LYU9904 Final Report

Multi Model Digital Video Library

Department of Computer Science and Engineering,
The Chines University of Hong Kong

Supervisor:
Prof. Michael Lyu

Student:
Chung Kit San
S97745893
kschung@cse

Abstract

Our project, titled "Multi Model Digital Video Library", is targeted to learn issues about digital video libraries and implement a small-scale model. Digital video library becomes more and more important as the Internet technology evolves, and will soon affecting the mode which people consumes media data, and thus the media industry. The recent development of digital video libraries is to employ various new technologies for building the database, indexing the contents, searching and retrieving the video resources in effective and efficient way. The breakthrough is to extract and index the "semantic" of a video data, which create a new "Interactive Video" view against the traditional "Interrupted Video" view. In this project, we have built a small-scale Chinese-based digital video library with some basic features: Internet-based, data indexing, video searching and retrieval, real-time video playback and synchronized video transcript.

Table of Content

Introduction

The raise of Internet in the 20th Century changes the life style of people. Estimation shows that there are 50 million people using the Internet on a regular basis. Entertainment, education, commercial activities, etc. is being merged with the irresistible trend of Internet. Multimedia information is making a revolution in this trend. New technologies realized the concept of digital video library. And this initiates our project.

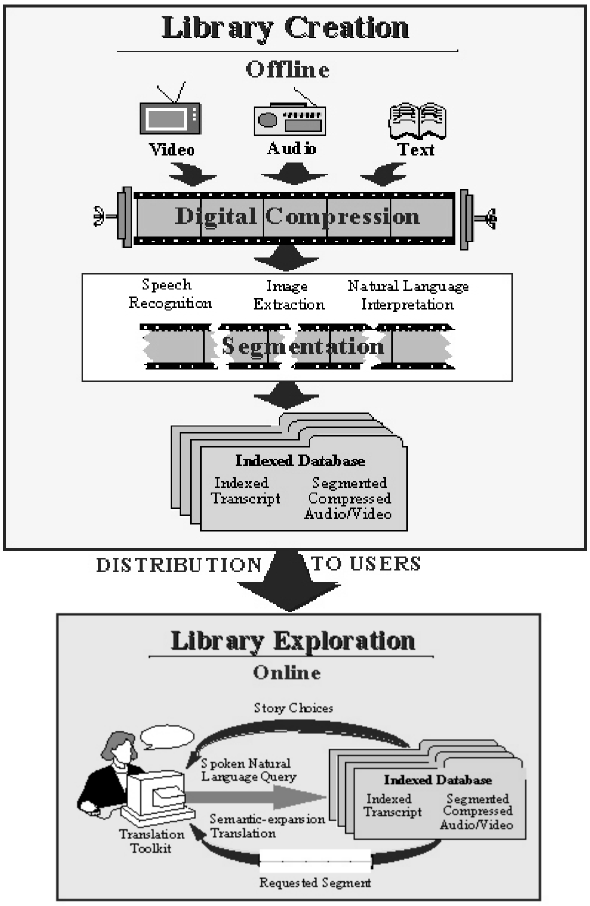
In this report, we will first introduce the background of digital video library. After reviewing the general ideas of DVL, we will introduce our work, sectioned into: considerations and approach we taken, details on the design and implementation, introduction on the functionality followed by the library content, discussions on our work, and end with a conclusion. Lastly, there are program codes attached at the end of this report.

Backyround

Video is not like pure texts or images, it is large in size and contains audio and sequence of images. Moreover, while page description languages may be more efficient, if the page contains many images, a raster image may be the only choice for representation. Video is not only imagery, but consists about 30 images per second. It is really true that "a picture is worth a thousand words". Detail descriptions of video images can be many thousands of words and even a short video clip description can be massive. Therefore, it is much more complex to handle video in computer world.

There are many questions arise before establish a digital video library. How do you build a vast video database? How do you index the video contents? How can you search and retrieve the video resources efficiently? How can you let users to view the resources conveniently and effectively? The issues on creating a digital video library (gathering video, representing its contents, and segmenting it appropriately) and utilizing and exploring the library (retrieving and browsing effectively) are also challenging parts in this topic. The following paragraphs will introduce these few aspects briefly.

Here is the overview of digital video library system:



Building Video Databases

Digital video takes a tremendous amount of space. Therefore, in order to build a video database, we need to consider the video format for the databases. It is important to choose a video format which can save space but still maintain the quality of video. A single high quality, uncompressed video channel would require a bandwidth of 200 million bits per second. Such bandwidth requirements are not practical today or perhaps ever, so the quality of the video may be reduced and compression schemes used to make possible the inclusion of video into digital libraries.

Even before the video can be digitized and placed into the library, a number of intellectual property rights issues need to be resolve. New legal rules will likely be established and evolve as consumers and publishers move fully into the electronic age.

Another consideration in the creation of a digital library is enabling access to the resources in the databases. Even with MPEG1 compression, a thousand hours of video will take approximately a terabyte (1024 gigabytes) of storage. It is so unlikely that user workstations will have the complete library stored locally at their machines. Rather, a key element of on-line digital video libraries will be the communication fabric through which media servers and satellite (user) nodes are interconnected. Traditional modem-based access over voice-grade phone lines is not adequate for this multimedia application, as evidenced by the difficulty in trying to move VHS-quality video between arbitrary sites on the Internet. The ideal fabric has the following characteristics:

- communication should be transparent to the user. Special-purpose hardware and software support should be minimized in both server and slave nodes.
- communication services must be cost effective, implying that link capability bandwidth) be scalable to match the needs of a given node. Server nodes, for example, will require the highest bandwidth because they are shared among a number of satellite nodes.
- the deployment of a custom communication network should be avoided. The most cost effective, and timely, solution will build on communication services already available or in field-test.

Indexing the Video Contents

Information is found best on the Internet when the providers augment the information with rich keywords and descriptors, provide links to related information, and allow the contents of their pages to be searched and indexed. There is a long history of sophisticated parsing and indexing for text processing in various structured forms, from ASCII to PostScript to SGML and HTML. However, it is not as simple to index video content.

An hour-long motion video segment clearly contains some information suitable for indexing, so that user can find an item of interest within it. The problem is not the lack of information in video, but rather the inaccessibility of that information to our primarily text-based information retrieval mechanisms today. In fact, the video likely contains an overabundance of information, conveyed in both the video signal (camera motion, scene changes, colors) and the audio signal (noises, silence, dialogue). A common practice today is to log or tag the video with keywords and other forms of structured text to identify its contents. Such text descriptors actually have many limitations, such as:

- Manual processes are tedious and time consuming;
- Manual processes are seriously incomplete;
- Cinematic information is complex and difficult to describe, especially for non-experts. For example, in an establishing shot that zooms from a wide angle to a close-up, determining the point when the scene changed is open to interpretation.

Breaking the Video into Segments

Anyone who has retrieved video from the Internet may realize that it takes a long time to move a video clip from one location to another because of its size. If a library consists of only 30 minutes clips, when users check one out, it may take them 30 minutes to determine whether the clip met their needs. Returning a full one-half hour video with only one minute is relevant is much worse than returning a complete book with one chapter is needed. With a book, tables of contents allow users to quickly find the material they need. However, since the time to scan a video cannot be dramatically shorter than the real time of the video clips, a digital video library should be efficient at giving users the relevant material. To make a faster retrieval and viewing, the digital video library will need to support:

- partitioning video into small-sized clips
- alternate representations of the video

Video Paragraphing

Just as textbooks can be decomposed into paragraphs with different chapters and subtitles, video library can be partitioned into video paragraphs. There are difficulties arise in how to carry out video paragraphing. Analogous structure is contained in video through scenes, shots and camera motions.

The boundaries of paragraph could be done by parsing and indexing on the video segment. Some videos, such as news broadcasts, have a well-defined structure which could be parsed into short video paragraphs for different news stories, sports and weather. Techniques monitoring the video signal can break the video into sequences sharing the same spatial location. These scenes could be used as paragraphs.

However, physically decomposing a video library into fixed number of small video files will not meet the future needs of the library user. A more flexible alternative is to logically segment the library by adding sets of video paragraph markers and indices, but still keeping the video data intact in its original context. This improvement allows later enrichment of the description of the video content. The original material can be retrieved easily and dynamically without redundancy for the user if desired.

Alternate Representations for Video Clips

In addition to trying to size the video clips appropriately, the digital video library can provide the users alternate representations or layers of information for the video. Users could then review a given layer of information before deciding whether to incur the cost of richer layers of information or the complete video clip. For example, a given half hour video may have a text title, a text abstract, a full text transcript, a representative single image, and a representative one minute “skim” video, all in addition to the full video itself. The user could quickly review the title and perhaps the representative image, decide on whether to view the abstract and maybe the full transcript, and finally the user may decide whether to retrieve and view the full video.

Retrieving Video

It is important that the retrieved videos are those user wants to have. However, it is not an easy task. For general purposed use, there may not be enough domain knowledge to apply to the user’s query and to the library index to return only a very small subset of the library to the user matching just the given query. For example, in a soccer-only library, a query about goal can be interpreted to mean a score, and just those appropriate materials can be retrieved accordingly. In a more open context, goal could mean a score in hockey or a general aim or objective. A larger set of results will need to be returned to the user, given a less domain knowledge from which to leverage.

In attempting to create a general purposed digital video library, the result set may then become quite large, so the user may need to filter the set and decide what is important. Three principle issues with respect to searching for information are:

- how to let the user quickly skim the video objects to locate sections of interest;
- how to let the user adjust the size of the video objects returned;
- how to aid users in the identification of desired video when multiple objects are returned.

-

Returning Small Pieces

There are about 150 spoken words per minute of “talking head” video. One-hour video will contain 9000 words, which is about 15 pages of text. If a user issues a query and receives ten half-hour video clips, it could take him/she hours to review the results to determine the relevance. If the results set were instead ten two-minute clips, then the review time is reduced considerably. In order to return small and relevant clips, the video contents need to be indexed well and sized appropriately.

Information Visualization

When a user searches for a specific piece of information in hours of audio or video, the results from his/her query may be too large to be effectively handled with conventional presentations such as a scrollable list. To enable better filtering and browsing, the features deemed important by the user should be emphasized and made visible. That returns us back to the problem of identifying the content within the video data and representing it in forms that facilitate browsing, visualization, and retrieval.

Consideration and Aррroach

Focus and Target

To build a product-quality Digital Video Library that fulfills all the criteria that mentions in the previous section will need years of work and a large working team. There are many related techniques needed to develop for a well-established DVL, such as speech recognition, image analysis, natural language processing, etc. In order to define targets which is suitable for a Final Year Project, we have to divide the project into modules that can be implemented in stages. And we also have to focus on building some components that will be reusable in the future development. We define some essential parts for a Digital Video Library that we can focus on:

- Build a component for video playback
- Build a component for user query
- Build a component to serve the query
- Build a small scale library content
-

For *video playback*, there are various video file formats that we can choose from. In the current stage, we are using the MPEG1 file format. MPEG1 format is not the only choice, Apple QuickTime format and Real Video format are possible alternatives. Reasons for we to choose MPEG1 format are its reasonable compassion rate and video quality, and also the free license for encoding.
For the *user side*, we target to build a simple interface for user to input query. We also implement the synchronization of video and the corresponding text scripts. We decide to enhance our DVL to be accessed through browser after the fundamental functions are developed.

And for the *server side*, we implement a full-text search for the transcripts of the videos. Although there are various searching methods, but full-text searching is the fundamental kind and useful in many applications, and is relatively easy to implement. Other kinds of queries will be added after the basic things have been developed.

For the *video library content*, we need to build a video collection for our library system. All video are in Chinese and encoded with MPEG1 algorithm with the corresponding Chinese full-text script and some descriptions.

Programming Environment

Platform

We are now developing our project using Microsoft Windows 98/NT platform. It is because up to now the Windows platform is still the most popular end-user OS systems. And, besides of that, Windows has better support for entertainment applications, so we got more convenience in developing the multimedia application.
But, instead of sticking to the Microsoft Windows, cross-platform operability also affecting our decision. There are a lot of advantages for the capability to move the application to another platform, especially when we consider the stability of Unix system and the raise of Linux system.

Programming Language

We'd take Java™ as the programming language tools for this project. There are certain aspects of Java that makes it favorite for the project.

Platform independence:
Platform independence means that the software written in Java can be run on any machines which has a Java Virtual Machine, no matter that's a PC, Unix, Linux or a Macintosh. This is obviously an advantage that the possible user group becomes larger.

Network ready
As in the very beginning, Java is considered as a language to work over the network, therefore its support in networking is good. Various kind of network model can be implemented with the network classes provided in Java, and is relatively easier to develop the same thing for other programming languages. Using Java may benefit when we implement the client/server model of the DVL. In case, we can even implement the Client as a Java Applet, which can be accessed by the web browsers on the WWW.

Classes for GUI and video playback
There are classes in Java which support GUI building and, most important, the video playback function. The simple but useful API free the programmers from low level programming details and can play more effort on the higher level system architecture etc.

Support XML technology
We have used the XML technology in our project. Java provides platform APIs for developing program with XML which is a format that can represent structured and unstructured data, along with rich descriptive delimiters, in a single atomic unit.

Nothing is ever perfect, there are drawbacks of using Java as programming language also.

Interpreted language
Java is an interpreted language, programs written in Java won't be as fast as those compiled languages such as C/C++

Immature
Java is a young language, it's API is likely to be changed in the successive versions, which may lead to difficulties in maintain the program.

Using of extended API
The classes provide video playback function, Java Media Framework API, is not included in the standard Java Runtime Environment, nor a standard install of Java Development Kit (JDK). The extra package has to be downloaded and installed before running the client program.

Programming Tools

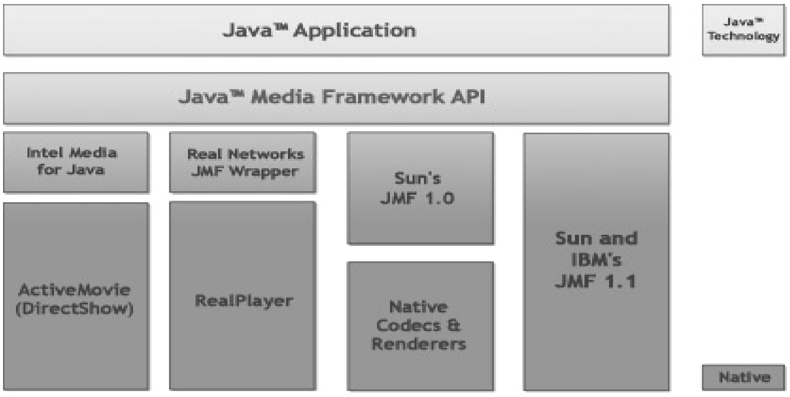
Java Multimedia Framework (JMF) API

The Java Media Framework (JMF) is an Application Programming Interface (API) for incorporating media data types into Java applications and applets. It is specifically designed to take advantage of Java platform features.
The version of JMF API we are using is JMF 2.0 API. It provides a platform-neutral multimedia solution that runs on Java platforms that support JDK 1.1.5 or later. Here are some of its features:

- Present time-based media in Java programs
- Support for capturing and storing media data
- Control the type of processing that is performed during playback
- Perform custom processing on media data streams

High-Level Architecture

JMF uses the traditional model for recording, processing, and presenting time-base media. It is quite the same as playing a movie using a VCR:
You provide the media stream to the VCR by inserting a video tape. The VCR reads and interprets the data on the tape and sends appropriate signals to your television and speakers.



