LYU9904 Final Report

Multi Model Digital Video Library

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

Supervisor:
Prof. Michael Lyu

Student:
Jacky Ma
Joan Chung

Author of report:
Jacky Ma

Abstract

This project, titled “Multi-Model Digital Video Library”, is targeted to built the skeleton of a DVL system. We are aimed at an extensible framework that is modular in design, such that enhancements can be implemented incrementally. In our implementation, the system includes over 1Gb of video content, a query server, and a Java Applet Client which users can do text-based search, browsing the result set quickly, and select videos to playback.

In this report, we will talk about the backgrounds of Digital Video Library Systems, including the important issues and the techniques addressing the issues. Then we will move onto the approach we taken in this project: how we defined our focus, what kind of tools did we use, and also some possible system models we could chosen from. After introducing the approach, we will draft an outline of our implementation, both in a system perspective and a modular perspective, as well as introduce our user interface to end this section. Then we will go through a discussion some issues on the implementation and come to a conclusion finally.

Contents

1. Backgrounds

Vast digital libraries of information will soon be available on the Information Superhighway as a result of emerging technologies for multimedia data processing. These libraries will profoundly impact the conduct of business, professional, and personal activity. However, it is not enough to simply store and play back video (as in currently envisioned commercial video-on-demand services); to be most effective, new technology is needed for searching through these vast data collections and retrieving the most relevant selections.

The rise of Digital Video Library is aimed at addressing these needs: Developing new technologies for data storage, search, and retrieval, and embedding them in a video library system for use in education, training, sports and entertainment. Additional applications include the recording and subsequent search of corporate meetings and professional conferences. The new digital video library technology will allow more independent, self-motivated access to information for self-teaching and exploration, which can bring about a revolutionary improvement in the way education and training are delivered.

1.1 Issues about Digital Video Library

Video is not like pure texts or images, it is large in size and contains audio and sequence of images. It will be 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? New techniques are needed to organize and search these vast data collections and retrieve the most relevant selections.

Video causes unique problems because of the difficulties in representing its contents. It is well known that if you electronically scan a page of a book into a raster image, the image will use a significantly greater amount of memory space than an ASCII representation of the original text. 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. The adage “a picture is worth a thousand words” was never more appropriate. Details descriptions of video images can be many thousands of words and even a short video clip description can be massive. However, the alternative of no description leaves even the shortest video clip a black box, giving the user no way to know what is within it. The issues on creating a digital video library and utilizing and exploring the library are also challenging parts in this topic.

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. For example, the MPEG algorithm for video compression was designed to deliver good quality at a very high compression ratio and random access to various points within the sequence. It is a scalable algorithm allowing more quality at the expense of requiring greater bandwidth. The MPEG1 SIF resolution will work for standard CD-ROM bandwidth requirements (1.2 Megabits per second), allowing 352 x 240 resolution at 30 frames per second or 352 x 288 resolution at 25 frames per second, thus delivering VHS quality NTSC/PAL video.

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

A library cannot be very effective if it is merely a collection of information without some understanding of what is contained in that collection. Without that understanding it could take hundreds of hours of viewing to determine if an item of interest is in a 1000-hour video library.

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.

But there are insufficiency of text descriptors, not only the manual preprocessing is tedious and time consuming, but the text descriptors are seriously incomplete to the video itself, identity of persons and objects and cinematic information are almost surely to be left out. Also, text descriptors are biased by the ambiguity of natural language. For example, one indexer may decide to label a particular video segment as occurring in a public vehicle. Another may decide to label the same segment as occurring in a bus. These different tags have implications for later browsing and retrieval of the video.

In order to reduce these limitations, there should be some technology supports to handle the indexing automatically instead of manually.

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 and some alternate representations of the video.

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, which allows later enrichment of the description of the video content. Moreover, the video clip to return to the user can be based dynamically on user and query characteristics, with more annotations allowing more possible segmentations of the video data. In order for a digital video library to be logically segmented as such, the system must be capable of delivering a subset of a movie (rather than having that subset stored as its own movie) quickly and efficiently to the user. Video compression schemes will have to be chosen carefully for the library to retain the necessary random access within a video to allow it to be logically segmented.

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 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

An important part of the function of a digital video library is that it should provide the utility for users to get the information they need easily and efficiently. There are two standard to measure the performance in information retrieval: recall and precision. Recall is the proportion of relevant documents that are actually retrieved. Precision is the proportion of retrieved documents that are actually relevant. These two measures may be traded off one for the other. Returning one document that is a known match to a query guarantees 100% precision, but fails at recall if a number of other documents were relevant as well. On the other hand, returning all of the library’s contents for a query guarantees 100% recall, but fail at precision and filtering the information. The goal of information retrieval is to maximize both recall and precision.

In many information systems, precision is maximized by narrowing the domain considerably, extensively indexing the data according to the parameters of the domain, and allowing queries only via those parameters. For example, a CD-ROM on animals

might fully index the data by genus, species, habitat, diet, growth rate, estimated population, and other biological and environmental factors. The data becomes very useful for its given purpose (e.g. studies on animals), but this approach lost the generality of properties. A user may not be able to find all birds that are blue if color is not one of the attributes which were indexed.

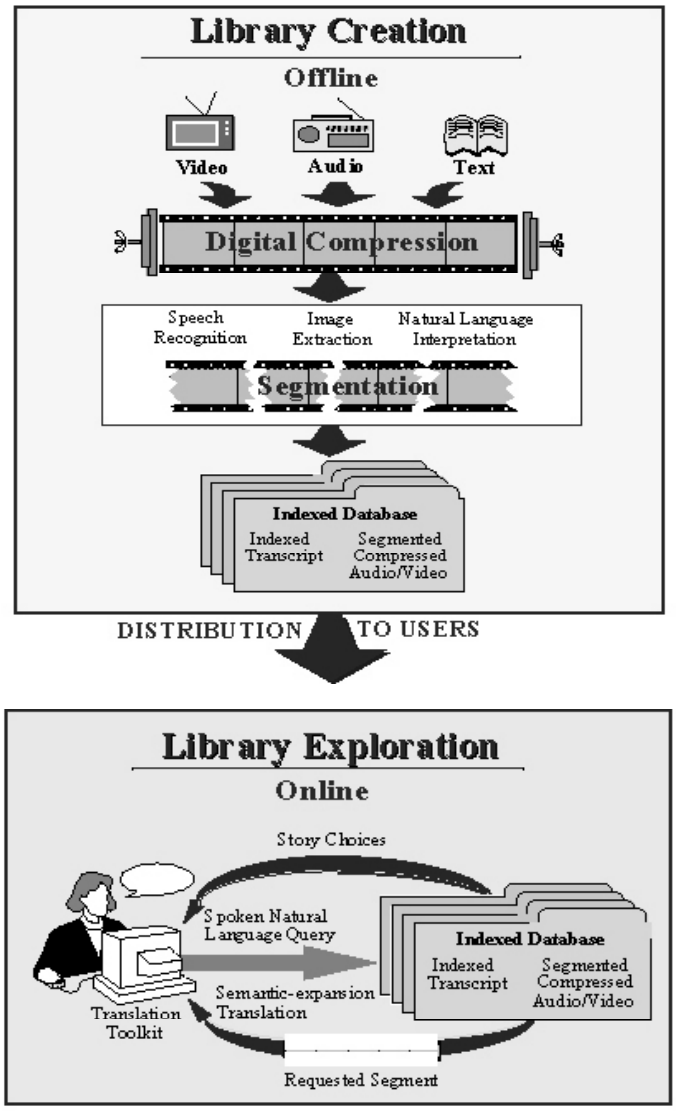
In attempting to create a general purpose digital video library, precision may have to be sacrificed in order to ensure that the material the user is interested in will be recalled in the result set. The result set may then become quite large, so the user may need to filter the set and decide what is important.

In order to let user quickly skim the video objects to locate sections of interest and to aid users in the identification of desired video when multiple objects are returned, information visualization is required. Users often wish to peruse video much as they flip through the pages of a book. Unfortunately, today's mechanisms for this are inadequate. The results from a query may be too large to be effectively handled with conventional presentations such as a scrollable list. To enable better filtering and browsing, features that are important to the user should be emphasized and made visible. What are these features, though, and how can they be made visible, especially if the digital video library is general purpose rather than specialized to a particular domain? These questions return us back to the problem of identifying the content within the video data and representing it in forms that facilitate browsing, visualization, and retrieval.

1.2 Techniques address DVL issues

To establish a digital video library, initially, there are raw materials of videotapes with audio and video part. By using speech recognition and natural language processing technologies, generates a corresponding text transcript of each of the video file automatically. In addition to the generated text scripts, there may be some other information given. Composing these sub-products, the part of text indexing is completed. With the combination use of audio and image analysis techniques, the

segmentation and "paragraphing" of compressed video clips can be done. The whole indexed video database is then built. The creation part of the database is offline. It is just like the printing and binding procedure of book publishing. On the other hand, exploration and retrieval of library resources is in real-time. User makes a textual query or spoken query. The speech recognition is used in the user interface. The natural language analysis technique is used for the searching part. User can online either watch the returned video segment he/she wants or store it. Here is the overview of digital video library system:



Overview of the Informedia Digital Video Library System [Informedia CMU]

Text description of Video

Text description of video is important for retrieval and user's searching. It can be the title of a video, or a brief outline of the video. Close-caption recorders and OCR technology can be used to convert this information into an electronic text representation, suitable for processing and augmentation by the other techniques described in this section. Even if no other automation techniques are used, a human indexer would produce more accurate and complete text indices, or tags, for the video if given this supplemental information rather than just a title or nothing at all.

For better text description of the video, they can be kept in separate fields where the semantics of origin can be preserved, so that a user can filtering out the unwanted fields; also the text description needs to be associated with the video it describes closely such that the descriptions can be used to retrieve more precise, shorter duration video clips.

Speech Analysis

Much of the information conveyed in the audio for a given movie is captured in its close-caption text. Even though much of broadcast television is close-captioned, many other video and film assets are not. More importantly, typical video production generates 50 to 100 more data that is not broadcast and therefore not captioned. Clearly, effective use and reuse of all these video information assets within digital video libraries will require automatic generation of transcripts in order to make the information in the audio more accessible. Speech recognition technology can be applied to automatic transcript generation, but the accuracy of speech recognition is low. It still needs more time to improve the accuracy.

The audio conveys other information besides just dialog. Researchers have made progress in identifying pauses and silence, as well as specialized audio parsers for music, laughter, and other highly distinct acoustic phenomena. This information can supplement the other structured descriptors, and some such as pauses may be especially useful to identify natural start and stop times for video paragraphing as well as allowing for a degree of compression in presenting a “skim” video.

