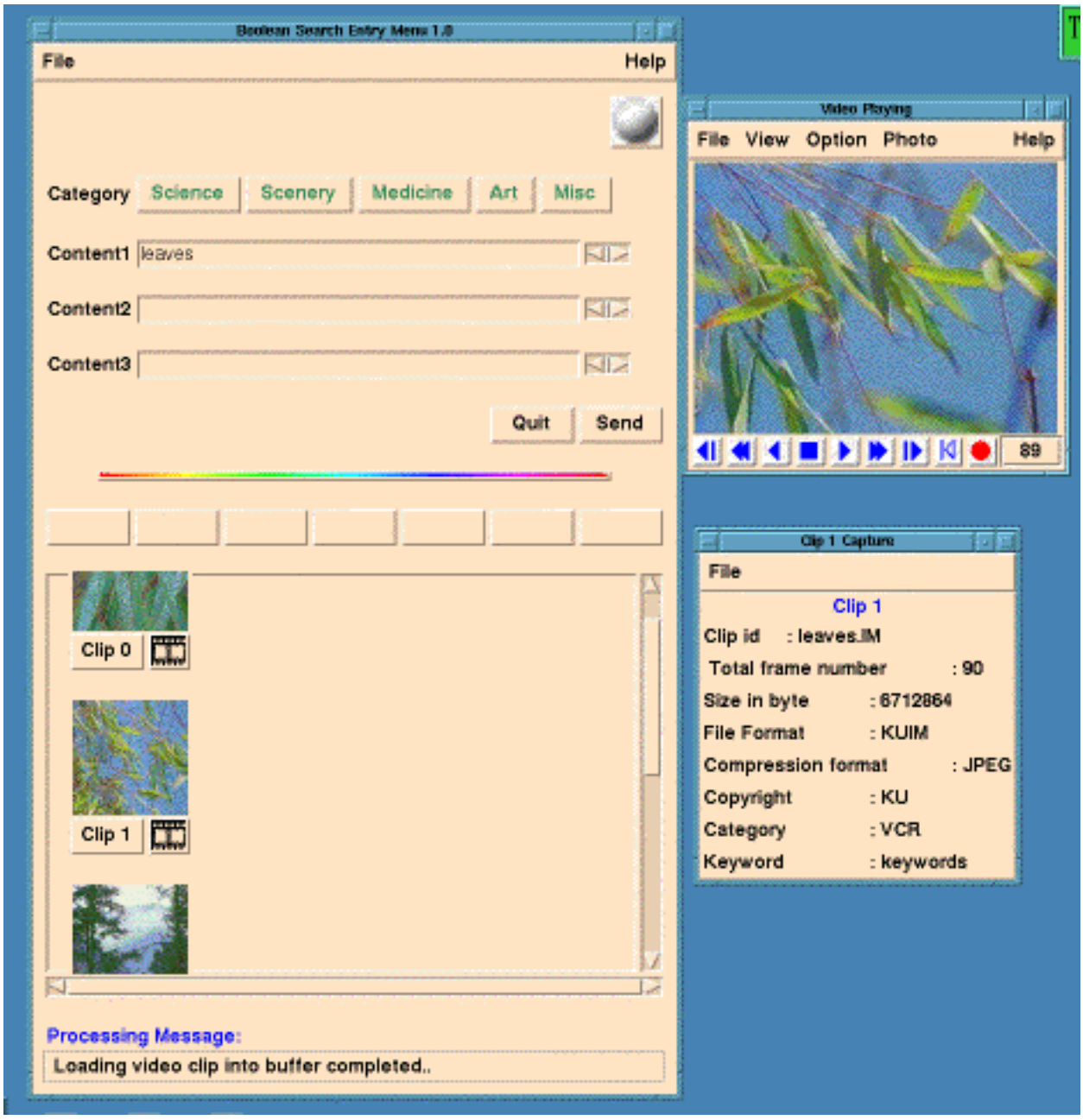

Figure 3.12: The Screen Dump of The Client Performing A Boolean Search

development. For example, many advanced windowing applications can be written entirely as Tcl scripts with no C code at all, using a windowing shell called *wish*. This allows programmers to code at a much higher level than C or C++, and many of the details that C programmers must address are hidden. Another benefit with Tcl/Tk is that Tcl is an interpreted language. When you use a Tcl application such as *wish* you can generate and execute new scripts on the fly without recompiling or restarting the application. New ideas can be tested and their bugs can be fixed very rapidly. The only drawback of this is that the resulting code runs slower than compiled C code, but this is generally offset by the speed of modern workstations.

Writing Tcl Applications in C

In order to make a Tcl application as flexible and powerful as possible, its C code should be organized as a set of new Tcl commands that provide a clean set of primitive operations. The central structure manipulated by the Tcl library is a C structure of type Tcl_Interp. Almost all of the Tcl library procedures take a pointer to a Tcl_Interp structure as an argument. An interpreter embodies the execution state of a Tcl script, including commands implemented in C, Tcl procedures, variables, and an execution stack that reflects partially-evaluated commands and Tcl procedures.

Each Tcl command is represented by a command procedure written in C. When the command is invoked during script evaluation, Tcl calls its command procedure to carry out the command. In order for a command procedure to be invoked by Tcl, we must register the code C by calling Tcl_CreateCommand. The

following shows how a command procedure called **EqCmd** could be registered with Tcl_CreateCommand Tcl function.

Tcl_CreateCommand(interp, "eq", EqCmd, (ClientData *) NULL,
(Tcl_CmdDeleteProc *) NULL);

The first argument to Tcl_CreateCommand identifies the interpreter in which the command will be used. The second argument specifies the name for the command and the third argument specifies its command procedure. A set of new Tcl commands are implemented in this Client GUI. They are:

- **avi.c**

  This is a new command procedure which creates a video widget to display an image frame of AVI video file. Together with a Tk procedure called **movie_controls**, it is the external representation of the video playback feature used for AVI video playback. J300 video board and MME drivers are called for decompression and audio output setup.

- **video.c**

  This is a new command procedure which creates a video widget to display an image frame in KUIM video file format. This video widget has basically the same functionality as avi.c except that it does not has audio playback.

- **tcl_boolean.c**

  This is a set of command procedures which are used for handling non-textual and textual data traffic between the network socket and the Client GUI.

- **tcl_client.c**

  This is a set of command procedures which comprise the network software

package used to establish the connection between the Remote Client and the Query Server.

For flexibility and more powerful applications, all the C coded programs in the client software have been registered as command procedures so that they can be used in the Tcl/Tk environment more easily.