In JMF, a *data source* encapsulates the media stream much like a video tape and a *player* provides processing and control mechanisms similar to a VCR. *Data sources* and *players* are integral parts of JMF's high-level API for managing the capture, presentation, and processing of time-based media.

Data Sources

JMF media players usually use DataSources to manage the transfer of media-content. A DataSource encapsulates both the location of media and the protocol and software used to deliver the media. As media data can be obtained from a variety of sources, such as local or network files and live broadcasts. JMF data sources can be categorized according to how data transfer is initiated:

- Pull Data-Source - the client initiates the data transfer and controls the flow of data from pull data-sources. Established protocols for this type of data include Hypertext Transfer Protocol (HTTP) and FILE.
- Push Data-Source - the server initiates the data transfer and controls the flow of data from a push data-source. Push data-sources include broadcast media, multicast media, and video-on-demand (VOD). For broadcast data, one protocol is the Real-time Transport Protocol (RTP), under development by the Internet Engineering Task Force (IETF). The MediaBase protocol developed by SGI is one protocol used for VOD.

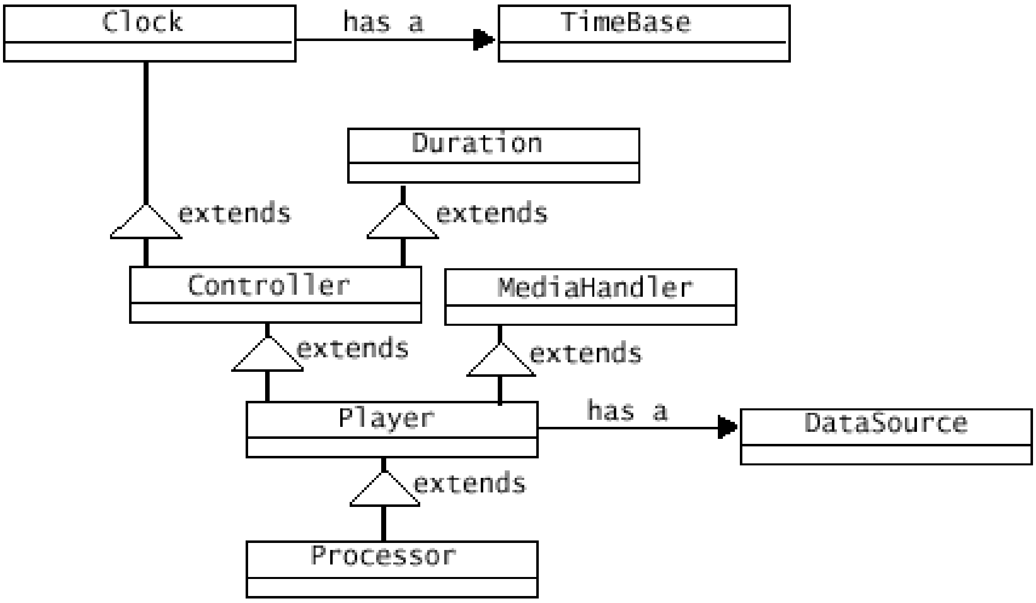
The degree of control that a client program can extend to the user depends on the type of data source being presented. For example, an MPEG file can be repositioned and a client program could allow the user to replay the video clip or seek to a new position in the video. In contrast, broadcast media is under server control and cannot be repositioned. Some VOD protocols might support limited user control, for example, a client program might be able to allow the user to seek to a new position, but not fast for ward or rewind.

Data Formats

The exact media format of an object is represented by a Format object. The format itself carries no encoding-specific parameters or global timing information, it describes the format's encoding name and the type of data the format requires.
An AudioFormat describes the attributes specific to an audio format, such as sample rate, bits per sample, and number of channels. A VideoFormat encapsulates information relevant to video data. Several formats derived from VideoFormat to describe the attributes of common video formats are:

- IndexedColorFormat
- RGBFormat
- YUVFormat
- JPEGFormat
- H261Format
- H263Format

Presentation



In JMF, the presentation process is modeled by the Controller interface. Controller defines the basic state and control mechanism for controls, presents, or captures time-based media. It defines the phases that a media controller goes through and provides a mechanism for controlling the transitions between those phases. A number of the operations that must be performed before media data can be presented can be time consuming, so JMF allows programmatic control over when they occur.
To present time-based media such as audio or video with JMF, we can use a Player which implements the Controller. Playback can be controlled programmatically, or by display a control-panel component that enables the user to control playback interactively.

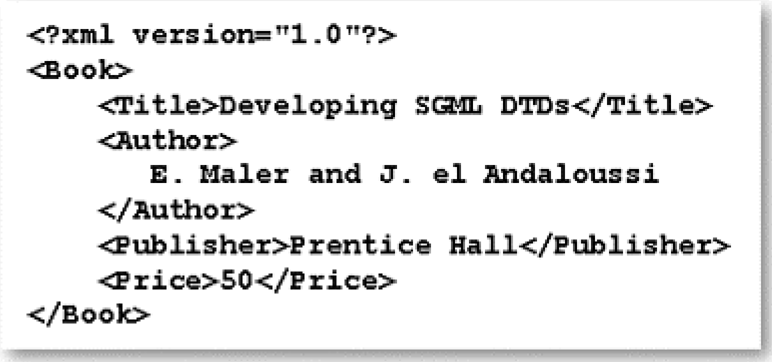
A Player generally has two types of user interface components, a visual component and a control-panel component. A visual component is where a Player presents the visual representation of its media, if it has one. Even an audio Player might have a visual component, such as a waveform display or animated character; A control panel component allows the user to control the media presentation. For example, a Player might be associated with a set of buttons to start, stop, and pause the media stream, and with a slider control to adjust the volume. Some Player implementations can display additional components, such as volume controls and download-progress bars.

When several media streams are to be play, a separate Player for each one will be used, to play them in sync, we can use one of the Player objects to control the operation of the others.

Extensible Makeup Language (XML)

XML is the meta language defined by the World Wide Web Consortium (W3C) that can be used to describe a broad range of hierarchical mark up languages. It is a set of rules, guidelines, and conventions for describing structured data in a plain text, editable file. Using a text format instead of a binary format allows the programmer or even an end user to look at or utilize the data without relying on the program that produced it. However the primary producer and consumer of XML data is the computer program and not the end-user.

XML is syntax for developing specialized markup languages, which adds identifiers, or tags, to certain characters, words, or phrases within a document so that they may be recognized and acted upon during future processing. "Marking up" a document or data results in the formation of a hierarchical container that is platform-, language-, and vendor-independent and separates the content from any environment that may process it.

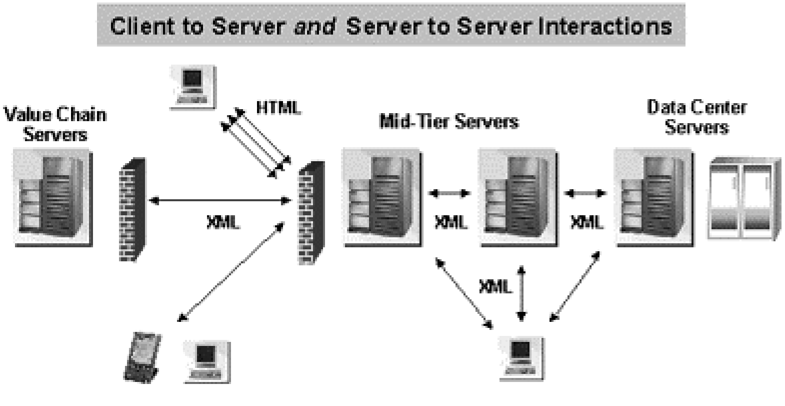


```
<?xml version="1.0"?>
<Book>
    <Title>Developing SGML DTDs</Title>
    <Author>
       E. Maler and J. el Andaloussi
    </Author>
    <Publisher>Prentice Hall</Publisher>
    <Price>50</Price>
</Book>
```

Example of XML

There are some advantages using XML:

Cross-platform Capabilities
Implement XML technology using the Java programming language can even got something more powerful: XML with cross-platform capabilities built in at the binary level, so that we have a platform independent solution from backend to frontend. When code and data are combined in the right ways, the pair becomes "portable objects" – which is really an effective way to design large scale distributed systems. In a sence, XML technology makes the information exchange possible, and Java technology makes the automation feasible.



XML data can be shuttled between many different types of servers and clients

Platform independent

Information in an XML document is stored in plain-text. This might seem like a restriction if were thinking of embedding binary information in an XML document. But this is the main reason for it to maintain the interoperability. By accepting and sending information in plain text format, programs running on disparate platforms can communicate with each other. This also makes it easy to integrate new programs on top of older ones (without rewriting the old programs), by simply making the interface between the new and old program use XML.

An example is web enabling legacy systems. It is very feasible to create a Java web ennoblement application server that simply uses the services provided by the underlying legacy system. Instead of rewriting the legacy system, if the system can be made to communicate results and parameters through XML, the new and old system can work together without throwing away a company's investment in the legacy system.

And since XML is not a binary format, you can create and edit files with anything from a standard text editor to a visual development environment. That makes it easy to debug your programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides scalability for anything from small configuration files to a company-wide data repository.

Structured

XML documents benefit from their structure.

As XML allow users to define their own tags and create the proper structural relationships in the information (with a DTD - Document Type Definition), the validity and integrity of the data can be checked with any XML parser easily. This makes the application code more reliable and quick to develop by providing validity checking on the XML documents with help of a DTD.

Moreover, the hierarchical structure also benefits the usage of XML from speed and simplicity for creation and modification of XML documents.

And since the structure of the XML document can be specified in DTDs they provide a simple way to make it easier to exchange XML documents that conform to a DTD. For example, if two software systems need to exchange information, then if both of the systems conform to one DTD, the two systems can process information from each other.

Information Management

In XML, documents can be seen independently of files. One document can comprise many files, or one file can contain many documents. This is the distinction between the physical and logical structure of information. XML data is primarily described by its logical structure. In a logical structure, principal interest is placed on what the pieces of information are and how they relate to each other, and secondary interest is placed on the physical items that constitute the information.

Rather than relying on file headers and other system-specific characteristics of a file as the primary means for understanding and managing information, XML relies on the markup in the data itself. A chapter in a document is not a chapter because it resides in a file called chapter1.doc but because the chapter's content is contained in the <chapter> and </chapter> element tags. When the elements carry self-describing metadata with them, systems that understand XML syntax can operate on those elements in useful ways. And as XML markup provides metadata for all components of a document, not merely the object that contains the document itself. This makes the pieces of information that constitute a document just as manageable as the fields of a record in a database.

Java API for XML Parsing (JAXP)

Java API for XML Parsing (JAXP) is a package of two vendor-neutral classes `SAXParserFactory` and `DocumentBuilderFactory` which can instantiate a SAX Parser and a `DocumentBuilder` respectively. The `DocumentBuilder`, in trun, creates DOM-compliant `Document` objects. The factory APIs enable XML implementation of another vendor to plug in without changing your source code. It is default to use the Sun's reference implementation of SAX Parser and DocumentBuilder.

SAX

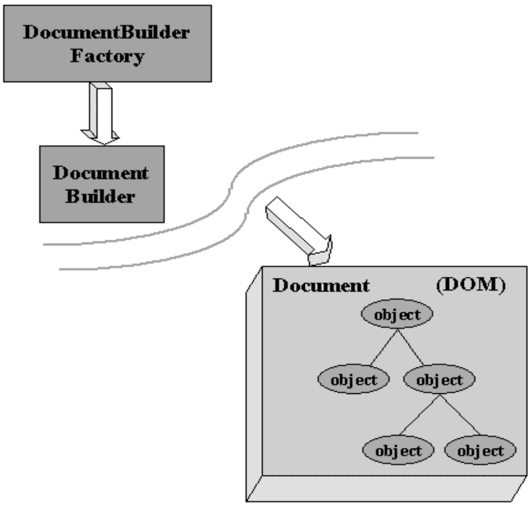
SAX stands for ‘Simple API for XML’. This API was actually a product of collaboration on the XML-DEV mailing list, rather than a product of the W3C. Thoguh it has the ‘final’ characteristics as a W3C recommendation.

You can also think of this standard as the "serial access" protocol for XML that does element-by-element processing. This is the fast-to-execute mechanism you would use to read and write XML data in a server, for example. This is also called an event-driven protocol, because the technique is to register your handler with a SAX parser, after which the parser invokes your callback methods whenever it sees a new XML tag (or encounters an error, or wants to tell you anything else).

DOM

Document Object Model (DOM) represents an XML document into a tree structure of objects in the program. You can then manipulate the object model in any way that makes sense. This mechanism is also known as the "random access" protocol, because you can visit any part of the data at any time. You can then modify the data, remove it, or insert new data.

The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user. On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive.



Above shows the function of a DocumentBuilder. First, the javax.xml.parsers.DocumentBuilderFactory class is used to get a DocumentBuilder instance (upper left), and use that to produce a Document (a DOM) that conforms to the DOM specification (lower right).

The builder's newDocument() method can be used to create an empty Document. Alternatively, you can use one of the builder's parse methods to create a Document from existing XML data. The result is a DOM tree like that shown in the lower right corner of the diagram.

After we have a Document object, we may apply different operations to it, including creating, removing, changing, and tranversing nodes; or setting and creating attributes etc. This tree model can be even visualize using the JTree interface in javax.swing.

System Consideration

Our Digital Video Library consists of two major components, the client and the server. From a user perspective, the client program is all that he/she will have interaction with, while the server should be transparent to the user. Therefore, the client program will accept queries/commands from user, and play the video clips wanted; what the server have to handle is to process the queries and return the corresponding video clip to the clients.

User Aspect

All of the user's action is done on the client program, and other parts of the system are transparent to the user, so that the user would not need to take care of other system's details. Since the usage is video oriented, high performance machine is suggested to use.

Some basic abilities for the client program:

- provides user interface
- gets query input from user
- shows results related to the query
- playbacks video

There are some qualities to provide a better software design for our DVL system:

- browser-enabled
  It only is meaningful for user to access digital library remotely not locally. Therefore, it is convenient that user can view the video through Internet.
- provides clear and easy-to-use interface for user
  Obviously, a complex interface only makes the user confused.
- Support Chinese
  Our library is a Chinese digital video library. Therefore, it is important to show Chinese even under an English Operating System.