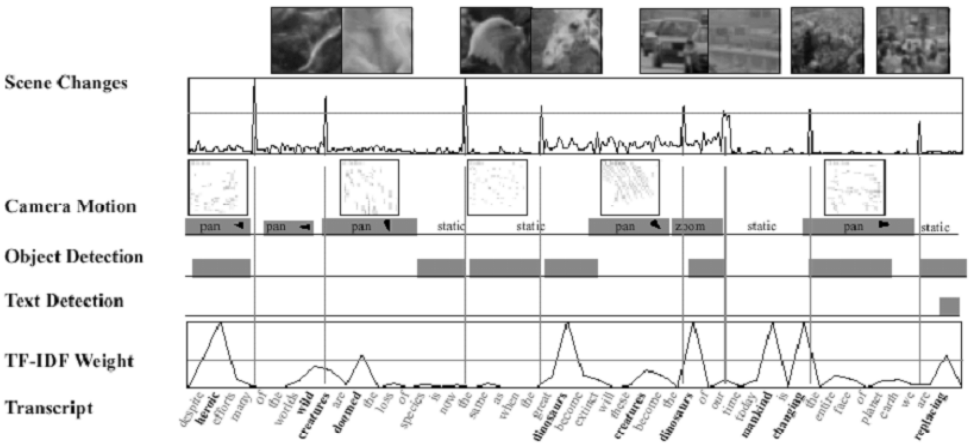
Image Analysis

Image analysis is primarily used for the identification of breaks between scenes and the identification of a single static frame icon that is representative of a scene. Image

analysis can be mainly use in the segmentation of video. Identifying camera pans and zooms, edit effects like fades, cuts, and dissolves, can be useful for segmenting, or "paragraphing", the video into a group of frames when video library is formed. Each group can be reasonably abstracted by a representative frame.

Video is segmented into scenes through the use of comparative difference measures. Images with small histogram disparity are considered to be relatively equivalent. By detecting significant changes in the weighted color histogram of each successive frame, image sequences can be separated into individual scenes. With image analysis, it can interpret camera motion which is one important method for video segmentation and description.

However, a more efficient digital video library needs content-based video paragraphing methods, and image analysis by itself cannot determine all of the information. Some information system developers parse video in a particular domain, such as news footage, to supplement the image analysis with more structure and semantics, while others use human indexers to document video content, including space, time, weather, characters, objects, character actions, object actions, relative position, screen position, and cinematography. The digital video library user is interested in subject or content retrieval, not just "image" retrieval. The subject consists of both image content and textual content (from audio and other sources); the combination specifies the content. Any textual information attached is useful to quickly filter video segments locating potential items of interest. But subsequent query is usually visual, referring to image content. For example, "Find video with similar scenery," "Find the same scene with different camera motion," "Find video with the same person," and so on. Although part of the capability can be realized by content-free methods, such as histogram comparison, it is a long-term challenge to this field of computer vision research on the real solutions in content-based image search.



Natural Language Processing

Natural language queries allow straightforward description of the subject matter of the material desired. An initial query may be textual, entered either through the keyboard, mouse, or spoken words entered via microphone and recognized by the system. Subsequent refinements of the query, or new, related queries may relate to visual attributes such as: “find me scenes with similar visual backgrounds.” Current retrieval technology works well on textual material from newspapers, electronic archives and other sources of grammatically correct and properly spelled written content. However, the video retrieval task, based upon searching errorful transcripts of spoken language, challenges the state of the art. Even understanding a perfect transcription of the audio would be too complicated for current natural language technology.

There are several ways for natural language processing to improve the utility of a digital video library:

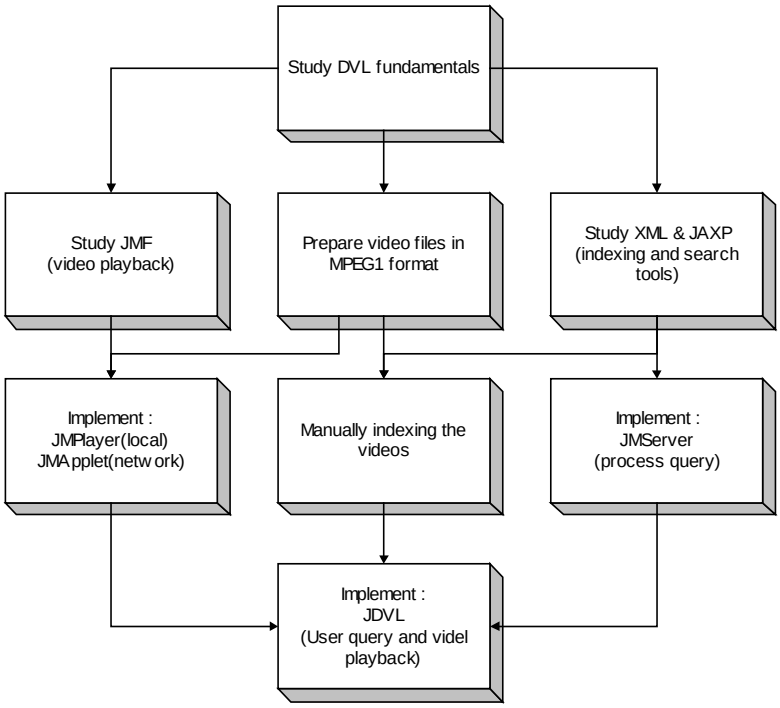
- Query processing: the user must be able to specify a subject or content area for search without having to resort to specialized syntax or complicated command forms.
- Retrieval: once the system has digested a user query, the corresponding text objects must be located, scored, and ranked according to user interest.
- Display: the video segments associated with each relevant text object must be located, and appropriate scene boundaries identified for each video object (visual sentence, paragraph or page) used to generate a menu of visual segments for user selection.

2. Approach

2.3 Focus and Targets

A Digital Video Library involves a lot of advance technologies and also a lot of human resources, so we have to define a more practical target for ourselves as a Final Year Project. We would like to implement our project as the skeleton modules of a DVL, where advance features can be plug into the project when they are ready later on. For these ‘skeleton’ components, video playback and user query are the two core features that we have to implement. Networking capability is another important issue that we have to consider because it will affect the application structure which will affect the on going of the project.

To carry out our project, we first need to understand the fundamentals of a DVL system. Then we have to start to search for a suitable platform, API and other tools that help us to build the platform. In order to implement the DVL progressively, we will first develop a simple application for video playback; then a application that could retrieve and playback video over the network; lastly we will have a server that can process simple user query; and the integration of server with client to a simple DVL model.



workflow of our project

2.4 Equipments & Facilities

A PC(pc89184) act as our server, with Windows NT4.0 English version installed. The PC is equipped 16Gb of harddisk, while 10Gb of the space is used to install another DVL Demo 'Informedia' from the Carnegie Mellon University. Another removable harddisk with Windows98 Chinese version is used to develop the client, which should display Chinese messages.

To prepare the MPEG1 video files from VHS tapes, we use a PC in multimedia lab which equipped with a hardware video capture and encoding card. The video source is mainly daily news from the TV, such that the domain is narrowed to achieve higher precision and easier to be indexed manually.

To enable the video files to be retrieve from the network, we use a http server to serve the purpose. At first we use a freeware SAMBER server, actually the server program is good enough, but for better integration with the system, we change to the IIS4.0 server from Microsoft, which we can monitor the network traffic more easily.

2.5 Programming Tools

Before we can start the implementation, we have to evaluate the cost and difficulties of choosing different programming tools. JAVA is chosen as the programming language of our project as we found it's both convenient and powerful as well in the area of our interest. It comes with some advantages over other programming environment in these aspects:

- Platform independence – Java can be run on any machine any platform where Java Virtual Machine is available. 'Write Once Run Many' capture the idea of Internet revolution, and this is a trend.

- Network ready – Java is designed for network environment. It supports various networking model, from simple connection to complex and powerful distributed computing. This provides a scalable network model for the DVL system; we can extend the library onto a network without too much change in the code. Moreover, we can (and we did) implement the Client as a Java Applet, which can be accessed using web browsers on the WWW. This brings convenience to the users.

- International appeal – Unicode is integrate into the Java. With appropriate fonts installed, different languages can be process and display with no difficulties. And we make use of this feature to display Chinese scripts and messages on a browser of an English version Windows NT platform.

- Modular GUI design – GUI Building is one of the well-known features of Java. The GUI support in Java is simple and efficient, that can free the programmers from low level programming details and can play more effort on the higher-level system architecture etc.

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

- Speed - 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. The Virtual Machine is not efficient and reliable enough, and the worse is that its API may subject to changes in successive versions, which may lead to higher cost for maintaining the program.

But these drawback does not prevent us from using Java, because both disadvantages will be not critical when the project going on – machines will be fast enough, Java will be mature and the virtual machine will be efficient and reliable as well in the future.

Video Playback API - JMF

The Java Media Framework API (JMF) is a collection of classes that enable the display and capture of multimedia data within Java applications and applets. It specifies a unified architecture, messaging protocol and programming interface for playback, capture and conferencing of compressed streaming and stored timed-media including audio, video, and MIDI across all Java enabled platforms. Without this API, it's impossible for us to use Java as our development environment as the very first task of our project is to playback video files (and this is an ESSENTIAL feature!).

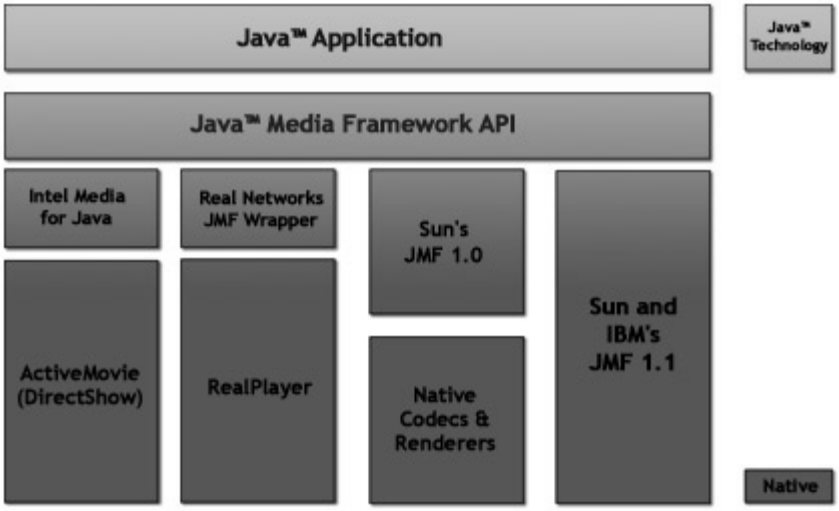
JMF is being released as three packages: Player, Capture, and Conference. And we will focus on the Player API that was used in our project for media playback.

Overview of Java Media Player

Java Media Player provides a platform-neutral framework for building media players. Using Java Media Players, a programmer can synchronize and present time-based media from diverse sources. In addition, developers can create applications using their own media types or take advantage of a wide array of the integrated audio and video media types including QuickTime, MPEG-1, MPEG-2, AVI, H.261, MIDI, AU, WAV, and AIFF files.