Colormap Issues

Both AVI and KUIM format representing pixels using 8-bits which is sufficient for 256 colors. Each uses a private bitmap color table (an array of 256 entries) for each video clip. Each entry is a 24-bit RGB[4] structure representing the true color of the pixel. The RGB value for a given pixel can be easily retrieved using table look-up.

The colormap is used to map the pixel values of video clips to the display colors found in the system hardware. It is used to translate each pixel's value into the visible colors we see on the screen. There are two ways in which a pixel value can be mapped into a display colors. These two methods are using a default colormap or a private colormap to find the display colors for each pixel.

Usually a private colormap for each widget is used to obtain a high quality of image display, and to avoid the problem of running out of color entries. In this private colormap method, a colormap which has 256 entries is created for each widget. Since the maximum number of color entries needed to display an AVI or KUIM format video clip is 256, the newly created private colormap will always have enough entries to map each entry found in the bitmap color table of the video

[4]Three primary color components: Red, Green, and Blue

clip into its own display color entries. This private colormap can only be used by the assigned widget and is not sharable. The only drawback of using private colormap for image display is the flashing phenomenon when the cursor is moved from one widget to another widget. This flashing occurs when the cursor moves from one widget to another, where the second widget uses a different colormap. All the colors on the screen get reinterpreted using the new colormap, resulting in often drastic color changes.

In order to avoid the unpleasant flashing phenomenon, we chose to use the default colormap method for video displaying purposes. In this approach, all the widgets of a running application share a common colormap with 256 color entries. To display a video clip, each widget will try to allocate enough color entries in the default colormap, based on the bitmap color table of the video clip. Hence, a color entry of the default colormap could be allocated by one widget and shared by another widget. The drawback to using the default colormap is that, with only 256 color entries, we can run out of colors very quickly when there is more than one widget sharing the default colormap. When the colormap is full, instead of adding a new color, we must find the closest matching colormap entry. To do this, the Euclidean Distance Algorithm, which calculates the square root of the total sum of square of each primary color component of a color pixel, (i.e., green, blue and red) is used. We are able to get a high quality of video display when video clips are played one at a time. The quality is greatly reduced when multiple video clips are playbacked simultaneously.

3.5 Networking

Sockets are an interprocess communication mechanism which allows processes to talk to each other, even if they are on different machines. It is this across-network capability that makes them so useful. Process communication via sockets is the basis of the client-server model. One process, known as a server process, creates a socket whose name is known by other client processes. These client processes talk to the server process via a connection to its named socket. To do this, a client first creates an unnamed socket and then requests that it be connected to the server's named socket. A successful connection returns one file descriptor to the client and one to the server, both of which may be used for reading and writing. The socket connections are bidirectional which means data can be read or written at either end.

Once a socket connection is made, it is quite common for the server process to fork a child process to converse with the client, while the original parent process continues to accept other client connections. In other words, the server can have multiple remote clients connected at the same time. The domain of a socket indicates where the server and client sockets may reside. We are implementing this client-server model in AF_INET which allows the clients and server may be anywhere on the Internet. Also, as mentioned in the previous section, the data format in the network is a byte stream. Therefore, the SOCK_STREAM type of socket is used which supports sequenced, reliable, two-way connection base and variable length streams of bytes.

Figure 3.12 shows a time line for typical scenario that resulting a connection-oriented transfer. First the server is started, then sometime later a client is started

that connects to the server.