Server Aspect

The server work mainly is to find out the result(s) that matches the query from the client side. There are possible more than one client wants to serve. So, server has to:

- have searching ability
- be equipped with indexed video data for searching
- return all results to client
- support multi-client
- stable performance

Server machine need not to have the video storage of the library because the video files can put in other server(s). However, the server should be a high performance one which can reduce the searching time.

Network Issues

Transmitting MPEG1 video signals requires a large bandwidth (approximate 350k bit/s). Therefore we test the program mainly in the LAN environment. For playback of media files at remote sites, the video will be buffered before enough data have been received for continuous playback. When streaming is used, user need not to wait the whole video to be loaded to his/her own machine and still view the video.

Video Collections

There must be a large memory to store the video files. We decide to encode the video recorded in tapes into MPEG1 video format. We choose MPEG1 because of its reasonable compassion rate and video quality, and also the free license for encoding.

Apart from the choice of digital format of video, the copyright of video source has to be considered when the video collection has to scale up.
With the concept of building large-scaled digital video library, we also will follow

the steps to build our small-scaled library:

- prepare Chinese video sources
- digitize the video into MPEG1
- segment video into small pieces
- prepare the video-related descriptions, full-text scripts, etc.
- index all video resources

Without techniques assisting, we only can do these steps manually at this stage.

Equipment Used

Our Digital Video Library Server (DVLS) required relatively a large amount of space for video storage. We use a PC with static IP to build our DVLS, which makes the network programming easier. The PC is equipped with about 16Gb of harddisk, and more than 10Gb of the space is used to install a sample digital video library 'Informedia' from Carnegie Mellon University. When the amount of video in our library grow, the need for more space may be necessary. Currently, all the video files are stored in the machine pc89184 in CSE Department. The link is "pc89184.cse.cuhk.edu.hk/jmdata".
To prepare the video files, we use the facilities in Multimedia Lab of CSE Department to convert VHS tape video to MPEG1 files. With the help of hardware encoding card, the encoding can be done in real time, which is quite satisfactory.

Design and Implementation

System Design

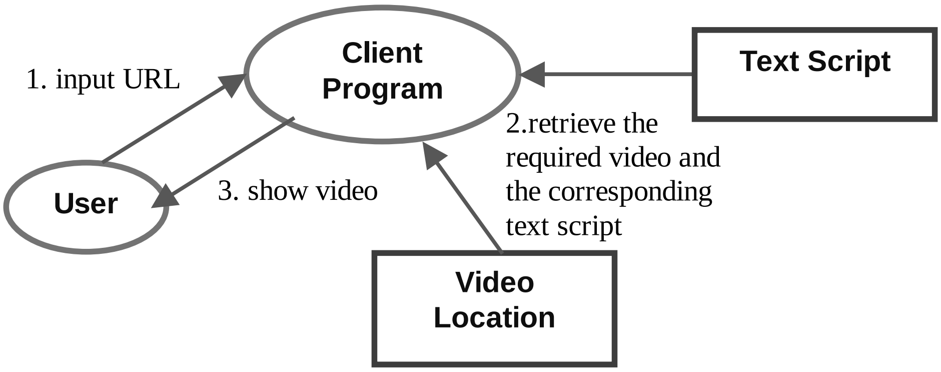
Overview of Different System Models

In this whole year, we have built totally three versions for the digital video library project. They are the Initial Version (JMPlayer), the Applet Version (JMApplet), and the Final Version (JDVL). In the following, there will be detail description on each of the version.

Initial Version -- JMPlayer

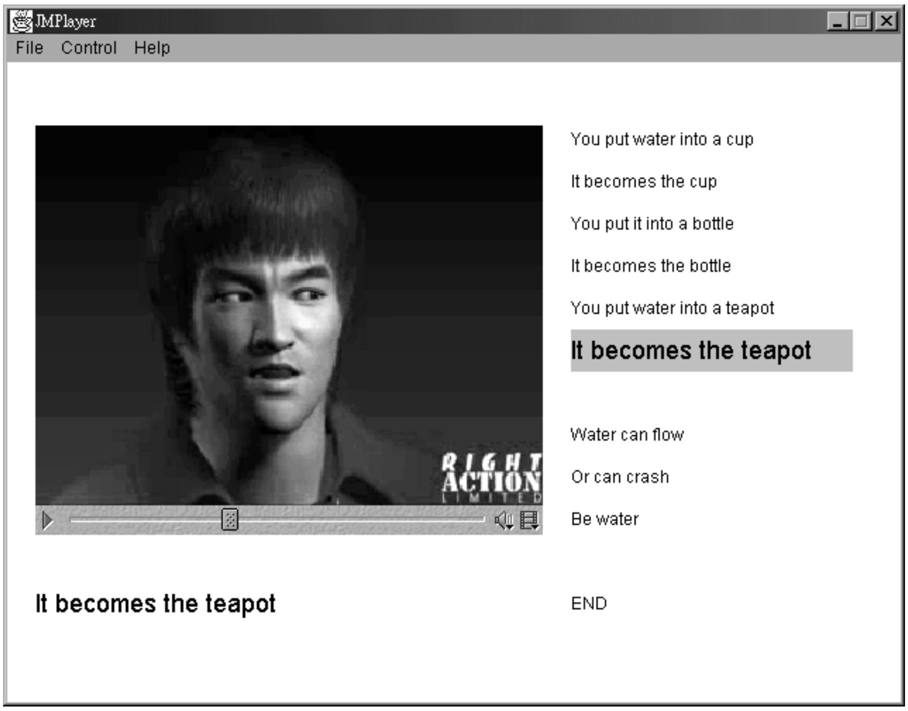
JMPlayer is our test model letting us practise Java programming and familiar to the video playback. Before building JMPlayer, we started to learn Java language and also the Java Multimedia Framework JMF API which is a Java extended package used for developing the media playback task.

Here is the JMPlayer Overview:



In the JMPlayer, there is no client/server concept. Instead, it is an application which gets user's URL input and then retrieves the video from the URL with the corresponding text script (the URL can be local file directory or remote file site). Lastly it playbacks the retrieved video and shows synchronized script beside the video screen.

Here is JMPlayer interface:



You put water into a cup

It becomes the cup

You put it into a bottle

It becomes the bottle

You put water into a teapot

It becomes the teapot

Water can flow

Or can crash

Be water

It becomes the teapot

END

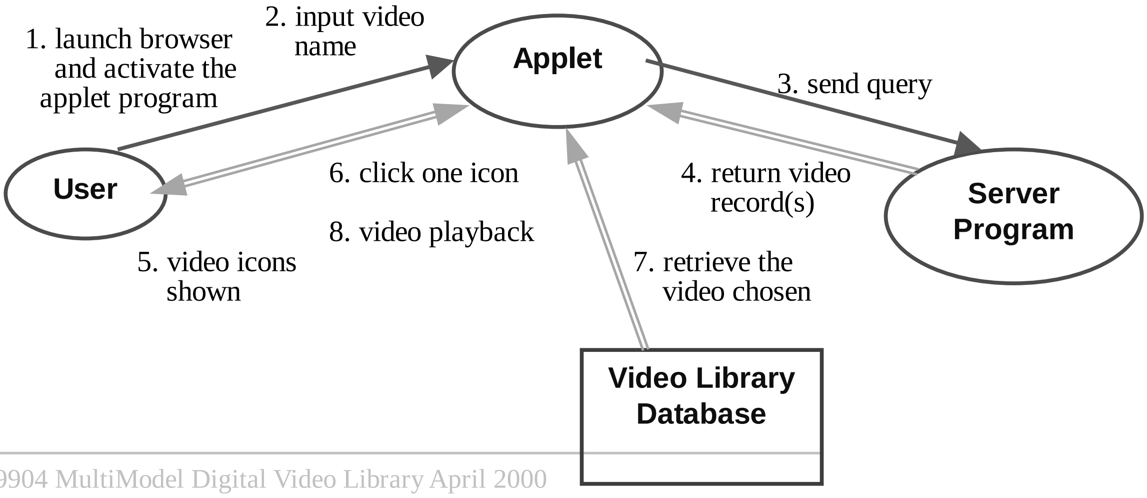
Some simple features are available:

- Synchronized script
- “File” Menu (get file location)
- Play control (video play or pause)
- Sound control (play sound or mute)

Applet Version -- JMApplet

JMApplet is our second system version. It inherits some features from the JMPlayer, including Synchronized script, play control and sound control. As its name, we have enhanced the first version with browser-accessible ability. In this version, we also have applied client/server concept on it.

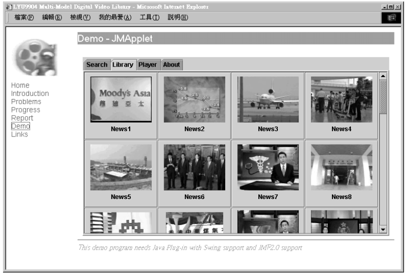
Again, let see the system model of JMApplet:



In this version, user can through Internet to retrieve video with JMApplet. The client program - applet mainly provides a space for user interaction and gets the query input by user. Then client side sends the query to the server side. Since we hope to have a testing mode on server side, the server program in this version performs a simple task. It purely matches the input query word with the video file names. If it finds they are fully matched, then return the representative icons to client.

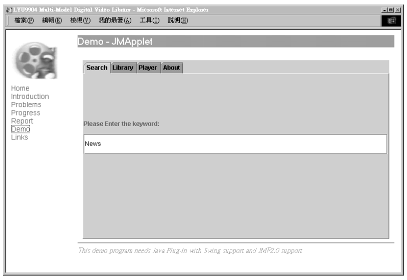
The following is the JMApplet interface using Windows Internet Explorer:

Library Panel



This demo program needs Java Plug-in with Swing support and JMF2.0 support

Query Panel

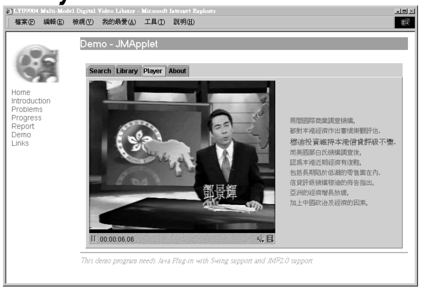

LYU9904 Multi-Model Digital Video Library - Microsoft Internet Explorer
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
Demo - JMApplet
Search
Library
Player
About
Home
Introduction
Problems
Progress
Report
Demo
Links

Please Enter the keyword:

News

This demo program needs Java Plug-in with Swing support and JMF2.0 support

Player Panel


LYU9904 Multi-Model Digital Video Library - Microsoft Internet Explorer
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
Demo - JMApplet
Search
Library
Player
About
Home
Introduction
Problems
Progress
Report
Demo
Links
鄧景輝
00:00:06.06

兩間國際商業調查機構,
都對本港經濟作出審慎樂觀評估.
穆迪投資維持本港信貸評級不變.
而美國鄧白氏機構調查後,
認爲本港近期經濟有復甦,
包括長期陷於低潮的零售業在內.
信貸評級機構穆迪的佈告指出,
亞洲的經濟增長放緩,
加上中國政治及經濟的因素,

This demo program needs Java Plug-in with Swing support and JMF2.0 support

Query Panel - input query
Library Panel - show all matched results
Player Panel - video playback

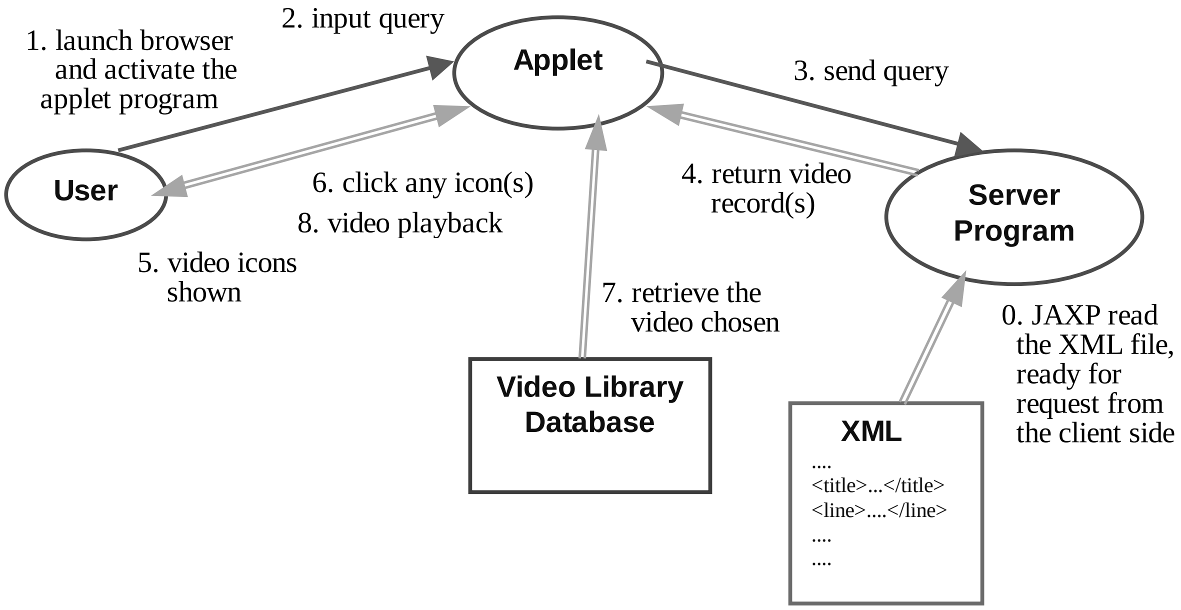
New Features in this version apart from Initial version:

- browser-accessible
- thumbnails in Library Panel
- client/server concept applied

Final Version -- JDVL

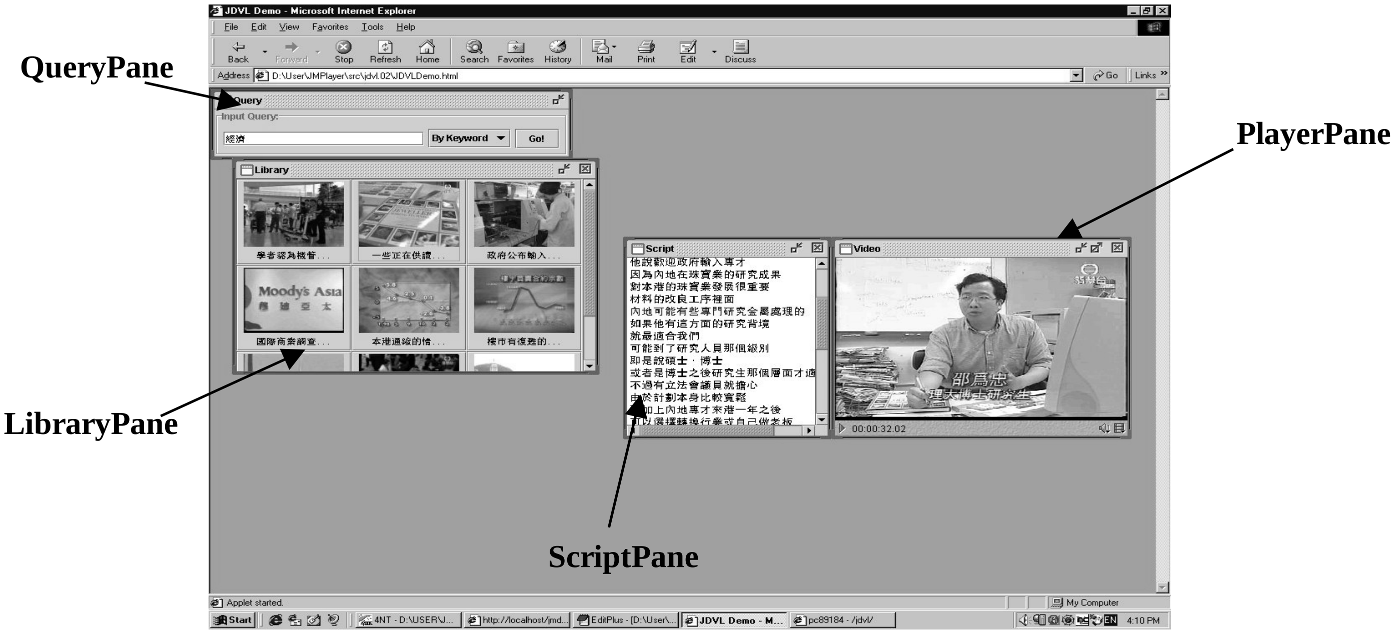
The name JDVL is get from "Java Digital Video Library". The system can be called as Digital Video Library at this stage but not in previous stages. In JDVL, for the client part, we change the interface of JMApplet for multi-video ability and a more convenient usage. We also enhance the video screen with resize ability. On the server side, we develop the searching feature with XML technology. It inherits the browser-accessibility, script synchronizability, video-playback feature, thumbnails and client/server concept from JMApplet.