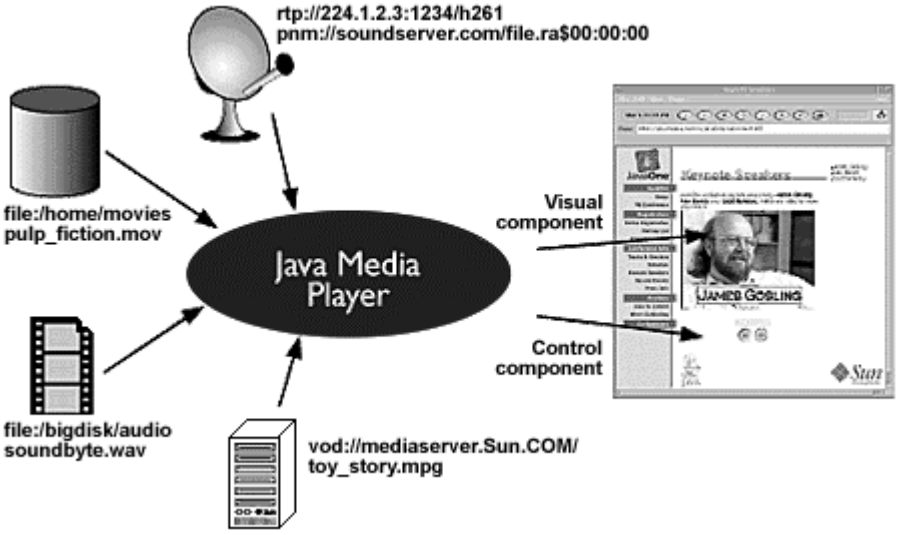
Existing Players on desktop computers are heavily dependent on native code for compute-intensive tasks like decompression and rendering. Some Java Media Players require native code to support the features of a specific hardware device or operating

system, or to maintain compatibility with existing multimedia standards. Since Java accommodates both Java byte code and native methods, developers and users can choose among different Player implementations that use both Java and native objects. Even for some specific media type, like Real Audio or Real Video, they can be handled by JMF as other common media types because the developer Real Networks can (and they do) provide a Java Media wrapper library on top of their development tool kit. With the wrapper library, a programmer can handle all supported media types in the same manner.



Interaction between a Java technology-based application, Java Media Framework API, and native code

Working on the network environment, developers can use the Java Media Framework API to implement a versatile media player that receives and plays multimedia data from sources stored locally or on the network. It allows for cross-platform rendering, control, and synchronization of supported media types independent of the network protocol. And what's important is that the networking model is so simple that programmer can create a player just by giving the URL of media source. This greatly reduces the workload of the programmer. From the client's perspective, the applets built with JMF can be run and does not have to worry about client-side support or plug-ins. This is done by a Web server implementation of JMF, which allows developers to deploy multimedia applets from a web server.



Lastly, with the concept of a media player as a Java Beans technology-based component, the power and facility of the Java Media Framework comes to the reusable, component driven world of beans. This hides the complexity of that underlying code, so that the developers can focus more on the system architecture level.

Media Sources

For starting to use a Java Media Player, we first have to understand the concept of a media source.

A Java Media Player encapsulates its media source. It is designed to present a particular media source, identified by a universal resource locator (URL) and the media data may be obtained from a variety of sources, such as local or network files and live broadcasts. JMF categorize the media sources into 'reliable' and 'streaming'.

- Reliable – the client is guaranteed to receive every packet from a reliable data source such as a local or network file. The established protocols for reliable data are Hypertext Transfer Protocol (HTTP) and FILE.

- Streaming – the data from a streaming media source is not guaranteed to be delivered reliably and clients are expected to recover from gaps in the data. Streaming data sources include broadcast media, multicast media, and video-on-demand (VOD). Some example protocols are the Real-time Transport Protocol (RTP) and the protocol used for VOD.

The degree of control that a client program can extend to the user depends on the type of media source being presented. For example, a reliable media source such as a file

can be repositioned, allowing the user to replay the media stream or seek to a new location in the stream. A broadcast media source, however, is under server control and cannot be repositioned. Similarly, a VOD source might support limited user control, but probably not the degree of control available with a reliable data source.

Players

A Player is a software machine that processes a stream of data, reading data from a media source and rendering it at a precise point in time.

Players share a common model for timekeeping and synchronization. A Player's `media time` represents the current position in the media stream. Each Player has a `time base` that defines the flow of time for the Player. When a Player is started, its media time is mapped to its time-base time. To be synchronized, Players must use the same time base.

A Player's user interface can include both a visual component and a control-panel component. A custom user-interface for a Player can be implemented or the Player's default control-panel component can be used.

A Player must perform a number of operations before it is capable of presenting media. Because some of these operations can be time consuming, JMF allows you to control when they occur by defining the operational states of a Player and providing a control mechanism for moving the Player between those states.

Media Events

The JMF event-reporting mechanism allows your program to respond to media-driven error conditions, such as out-of-data or resource unavailable conditions. The event system also provides an essential notification protocol so that when your program calls an asynchronous method on a Player, it can only be sure that the operation is complete by listening for the appropriate event.

A `Controller` can post a variety of events that are derived from `ControllerEvent`. To receive events from a controller such as a Player, you implement the `ControllerListener` interface. The following figure shows the events that can be posted by a `Controller`.

Controller events fall into three categories: change notifications, error events, and transition events:

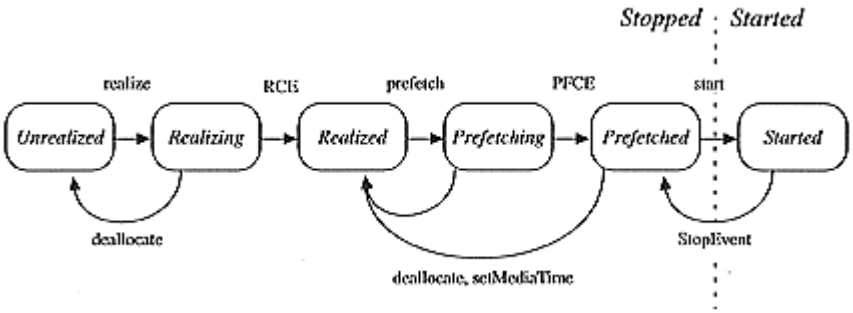
- Change notification events such as `RateChangeEvent` and `DurationUpdateEvent` indicate that some attribute of the Player has changed, often in response to a method call. For example, the Player posts a `RateChangeEvent` when its rate is changed with a `setRate` call.

- `ControllerErrorEvents` are posted by a Player when it has encountered a problem and cannot recover. When a Player posts a `ControllerErrorEvent`, it is no longer usable. You can listen for `ControllerErrorEvent` so that your program can respond to Player malfunctions, minimizing the impact on the user.

- `TransitionEvents` allow your program to respond to changes in a Player's state. A Player posts transition events whenever it moves from one state to another.

Player States

A Java Media Player can be in one of six states. The `Clock` interface defines the two primary states: Stopped and Started. `Controller` breaks the stopped state down into five standby states: Unrealized, Realizing, Realized, Prefetching, and Prefetched.



Progressing of states of a Player

In normal operation, a Player steps through each state until it reaches the Started state:

- A Player in the Unrealized state has been instantiated, but does not yet know anything about its media other than its URL. When a media Player is first created, it is Unrealized.

- When `Realize` is called, a Player moves from the Unrealized state into the Realizing state. A Realizing Player is in the process of determining its resource requirements. During realization, a Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources.

- When a Player finishes Realizing, it moves into the Realized state. A Realized Player knows what resources it needs and something about the media it is to present. Because a Realized Player knows how to render itself, it can provide its visual components and controls. Its connections to other objects in the system are in place, but it does not own any resources that would prevent another Player from starting.

- When Prefetch is called, a Player moves from the Realized state into the Prefetching state. A Prefetching Player is preparing to present its media. During this phase, the Player can preload its media data, obtain exclusive-use resources, and anything else that it must do every time it prepares to play. Prefetching might have to recur if a Player's media presentation is repositioned, or if a change in the Player's rate requires that additional buffers be acquired or alternate processing take place.

- When a Player finishes Prefetching, it moves into the Prefetched state. A Prefetched Player is ready to be started; it is as ready to play as it can be without actually being started.

- Calling `start` or `syncStart` puts a Player into the Started state. A Started Player's time-base time and media time have been mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

A Player posts `TransitionEvents` as it moves from one state to another. The `controllerListener` interface provides a way for your program to determine what state a Player is in and to respond appropriately. This mechanism allows you to manage Player latency by controlling when a Player begins Realizing and Prefetching. It also provides a way that you can ensure that the Player is in an appropriate state before calling methods on the Player.

Creating and Displaying a Player

To Create a Player, you request it from the Manager by calling createPlayer. The Manager uses the media URL that you specify to create an appropriate Player.

JMF specifies the timing and rendering model for displaying a media stream, but a Player's interface components are actually displayed using java.awt. A Player can have two types of components, its visual component and its control components. To displaying a Player's Visual Component, which a Player displays its media data, you

first have to get the component by calling `getVisualComponent` and then add it to the applet's presentation space or application window. The layout of the Player components is controlled through the layout manager. For the Player's Control Component, the procedure is similar to that of a Visual Component, but you get the component by calling `getControlPanelComponent` this time.

Finally, after the interface components are added, methods that control the playback of media (e.g. `start, stop, setMediaTime`, etc) are ready to control the presentation to the user. A simple call to the start can start the playback of media now.