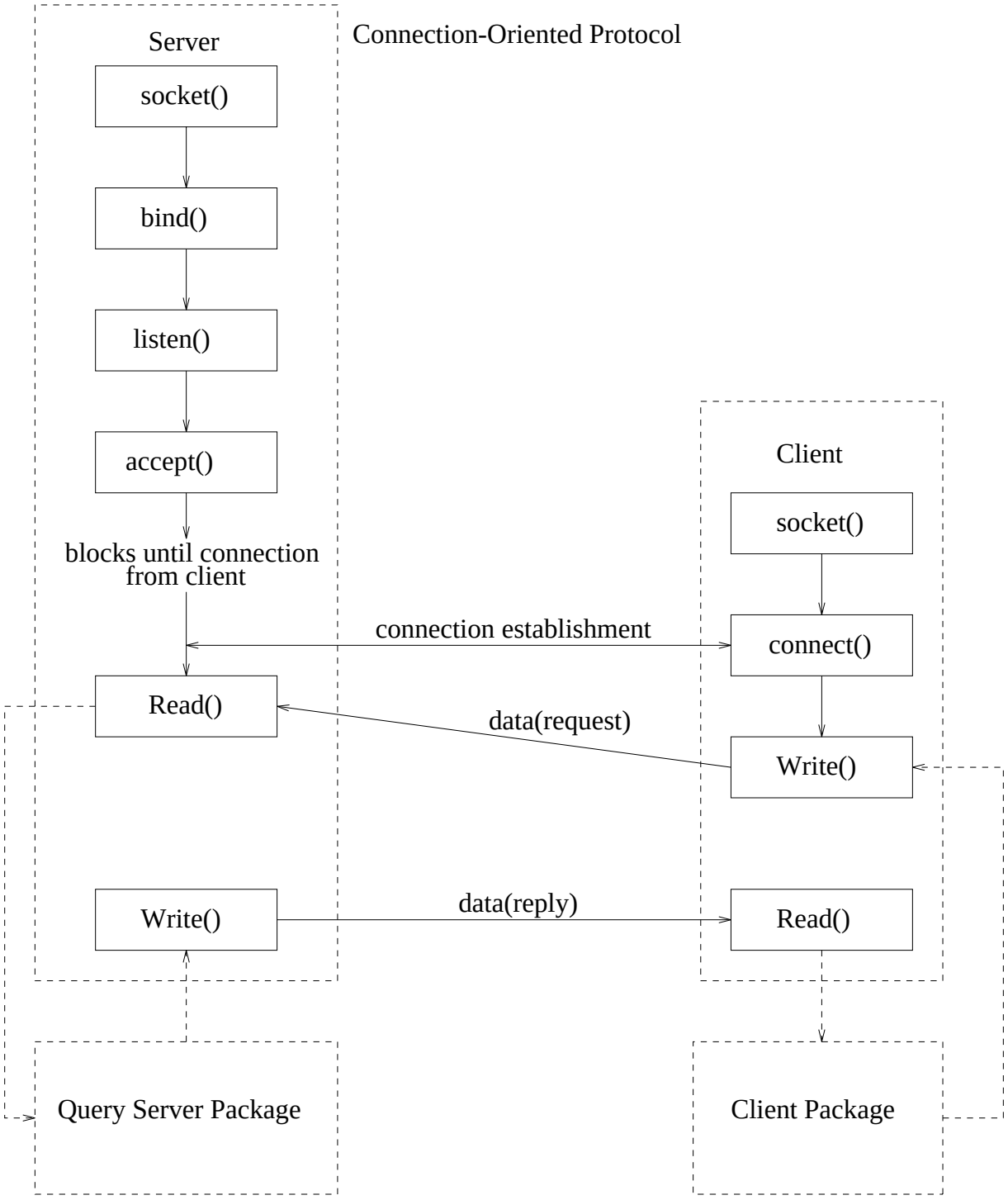

Figure 3.13: Socket System Calls for Connection-Oriented Protocol

Chapter 4

System Testing and Future Work

4.1 System Testing

4.1.1 Prototype Verification and Validation

Verification and validation is the generic term for testing to ensure that the software meets its requirements and that the requirements meet the needs of the software procurer. Verification and validation is a whole life cycle process. It starts with requirements reviews and continues through regular and structured reviews to product testing. In our case, this process is used to check the prototype to ensure that it meets the software and hardware requirements discussed in Chapter 2. Since the goal of this thesis is to build a prototype but not a complete product, the validation process is omitted from the prototype testing process.

4.1.2 The Program Testing

Although other techniques exist, Program Testing is still the predominant Verification and Validation technique. Testing is normally carried out during implementation and also, in a different form, when implementation is complete. Testing

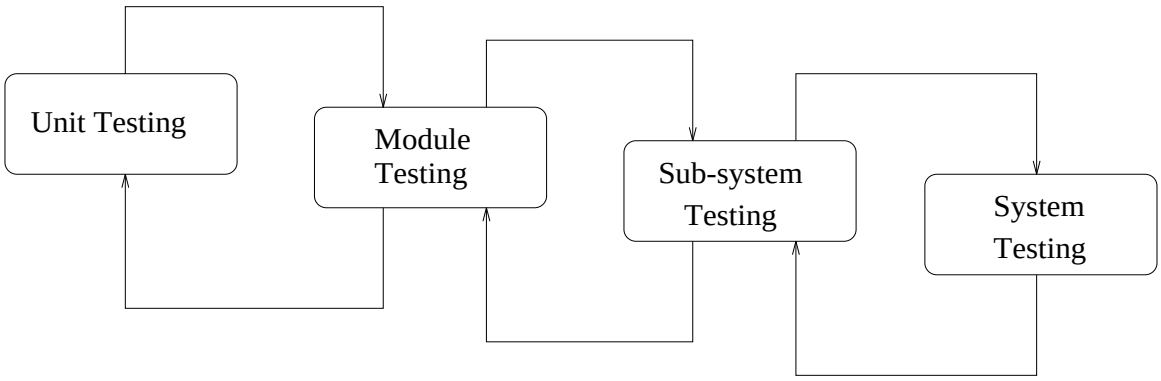

Figure 4.1: Stages of Prototype Testing Flow Chart

involves exercising the program using data similar to the real data expected by the program. Nevertheless, program testing can only demonstrate the presence of defects in a program, it cannot demonstrate that the program is error-free.

Systems should not be tested merely as a single unit. A large system usually consists of several sub-systems, built as modules, which are composed of procedures and functions. Hence, the testing process should proceed in stages where testing is carried out incrementally in conjunction with system implementation. Figure 4.1 shows a flow chart of the stages of program testing used in my prototype development. As can be seen in this figure, a repeat loop from one stage to the other is needed when defects are discovered at that stage. In other words, the testing process is an iterative one with testing information being fed back from later stages to earlier parts of the process.

Since this prototype is designed and implemented using top-down techniques, a top-down testing technique is also used for program testing purposes. The following lists the sequence of prototype implementation and its testing process:

1. **A simple client/server network model**

The first step in implementing the VISION prototype was to set up a network connection between a server program and a client program. There was no GUI in the client program and the server was just a shell. The purpose of this simple client and server program was to demonstrate that the server can be reached and connected to by a client through the Internet. For testing purposes, ASCII files were used to transfer data from the server to the client.

2. **Adding Query Server Software Component**

   The second step of building the VISION prototype was to add the query server software component to the client/server model. In order to test the functionality of the Query Server, a few query signals such as searching for a file in the server directory given a filename in the client side were added. In this model, image files such as KUIM image files were requested by the client. The server received the signal and performed searches using the filename provided by the client. Then the file was transferred to the client, if it was found.

3. **Adding Client GUI to the client software**

   Client GUI software implemented using Tcl/Tk software package was added to the client software. The Client GUI provided an interactive user interface to the user for filename and display images. In this version of prototype, we tested that the user could input any image filename request in the Client GUI and send the message to Query Server through the Internet. The Query Server then forked a process to find the file identified by client. If the file was found in the specified directory, it was transferred to the client. The client displayed the image file using a Tcl/Tk image widget. Otherwise, the Query Server returned a flag indicating file was not found.

4. **Connecting the database server with Query Server**

   The next step was to establish a connection between the database server, which serves as an information retrieval unit, and Query Server. The video segments were hand selected and manually indexed. This prototype was able to handle Boolean search queries from the client and return results to the client for display. Thus, this version of prototype was able to do content-based retrieval of video segments.

5. **Handling Compressed Video Segments**

   The previous version of the prototype assumed that the video received was not compressed. The next goal was to ensure that the client can support both hardware and software decompression of JPEG format, allowing for much faster data transfer over the network. A video widget was implemented using C code which performed the video playback function. This included the decompression of each video frame and conversion of the data into Ximage format to be displayed using Xlib routines. The testing process confirmed that the playback rate was between 25 frame/sec and 35 frame/sec.

6. **Handling audio palyback**

   The audio stream was also extracted from the video storage system and decompressed using hardware or software. The decompressed audio data was sent to an audio playback driver, in which the digitized audio data was converted to an analog signal and sent to the speakers. The main testing process in the prototype was the synchronization between the audio and video streams.

7. **Adding Video Editing Features**

Next, we added more features to the video display utilities such as changing the size of the video widget and taking a snapshot of a video clip. This required changes to the Client software with no modification to the server.

8. **Digitizing An Hour Of Video Data For Demonstration**

   The final step in the prototype fulfilled the system requirements discussed in Chapter 2. Here, an hour of video and the accompanying audio were digitized, compressed, manually indexed and stored to provide a small database for demonstration purposes. We chose to digitize an hour of movie called *Star Wars*.