Let see the system model:



When the server is started up, JAXP (the Java API for XML Parsing) first reads the XML (the Extensible Makeup Language) file prepared before accepts any request from client. This step is only run once when the server is started. The details of JAXP and XML will be talked in the section “System Implementation”. The other steps are shown in the above figure indicated with number. Also, in the later Chapter “Our DVL System”, we will introduce the functionality and library content.

Here is the user interface:



Address D:\User\JMPlayer\src\jdvl.02\JDVLDemo.html

Input Query: 經濟

他說歡迎政府輸入專才
因為內地在珠寶業的研究成果
對本港的珠寶業發展很重要
材料的改良工序裡面
內地可能有些專門研究金屬處理的
如果他有這方面的研究背境
就最適合我們
可能到了研究人員那個級別
即是說碩士．博士
或者是博士之後研究生那個層面才適
不過有立法會議員就擔心
由於計劃本身比較寬鬆
加上內地專才來港一年之後
可以選擇轉換行業或自己做老板

00:00:32.02

JDVL User Interface

New features in JDVL:

- resize of video screen
- multi-video screens
- support multi-client
- use with XML technology in Server side
- allow full-text, keyword and title search

Modules of System in Last Version

After talking about the system models, I would like to look into the modules of the final version JDVL.

List of Modules

Here is the list of modules in JDVL. There are module names, module functionality and constructor of each module.

Module Name	Constructor	Main Function
JDVLMsg	JDVLMsg(int type, int id, String msg)	as a data structure
JDVLVideo	JDVLVideo(Node video)	as a data structure
QueryPane (client side)	QueryPane(JDVLApplet parent)	1)implement the query window 2)let user to input query 3)let user to choose the searching type
LibraryPane (client side)	LibraryPane(JDVLApplet parent)	1)implement the library window 2)display the video icons 3)responsible for user’s mouse actions
PlayerPane	PlayerPane(JDVLApplet parent)	1)implement the video playback issues 2)responsible for the resize of video window
ScriptPane (client side)	ScriptPane(JDVLApplet parent)	1)show the corresponding script 2)responsible for script synchronization

JDVLApplet (client side)	JDVLApplet()	1)initialize the classes used in interface 2)Determine which fonts support Chinese
JDVLServer (server side)	JDVLServer(String xmlLocation)	1)read data file from the xmlLocation 2)respond for any new client 3)searching for the received request

Communication among the Modules

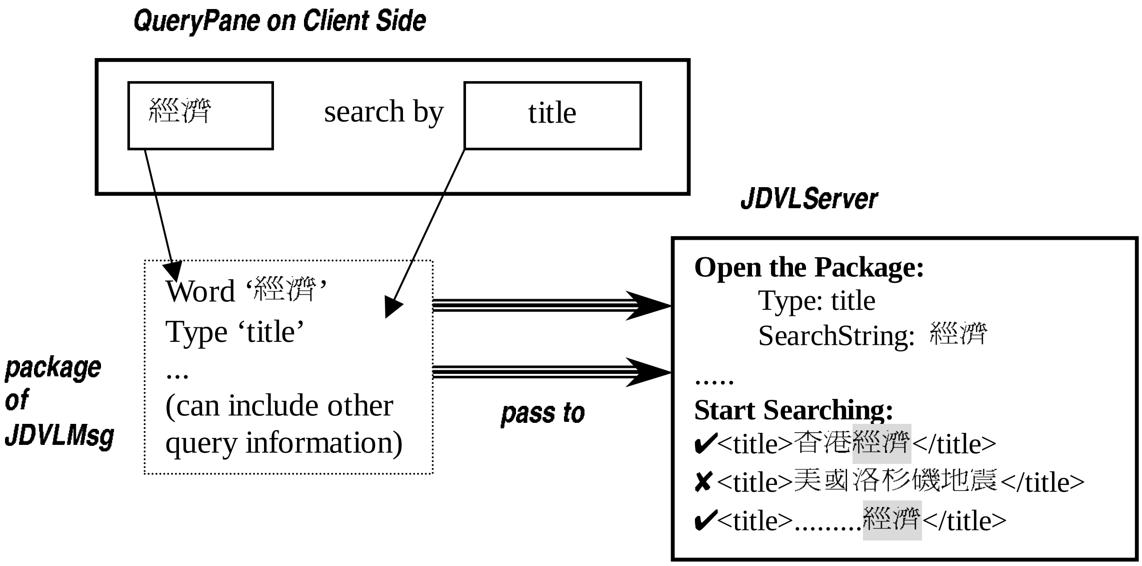
We have designed two classes act as package-like object for the ease of communication between the other modules. They are JDVLMsg class and JDVLVideo class.

JDVLMsg

Parameters in the interface:

- int type: type of the package
- int id: id of the message, reserved for further use
- String msg: the words input by user

It is used by the classes QueryPane and JDVLServer. You can just think that all the query information, e.g. input word and the searching type, are packed into a data structure - the class JDVLMsg.
Then the package is passed to the Server side - the class JDVLServer. Server will retrieve the information stored in this package and performs further task.



JDVLVideo

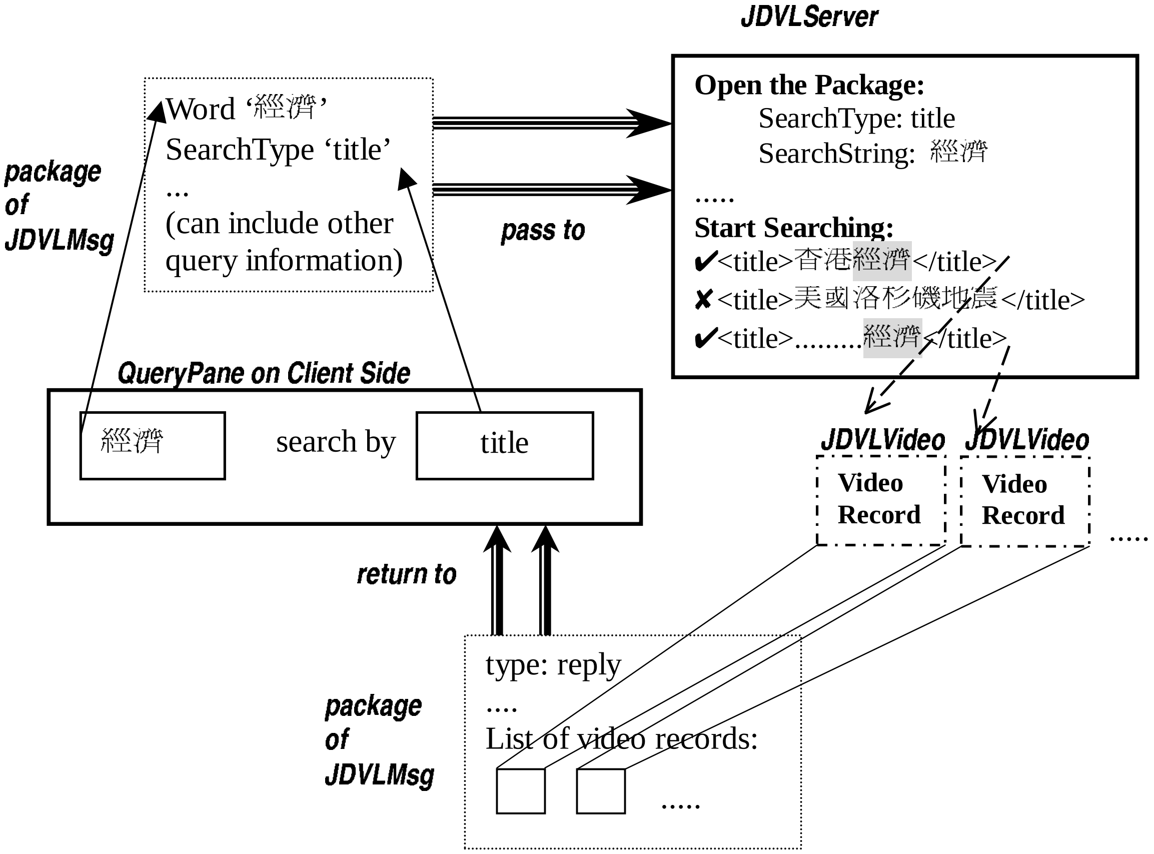
Parameter in the interface:

- Node video: Since the class of Node is an object used in javax.xml.parsers, other modules apart from JDVLServer do not know this class after through the network. Therefore, we need to have a new class JDVLVideo to replace the type of Node which is storing all data of a video record got from a xml file.

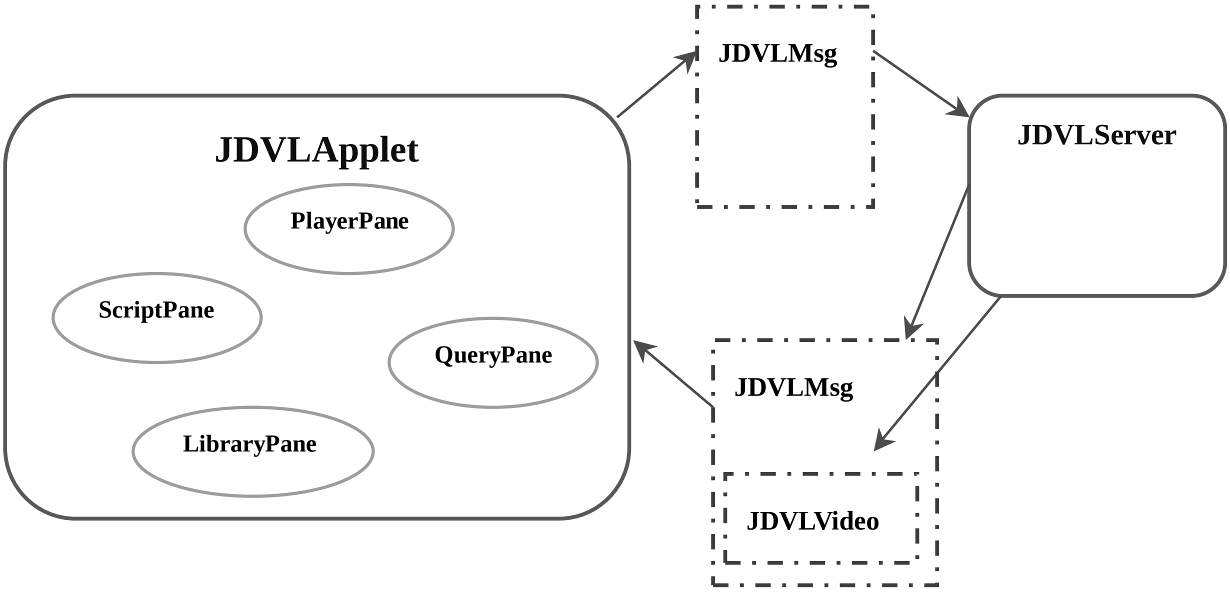
Content in JDVLVideo:

- String title
- String keyword
- String script (script marked with time)
- String fullscript
- iconsrc (the location of image file for the video icon)
- videosrc (the location of the video)

There are the corresponding methods for other modules to retrieve the contents in JDVLVideo class. You can reference to the codes attach in the end of this report. JDVLVideo can be thought as a record of one video. In the Server Side, it is put into the package of JDVLMsg which will be returned to the Client Side. All the 'Pane's can then call the methods to get the field they need to reference.



Graphical Description of Module Communication in the Whole System



In order to cooperate with modules in the system, there must be interface between module and module provided. In the following, I will show you the methods of each Modules.

JDVLMsg

Methods

Return type	**Interface**	**functions**
int	type()	return the type of searching
String	msg()	return a string of message
void	add(Node)	get a Node object and store it as a suitable format in itself (JDVLMsg)

JDVLVideo

Methods

Return type	**Interface**	**functions**
String	title()	retrieve the title
String	script(int)	get the time (the input integer value) and retrieve the corresponding line of script
String[]	script()	return full script
URL	iconsrc()	return the url video icon
URL	videosrc()	return url of video

JDVLApplet

Methods

Return type	Interface	functions
void	loadLibrary(JDVLMsg)	Initialize the Library Pane Show out Library Pane
void	loadMovie(JDVLVideo)	initialize and show the Player Pane and Script Pane in interface

Objects

Object type	Name
Font	cFont

QueryPane

There is no method used by other modules.
Instead, class RequestHandler is a sub-module in QueryPane.

LibraryPane

Methods

Return type	Interface	functions
void	loadLibrary(JDVLMsg)	Get all icons from the JDVLMsg object

PlayerPane

Methods

Return type	Interface	functions
void	doResize()	resize the Player Pane
void	loadMovie(JDVLVideo)	start media player and timer

ScriptPane

Methods

Return type	Interface	functions
void	loadScript(JDVLVideo)	retrieve script from the JDVLVideo object

JDVLServer

Other modules will not access the methods or attributes in Server side, again, there is no common methods shown here.