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();
```

Structured Data - 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

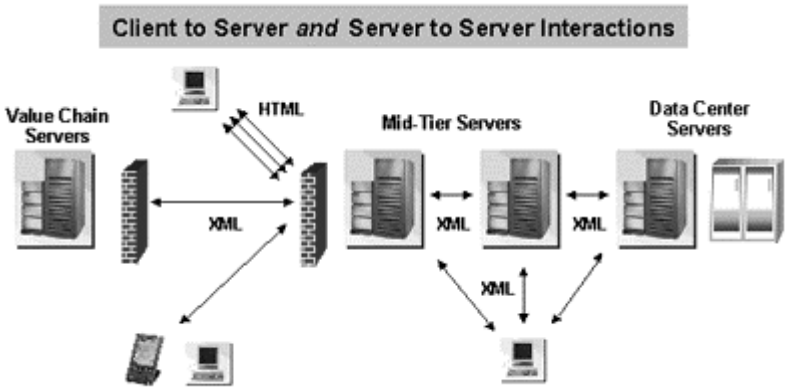
user to look at or utilize the data without relying on the program that produced it. However the primary producer and consumer of XML data is the computer program and not the end-user.

Overview of XML

The Extensible Markup Language (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.

Like HTML, XML makes use of tags and attributes. Tags are words bracketed by the '<' and '>' characters and attributes are strings of the form 'name="value"' that are inside of tags. While HTML specifies what each tag and attribute means, as well as their presentation attributes in a browser, XML uses tags only to delimit pieces of data and leaves the interpretation of the data to the application that uses it. In other words, XML defines only the structure of the document and does not define any of the presentation semantics of that document.

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 front-end. 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 sense, 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.

Here are some new rising applications that make use of XML:

- Traditional data processing – where XML encodes the data for a program to process
- Document-driven programming – where XML documents are containers that build interfaces and applications from existing components
- Archiving – the foundation for document-driven programming, where the customized version of a component is saved (archived) so it can be used later
- Binding – where the DTD or schema that defines an XML data structure is used to automatically generate a significant portion of the application that will eventually process that data

While in this project, we are using XML for the first purpose; we use XML to encode the datasource for the server to process. And we will go into some advantage of using XML as a datasource of an application.

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), 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. 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.

The focus on information rather than documents from XML offers some important and capabilities:

- Inline Reusability –XML documents can be composed from separate entities. And the entities can be included "in line" in a document. The included sections look like a normal part of the document, the whole document can be searched at one time and can be downloaded as one piece. The modularization make it possible to single-source a section so that an edit to it is reflected everywhere the section is used, and yet a document composed from such pieces looks for all the world like a one-piece document.

- Information harvesting – By enabling people to focus on information components that make up documents rather than on the documents themselves, these systems can identify and capture useful information components that have ongoing value "buried" inside documents whose value as documents is limited. That is, a particular document may be useful only for a short time, but chunks of information inside that document may be reusable and valuable for a longer period.

Committed to a persistence layer

XML documents may be stored in files or databases. When stored in files, XML documents are simply plain text files with tags (and possibly DTDs). It is very easy to save your XML documents to a text file and pass the text file around to other machines, platforms and programs (as long as they can understand the data). In case, XML documents (files) can be viewed in a text editor on just about any platform.

Apart from plain text files, XML documents can also committed to a database (relational or object) or any other kind of XML document store. There are commercial products available which allow you to save XML documents to an XML storage layer. What's more is that XML documents can be retrieved from a persistence layer (databases, file systems, XML stores). This lends XML to be used in real world applications where the information being used by different parts of a system is the most important thing.

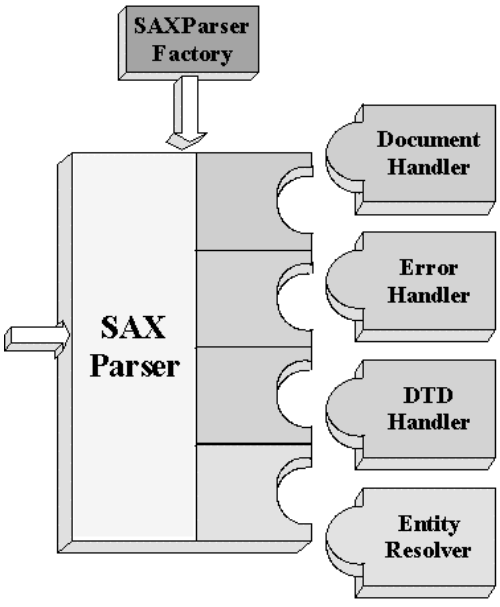
Data management – JAXP

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 turn, creates DOM-compliant `Document` objects. The factory APIs enables 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. Though 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).



Outline of a SAX Parser API

Above is basic outline of a SAX parser. First, the `SAXParserFactory` at the top

generates an instance of the `parser`. Then the XML text is shown coming in to the parser from the left. As the data is parsed, the parser invokes one of several callback methods defined by the interfaces `DocumentHandler, ErrorHandler, DTDHandler, and EntityResolver.` The events handled by each handler is explained below:

- DocumentHandler – Methods like `startDocument, endDocument, startElement`, and `endElement` are invoked when an XML tag is recognized. This interface also defines methods `characters` and `processingInstruction`, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

- ErrorHandler – Methods `error`, `fatalError`, and `warning` are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). Sometimes, the application may need to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.

- DTDHandler – Methods defined in this interface are invoked when processing definitions in a DTD. When the parser sees an unparsed entity or a notation declaration, it does nothing with the information except to pass it along to the application using the DTDHandler interface. That interface defines two methods: `NotationDecl` and `unparsedEntityDecl`. Whether the notation reference is used to describe an unparsed entity or an attribute, it is up to the application to do the appropriate processing.

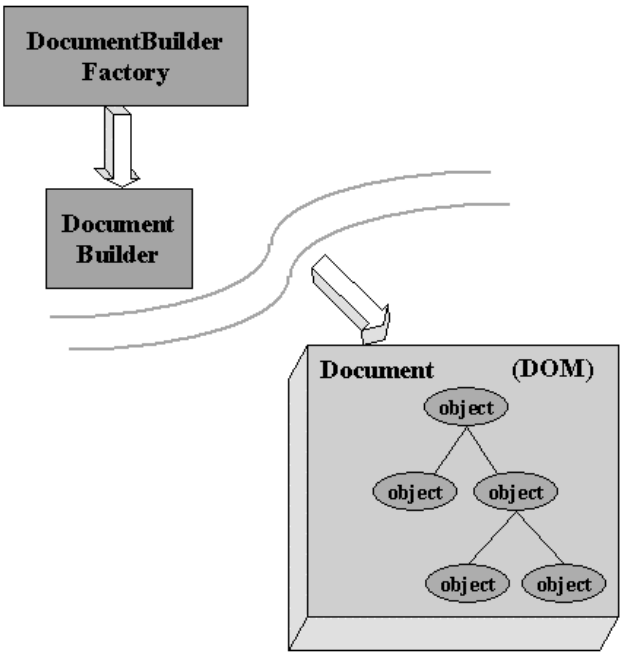
- EntityResolver – The `resolveEntity` method is invoked when the parser must identify data identified by a URI. In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN - a public identifier, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The `EntityResolver` can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.

A typical application provides a DocumentHandler, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to provide an ErrorHandler to report more errors or report them differently.

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.



Function of a DocumentBuilder

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 transversing nodes; or setting and creating attributes etc. This tree model can be even visualized using the JTree interface in javax.swing.

2.6 Java XML Solution Models

XML and Java can certainly be used to create some very interesting applications from application servers to searchable websites. An 'Big picture' is given here to show a possible arrangement of most 'pieces' of objects for a Java XML project.

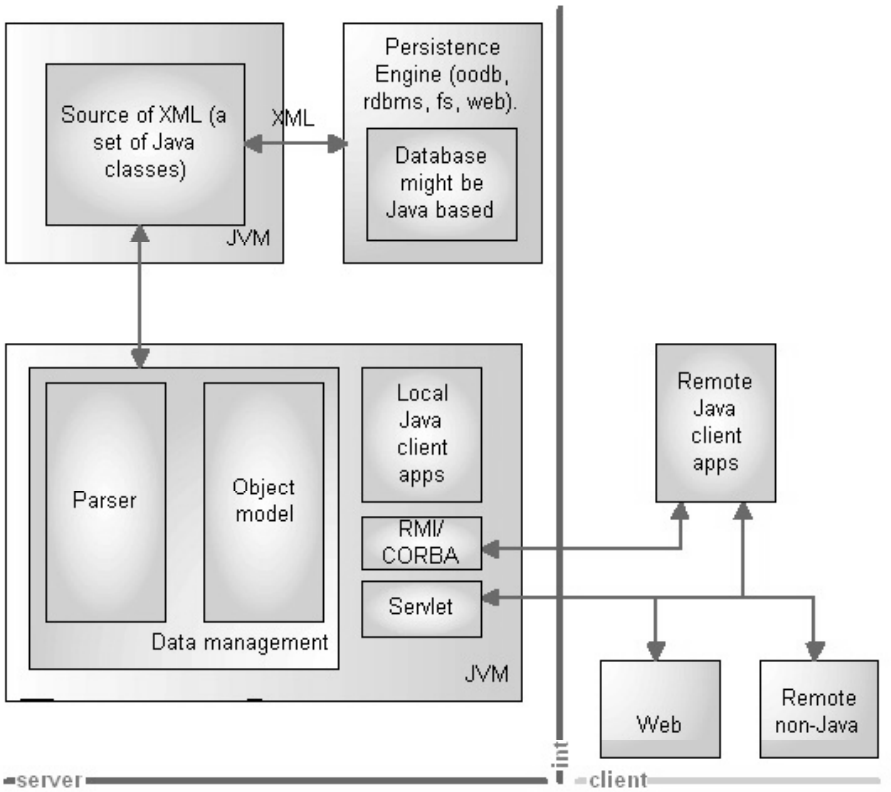


Figure 1 : Where does everything fit?

Components in a Java XML solution

In the model, we have a ‘Source of XML’ which is actually a set of Java classes, that reads a ‘XML’ document from either a file system, database system, or even from the web, and that could be either a XML file or some other custom format which could be convert to XML data by this component. This ‘Source of XML’ serves the application server program with real XML data to process.