4.2 Results

The prototype achieved the following results:

- A system and network architecture to support the delivery of video library services on the Internet.
- Search and retrieval engines to provide content-based access to video segments of interest to the user.
- User interfaces to support the use of digital video library.
- A valuable digital video library of approximately one hour of video, made accessible to a testing client.

4.3 Prototype Demonstration

The following scenario illustrates the thread of events when a user performs a search for sports videos using VISION.

1. The user enters the word *sports* in the appropriate field of the query window.

2. The user indicates that they are ready to send by clicking on the *Send* button in the query window. This causes Boolean query input to be built.

3. A connection is established with Query Server.

4. The Boolean query is sent to the Query Server by writing the Boolean query string to the socket in byte stream format.

5. The Query Server reads the Boolean query from the socket. This Boolean query is formatted as an SQL query.

6. A connection is established with the Illustra Server and the SQL statement is sent to the Boolean search engine of the Ilustra Text DataBlade.

7. The search results are stored in a *saveset*. This is a table in which each row represents a retrieved clip, or a hit. Each column consists of the *id* of the video clip, the icon representation of the first frame, the size of the video clip, the video file format of the video clip, the compression scheme of the video data, and the category of the video clip. This information is written to the socket. On the client side, a reverse function reads the data from the socket and stores it in a local array. After the data transfer process is complete, the client displays all the icons of the retrieved video clip. Figure 4.2 shows a screen dump of the client after the query is processed. The icons of each video clips are shown in a widget at the bottom half of the main search widget. A smaller display widget named *Clip Capture* appears at the right of the main search widget to display textual information about a video clip when the button named *Clip Num.*, which is at the bottom left of the

icon, is pressed. This textual information includes the filename and size of the video clip.

8. The Client closes the connection with the Query Server

The user can preview the search results by reading the textual information about the video clips by pressing the *Clip Num.* button provided at bottom left of each icon, and viewing the icon. He can request the entire clip by selecting the button at the bottom right of the icon.

To view a clip, the following occurs:

1. The user clicks on the video playback button of a chosen icon.

2. The video clip transfer signal is written to the socket, followed by the id of the video clip and the size of its file.

3. The Query Server reads the id and size of the video clip and starts the video file transfer process by opening the correct video file in the Video Storage System and writing the number of bytes specified to the socket. On the client side, there is a reverse function which reads a same total number of bytes of data from the socket and stores them in a new AVI file.

4. The Client closes the connection with the Query Server.

5. A video display module decompresses the video and audio data for video playback event. Figure 4.3 shows a screen dump of the Client when user clicks the video playback button at the bottom right of the icon. Also shown in this figure is a storage widget which prompts the user for output filename and the format of video file to be saved. A screen dump showing a

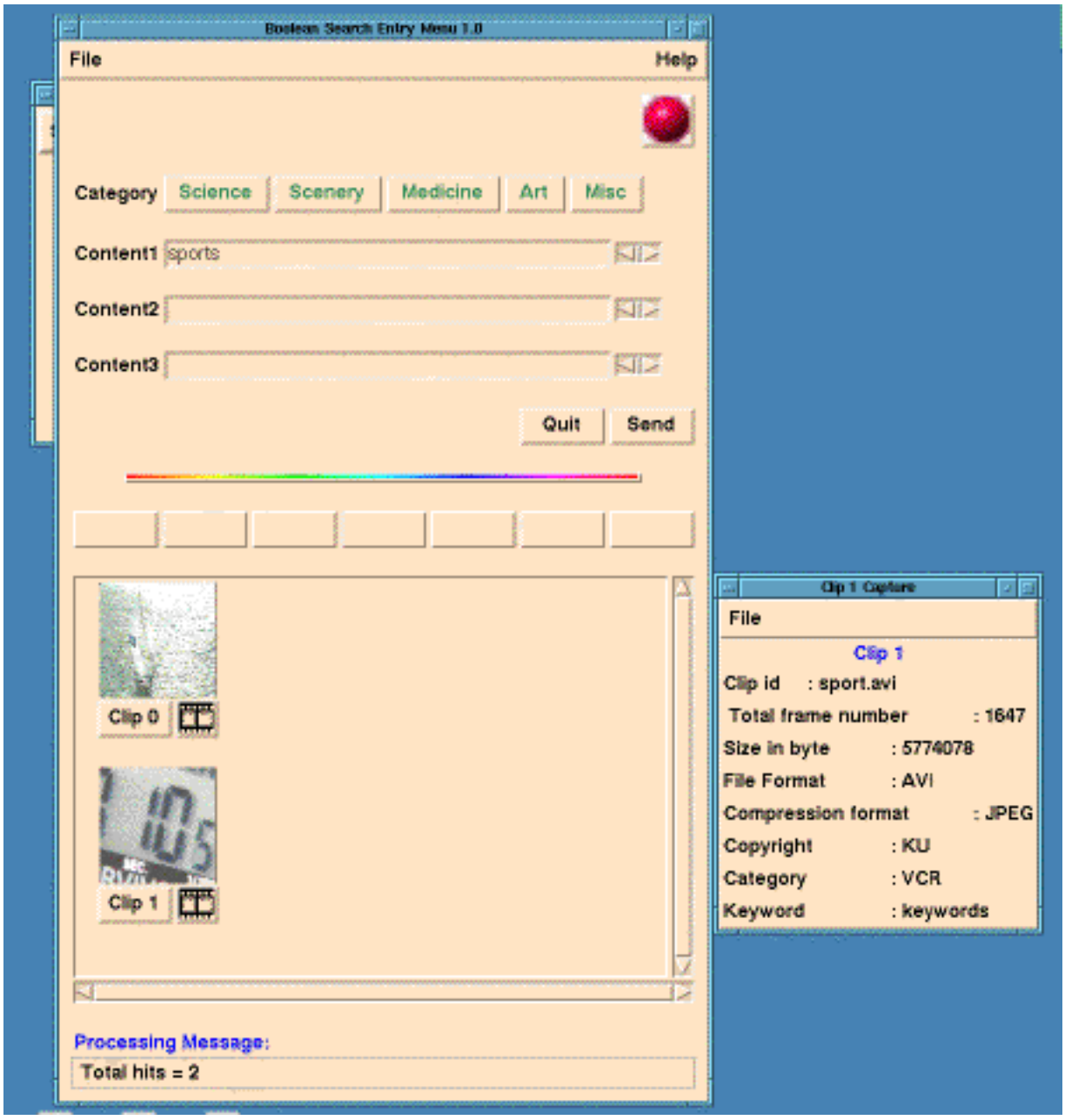

Figure 4.2: The Screen Dump of The Client Performing A Boolean Search

frame from a retrieved video clip is shown in Figure 4.3. Storage functions are provided for the user to save the retrieved video clip or the chosen frame.