System Implementation

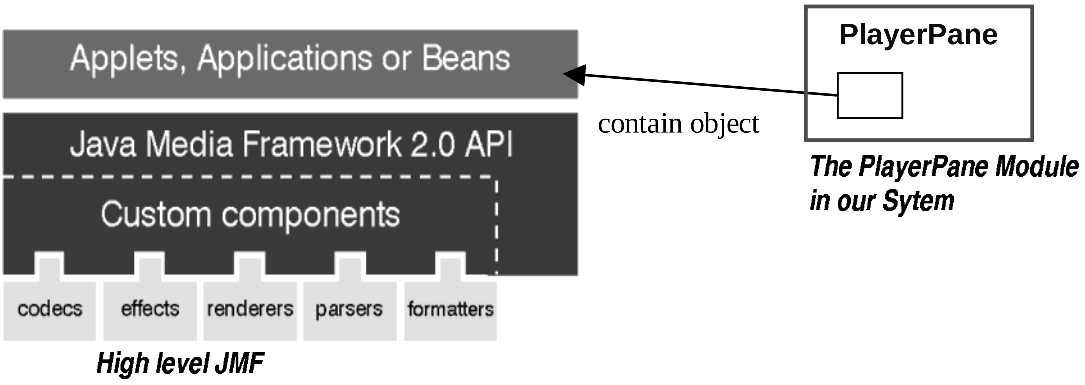
Client Program

On the client side, we have developed the program with the help of extension Java packages. They are javax.swing and Java Multimedia Framework JMF API. javax.swing is used in the building of interface, such as the 'Pane's (QueryPane, LibraryPane...). JMF is responsible for the video playback in the system. In the following section, I would like to introduce JMF API.

Video Playback - JMF API

For the client program of a Digital Video Library, one of the core part is to implement the playback the media data.

In this stage of our work, we have taken the advantage of its feature of presenting time-based media. The package javax.media and javax.media.beam.playerbean is most useful for playback of video files. These packages are included in the Module PlayerPane for the video playback.



MediaPlayer Java Bean

Using the MediaPlayer Java Bean is the simplest way to present media streams. MediaPlayer encapsulates a full-featured JMF Player in a Java Bean. You can either use the MediaPlayer bean's default controls or customize its control Components. One key advantage to using the MediaPlayer bean is that it automatically constructs a new Player when a different media stream is selected for playback. This makes it easy to play a series of media clips or enable the user to select the media clip that they want to play.

To play a media clip with the MediaPlayer bean:

- Construct an instance of MediaPlayer –
MediaPlayer mp1 = new javax.media.bean.playerbean.MediaPlayer();
- Set the location of the clip you want to play:
mp1.setMediaLocation("http://jvideo/Sample1.mov");
- Start the MediaPlayer:
mp1.start();
- You can stop playback by calling stop on the MediaPlayer:
mp1.stop();

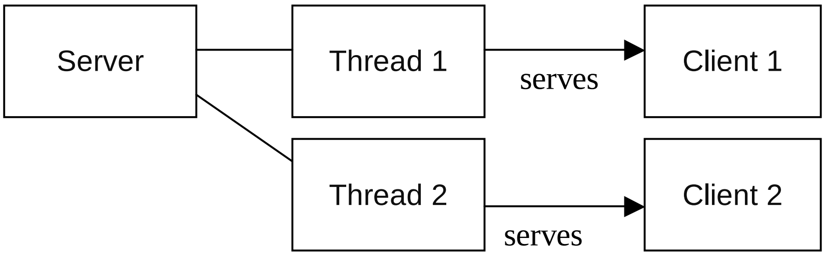
The classes in javax.media.rtp, javax.media.rtp.event, and javax.media.rtp.rtcp provide support for RTP (Real-Time Transport Protocol). RTP enables the transmission and reception of real-time media streams across the network. RTP can be used for media-on-demand applications which may be quite useful in the future versions of the project.

Server Program

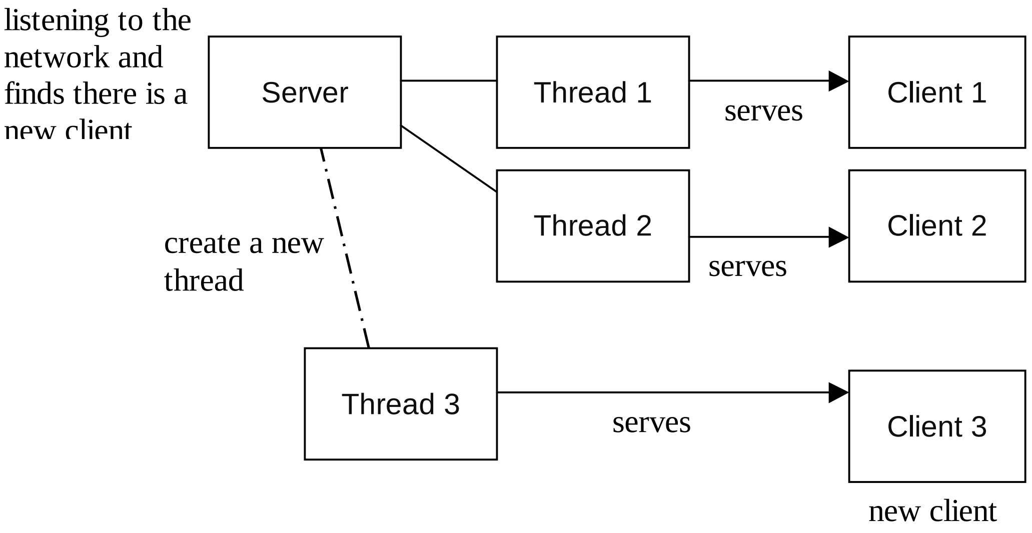
On the server side, it supports multiple clients by using the Java class Thread. We also have developed the program with the help of XML and JAXP on structuring media data. In the later sections, there will be detail explanation on these implementations.

Multi-client Technique

Java Language provides a simple and clear usage on the Java Class - Thread. Whenever the Server listens signal that there is a new client wanting service of Server through the network, Server will create a new Thread which is unique for serving that new client.



Two Clients are connecting



New Client Requesting Service

Structuring and Managing Data

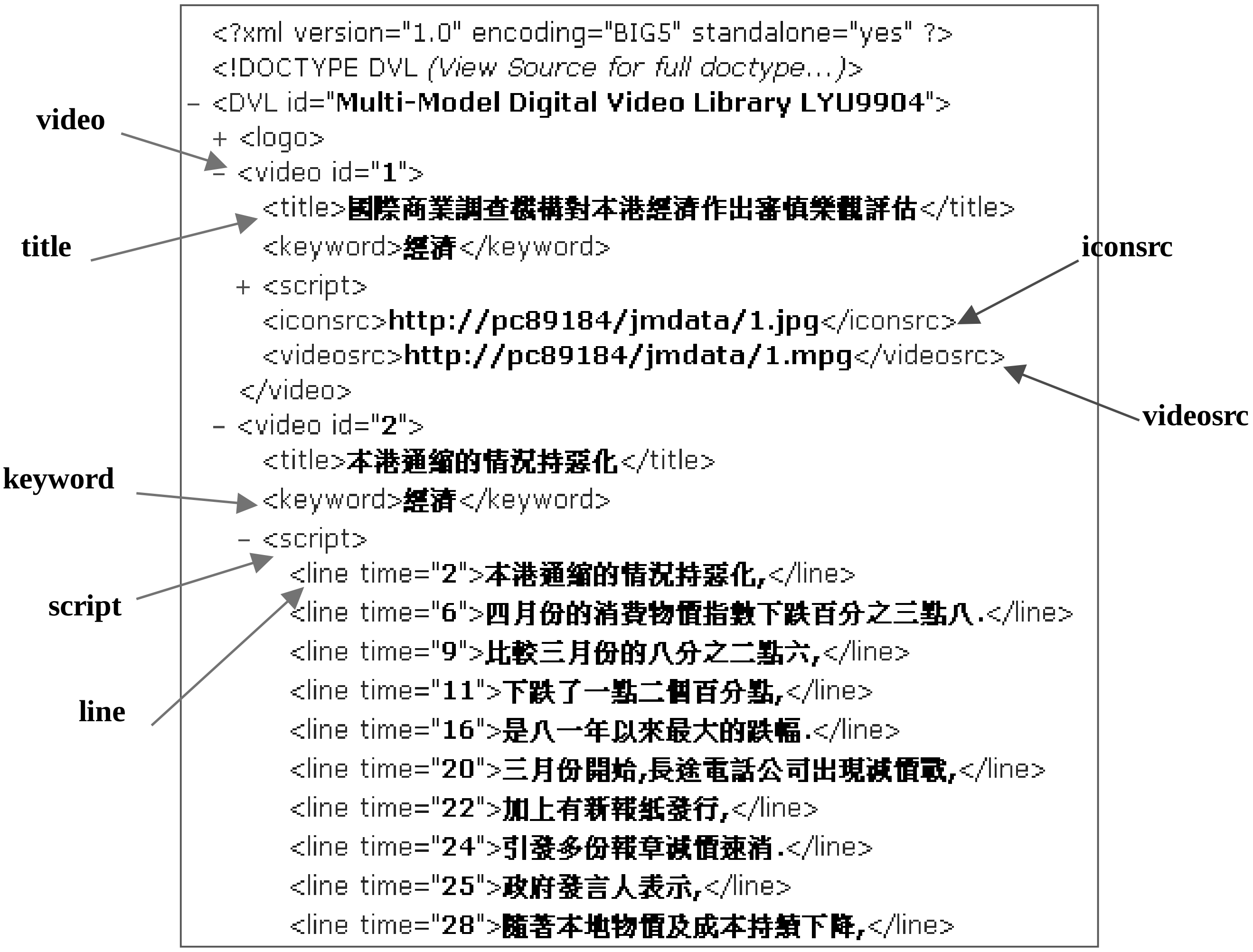
All of the code that we write (in the Java classes) might be considered the Java application layer. Other layers are the XML Parser layer, the XML source (that supplies the XML data that is necessary), and the persistence engine (where the data is actually stored and retrieved by the source).

In our server program, it has to make use of the DOM (Document Object Model) or SAX (Simple API for XML) API and the XML parser in order to access the information in XML documents (that come from the source). The source might be responsible for pulling data from different persistence engines (relational or object databases) and even the web (dynamically generated websites that supply only XML data).

Structuring data - XML

XML - the Extensible Makeup Language which we have used it as the tool for structuring our video data. You can think it as a plain text but organizing the written data in well structure. The reasons why we use XML have been introduced. As said before, there can be persistence engine (e.g. Database engine) to store data and retrieved to XML. But in our project, we manually prepare the XML file as our video database.

Here is a part of our XML data file:



```
  <?xml version="1.0" encoding="BIG5" standalone="yes" ?>
  <!DOCTYPE DVL (View Source for full doctype...)>
- <DVL id="Multi-Model Digital Video Library LYU9904">
  + <logo>
  - <video id="1">
      <title>國際商業調查機構對本港經濟作出審慎樂觀評估</title>
      <keyword>經濟</keyword>
    + <script>
      <iconsrc>http://pc89184/jmdata/1.jpg</iconsrc>
      <videosrc>http://pc89184/jmdata/1.mpg</videosrc>
    </video>
  - <video id="2">
      <title>本港通縮的情況持惡化</title>
      <keyword>經濟</keyword>
    - <script>
        <line time="2">本港通縮的情況持惡化,</line>
        <line time="6">四月份的消費物價指數下跌百分之三點八.</line>
        <line time="9">比較三月份的八分之二點六,</line>
        <line time="11">下跌了一點二個百分點,</line>
        <line time="16">是八一年以來最大的跌幅.</line>
        <line time="20">三月份開始,長途電話公司出現減價戰,</line>
        <line time="22">加上有新報紙發行,</line>
        <line time="24">引發多份報章減價速消.</line>
        <line time="25">政府發言人表示,</line>
        <line time="28">隨著本地物價及成本持續下降,</line>
```

XML lets us to define our own tags to describe data. In our XML document, we define the tags named:

- video
  This tag has an attribute ‘id’ which is unique for each mpg video file. Therefore, if we have 100 mpg video files, there will be 100 ‘video’ tags set with id from 1 to 100. Between the starting tag <video> and ending tag </video>, there are other tags describing this video, including title, keyword, script, line, iconsrc and videosrc. Each video tag set in our XML file contains all of these description fields.

- title
  Between the tag of <title></title>, there is the Chinese title name briefly describing the video.

- keyword
  Between this tag, there is a keyword related to the video. If there are more than one keyword describing the video, each single keyword will have a set of keyword tag

- script
  In this tag, there is the full-text script for the video. Each line of script is tagged with <line> tag.

- line
  The line tag has an attribute 'time'. The time number in the 'time' attribute is used for the synchronization of video playback and the whole video text script.

- iconsrc
  The content in this tag indicates the location of the video icon.

- videosrc
  Similar to iconsrc, it shows the location of the video file.

The above different fields are designed for our system. XML is extensible, so, if we want to add more descriptions for video, e.g. the date of news, we just simply define a new tag for 'date'. You can use XML to model data to any level of complexity and add any tags as needed.

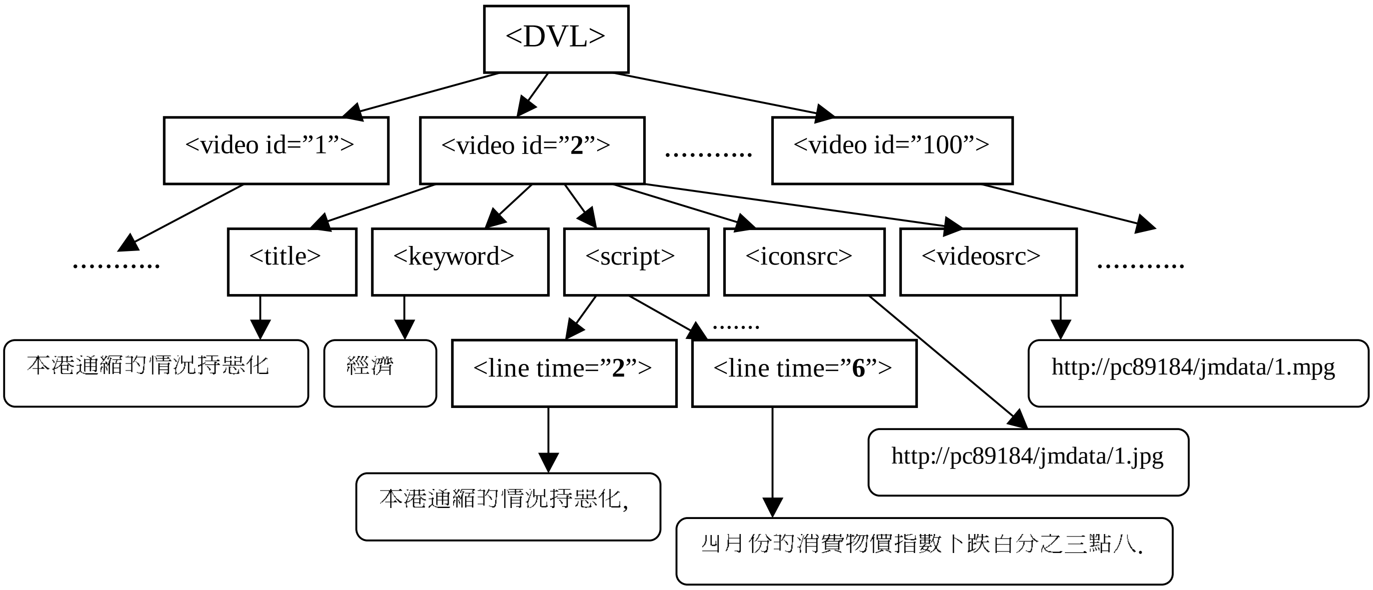
Managing XML Data - JAXP

JAXP - Java API for XML Parsing, is a package of two vendor-neutral classes `SAXParserFactory` and `DocumentBuilderFactory` which can instantiate a SAX Parser and a `DocumentBuilder` respectively.
In our project, we have used the DOM API and JAXP to access the information in our XML document.