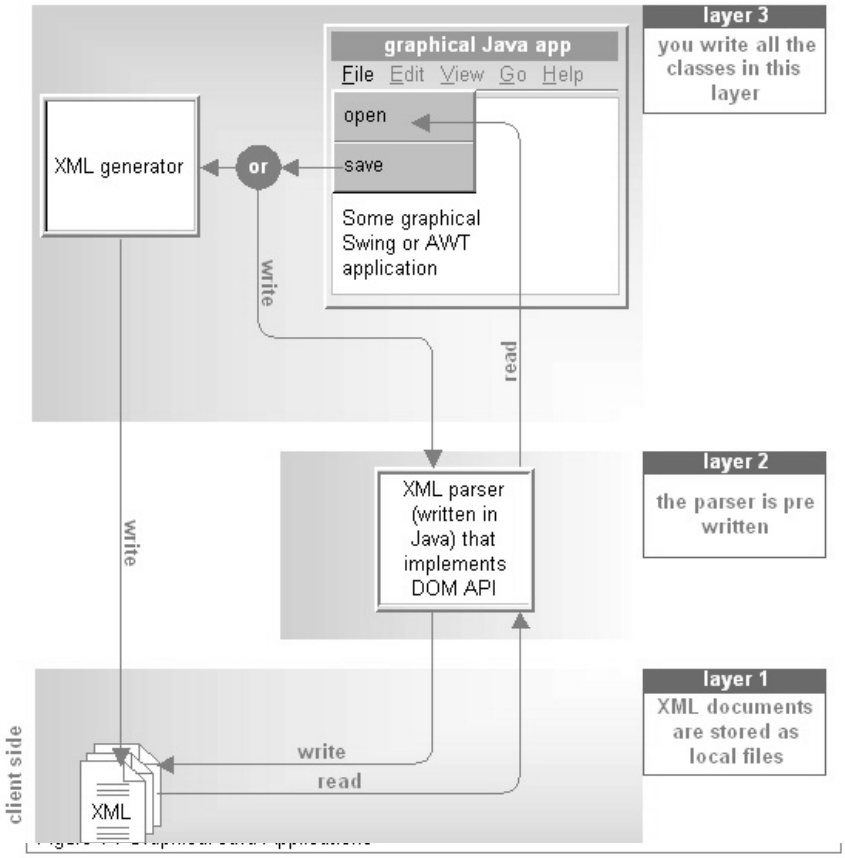
On the application server, it reads XML data from the Source, parse it (by Parser in JAXP!), and keep it as Object model in memory, where various manipulation can be carried out here. An object model is basically a set of classes (and interfaces) that you have to define in order to represent the information in your XML documents. If you use SAX, you have to write a DocumentHandler implementation class that is used by the SAX parser to create your own object model. If you use a DOM parser, then a default object model is provided.

For the client side, if it is a local application, than the entire user interaction is also implemented here; and if it’s a remote client, RMI/CORBA or Servlet will be used for communication through the Internet. And the clients on the remote side can either a Java Application or an Applet, or even just an ordinary Web browser.

There are different types of presentation layers, namely web based and Java based. The Servlet API is perfectly suited for creating web based user interfaces, in addition with a Servlet enabled web server. The Swing API is very good for creating Java based client apps.

These are the ‘big picture’ of a Java XML project, and different variation is possible.

Client side - Graphical Java Applications



Program flow of Graphical Java Applications

The simplest category of XML Java applications is the kind of Java application that stores information in XML documents (files). By using XML to create your own markup languages (i.e. your own file formats for your information) in an open way, you don't have to use proprietary and binary file formats. Using XML over proprietary binary file formats, allows your applications to have immense inter operability across platforms.

Client and Server side - Application Servers

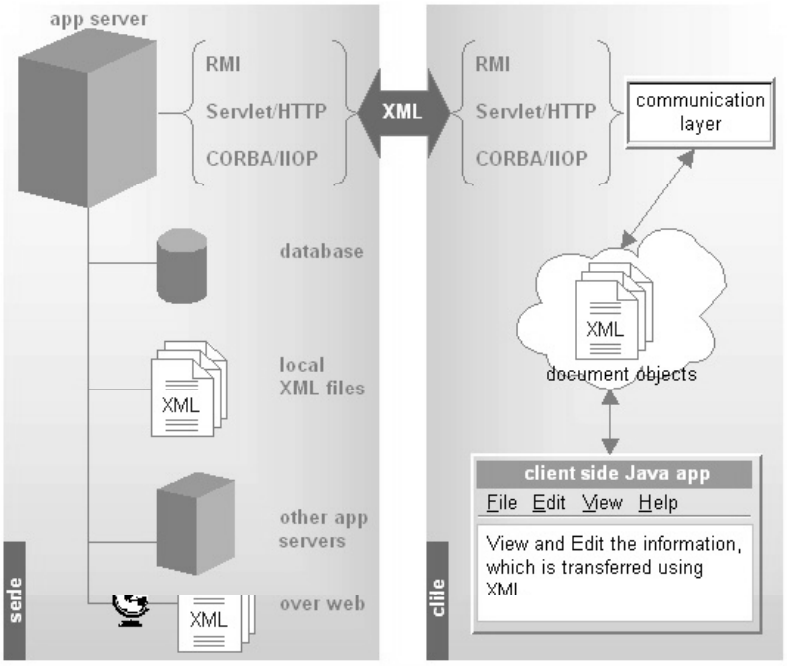


Figure 2 : XML Application Servers

Overview of Java Application Server and the Client

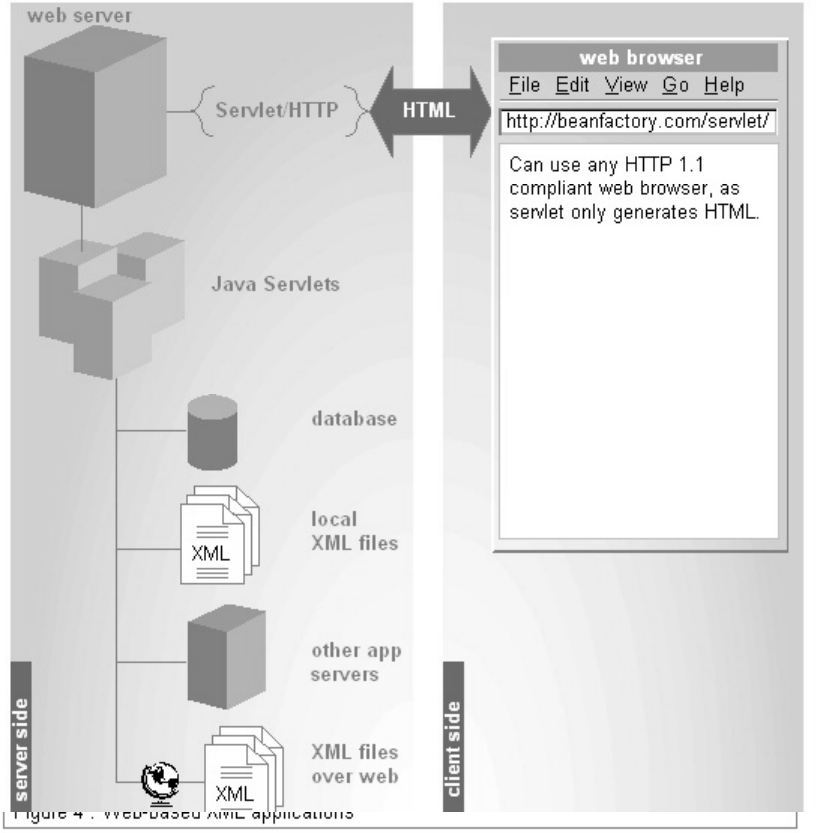
Another category of Java applications called Java Application Servers (app server) and they make good use of XML. Unlike client side graphical Java apps, which are very standalone in their operations, app servers tie many different networked software components together in order to provide information from multiple sources to a set of client side Java apps or web browsers. An app server is actually a conglomeration of several distributed and client/server software systems. Such and app server is actually a system that makes many different networked software systems work together, to process information that comes from various sources, and distribute this information to a set of client apps that maybe running on different devices and platforms.

App servers traditionally give their client apps access to information in remote databases, remote file systems, remote object repositories, remote web resources, and even other app servers. All these information sources don't even need to reside on the machine that hosts the app server. These remote resources may be on other machines on the Intranet or the Internet, and maybe even on different platform, and different kind of app servers.

XML allows these systems to talk with each other without requiring any special binary

information format converters or other service layers to translate between binary formats (for encoding data). Also, since HTTP already supports transmission of plain text, it is completely natural to move XML around using the Hyper Text Transfer Protocol through firewalls and disparate networks. Where it brings simplicity for exchange of data.

Web-based Applications



Overview of Java XML solution in Web-based Applications

Web-based applications are similar to app servers, except for one thing: Web-based applications don't have client apps; instead they use web browsers on the client side. They generate their front ends using HTML, which is dynamically generated by the web-based app. In the Java world, Servlets are best suited for this job.

Web-based apps might themselves rely on another app server to gather information that is presented on the client web browser. Also, you can write Servlets that get information from remote or local databases, XML document repositories and even

other Servlets. One good use for web-based apps is to be a wrapper around an app server, so that you can allow your customers to access at least part of the services offered by your app server via a simple web browser.

In a real world scenario, both a web-based app and app server may be used together, in order to provide your customers access to their information. In an Intranet setting, you might deploy the clients that come with the app server, and in an Internet setting it would be better to deploy a web-based app that sits on top of this app server, and gives your customers relatively limited access to their data over the web via a simple web browser.

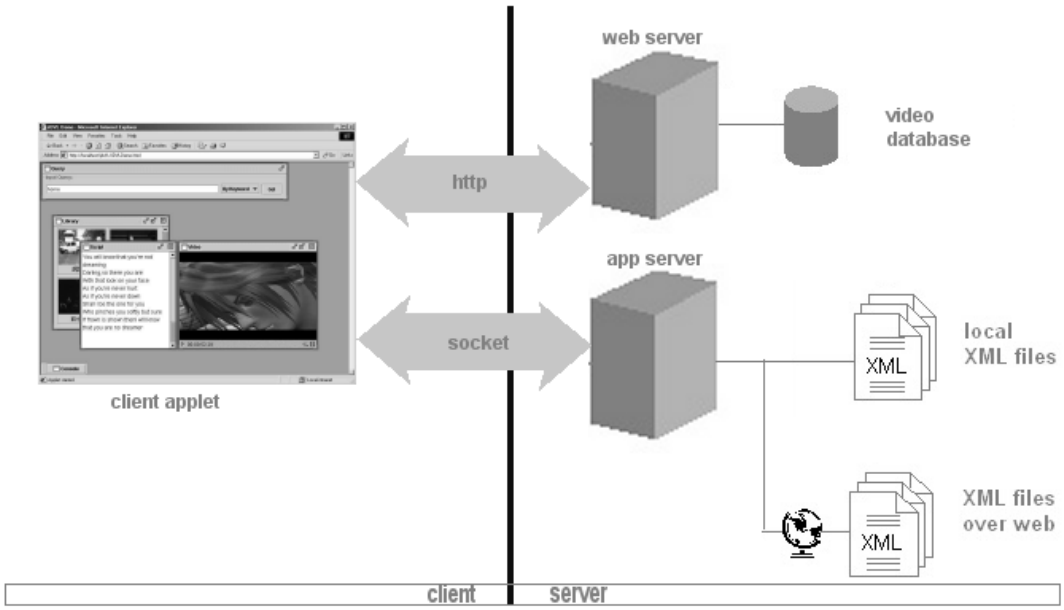
3. Design & Implementation

3.7 System Design

For our project, we target to build a simple Digital Video Library that should have good extensibility for adding advance features in the future. To achieve this, we design our system in modules that could be implement in steps.

Overview

In our design, we have a client/server model over the Internet. There are a number of reasons for we to implement the DVL as a client/server model:



Overview of the JDVL system

- The video database is huge (>1Gb), definitely not suitable to install on the client computer.
- Multiple clients can access the server's video storage concurrently, this increase the video's availability.
- Update of database (add/remove videos) is only done on the server.

And this model also benefits some extension of the system.

- With a network of server, the amount of video storage can increase dramatically, while this is not possible with a client side application.
- Access control can be enforced by centralize administration.
- Different kind of server may exist (different platform, different backend support, specialized video collection, etc.) while the client side can be kept unchanged.

Client Applet

In this model, we have the client as a Java Applet that can be run in a Web-browser that supports Java Plug-in. This can make the client available on machines of different configuration while still keeping the power to perform all it's necessary functions.

As an Applet is only allowed to have socket connection with the machine where it's origin loaded from, so the files are reside on the same machine where the App Server hosted.

The Client Applet connects with the App Server to perform queries and to gather the results through TPC sockets. And then the video is retrieved from a Web Server with HTTP streaming to playback. This is supported by JMF for MPEG1 file type.

App Server

The App Server is a Java application, hosted on a WindowsNT 4.0 Platform. It's responsible for parsing the XML database, and keeping all the records in its tree structure Object Model.

Whenever there is a client connected to the server, a separated thread will be created to handle the request. According to the query type, the thread will search the tree and gather the result to return to the client.

Web Server

Currently we are using the Microsoft Internet Information Services (IIS) 4.0 on WindowsNT 4.0 as our Web server. It's responsible to serve the XML file to the App Server.