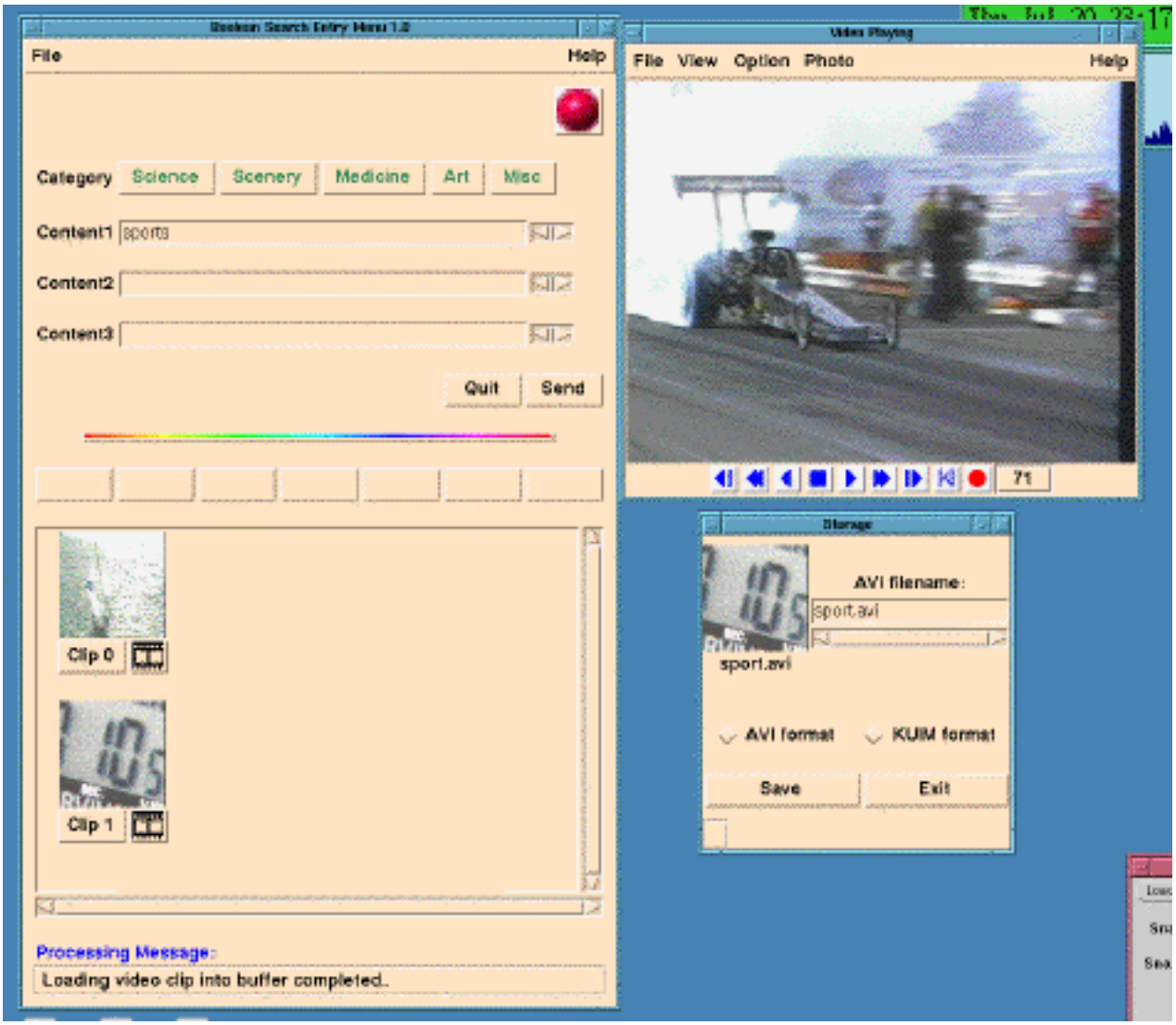

Figure 4.3: The Screen Dump of The Video Clip Retrieving Process

Figure 4.5 and 4.6 show the thread of events for both processes.