Document Object Model (DOM)

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, it is like a "forest" or "grove", which can contain more than one tree. With this tree-like structure of DOM, when we implement the program, we consider the XML document content as a tree. However, the DOM does not specify that documents must be implemented as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner.

The DOM represents our previous example of XML document data like this:



DOM representation of the XML document example

After we understand the logical structure of DOM, we then implement the code in our server program JDVLServer for searching.
Apart from importing some basic java packages, in order to deal with xml and dom, we have to include: 1) javax.xml.parsers.* and 2) org.w3c.dom.*.

DOM benefits that we can easier to retrieve some data in the structure. For example, we need to separate the fields of title, keyword and script in the XML document into three lists: titleList, keywordList and scriptList, we can simply use the methods provided in the included packages:

```
doc = docBuilder.parse (xmlLocation);
scriptList = doc.getElementsByTagName("line");
                                        //preloaded Search by script
keywordList = doc.getElementsByTagName("keyword");
                                        //preloaded Search by keyword
titleList = doc.getElementsByTagName("title");
                                        //preloaded Search by keyword
```

Then, we can use the three lists for searching. Which list we will use depends on what query type user requires.

Since we have studied XML for a very short time, we just take very little part of advantages with XML and JAXP. We can develop our system by using this new language in a wide range of aspects.

Library Preparation

In our digital video library, we need to have a collection of Chinese video. In order to simplify some tasks in preparation of the library, we start with video recorded from news report. To establish the library content, we have to:

- prepare video tapes
  We have a VHS tape recorded with ATV news in Chinese provided by ATV HK. In the second semester, we need to prepare more video, therefore we recorded some Chinese news reports of ATV and TVB from television. We use the video for self use, but do not provide to the public. However, if the system will be provided to public at later time, there must be agreement from these two TV companies before doing so.
- Digitize Video
  We then need to digitize the VHS tape into digital format. We have chosen MPEG1 for encoding video.
- Segmentation of Video
  After digitized the news video, we segment them into small pieces manually. In each piece, there is only talking about one topic of news. Since our project is newly started in this year, it can make the indexing less difficult. Techniques may be applied on the system in the future stages.

Now, we can ready for the preparation of different kind of descriptions for video segments. They are:

- Scripts Preparation
  We have full-text searching and synchronization functions in our system, so we need to prepare the whole script which is what the news reporter speaking in the video.
- Keywords, Titles Preparation
  According to the content of each video, we give some keywords and a title for it.
- Editing Time Stamps
  For the synchronization of script, we need to assign the time (in second unit) on each line of video script.
- Capture of Icons
  In the LibraryPane of the user interface, there are thumbnails shown for user to choose the one he/she needs, the thumbnails are prepared by capturing a representative frame from each video and changing it to jpeg format.

Our DVL System

Functionality of our DVL

In this part, I would like to introduce the functionality of our DVL System.

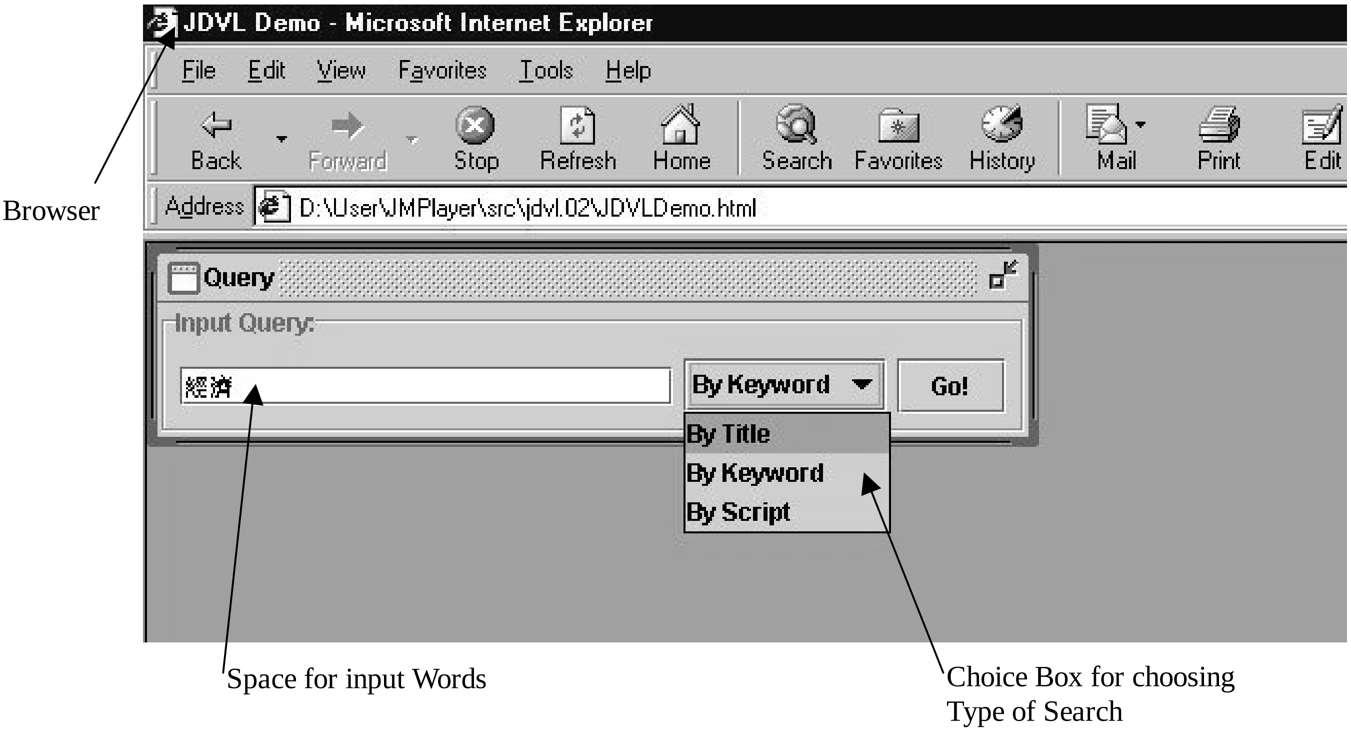
Web-based Application

User can use browser to activate the applet program of our system. User can retrieve video from a remote location through Internet.

Support Chinese on both Chinese/English OS

User can read Chinese words even under English Environment

Searching ability with Keywords, Title, Full Scripts


JDVL Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address D:\User\JMPlayer\src\jdvl.02\JDVLDemo.html
Query
Go!
By Title
By Keyword
By Script
Browser
Space for input Words
Choice Box for choosing Type of Search

Input Query: 經濟

By Keyword

Zoom Appearance of the User Interface when the Applet is started up

Thumbnails

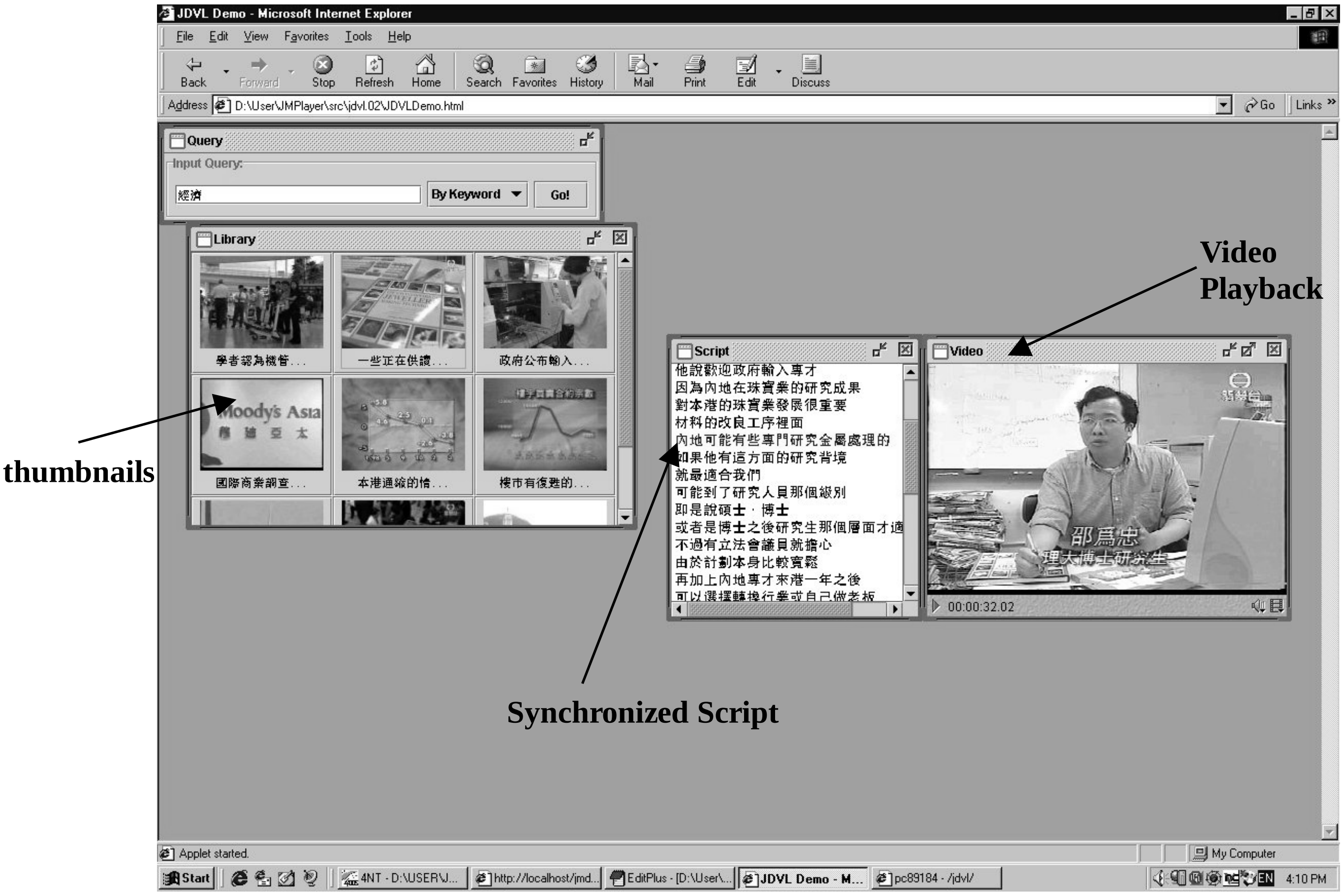
After Searching, LibraryPane shows the video results in the form of icons.   When user click on one of the icon, the video is played back.

Video Playback

Video playback in the PlayerPane.

Synchronized with Scripts

Synchronized Script is shown in the ScriptPane. Current line is highlighted. Which Video is paused, the script also is paused.


JDVL Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address D:\User\JMPlayer\src\jdvl.02\JDVLDemo.html
Query
Library
Script
Video
Video Playback
thumbnails
Synchronized Script
Applet started.
My Computer
Start
4:10 PM

Input Query: 經濟 By Keyword Go!

學者認為機管...
一些正在供讀...
政府公布輸入...
國際商業調查...
本港通縮的情...
樓市有復甦的...

他說歡迎政府輸入專才
因為內地在珠寶業的研究成果
對本港的珠寶業發展很重要
材料的改良工序裡面
內地可能有些專門研究金屬處理的
如果他有這方面的研究背境
就最適合我們
可能到了研究人員那個級別
即是說碩士，博士
或者是博士之後研究生那個層面才適
不過有立法會議員就擔心
由於計劃本身比較寬鬆
再加上內地專才來港一年之後
可以選擇轉換行業或自己做老板

00:00:32.02

MouseOn Descriptions

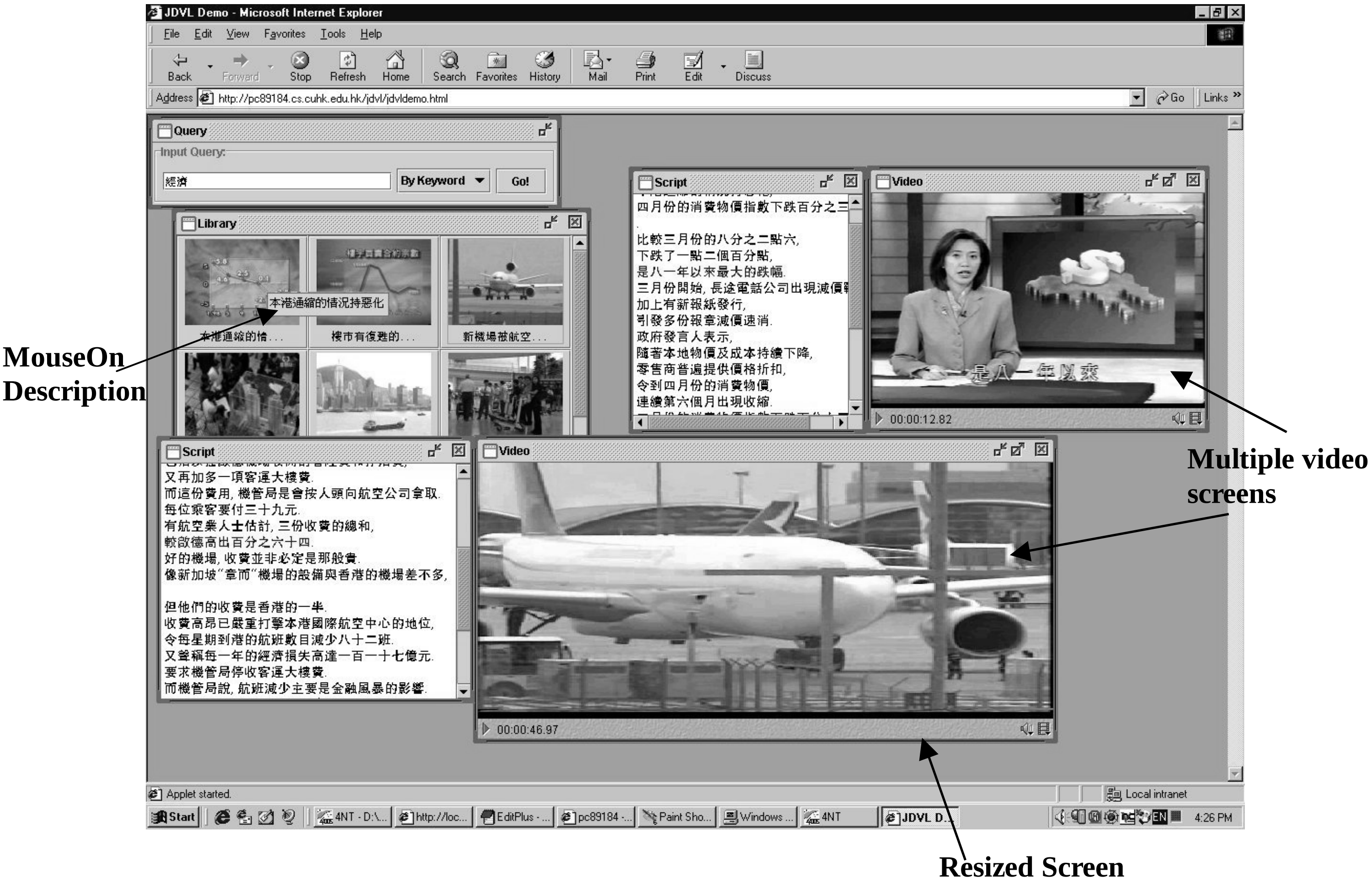
If user put his/her mouse on one of the video icon, there will be a description shown up.