It also serves the MPEG1 video files to the Client Applet through Hypertext Transfer Protocol (HTTP). This is feasible in a network of sufficient bandwidth as the JMF use in the Client Applet supports HTTP Streaming for MPEG1 files.

Implementation in Stages

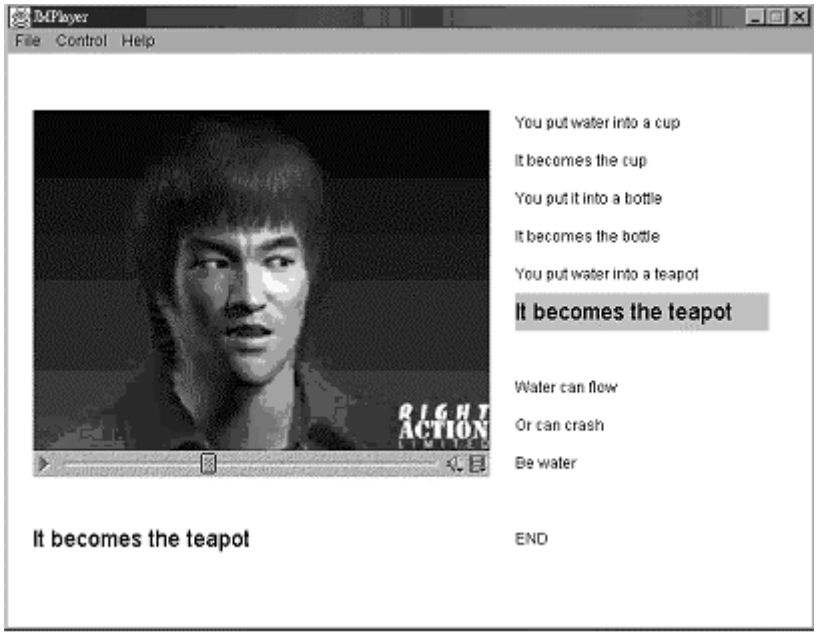
Our implementation was separated in three stages, from a client side application, to a java applet, and finally a client/server system.

Name	Type	Added Features
JMPlayer	Client side application	Playback of local media file
		Display synchronous script
JMApplet	Java Applet	Playback of remote media file through HTTP

		Thumbnail view of video collection
JDVL	Client/Server System	Query processing
		Multiple Document Interface, for open multiple video at same time

JMPlayer

This was the earliest version. It's an application that can playback a local video file and display a synchronous script at same time.



Interface of JMPlayer

In this implementation, the video file and the script file are related by their filename only. Whenever a video file is loaded, the program will look for the same filename with .txt extension as the script file.

JMApplet

In this version, we change the implementation as a Java Applet, which could be run in a browser. This is a step toward our final implementation. When the application becomes an Applet, the user won't need to download and install the program files by himself. This brings a level of convenience over the Java application.

Interface of JMApplet – video playback

With an Applet implementation, we can no longer open local files. Thus we have to switch from opening a local video file to open a remote video file. We do this by using the HTTP streaming feature of JMF, which allows media data streams to be retrieved from the remote computer using Hypertext Transfer Protocol. Therefore we setup a Web Server and place the MPEG1 files in the Web Server to serve this purpose.

Besides switch to the network environment, we also add a “Library view” as a feature. This “Library view” loads a set of thumbnails to represent the videos for user to visualize what is the video about before actually downloading the video. The icons are manually captured from the video segment. The binding of video file, icon file, and the script file are specified in a custom record file. This record file is changed to the XML file in the final version.

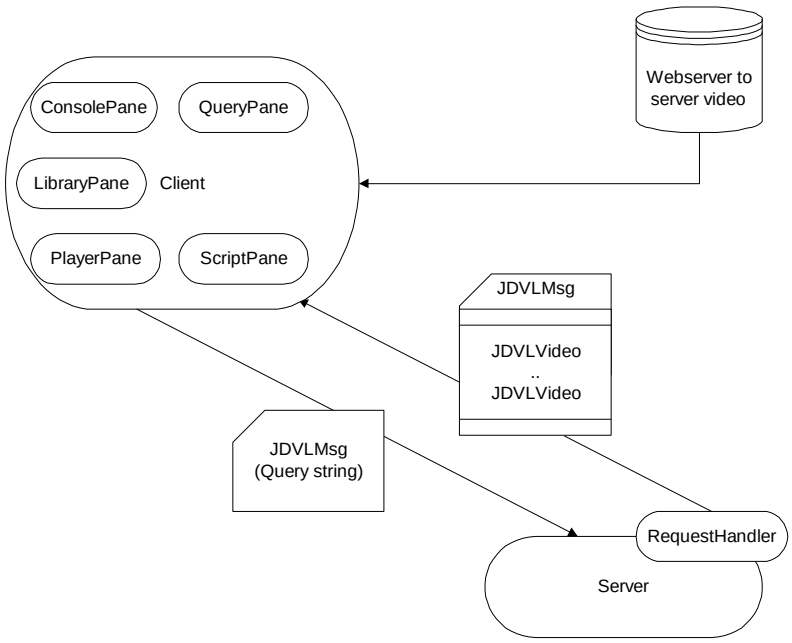
Interface of JMApplet – Library view

JDVL

This is our final version, which changes quite a lot compares with the previous versions. The query server is implemented: users can now search the database by keyword, title, or full script. For the client side, though it is still a Java Applet, but the user interface is rewritten using a desktop view to support more features.

We will go through the JDVL system in the following sections.

3.8 System Modules



Interaction of objects in JDVL

The JDVL system is mainly divided into three kinds of classes: server side, client side,

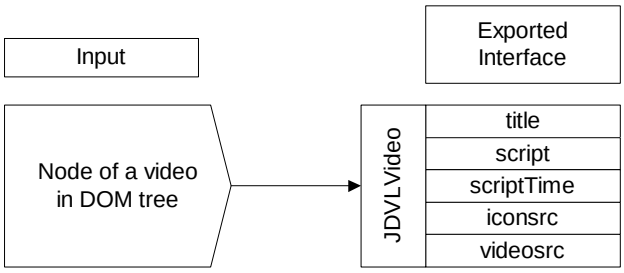
and the network communication class.

4 Module are presented here, namely: JDVLVideo, JDVLMsg, JDVLServer, and JDVLApplet.

In more detail, there is a sub-module RequestHandler for JDVLServer, and 5 sub-modules for JDVLApplet: ConsolePane, QueryPane, LibraryPane, PlayerPane, and ScriptPane.

JDVLVideo

This class encapsulates the information of a video's record in the XML file. It takes the subtree structure in DOM as input parameters that represent a record and convert it to its internal variables. This kind of object is used for the Server to pack the information of a matched record for sending back to the Client. The Client Applet read the attributes of this objects to retrieve the resources (video file, icon file) and also reads the embedded script information. Then this information will used to generate the presentation to the user.



Logic of JDVLVideo

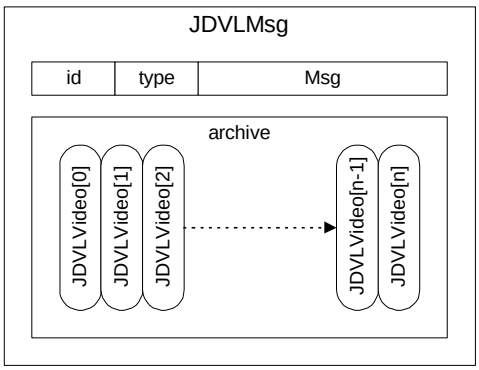
Name	Usage
Public JDVLVideo(Node video)	Constructor. Input parameter is the tree node structure of matched video
Public URL videosrc()	Return the video resource's URL
Public URL iconsrc()	Return the icon resource's URL
Public String title()	Return the title of the video
Public Vector script()	Return the full script of the video
Public int scriptTime(int timestamp)	Return the script index by input the corresponding media time

JDVLMsg

Object of this class is the only object that actually transmitted between the Server and the Client.

On a Query action initiate from the Client, a JDVLMsg will be construct, with the query type and query string as input, also an id to identify the message, and then send to the Server. The Server will then read these parameters to determine the action.

After the Server process the query and a collection of video record is ready, another JDVLMsg will be construct with the same id of the query message, the collection of video record will be added (in form of JDVLVideo instants), and again transmit through the network. Then the Client can get the records inside.

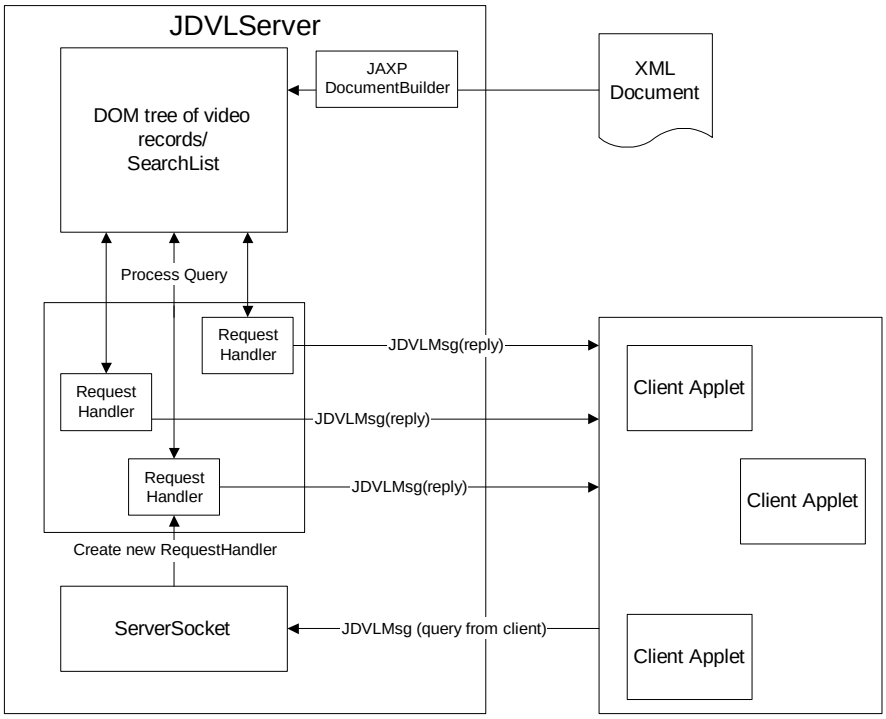


Name	Usage
Public JDVLMsg(int type, int id)	Constructor. type and id are the identifiers of the message
Public JDVLMsg(int type, int id, String msg)	Constructor. type and id are the identifiers of the message. Msg maybe a query string or an error message, depend on the type of the message
Public int id()	Return the id of the message
Public int type()	Return the type of the message
Public String msg()	Return the string message
Public Vector archive()	Return the archive of JDVLVideo carried by this JDVLMsg
Public void add(Node video)	Add an video record to the archive

JDVLServer

JDVLServer is the implement class of the App Server. It implements the main() function for starting the program. When it starts, it will load the XML file database, and use the DocumentBuilder of JAXP to build a DOM. The Object model will be used for the searching purpose. As keyword, script, and title are always used for searching, therefore the server will prepare a search list for them respectively.

After building the DOM, the server will listen on the server socket, whenever a client is connected, and a new thread, RequestHandler, will be create to process the request. By using a thread model, the server can handle multiple Client request as the same time.



Flow of JDVLServer

RequestHandler

RequestHandler is constructed when a Client connects to the Server. It will bind with a socket to communication with client.

As soon as it is constructed and run as a thread, it will read from the socket for a JDVLMsg object, and determined which search list will be used by the type of JDVLMsg. Then it will go through the corresponding search list and compare the item

with the query string, and get the matches.

Finally it will construct a JDVLMsg with same id as the incoming JDVLMsg, add the matched collection in. And send the message back to the client through the connected socket.

Name	Usage
Public RequestHandler(Socket s)	Constructor. Input parameter is the socket connected with the client

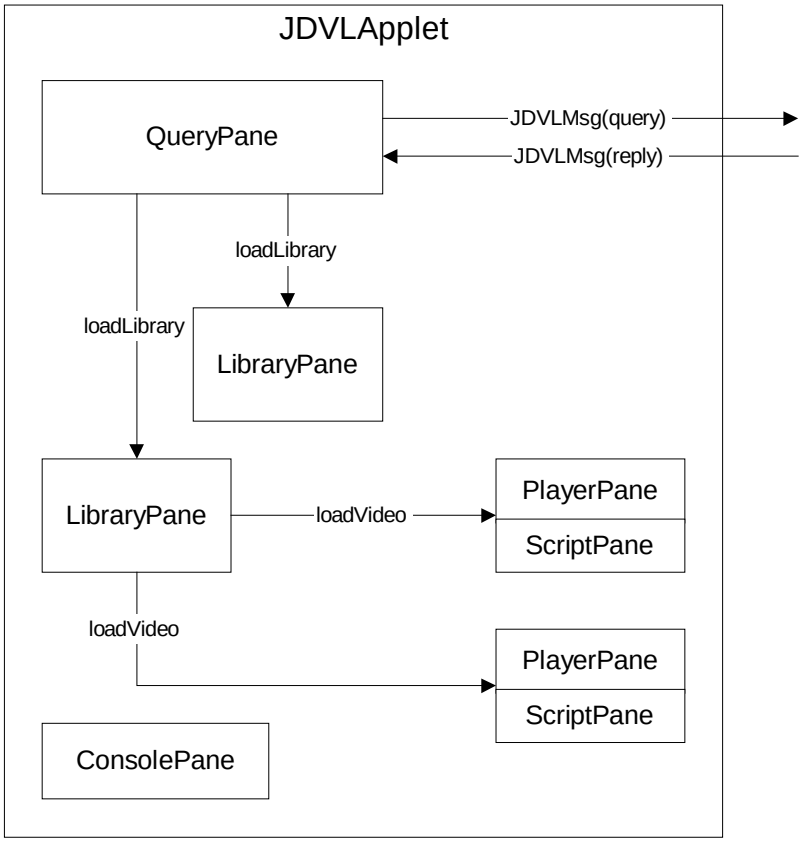
JDVLApplet

JDVLApplet is the implement class of Client Applet. It is the class to be invoked by the HTML file that is loaded by a client browser.