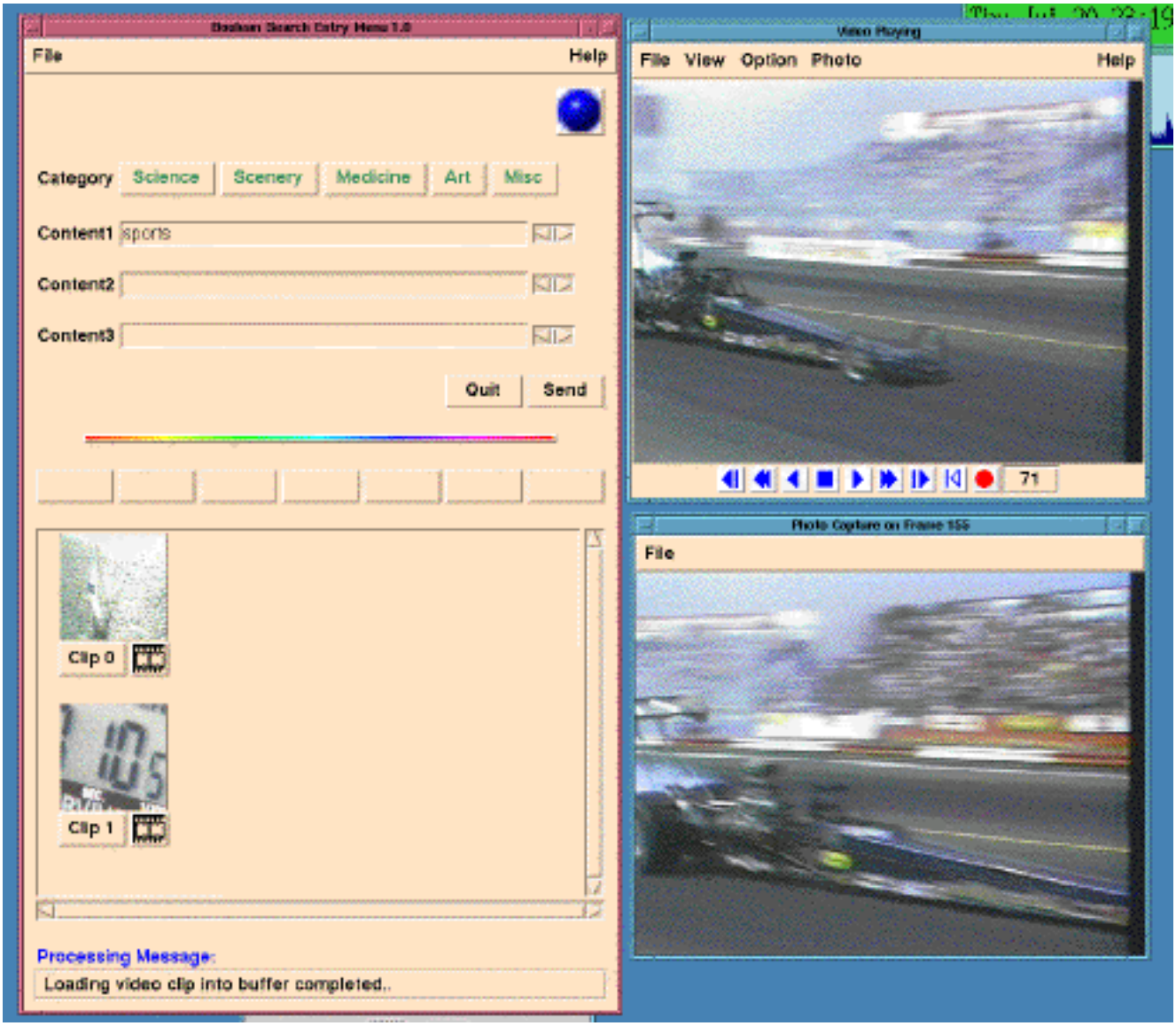

Figure 4.4: The Screen Dump of Images Retrieving Process

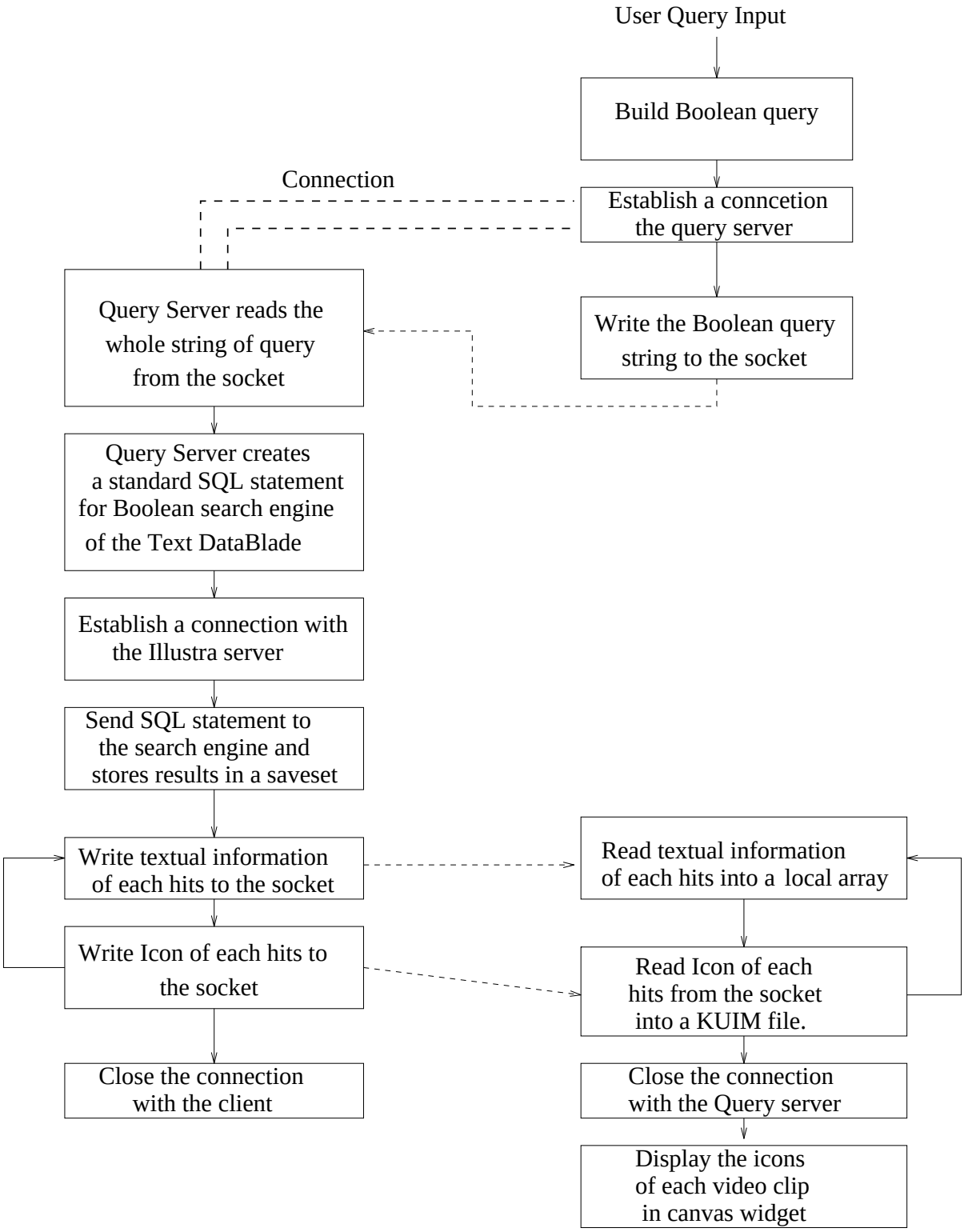

Figure 4.5: The Thread of Events of A Boolean Search

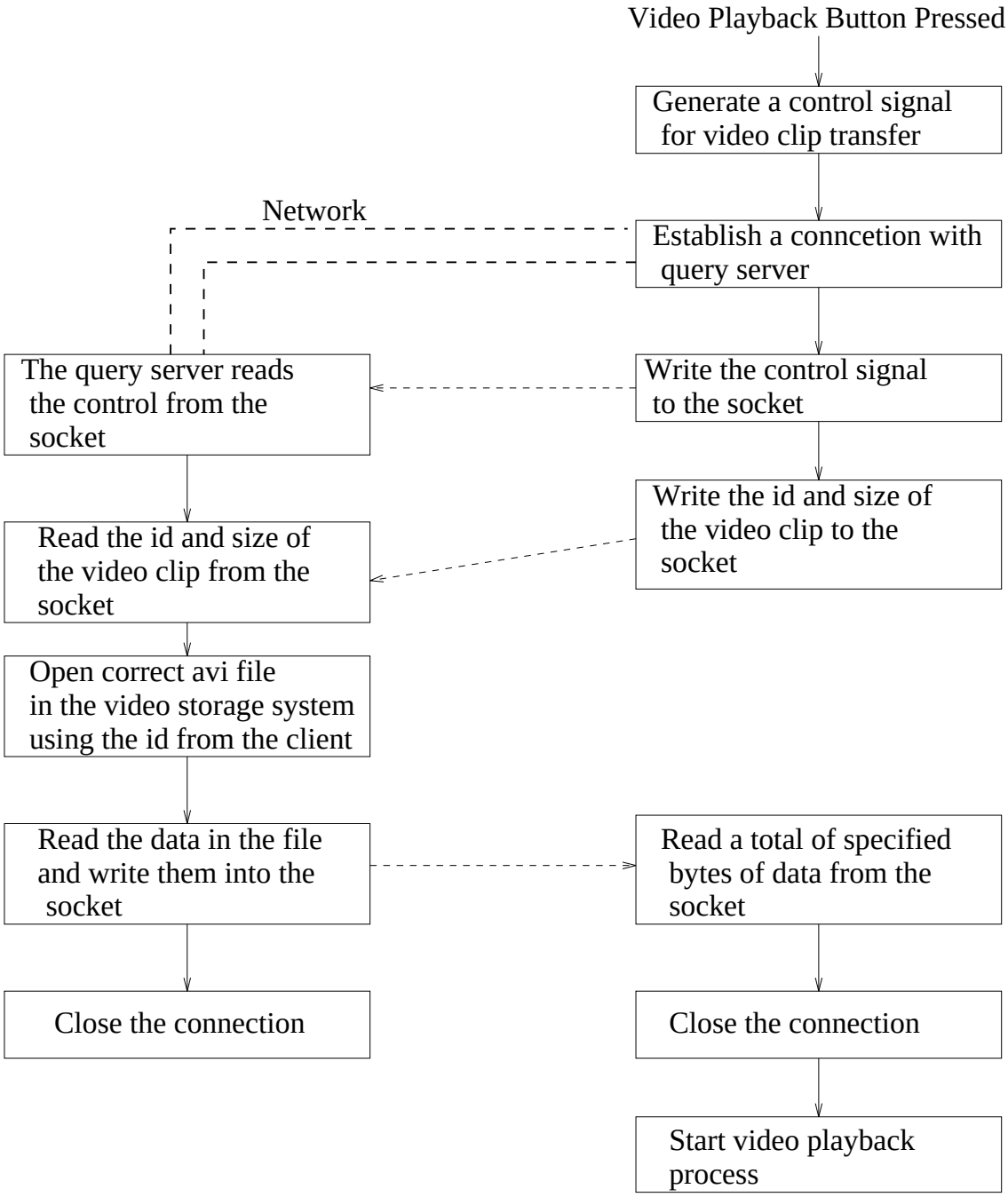

Figure 4.6: The Thread of Events of Video Clip Playback Signal

4.4 Future Work

The goal of this prototype is to demonstrate a system and network architecture to support the searching and retrieval of video data from a video library on the Internet. Each of the software components defined in Chapter 2 will be refined and modified in order to reach the requirement specifications of the fully functional VISION system. Hence, the future work on this VISION prototype includes the following:

1. **Video Data Streaming**

   Currently, the video data is handled as files. We need to refine this to be a stream that transfers one video and audio frame at a time from the Server to a Remote Client which displays it without storing it locally. The video data could be in compressed format which is decompressed in the Remote Client end; or it could be decompressed in the Server before it is sent to the Client for playback. In order to transfer uncompressed video in real time, we must test on an ATM network.

2. **Implement Client GUI using Motif and X Window package**

   Tcl/Tk scripts allow only *string* data type. Hence, unless there is a solution to handle pointer passing between two widgets in Tcl/Tk script, the Client GUI in the prototype written in Tcl/Tk package will be reimplemented using Motif and X Window package.

3. **Automatic Content-Based Indexing**

   The speech recognition of the audio track will provide more data to build search indices for each video clip. Closed caption information may also be available.

4. **Port the Client software to other workstations**

   Client software will be extended so that it is supported by multiple workstation architectures such as Alpha, Suns, SGI and Pentium.

5. **Design Efficient VSS**

   Redesign the Video Storage System to handle video data more efficiently. Speed and space will be the major concern.

6. **Complete the Video Processing System implementation**

   A first prototype of VPS is close to the completion under the work of Dr. Li. Refinement to select the appropriate compression ratio for storage and to improve the automatic segmentation is underway.

Chapter 5

Conclusion