Multiple Video Screens

You not only can open one video, but multiple video screens.

Resize of Video Screen

The video screen can be resized.



JDVL Demo - Microsoft Internet Explorer

Address: http://pc89184.cs.cuhk.edu.hk/jdvl/jdvldemo.html

Input Query: 經濟

本港通縮的情況持惡化

Script:

四月份的消費物價指數下跌百分之三

比較三月份的八分之二點六,
下跌了一點二個百分點,
是八一年以來最大的跌幅.
三月份開始, 長途電話公司出現減價
加上有新報紙發行,
引發多份報章減價速消.
政府發言人表示,
隨著本地物價及成本持續下降,
零售商普遍提供價格折扣,
令到四月份的消費物價,
連續第六個月出現收縮.

Script:

又再加多一項客運大樓費.
而這份費用, 機管局是會按人頭向航空公司拿取.
每位乘客要付三十九元.
有航空業人士估計, 三份收費的總和,
較啟德高出百分之六十四.
好的機場, 收費並非必定是那般貴.
像新加坡“章而”機場的設備與香港的機場差不多,

但他們的收費是香港的一半.
收費高昂已嚴重打擊本港國際航空中心的地位,
令每星期到港的航班數目減少八十二班.
又聲稱每一年的經濟損失高達一百一十七億元.
要求機管局停收客運大樓費.
而機管局說, 航班減少主要是金融風暴的影響.

Applet started.

Other features have not shown in the interface:

- Streaming
- Support Number of Video Formats

Library Content

We have made 70’s clips of video. Each of the video file talks about one topic of News in Chinese. There is a link in which you can read our xml file. In it., you can see our list of video collection and their other descriptions. That is:

http://pc89184.cs.cuhk.edu.hk/jmdata/jmdata.xml

If you want to use our system, client and server should have JMF2.0 and JDK1.2.2. For the server, it also needs JAXP1.0. You can visit the following sites:

http://pc89184.cs.cuhk.edu.hk/jdvl/JDVLDemo.html or
http://www.cse.cuhk.edu.hk/~lyu9904/

Discussion

Problem Encountered

Hard to Define a Suitable Scope of Our Work

During the summer holiday in 1999, we studied the papers about different aspect of digital video library. As there are many different technologies involved in DVL, and many of them needs advance knowledge and deep understanding of the specialized field, so it is impossible to implement the full feature DVL by ourselves within one year.

After the first semester began, we finally decided to develop a small scale DVL project, which started by implement a program that can play video files. In the second semester, we got a clearer direction and start to enhance the simple system done in first semester with better interface, browser-accessibility, client/server concept, searching function, and so on. Lastly, we have the system already shown to you in previous sections.

Choosing Programming Language

In first semester, right after we had defined our working direction, we felt into trouble of the techniques needed for implementation. For playback of media data, MPEG1 files in specific, we only knew that we might write our own MPEG-decoder or use the DirectX API in the beginning. But after some investigation, we found that writing a MPEG-decoder by ourselves will involve too much work, which will be too low level and deviated from the high-level system design and so our project target. For DirectX API, it is powerful, but from the experience of other FYP teams, it takes a long time to study; also, it is platform dependent, which is not desired for the interoperability of the software. After more investigation, we found the Java Media Framework API and realized it’s advantage over DirectX, including high-level of abstraction and ready to work over the network, which is very favorable for our project. And therefore we finally choose Java as the programming language for our project.

Preparing Video and Synchronized Transcript

In this stage, we prepared the videos and synchronous transcripts of our database manually, which was a tedious and time-consuming process. Not only digitizing the videotape consume time, but also segmenting the video files into small pieces.

But spending most of the process time was the preparation of text scripts. Since it is a full-content text script of video, we had to listen all the words spoken in the video carefully and typed them into the script files. Apart from that, timestamps will be added into the transcripts for synchronization. To finish all these pre-process on a two-minute video segment will spend more than half an hour.

Display Chinese in English OS

Our digital video library is in Chinese. User should be able to read the Chinese words when using our application. However, we find that our Java program cannot display Chinese word in Browser under English Operating System. We have tried to install different software which is used for reading Chinese in English OS (e.g. NJStar and 中文之星). Lastly, we solve this problem. Actually, the java program need to find out which type of Chinese font the machine on client side can support. Then program uses that font to display all Chinese words in the user interface.

Things Learned

We have learned the following things from our project:

- Different Issues of a Digital Video Library
- Building GUI with Java
- Building Applications with JMF API
- Building Applet program
- Some issues on Java network programming
- The concept of XML & JAXP
- The Java program implementation with JAXP
- Digitizing and cutting of Video

Possible Extensions

More functions on the User Interface

There can be more functions provided in the user interface, such as a world map in which lighting effect or something else will indicate the place (country or city) the video of news happened. Or if user point to a place on the map using a mouse, news icons happened in this place will be shown.

Also, there are still many searching type can be implement in later stages. Of course, the video library must be scaled up, and the indexing technique must be improved too.

Skimming of Video

For the current system we have built, after a user click on one of the video icons shown in LibraryPane, there is the video playback. However, we can improve the system with skimming-like technique. After an video icon is clicked, there can be some main frames of the video shown first.

Ranking of Video Results

After Searching, the video icons are shown. We can apply weighting techniques to identify the most relevant keywords and phrases in the transcript. Then rank the video icon corresponding to their weighting score.

Using Database Engine to Store XML documents

Now, we just use a single XML document to store video data. When the size of library is scaling up, a single document may be not efficient enough. The technique in exchanging of XML data may be needed.

The following are some extensions needed more technology support:

Automatically Derived Transcripts

Manually prepare the transcript of video is a time consuming job, which is not suitable for large scale DVL. Auto-transcribing should be implemented to reduce the manual work. Techniques of continuous speech recognizer, probably aided by the existence of auxiliary vocabularies from closed-captioning or available scripts, is needed to implement this feature,.

Language Processing of Queries and Transcripts

To further improve the accuracy of query results, linguistic indexing can be used to the derived transcript, generate lexicons, equivalence classes, and search indices. Techniques of natural language processing will be needed.

Content-based Image Manipulation

Image understanding plays a critical role for organizing, searching, and reusing digital video. Yet, the traditional database search by keywords, where images are only referenced, not directly searched for, is not appropriate or useful enough. Instead, digital video images themselves must be segmented, searched for, manipulated, and presented for similarity matching, parallel presentation, context sizing, and skimming, while preserving image content. For this extension, techniques including comprehensive image statistics, camera motion , object presence, object and scene understanding in 3D may be employed.

Conclusion

We have studied on issues of Digital Video Library for our project. After a year work, now, we have built a small-scaled Chinese digital video library. Our system has features of Browser-accessibility, a styled user interface, simple searching function, video playback and those functions mentioned before. We also produced 70's video segments of Chinese news for the video library. After working with this project for one year, we have learned the concepts of some technologies in this topic. We also have practised on the programming with Java and the programming tools needed. Although we have spent one-year time on this project, there are still lot of new techniques in DVL we do know. Hope that this project can be continuously developed in the future. We believe that there is really a great space for us to extend.

Acknowledgement

We would like to thank our supervisor, Prof. Michael Lyu, to give us valuable advice in our work. Our marker, Prof. Irwin King, gave us idea on our work.
We would also like to thank the following people who did give us a hand: Mr. Tim, Technical Staff, CSE Department, CUHK; Mr. Tony, Technical Staff, CSE Department, CUHK; Vincent Cheung, M.Phil Student, CSE Department, CUHK; Anson Lee, M.Phil Student, CSE Department, CUHK; Mole, Fellow Classmate, CSE Department, CUHK.

Reference

[Informedia CMU] http://www.informedia.cs.cmu.edu/
Informedia Digital Video Project, Carnegie Mellon University
[Java Online Document] http://java.sun.com/

[XML & JAXP] http://java.sun.com/xml/

[Quicktime for Java]
http://developer.apple.com/quicktime/qtjava/index.html

Appendix

JDVLMsg.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import org.w3c.dom.*;

/**
 * The class is encapsulate the message to be transmit across the network
 * @author Jacky & Joan
 * @version 2.0
 */
public class JDVLMsg implements Serializable{

  static final int UNDEFINED=0;
  static final int KEYWORD=1;
  static final int SCRIPT=2;
  static final int TITLE=3;
  static final int REPLY=4;
  static final int ERROR=5;
  static final int ADMIN=6;

  int id=0;
  HashSet archive=new HashSet();
  String msg=new String();
  int type;

  /**
   * Constructor
   * @type type of the message
   * @id id of this message, reserve for future use.
   */
  public JDVLMsg(int type, int id){
    this.id=id;
    this.type=type;
  }

  /**
   * Constructor
   * @type type of the message
   * @id id of this message, reserve for future use.
   * @msg message to be process, maybe error message or query string
   */
  public JDVLMsg(int type, int id, String msg){
    this.id=id;
    this.type=type;
    this.msg=new String(msg);
  }

  public int id(){
    return id;
  }

  public int type(){
    return type;
  }

  public String msg(){
    return msg;
  }

  public HashSet archive(){
    return archive;
  }
  public String toString(){
    return new String("[id:" + id + " type:" + type + " size:" + archive.size() + "]");
  }

  public void add(Node video){
    if(type==REPLY)archive.add(new JDVLVideo(video));
    //else threw an error!
  }
}
```

JDVLVideo.java

```
import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;
/**
 * The class is encapsulate the message to be transmit across the network
 * @author Jacky & Joan
 * @version 2.0
 */
public class JDVLVideo implements Serializable{

  String title;
  String keyword;
  String[] script;
  String[] fullscript;
  String iconsrc;
  String videosrc;


  public JDVLVideo(Node video){
    //videosrc=video.toString();

    if(video.getChildNodes() != null) {
      System.out.println("[Video]");
      for(int i=0; i<video.getChildNodes().getLength(); ++i) {
        String Nodename=video.getChildNodes().item(i).getNodeName();
        if( Nodename.equal("title")) {
          title = video.getChildNodes().item(i).getFirstChild().getNodeValue();
          System.out.println("title : " + title);
        }else if(Nodename.equal("keyword") {
          keyword = video.getChildNodes().item(i).getFirstChild().getNodeValue();
//        System.out.println("keyword : " + keyword);
        }else if(Nodename.equal("videosrc") {
          videosrc = video.getChildNodes().item(i).getFirstChild().getNodeValue();
          System.out.println("videosrc : " + videosrc);
        }else if(Nodename.equal("iconsrc") {
          iconsrc = video.getChildNodes().item(i).getFirstChild().getNodeValue();
          System.out.println("iconsrc : " + iconsrc);
        }else if(Nodename.equal("script") {

          Vector timeVector = new Vector();
          Vector sentenceVector = new Vector();
          int childNum = video.getChildNodes().item(i).getChildNodes().getLength();
//        System.out.println("childNum == "+ childNum);
          int lineCount=0;
          for(int j=0; j<childNum; ++j) {
            if(video.getChildNodes().item(i).getChildNodes().item(j).getNodeName()
== "line") {

 timeVector.add(video.getChildNodes().item(i).getChildNodes().item(j).getAttribute
s().getNamedItem("time").getNodeValue());

 sentenceVector.add(video.getChildNodes().item(i).getChildNodes().item(j).getFirst
Child().getNodeValue());
//            System.out.println("element at " + j + " is " +
timeVector.elementAt(lineCount) + "  " + sentenceVector.elementAt(lineCount));
              ++lineCount;
            }
          }