When it starts, it will get the origin of its host, where it could establish socket connection to the App Server. Then it'll examine the system fonts installed to find one that can display Chinese font, such that components that need to display Chinese characters can be assigned with this font.

The last thing it does is to setup the interface for user to interact with. JDVLApplet use a JdesktopPane to give a desktop environment, and the actual functions are carried out in the internal windows. Currently, ConsolePane and QueryPane is default opened in a new started Client Applet.

Name	Usage
public String serveraddr	Address of server. Sub-modules connect to this server when needed
public Font cFont	Chinese font. Interface components which need to display Chinese characters will be set to this font
public void loadLibrary(JDVLMsg searchResult)	Create a library window from a given set of search result
public void loadMovie(JDVLVideo video)	Create a video player window and a script window from a record of video



Sturcture of JDVLApplet

ConsolePane

This class simply displays a textarea and provides methods for others to write a string. It's exist is for debug purpose, and may be disabled for a 'user release'.

Name	Usage
public ConsolePane (JDVLApplet parent)	Constructor. JDVLApplet instant is passed as a parameter for calling and of Applet-wide methods
public void println(String msg)	Write a string on the ConsolePane

QueryPane

QueryPane is responsible for user query input. It consist of a Textfield for input the

query string, a drop-down list for user to choose the type of query (keyword, title, script), and a submit button.

On submit, it'll create a thread to communicate with the server, passing the required parameters as a JDVLMsg. When the server replies, it will call the JDVLApplet's loadLibrary() method to create a library view for the user to choose a video.

Name	Usage
public QueryPane (JDVLApplet parent)	Constructor. JDVLApplet instant is passed as a parameter for calling and of Applet-wide methods

LibraryPane

LibraryPane presents user a library view of thumbnails. Each thumbnail has a short-title and also pop-up hint for the full title, and is associated with the corresponding JDVLVideo object.

After it is constructed, by calling the loadLibrary() method in JDVLApplet will load the result set of JDVLMsg as thumbnails in the library view. This module gets each video record in the JDVLMsg, and then retrieves their icon picture from their iconsrc() property. Then construct a button with the icon and the title, and associate the button with the JDVLVideo object itself.

When use clicks one of the thumbnail buttons, the LibraryPane will call the loadMovie() method in JDVLApplet with the corresponding JDVLVideo object as parameter.

Name	Usage
public LibraryPane(JDVLApplet parent)	Constructor. JDVLApplet instant is passed as a parameter for calling and of Applet-wide methods
public void loadLibrary(JDVLMsg searchResult)	To load the library with result set in the JDVLMsg

PlayerPane

PlayerPane use JMF API to playback video. It constructs a javax.media.bean.playerbean.MediaPlayer and setup the status variables. When the loadMovie() method is called, it will load and play the video specified by videosrc() of the JDVLVideo object.

For adding a synchronous script display, the addScript() method will be called. This

will register a ScriptPane with this PlayerPane, so that the timer will take effect and call ScriptPane's synScript() method at each time interval (0.5 sec by default).

During the video is playing, user can resize the windows, can pause and restart the video, and can mute the sound channel also.

Name	Usage
public PlayerPane(JDVLApplet parent)	Constructor. JDVLApplet instant is passed as a parameter for calling and of Applet-wide methods
public void loadMovie(JDVLVideo video)	To load and play the video specified in JDVLVideo
public void addScript(ScriptPane scriptpane)	Register a ScriptPane with this PlayerPane, such that the synchronous event will be triggered

ScriptPane

ScriptPane is used to display a text script. It uses a JEditPane as the text component, so that it can highlight the script sentence currently playing.

To load a ScriptPane with text, use the loadScript() method, which will read the script() of the JDVLVideo object.

Method synScript() will re-print the script with highlight on the indexed line. It is usually called by a PlayerPane, which hook the media time and the sentence of scripts in synchronization..

Name	Usage
public ScriptPane (JDVLApplet parent)	Constructor. JDVLApplet instant is passed as a parameter for calling and of Applet-wide methods
public void loadScript(JDVLVideo video)	To load the script specified in JDVLVideo
public void synScript(int idx)	To re-print the script with high-light on the indexed line

3.9 Video Preparation and Indexing

Video, Icons, Titles and Scripts

To build the library database, we need video, icons, and scripts. And all of these are prepared manually, which could consider being most time consuming and tedious job of the entire project.

We record the daily news from TV by VHS tapes, and use the PC in Multimedia Lab, which equipped with hardware MPEG encoder card to convert the tapes into MPEG-1 files. Video segmentation is done manual, we cut the files by the natural boundary of each news pieces.

To prepare the icons, we browse the video to select a frame that can represent the subject of the video and scale it to a comfortable size. Similarly, but even more takes more time, we listen to the video and type the scripts by ourselves, digest the subject (to a curtain level) and draft the title for the video.

Obviously, to employ the project in large scale, this part must be automated. And the automation needs more advance technologies like image processing and speech recognition. Hopefully this could be achieved in the near future.

XML Database

The XML file is prepared using the resources mentioned in the previous section. Here we will explain some of the tags used inside:

- video – defines a video entity, all of the child nodes are attributes of this video. and attribute “id” is a unique identifier for the video in the database.
- title – the title of the video clip
- keyword – some predefined keywords are used, in this way, keyword have a more narrow scope of search, but with increased precision. Multiple keywords for a video are allowed.

- script – encapsulate the scripts of the video.
- line – the actual script, the attribute “time” is the media time of the video while this line of script should be in synchronizes.
- iconsrc – the URL of the thumbnail icon of this video
- videosrc – the URL of the video file

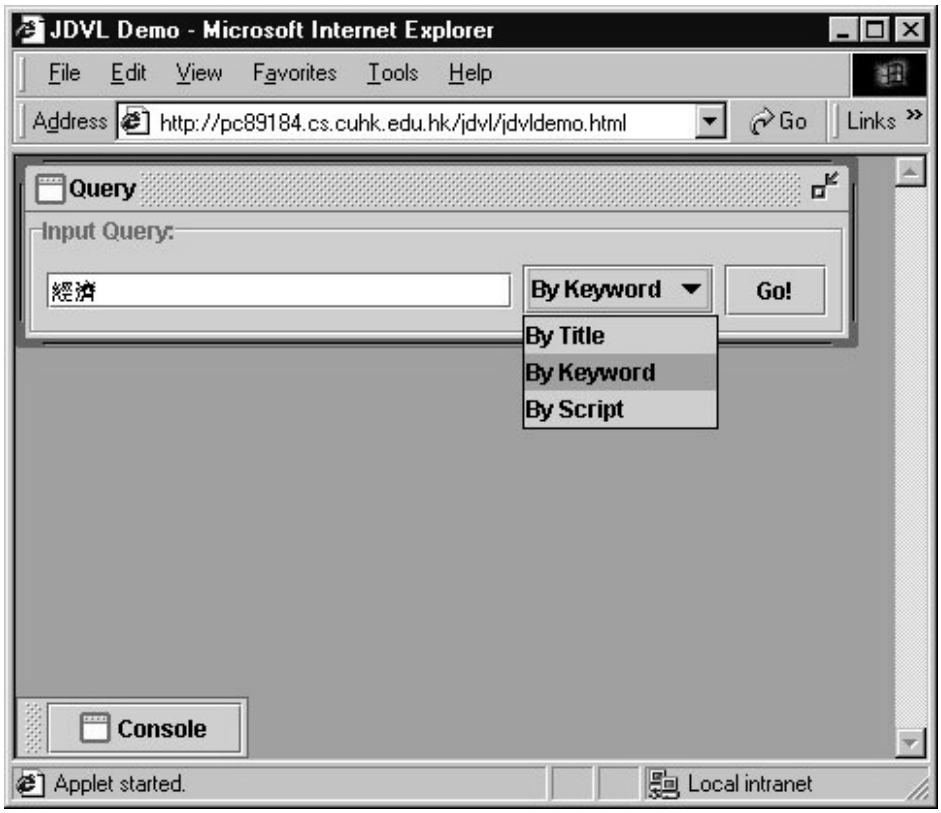
```
<?xml version="1.0" encoding="BIG5" standalone="yes" ?>
<!DOCTYPE DVL (View Source for full doctype...)>
- <DVL id="Multi-Model Digital Video Library LYU9904">
  + <logo>
  - <video id="1">
      <title>國際商業調查機構對本港經濟作出審慎樂觀評估</title>
      <keyword>經濟</keyword>
    + <script>
      <iconsrc>http://pc89184/jmdata/1.jpg</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>
```

The XML database structure

3.10 User Interface

In our Client Applet, a desktop view is used to provide greatest flexibility such that new

features can be added more easily by creating classes that implements the InternalFrame interface.



User Interface - Query Pane