A working VISION prototype with restricted functionality was designed and implemented. The work of this thesis serves as a basis for writing the specification for a production quality system. This thesis performs most of the important pilot work in developing the technologies necessary to provide automatic video indexing and digital video delivery via computer network. Most of the work involved in this VISION digital video library prototyping was to test and define software/hardware solutions for each of the sub-system components. This work includes studying and choosing a suitable database management system to be the VISION video database system, choosing a text database which has the capability to perform Boolean search as the Information Retrieval Engine, studying and using TCP/IP network protocol for the network traffic control, and using the Tcl/Tk software package for the Client GUI implementation. When the software or hardware solution for each sub-system component was defined, the next task was to design and implement each of the sub-system components in a working prototype which could perform the specified functionality.

We tried to detect and solve each of the design and implementation problems

that were encounted in the simpliest, but efficient way. Examples of the problems solved were colormapping for video clip playback in the Tcl/Tk environment, multimedia data handling in the network, video and audio synchronization in the video playback, etc. The techniques that are used in this prototyping will serve as the basic ideas and solutions to our future work in building a complete and productive system. The important technologies that we learned from this prototyping include video data processing techniques, multimedia data handling in the network, database system organization, Boolean search window layout, and video/audio playback synchronization. Our future work goal of extending this VISION prototype to a complete product will be an extension of these basic techniques.

Bibliography

[1] "AP Wire Service Article", *Broadcastning and Cable*, Vol. 123 n 39, September 27, 1993.

[2] Bacher, D. R., Christopher, J. L., "Content-based Indexing of Captioned Video on the ViewStation", *Laboratory for Computer Science*, Technical Report, MIT, 1995.

[3] Burks, C., M. Cassidy, et al. (1991). GenBank. Nucleic Acid Res. Christel, M., et al. (1994), "Information Digital Video Library", *Proceeding of ACM Multimedia 94*, October, 1994, p.480-481.