          //copy the content in Vectors to the script array
//        System.out.println("time length of script : " +
(String)timeVector.lastElement());
          int numberOfSecond = (new
Integer((String)timeVector.lastElement())).intValue();

          script = new String[numberOfSecond];
          int count=0;
          int time = (new Integer((String)timeVector.elementAt(count))).intValue();
          String sentence=new String("");
          sentence = (String)sentenceVector.elementAt(count);
          for(int k=0; k<numberOfSecond; ++k) {
            if(k == time) {
              ++count;
              sentence = (String)sentenceVector.elementAt(count);
              time = (new Integer((String)timeVector.elementAt(count))).intValue();
            }
            script[k] = sentence;
          }
          fullscript=new String[sentenceVector.size()];
          fullscript=(String[])(sentenceVector.toArray(fullscript));
        }
      }
    }
    else {
      System.out.println("[Empty]");
    }
  }

  public URL videosrc(){
    try{
      URL videoLoc = new URL(videosrc);
      return videoLoc;
    }catch(Exception x){
      System.out.println(x);
    }
    return null;
  }

  public URL iconsrc(){
    try{
      URL icon = new URL(iconsrc);
      return icon;
    }catch(Exception x){
      System.out.println(x);
    }
    return null;
  }

  public String title(){
    return title;
  }

  public String script(int time) {
    return script[time];
  }

  public String[] script(){
    return fullscript;
  }
}
```

QueryPane.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;


public class QueryPane extends JInternalFrame implements ActionListener{

  JDVLApplet jdvlapplet;
  JTextField textField;
  JComboBox combobox;

  public QueryPane (JDVLApplet parent){
    super("Query", false, false, false, true) ;
    setLocation(0,0);
    setVisible(true);

    jdvlapplet=parent;

    JPanel contentPane = new JPanel();

 contentPane.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtched
Border(), "Input Query:"));

    textField=new JTextField("", 40);
    contentPane.add(textField);

    String[] qryTypeStr = { "By Title", "By Keyword", "By Script"};
    combobox=new JComboBox(qryTypeStr);
    combobox.setSelectedIndex(1);
    contentPane.add(combobox);

    JButton submitButton=new JButton("Go!");
    submitButton.addActionListener(this);
    contentPane.add(submitButton);

    setContentPane(contentPane);
    pack();
  }

  public void actionPerformed(ActionEvent e){
    String cmd=e.getActionCommand();
    jdvlapplet.console.println(cmd);

    String qryTypeStr=(String)combobox.getSelectedItem();
    int qryType=JDVLMsg.UNDEFINED;

    if(qryTypeStr.equals("By Keyword")){
      qryType=JDVLMsg.KEYWORD;
    }else if(qryTypeStr.equals("By Script")){
      qryType=JDVLMsg.SCRIPT;
    }else if(qryTypeStr.equals("By Title")){
      qryType=JDVLMsg.TITLE;
    }

    JDVLMsg qryMsg=new JDVLMsg(qryType, 0, textField.getText());

    try{
      InetAddress iAdr=InetAddress.getByName(jdvlapplet.serveraddr);
      Socket s=new Socket(iAdr, jdvlapplet.serverport);

      OutputStream os=s.getOutputStream();
      ObjectOutputStream oos = new ObjectOutputStream(os);
      oos.writeObject(qryMsg);
      jdvlapplet.console.println("msg sent!");

      InputStream is=s.getInputStream();
      ObjectInputStream ois = new ObjectInputStream(is);
      JDVLMsg resultMsg=(JDVLMsg)ois.readObject();
      s.close();

      jdvlapplet.console.println("got reply!");
      jdvlapplet.console.println("info: " + resultMsg.toString());

      jdvlapplet.loadLibrary(resultMsg.archive());

    }catch(Exception x){
      jdvlapplet.console.println(x.toString());
    }
  }
}
```

LibraryPane.java

```
import java.awt.*;
import java.awt.event.* ;
import java.applet.*;
import javax.swing.*;
import java.net.*;
import java.util.*;

/**
 * This class shows icons for user to choose a video
 */
public class LibraryPane extends JInternalFrame implements ActionListener{

  JDVLApplet jdvlapplet;
  MyIconLibrary iconpanel;
  JScrollPane contentPane;
  Vector searchResult=null;
  MyIcon icon[];

  /**
   * Constructor
   * @param owner The owner of the dialog
   */
  public LibraryPane(JDVLApplet parent){

    super("Library", true, true, true, true) ;
    setVisible(true);
    setSize(300,200);
    jdvlapplet=parent;
    jdvlapplet.console.println("Library");
    contentPane = new JScrollPane();
//
 contentPane.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtched
Border(), "Query Results:"));

    iconpanel=new MyIconLibrary();


//
  contentPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    contentPane.setViewportView(iconpanel);
    setContentPane(contentPane);

    addComponentListener( new ComponentAdapter() {
      public void componentResized(ComponentEvent ce) {
        doResize();
      }
    });
  }

  public void doResize(){
    if(searchResult!=null){
      iconpanel.setPreferredSize(getSize());
//    iconpanel.setPreferredSize(new Dimension(getSize().width,
(searchResult.size()/(getSize().width/140))*200));
      iconpanel.revalidate();
    }
  }

  public void loadLibrary(HashSet searchResult){
    this.searchResult=new Vector(searchResult);
    JDVLVideo tempObject;
    icon=new MyIcon[this.searchResult.size()];

    iconpanel.removeAll();
    for(int i=0; i<this.searchResult.size(); i++){
      tempObject=(JDVLVideo)(this.searchResult.elementAt(i));
      icon[i]=new MyIcon(tempObject);
      icon[i].addActionListener(this);
      iconpanel.add(icon[i]);
    }
    doResize();
  }

  public void actionPerformed(ActionEvent e){
    if(e.getSource() instanceof MyIcon){
      MyIcon tempicon = (MyIcon)e.getSource();
      jdvlapplet.loadMovie(tempicon.source());
    }
  }
}


class MyIconLibrary extends JPanel implements Scrollable{

  public Dimension getPreferredScrollableViewportSize(){
    return getPreferredSize();
  }

  public int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int
direction){
    return 80;
  }

  public int getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int
direction){
    return visibleRect.height;
  }

  public boolean getScrollableTracksViewportWidth(){
    return true;
  }

  public boolean getScrollableTracksViewportHeight(){
    return false;
  }
}

class MyIcon extends JButton{
  JDVLVideo obj;
  public MyIcon(JDVLVideo obj){
    super(new ImageIcon(obj.iconsrc())) ;
    String title=obj.title();
    setToolTipText(title);
    try{
      title=obj.title().substring(0,6) + "...";
    }catch(Exception x){
    }
    setText(title);
    this.obj=obj;
    setVerticalTextPosition(BOTTOM);
    setHorizontalTextPosition(CENTER);
    setMargin(new Insets(0, 0, 3, 0));
//  setToolTipText(obj.title());
    setSize(120, 90);
  }

  public JDVLVideo source(){
    return obj;
  }
}
```

PlayerPane.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.net.*;
import java.io.*;
import javax.media.*;
import javax.media.MediaLocator;
import javax.media.bean.playerbean.MediaPlayer;

/**
 * This class shows icons for user to choose a video
 */
public class PlayerPane extends JInternalFrame{

  JDVLApplet jdvlapplet;
  JPanel contentPane;
  MediaPlayer mplayer;
  URL videosrc=null;
  Timer myTimer;

  /**
   * Constructor
   * @param owner The owner of the dialog
   */
  public PlayerPane(JDVLApplet parent){

    super("Video", true, true, true, true) ;
    setVisible(true);
    setSize(360, 280);

    jdvlapplet=parent;

    mplayer = new javax.media.bean.playerbean.MediaPlayer();
    mplayer.setControlPanelVisible(true);
    mplayer.setPopupActive(false);
    mplayer.setPlaybackLoop(false);
    mplayer.setFixedAspectRatio(false);

    setContentPane(mplayer);

    addInternalFrameListener(new frameHandler());
    addComponentListener( new ComponentAdapter() {
      public void componentResized(ComponentEvent ce) {
        doResize();
      }
    });
  }


  public void doResize(){
    Dimension d = getSize();
    if (mplayer != null) {
      mplayer.setBounds(0, 0, d.width, d.height);
      validate();
    }
   }

  public void loadMovie(JDVLVideo obj){
    videosrc=obj.videosrc();
    mplayer.setMediaLocator(new MediaLocator(videosrc));
    if (mplayer.getPlayer() == null)
      jdvlapplet.console.println("Could not create player for " + videosrc.toString());
    else
      mplayer.start();
    doResize();
//  mplayer.setBounds(0, 0, contentPane.getWidth(), contentPane.getHeight());
  }

  public void startMovie(){
    if(videosrc!=null) {
      mplayer.start();
    }
  }

  public void stopMovie(){
    if(videosrc!=null){
      mplayer.stop();
    }
  }

  public void destroy() {
    mplayer.close();
  }

  class frameHandler extends InternalFrameAdapter{

    public void internalFrameClosing(InternalFrameEvent e){
      destroy();
    }

    public void internalFrameDeiconified(InternalFrameEvent e){
      startMovie();
    }

    public void internalFrameIconified(InternalFrameEvent e){
      stopMovie();
    }
  }
}
```

ScriptPane.java

```
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;


public class ScriptPane extends JInternalFrame{

  JDVLApplet jdvlapplet;
  JEditorPane scriptArea;
  String script[];

  public ScriptPane (JDVLApplet parent){
    super("Script", true, true, false, true) ;
    setLocation(0,100);
    setVisible(true);
    setSize(250,280);

    jdvlapplet=parent;

    scriptArea = new JEditorPane("text/html; charset=BIG5", "");
    scriptArea.setEditable(false);
    scriptArea.setFont(jdvlapplet.cFont);
    scriptArea.setEditorKit(new javax.swing.text.html.HTMLEditorKit());

    JScrollPane areaScrollPane = new JScrollPane(scriptArea);

  areaScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    setContentPane(areaScrollPane);
  }

  public void loadScript(JDVLVideo video){
    script=video.script();
    String longString=new String("<html><head><title></title></head><body>\n");


    for(int i=0; i<script.length; i++)
      longString+=(script[i]+"<br>\n");

    longString+="</body></html>";

    scriptArea.setText(longString);
  }

  public void SynScript(int idx){
    String longString=new String("<html><head><title></title></head><body>\n");

    for(int i=0; i<script.length; i++){
      if(i==idx)
        longString+=("<a name=\"current\"><b>"+script[i]+"</b><br>\n");
      else
        longString+=(script[i]+"<br>\n");
    }

    longString+="</body></html>";

    scriptArea.setText(longString);
    scriptArea.scrollToReference("current");
  }
}
```

JDVLApplet.java

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import org.w3c.dom.*;
```

```
public class JDVLApplet extends JApplet{

  public int serverport=4000;
  public String serveraddr="pc89184.cs.cuhk.edu.hk";
  JDesktopPane desktopPane;
  public QueryPane queryPane;
  public ConsolePane console;
  public LibraryPane libraryPane=null;
  public PlayerPane playerPane=null;
  public ScriptPane scriptPane=null;
  public cFont=null;

  public void init(){
    serveraddr=getCodeBase().getHost();
    setupInterface();
  }

  public void start(){
    try{
      console.setIcon(true);
    }catch(Exception x){
    }
  }

  public void stop() {
    if(playerPane!=null)
      playerPane.stopMovie();
  }

  public void destroy() {
    if(playerPane!=null)
      playerPane.destroy();
  }

  public void setupInterface(){

    desktopPane=new JDesktopPane();
    /*JInternalFrame(String title,
             boolean resizable,
             boolean closable,
             boolean maximizable,
             boolean iconifiable)
    */


    queryPane=new QueryPane(this);
    console=new ConsolePane(this);

    desktopPane.add(queryPane);
    desktopPane.add(console);
    desktopPane.setDoubleBuffered(true);

    setContentPane(desktopPane);
  }

  void GetChineseFont(){
    // Determine which fonts support Chinese here ...
    Font[] allfonts =
GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
    int fontcount = 0;
    String chinesesample = "\u4e00";

    int j = 0;

    while((j < allfonts.length) && (cFont==null)){
      if (allfonts[j].canDisplayUpTo(chinesesample) == chinesesample.length())
        cFont=new Font(allfonts[j].getFontName(), Font.PLAIN, 12);
      j++;
    }
  }

  public void loadLibrary(HashSet searchResult){
    libraryPane=new LibraryPane(this);
    desktopPane.add(libraryPane);
```

```
      libraryPane.show();
      libraryPane.loadLibrary(searchResult);
    }

    public void loadMovie(JDVLVideo video){
      scriptPane=new ScriptPane(this);
      desktopPane.add(scriptPane);
      scriptPane.setLocation(100,200);
      scriptPane.show();
      scriptPane.loadScript(video);


      playerPane=new PlayerPane(this);
      desktopPane.add(playerPane);
      playerPane.setLocation(350,200);
      playerPane.show();
      playerPane.loadMovie(video);
    }


  }
```

JDVLServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class JDVLServer extends Object{
  int port=4000;
  boolean run=true;
  Document doc;
  NodeList scriptList, keywordList, titleList;

  public static void main(String[] args) {
    JDVLServer server;
    if(args.length==0){
      System.out.println("Use default:
[http://pc89184.cs.cuhk.edu.hk/JMData/jmdata.xml]");
      server=new JDVLServer("http://pc89184.cs.cuhk.edu.hk/JMData/jmdata.xml");
    }else{
      server=new JDVLServer(args[0]);
    }

  }

  public JDVLServer(String xmlLocation){

    try {
      DocumentBuilder docBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();

      //Load records from xml file
      System.out.print("Loading data...");
      doc = docBuilder.parse (xmlLocation);
//    doc.normalize();
      scriptList = doc.getElementsByTagName("line");  //preloaded Search by script
      keywordList = doc.getElementsByTagName("keyword");  //preloaded Search by
keyword
      titleList = doc.getElementsByTagName("title");  //preloaded Search by keyword
      System.out.println("ok");

      System.out.println("Server started.");
      //Listen to requests
      Listening();
      System.out.println("Server stoped.");

    }catch(Exception e){
      System.out.println(e);
    }



  }

  public void Listening(){
    try{
      ServerSocket acceptsocket=new ServerSocket(port);
      System.out.println("Server listening on port: " + port);

      do{
        Socket s = acceptsocket.accept();
        new RequestHandler(s);
      }while(run);
    }catch(Exception x){
      System.out.println(x);
    }
  }

  //RequestHandler to handle the requests
  class RequestHandler implements Runnable{
    Thread t;
    Socket s;

    public RequestHandler(Socket s){
      if(s!=null){
        this.s = s;
        t = new Thread(this, "RequestHandler");
        t.start();
      }
    }

    public void run(){
      try{
        InputStream is=s.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(is);
        JDVLMsg receiveMsg=(JDVLMsg)ois.readObject();   //read input

        System.out.println("[Receive " + receiveMsg + " ]");
        JDVLMsg replyMsg=process(receiveMsg);       //process query

        OutputStream os=s.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(os);
        oos.writeObject(replyMsg);             //reply

        System.out.println("[Reply " + replyMsg + " ]");
        s.close();                       //close connection
      }catch(Exception x){
        System.out.println(x);
      }
    }

    public JDVLMsg process(JDVLMsg msg){
      JDVLMsg reply=new JDVLMsg(JDVLMsg.REPLY, msg.id());
      String currentline;
      Node video;
      NodeList searchList=null;
      HashSet replyall=new HashSet();

      int type=msg.type();

      switch(type){
        case JDVLMsg.KEYWORD:searchList=keywordList;break;
        case JDVLMsg.SCRIPT:searchList=scriptList;break;
        case JDVLMsg.TITLE:searchList=titleList;break;
        default:break;
      }

      String qryStr=msg.msg();
      if (searchList!=null){
        System.out.println("[process query...]");
        for(int i=0; i<searchList.getLength(); i++){
          currentline=searchList.item(i).getFirstChild().getNodeValue();
          if (currentline.indexOf(qryStr)!=-1){
            if(type==JDVLMsg.SCRIPT)
              video=searchList.item(i).getParentNode().getParentNode();
  //line->script->video
            else
              video=searchList.item(i).getParentNode(); //title->video,
keyword->video
            replyall.add(video);
          }
        }
        Vector temp=new Vector(replyall);
        for(int i=0; i<temp.size(); i++)
          reply.add((Node)temp.elementAt(i));
      }

      return reply;
    }

  }

}
```