The figure shows our user interface. By default, the Query Pane is always on the desktop, and cannot be dismissed; and the Console Pane is minimized so that it won't bother the user a lot. After the use enters a query and press the "Go!" button on the Query Pane, a Library Pane will pops up, showing the result set of the query in thumbnails.

In the Library Pane, thumbnails are used to represent the video items such that user can quickly locate a video that can interest him, there are also pop-up hints showing the title of the videos for user to know more information about the videos before actually download and view the videos. When the user locates the interested video, he can simply click the associated icon to play the video.

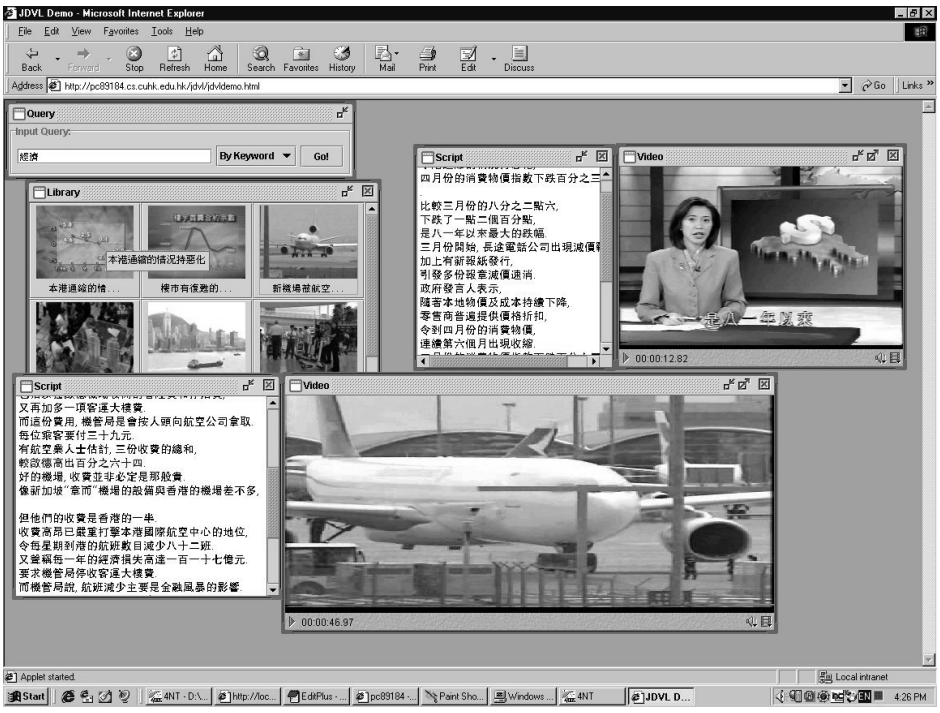
User Interface – Library Pane

When user click one of the icon, a Player Pane and a Script Pane will appear, showing the video and the script of the chosen video respectively. Synchronous script will be displayed during the playback of video; this allows the user to follow the video easily. There is a control bar at the bottom of a Player Pane, user can pause/resume the video playback, can mute the voice channel, and view the information of the video.

User Interface – Video playback and Script Pane

With a multiple-document interface, user can manipulate the Player Pane in anyway they like: minimize, maximize, or resize in any aspect ratio. This allows the user to view the video in a most comfortable way.

Another advantage of using desktop view is that we can open multiple videos at the same time. But this seems not suitable on computers that do not have enough system resources (CPU, ram, etc), where the slow down of system will be very noticeable.



User Interface – Multiple Video playback

4. Discussion

JMF API vs. QuickTime for Java API

Though we cannot test our applet on the Unix workstation due to technical problem (we are not the system admin and can not install some extended API package into the system!), but cross-platform interoperability is still our concern. Using QuickTime for Java, we have media playback feature on Microsoft Windows platform and the Macintosh platform. On the other hand, in the current JMF release, when we use MPEG-1 video format, it can only render on Unix platform and Microsoft Windows platform. Therefore we cannot claim either one have much advantage over another.

But from some discussion on the Internet, we know that QuickTime is more mature product, but is also tougher to use than JMF, it's quite natural to choose an API of lower entrance barrier. Moreover, Apple Computer is also interested in joining the JMF project, so there is a chance that JMF will support MPEG-1 on Macintosh also.

XML vs. Database

Now we have a XML as a plain text file and the records are loaded into the program as DOM, no modification will be made to the XML file, and so we don't have to consider the data consistence, concurrent update, etc. But for a fully automated system, updating is continuous and there are multiple servers connected together (reference the "Java XML Solution" section). Database backend is still a more feasible solution, data integrity, security, etc. may be important also in such a scenario.

Moreover, the system can be scale up easily by integrated with existing databases. E.g. we have a 10Gb video library from the Informedia project, which use ODBC as the database interface. If the schema of the Informedia and our JDVL can be inter-converted, then it's possible to write an XML-ODBC interface to make use of the

video library.

Display Chinese Fonts on English platform

We develop the client program on the Chinese Windows platform in the beginning. When we later port it to the English platform, we have difficult on displaying Chinese Fonts on the English platform.

We have tried the following methods, but all of them are not successful:

- Install Chinese plug-in on the English platform.
- Change the Java VM setting in the file font.properties.
- Changing the code page for the browser that runs the Client Applet.
- Change the coding of the source document.
- Change the font of the interface component.

Lastly, we solve the problem use the last method mentioned, ie. change the font of the interface component. But we do it wrongly at first: we hardcode the fontname which should be able to display Chinese. Actually, the correct way is to let Java to examine all the system fonts and test if the font could display a Chinese character using a method: canDisplayUpTo(chinesesample).

5. Conclusion

In this year of work, we have developed a simple DVL system. In our final JDVL system, we have a Browser-enabled Client interface, which is capable to display Chinese messages on non-Chinese platform; we have a server that use XML to keep the records, and can perform searching based on keyword, title, and the script of videos. We learnt and used many new advancement of programming technology, including JMF, XML, JAXP, etc. All of these are popular topics nowadays; having an understand on them meant to be having powerful tools on constructing many types of poplar applications. For a enterprise scale DVL system, there is still a long way to go, and a lot of advance features can be added: Automation in video segmentation and indexing, natural language processing in query, etc. On the other hand, we know that, as broadband Internet becomes more popular, application for video consumption on Internet will be as common too. Maybe, for some near future, we will even have a DVL client embedded into the Operation System.

Acknowledgement

We would like to thank our supervisor, Prof. Michael Lyu, to give us valuable advice in 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.

Appendix A: Resources

[Informedia]	http://www.informedia.cs.cmu.edu/
[XML]	http://www.xml.org/
[JMF]	http://java.sun.com/jmf/
[JAXP]	http://java.sun.com/xml/
[Multimedia]	http://java.sun.com/features/1999/09/multimedia.html, JMF 2.0 - Multimedia Takes Center Stage
[JMF White Paper]	http://java.sun.com/marketing/collateral/jmf.html, JMF API White Paper
[XML White Paper]	http://java.sun.com/xml/white-papers.html, White Papers about XML Technology
[CO-STAR]	http://java.sun.com/xml/co-stars.html, CO-STARS IN NETWORKING: XML and JAVATM TECHNOLOGIES
[Portable Tech]	http://java.sun.com/xml/ncfocus.html, Portable Data/Portable Code: XML & JavaTM Technologies
[XML and Web]	http://www.developerlife.com/dbsourceintro/default.htm, Introduction to XML, Java, databases and the Web
[Java XML]	http://www.developerlife.com/appoverview/default.htm, What is a Java XML Application Server (and web based app, etc)?

Appendix B: Code Statistic

JMPlayer

Line	Word	Character	File
113	246	2635	JMPlayer.java
58	117	1154	MyAboutDialog.java
44	116	909	MyFileDialog.java
37	99	872	MyIcon.java
125	216	2739	MyInterface.java
81	193	2547	MyLibraryDialog.java
33	39	692	MyMediaPlayer.java
94	143	2591	MyMenu.java
75	117	1485	MyQueryDialog.java
20	29	391	MyTimer.java
72	116	1420	MyURLDialog.java
752	1431	17435	total

JMApplet

Line	Word	Character	File
69	103	1712	JMApplet.java
40	65	700	MMObject.java
24	61	699	MyAboutPanel.java
70	136	1605	MyLibraryPanel.java
144	242	3398	MyPlayerPanel.java

113	334	10742	MySearchPanel.java
460	941	18856	total

JDVL

Line	Word	Character	File
34	59	783	ConsolePane.java
104	185	2490	JDVLApplet.java
72	208	1406	JDVLMsg.java
170	363	4674	JDVLServer.java
91	196	2216	JDVLVideo.java
87	159	1994	LibraryPane.java
121	218	2559	PlayerPane.java
95	178	2419	QueryPane.java
61	110	1729	ScriptPane.java
835	1676	20270	total