[4] Brunei, D.J., B. T. Cross, et al. "What if there were desktop access to the computer science literature?" *ACM Press*, 21st Annual ACM Comp. Sci. Conf., Indianpolis, 1993.

[5] Business Week, (1993), *Fox and Sony article*, July 5, p.98.

[6] Business Week, (1993), *Video Marketing article*, September 6, p.78.

[7] Christel, M., et al. (1995), "Information Digital Video Library", *Communication of ACM*, Vol.38, No.4, April, 1995, p.57-58.

[8] Crum, " University of Michigan Digital Project", *Communication of ACM*,April, 1995, Vol. 38, No. 4.

[9] Deane M., Nathan P., Fredric G., Chris S., " The University of California CD-ROM Information System", *Digital Libraries*, April 95, Communications of the ACM, Vol. 38 n 4, pp. 51-52.

[10] Digital Equipment Corporation, "Multimedia Services for DEC OSF/1 AXP Programmmer's Guide", *Digital*, Maynard, Massachusets, March 1995.

[11] Edward A. Fox, Zahid Ahmed, Robert M. Akscyn, Christine L., Michael Lesk, John L. Schnase, "Digital Libraries of the Future", *ACM Multimedia 94*, San Francisco, ACM Press, pp. 465-466.

[12] Entlich, R., et.al, "Making A Digital Library: The Chemistry Online Retrieval Experiment", *Communication of ACM*, April, 1995, Vol. 38, No. 4.

[13] Farshid Arman, Arding Hsu, and Ming-Yee Chiu. "Image Processing on Compressed Data for Large Video Databases", *ACM Multimedia 93*, New York, ACM Press, pp. 267-272.

[14] Fox, E. A., Akscyn, R. M., et al. "Digital Libraries", *Comm. of ACM*, Vol. 38, No. 4 (April 1995), pp.22-28.

[15] Fox, E., (1991) "Advances in Digital Multimedia Systems", *IEEE Computer*, Vol.24, No.10, October 1991.

[16] Fox, E. A., D. Hix, et al. (1993). "Users, User Interfaces, and Objects:Envision, a Didital Library", *JASIS 44(8)*, p.474-479.

[17] Garrett, J. R. and P. A. Lyons (1993), "Toward an Electronic Copyright Management System", *JASIS 44(8)*, p.468-473.

[18] Gauch, S., Aust, R., et al, "The Digital Video Library System: Vision and Design," *Digital Libraries'94*, June 1994, College Station, TX, pp.47-52.

[19] Hampapure, A. , Jain, R. and Weymouth, T. "Digital Video Segmentation", *ACM Multimedia 94*, San Francisco, Oct. 1994, pp.357-364.

[20] Haralick and Shapiro, (1992), "Computer and Robot Vision", *Addison Wesley.*

[21] Heath, L. S., Hix, D., et al (1995). "Envision: A user-centered database of computer science literature", *Communications of ACM*, Vol.38, NO.4, April, 1995, p.52-53.

[22] Hoffman, M. M., L. O'Gorman, et al. (1993) "The RightPages Service", *JASIS 44(8)*, p.446-452.

[23] Jeffay, Stone and Smith. (1992), "One kernel support for real-time multimedia applications", *Proceeding of 3rd IEEE Workshop on Wordstation Operating Systems*, April 1992.

[24] J.K. Wu, Y.H. Ang, P.C. Lam, S.K. Moorthy, and A.D. Narasimhalu. "Facial Image Retrieval Identification, and Inference System", *ACM Multimedia 93*, New York, ACM Press, pp. 47-56.

[25] James, D.A. and Young, S.J., "Wordspotting", Proc. ICASSSP, 1994, Adelaide.

[26] John K. Ousterhout, "Tcl and the Tk Toolkit", *Addision-Wesley Publishing Company, Inc*, Berkeley, 1993.

[27] Kahle, B., H. Morris, et al. (1993). Interfaces for Distrubuted Systems of Information Serves.", *JASIS 44(8)*, p.453-467.

[28] Laura Teodosio and Walter Bender. "Salient Video Stills: Content and Context Preserved", *ACM Multimedia 93*, New York, ACM Press, pp. 39-46.

[29] Michael C., Scott S., Howard W., "Information Digital Video Library", *ACM Multimedia 94*, San Francisco, ACM Press, pp. 480-481.

[30] Nicolaou, C., "An Architecture for Real-Time Multimedia Communication Systems", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April 1990.

[31] P. Bruce Berra, Foruzan G., Rajiv M., Olivia R. Liu Sheng, " Guest Editors' Introduction Multimedia Information Systems", *IEEE Transcations On Knowledge and Data Engineering*, August 1993, 1993 IEEE, Vol. 5 n 4, pp. 545-547.

[32] Pissinou, N., K. Makki, et al. "Towards The Design And Development Of A New Architecture For Geographic Information Systems", *CIKM 93*, Washington DC, ACM Press, 1993.

[33] Purday, J. (1995), "The British Library's Initiatives for Access Projects", *Communications of ACM*, Vol.38, No.4, April, 1995, p.65-66.

[34] Rangen, P.V., and Vin, H.M., "Efficient storage techniques for digital continuous multimedia", *IEEE Trans. on Knowledge and Data Engineering Special Issue on Multimedia Information Systems, August 1993.*

[35] Rawlins, G. J. E. (1993). "Publishing over the next decade, *JASIS 44(8)*, p.474-479.

[36] Schatz, B. (1995), "Building the Interspace: The Illinois Digital Library Project", *Communications of ACM*, Vol.38, No.4, April, 1995, p.62-63.

[37] Smith, T.R. and Frew, J., "Alexandra Digital Library", *Communication of ACM*, April, 1995, Vol. 38, No. 4.

[38] Takebayashi, Y., H. Tsuboi, H. Kanazawa, (1991), "A robust speech recognition system using word-spotting with noise immunity learning", *Proceedings of ICASSP 91*, Toronto, 1991, p.905-908.

[39] Tat-Seng Chua, Swee-Kiew Lim, Hung-Keng Pung, "Content-based Retrieval of Segmented Images", *ACM Multimedia 94*, San Francisco, ACM Press, pp. 211-218.

[40] The Stanford Digital Libraries Group, " The Stanford Digital Library Project", *Digital Libraries*, August 95, Communications of the ACM, Vol. 38 n 4, pp. 59-60.

[41] Wilensky, R., "UC Berkeley's Digital Library Project", *Communication of ACM*, April, 1995, Vol. 38, No. 4.

[42] W. Richard Stevens, " UNIX Network Programming", *Prentice Hall Software Series*, Englewood Cliffs, New Jersey, 1990.